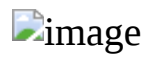


Breaking Ransomware

Explore ways to find and exploit flaws in a ransomware attack

Jitender Narula
Atul Narula





Breaking Ransomware

*Explore ways to find and exploit
flaws in a ransomware attack*

Jitender Narula

Atul Narula

www.bpbonline.com

Copyright © 2023 BPB Online

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the author, nor BPB Online or its dealers and distributors, will be held liable for any damages caused or alleged to have been caused directly or indirectly by this book.

BPB Online has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, BPB Online cannot guarantee the accuracy of this information.

First published: 2023

Published by BPB Online

WeWork

119 Marylebone Road

London NW1 5PU

UK | UAE | INDIA | SINGAPORE

ISBN 978-93-5551-362-5

www.bpbonline.com

Dedicated to

Our parents.

We have always seen God in them.

About the Author and Co-author

- **Author: Jitender Narula** has 20+ years of cyber security industry experience on the projects of AT&T, Citrix, Google, Boeing, SEDENA Mexico, IPolicy Networks (Tech Mahindra now), Conexant, HFCL, iiCyberSecurity, SKY, Delhi Police, Delhi University and Latin American government agencies. He has published technology articles, research and interviews in the area of cyber security on Security Newspaper (www.securitynewspaper.com), NoticiasSeguridad (noticiasseguridad.com) and contributed to the Vishvas News, which is part of Dainik Jagran (Indian Hindi language daily newspaper). Narula has also published a book on Reverse Engineering with the renowned publication house “BPP publications” with the title “Implementing Reverse Engineering”.

Media Appearance:

<https://www.securitynewspaper.com/2021/05/13/step-by-step-process-of-hacking-atms-using-black-box-atm-jackpotting/>

<https://www.vishvasnews.com/english/viral/fact-check-post-claiming-good-morning-messages-can-hack-private-information-is-misleading-cybersecurity-expert-shares-steps-to-protect-your-privacy-online/>

<https://www.iicybersecurity.com/intercept-satellite-communications.html>

<https://www.securitynewspaper.com/2022/07/26/windows-enables-default-account-lockout-policy-for-rdp-remote-desktop-protocol-to-reduce-ransomware-attacks-based-on-brute-forcing-rdp/>

<https://www.securitynewspaper.com/2022/07/04/netherlands-university-paid-e200000-in-bitcoins-to-a-ransomware-gang-now-got-its-money-back-except-now-the-bitcoin-was-worth-e500000/>

- **Co-author: Atul Narula** Co-author Atul Narula has 16+ years of experience in the implementation of cyber security services and solutions for different companies such as Accenture, Hexaware, iiCyberSecurity, Idemia, Air Canada, Telcel, Unisys, Petronic, Sectur and Mexican and other latin american Government agencies. He is proficient in English and Spanish. He has been awarded YouTube Silver Play Button for cyber security channel NoticiasSeguridad Informatica. He has also published articles, research papers on NoticiasSeguridad (noticiasseguridad.com), Exploit One (www.exploitone.com), Cibertip (www.cibertip.com) and contributed to the Televisa Telemundo News and El Pais news.

Media Appearance:

<https://www.telemundo.com/noticias/edicion-noticias-telemundo/ciencia-y-tecnologia/video/este-es-el-alcance-que-podrian-tener-las-fallas-de-seguridad-denunciadas-en-twitter-tmvo10827527>

<https://www.telemundopr.com/noticias/mexico/bandas-criminales-se-retan-por-mensaje-en-redes-sociales-donde-ya-presumian-excentricidades/1837462/>

<https://www.telemundo52.com/fotosyvideos/advierten-sobre-posibles-intrusos-que-invaden-clases-virtuales/2108065/>

<https://www.telemundowashingtondc.com/noticias/local/bandas-criminales-se-retan-por-mensaje-en-redes-sociales-donde-ya-presumian-excentricidades/70509/>

<https://elpais.com/mexico/2021-06-01/un-grupo-de-hackers-de-origen-ruso-secuestra-informacion-confidencial-de-la-loteria-nacional.html>

<https://noticiasseguridad.com/tutoriales/como-robar-de-banca-telefonica-clonando-voces-de-clientes-y-hackeando-reconocimiento-de-voz/>

<https://www.securitynewspaper.com/2021/06/14/how-to-hack-banks-voice-recognition-system-voice-biometrics-with-deepfake-voice-cloning/>

<https://www.exploitone.com/cyber-security/100-urls-and-mitre-attck-techniques-that-you-should-block-in-your-firewalls-to-avoid-conti-ransomware/>

<https://www.cibertip.com/virus/lorenz-ransomware-hackea-la-red-empresarial-a-traves-de-sistemas-telefonicos-voip-asegura-su-servidor-voip/>

About the Reviewer

Rafael Beda is a seasoned cyber security leader and professional with more than 15 years of experience in the industry. He is currently a Senior Cybersecurity professional in the Financial Market, and a Professor and Mentor in Defensive, Operations and Security Management. Rafael holds an MBA in Information Security Management and a postgraduate in Offensive Security.

Acknowledgements

First and foremost, praises and thanks to my Dad, Mom and God, for showering blessings throughout my work to complete the book successfully.

I would like to express my deep and sincere gratitude to Atul Narula, for helping in the writing of this book. I am extremely grateful for what he has offered me. I would also like to thank him for his friendship, empathy, and great sense of humor.

I am extremely grateful to my parents (Ramesh Narula and Mohini Narula) for their love, prayers, caring and sacrifices for educating and preparing me for my future. I am very much thankful to my wife and my son for their love, understanding, prayers and continuing support to complete this book. Also, I express my thanks to Dr. Shilpi Sahi, Dr. Eva Andrea & Om Narula for their support and motivating me throughout this process of writing. Once again, I would like to thank my family for putting up with me while I spent many nights on writing. I could have never completed this book without their support.

Finally, I would like to thank the team at BPB Publications for giving me this opportunity to write my second book with them.

— *Jitender Narula*

Preface

Ransomware has become a major threat to organizations and citizens of any nation. There is a need for experts who can help organizations and national law enforcement agencies in mitigating this risk. To mitigate ransomware risk, internal secrets of ransomware should be known to professionals. This book will help professionals across the globe to get insights of the ransomware working, its architecture and furthermore, with the help of a few examples, it is demonstrated that sometimes a flaw in ransomware design or program can lead to break its functionality. This book is divided into 5 sections. The first section talks about the basic concepts required for further chapters in this book, and covers basics related to ransomware internals, ransomware infection vectors from phishing emails, compromised websites, online advertising, vulnerabilities and many more. The second section walks over the ransomware internal details and how key management is being done by malware writers. The third section talks about the techniques used to perform ransomware analysis on a sample. The fourth section demonstrates how a loophole in ransomware can lead to further break in its functionality. The last section discusses how to respond to ransomware attacks without panicking and steps to be taken in case of a disastrous situation.

Section I: Ransomware Understanding – covers the basic concepts required to understand further chapters in this book, and the basics related to ransomware.

Chapter 1: Warning Signs, Am I Infected? – The Internet has contributed so much in the development of mankind; this was never imagined when computers were born. In those times, when smart geeks were developing good things for mankind, there were some who were involved in developing something to break it. Viruses, Trojans, and Malwares – all these buzz words were trending in the underground world. With time, malwares were weaponized and used as a tool for extortion. In this chapter, we will talk on the basics of ransomware and how hackers are infecting victims worldwide.

Chapter 2: Ransomware Building Blocks – covers details about Ransomware, its internal working and its symptoms. Terms associated with ransomwares such as Bitcoin, Crypto currency, Crypto mining, TOR and other related terms are also explained in this chapter.

Chapter 3: Current Defense in Place – Ransomware is now a major threat in the current cyber security era and it has matured over time and turned into a sophisticated attack against the organizations. There are many proposed solutions in the market to protect organizations user data against ransomware attacks. In this chapter, we will talk about the current solutions in place to protect against Ransomware.

Chapter 4: Ransomware Abuses Cryptography – presents the concepts of cryptography, different types of cryptographic algorithms and how these cryptographic encryption and decryption algorithms are fitting into the current Ransomware architectures.

Chapter 5: Ransomware Key Management – The cores of a Ransomware are its keys, which it uses to encrypt and decrypt user data. It is a prerequisite for a malware writer to implement a strong key management solution in Ransomware. Key management has evolved over time. In this chapter, we will talk about the concept of key management in Ransomware.

Section II: Ransomware Internals – covers the ransomware architectural details and how key management is being done by malware writers.

Chapter 6: Internal Secrets of Ransomware – Cryptographic libraries were introduced to add cryptographic capabilities in the applications or software. There were cryptographic libraries introduced by Microsoft. This chapter focuses on how the Cryptographic libraries are being misused by malware writers to code ransomwares.

Chapter 7: Portable Executable Insides – Critical aspect of cyber security defense involves Ransomware analysis. There are different techniques used by researchers to study Ransomware; most of them include analysis of the Portable executable file. Portable Executable (PE) files are

the important file format in the Windows operating system. Thus, this chapter walks the structure of Portable Executable file format in detail.

Chapter 8: Portable Executable Sections – Analyzing imported functions by a portable executable reveals the nature of the file. This chapter focuses primarily on the import fields in Portable Executable. It explains the important concepts related to Import Directory and Import Address Table. Analysis of some Ransomware code demonstrated that the malware writers have replaced clean DLL with malicious DLL to call export functions of malicious DLL. This chapter focuses primarily on the export fields in Portable Executable. It also explains the important concepts related to Export Directory and Export Address Table.

Section III: Ransomware Assessment – gives special attention to the techniques used to perform ransomware assessment on a sample.

Chapter 9: Performing Static Analysis – Analyzing Ransomware without actually running the ransomware is where static analysis plays an important role in malware analysis. It's the primary step towards malware analysis, which gleans many details about the malware, without going over the code. This chapter talks about different techniques used for ransomware static analysis.

Chapter 10: Perform Dynamic Analysis – Sometimes static analysis does not relieve much information about the malware or ransomware. The only option left in that case is to analyze the malware in running state. For running ransomware, a sandbox environment is required. Dynamic analysis chapter walks over the techniques used to analyze the ransomware in running state.

Section IV: Ransomware Forensics – demonstrates how a loophole in ransomware can lead to further break its functionality.

Chapter 11: What's in the Memory – explains the importance of physical memory forensics during ransomware execution. Key management in ransomware is implemented using symmetric and asymmetric algorithms. Sometimes, in order to study the key management, memory forensics of ransomware process memory is required.

Chapter 12: LockCrypt 2.0 Ransomware Analysis – takes a ransomware sample and does a static analysis on the ransomware. In the preceding chapter, we study how key management plays a vital role in ransomware development. However, in this analysis, we will analyze one ransomware sample and do some static & dynamic analysis to find loopholes in the key management process of ransomware.

Chapter 13: Jigsaw Ransomware Analysis – will take another ransomware sample to analyze its internal architecture covering key management and its working. A small mistake by a ransomware developer can bring down the whole mission of underground mafia. In order to break the ransomware, we either need to break its encryption or extract keys. In this chapter, we will see this ransomware case study.

Section V: Ransomware Rescue – discusses how to respond to a ransomware attack without panicking and steps to be taken in case of a disastrous situation.

Chapter 14: Experts Tips to Manage Attacks – Ransomware has evolved over the decade and it has become the easiest way to fulfill the financial urge of the underground mafia. Shift in ransomware terror from desktop users to organization level has paved a way to new opportunities for malware writers. In the earlier chapter, we spoke about ransomware, but what needs to be done when you in your organization or at home are a target of ransomware attack. What steps should be taken to neutralize the situation?

Code Bundle and Coloured Images

Please follow the link to download the *Code Bundle* and the *Coloured Images* of the book:

<https://rebrand.ly/qezb5ry>

The code bundle for the book is also hosted on GitHub at <https://github.com/bpbpublications/Breaking-Ransomware>. In case there's an update to the code, it will be updated on the existing GitHub repository.

We have code bundles from our rich catalogue of books and videos available at <https://github.com/bpbpublications>. Check them out!

Errata

We take immense pride in our work at BPB Publications and follow best practices to ensure the accuracy of our content to provide with an indulging reading experience to our subscribers. Our readers are our mirrors, and we use their inputs to reflect and improve upon human errors, if any, that may have occurred during the publishing processes involved. To let us maintain the quality and help us reach out to any readers who might be having difficulties due to any unforeseen errors, please write to us at :

errata@bpbonline.com

Your support, suggestions and feedbacks are highly appreciated by the BPB Publications' Family.

Did you know that BPB offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at www.bpbonline.com and as a print book

customer, you are entitled to a discount on the eBook copy. Get in touch with us at: business@bpbonline.com for more details.

At www.bpbonline.com, you can also read a collection of free technical articles, sign up for a range of free newsletters, and receive exclusive discounts and offers on BPB books and eBooks.

Piracy

If you come across any illegal copies of our works in any form on the internet, we would be grateful if you would provide us with the location address or website name. Please contact us at business@bpbonline.com with a link to the material.

If you are interested in becoming an author

If there is a topic that you have expertise in, and you are interested in either writing or contributing to a book, please visit www.bpbonline.com. We have worked with thousands of developers and tech professionals, just like you, to help them share their insights with the global tech community. You can make a general application, apply for a specific hot topic that we are recruiting an author for, or submit your own idea.

Reviews

Please leave a review. Once you have read and used this book, why not leave a review on the site that you purchased it from? Potential readers can then see and use your unbiased opinion to make purchase decisions. We at BPB can understand what you think about our products, and our authors can see your feedback on their book. Thank you!

For more information about BPB, please visit www.bpbonline.com.

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

<https://discord.bpbonline.com>

Table of Contents

Section I: Ransomware Understanding

1. Warning Signs, Am I Infected?

Introduction

Structure

Objectives

Proactive steps

Symptoms

Immediate actions

Disconnect the infected computer

Check the scope of infection

Check which ransomware infected you

Plan for response

Conclusion

2. Ransomware Building Blocks

Introduction

Structure

Objectives

Defining ransomware

Cryptocurrency

Bitcoin (BTC)

Ethereum (ETH)

Cryptomining

TOR (Anonymous browsing)

Ransomware as a Service (RaaS)

How RaaS works

RaaS business models

Affiliate-based

Subscription-based

Licensing for lifetime

Threat actors

[Vulnerability, exploit and payload](#)
[Ransomware attack vectors](#)
[Email](#)
[Phishing scam](#)
[Unsolicited advertisements](#)
[Email spoofing](#)
[Spear phishing](#)
[Exploit kits](#)
[USB and removable media](#)
[Malvertising](#)
[Stages of ransomware](#)
[Conclusion](#)

3. Current Defense in Place

[Introduction](#)
[Structure](#)
[Objectives](#)
[Existing solutions in place](#)
[*Backup solutions*](#)
[*Static or signature-based solutions*](#)
[*Dynamic behavior-based solutions*](#)
[*Entropy-based products*](#)
[*Machine learning products*](#)
[*Honeyfile products*](#)
[User awareness trainings](#)
[Vulnerability management tools](#)
[Cryptographic interceptors](#)
[Analysis of current solutions](#)
[*R-Locker*](#)
[*PayBreak*](#)
[*Redemption-anti-ransomware*](#)
[*Microsoft controlled folder access*](#)
[Conclusion](#)

4. Ransomware Abuses Cryptography

[Introduction](#)
[Structure](#)

[Objectives](#)

[Concept of cryptography.](#)

[Types of cryptography algorithms](#)

[Symmetric algorithms](#)

[Categories of symmetric encryption algorithm](#)

[Asymmetric algorithms](#)

[Hybrid encryption method](#)

[How ransomware abuses cryptographic algorithms](#)

[Conclusion](#)

5. Ransomware Key Management

[Introduction](#)

[Structure](#)

[Objectives](#)

[Key management techniques](#)

[No key management or scareware](#)

[Key on the victim machine](#)

[Key on hacker machine or attacker network](#)

[Conclusion](#)

Section II: Ransomware Internals

6. Internal Secrets of Ransomware

[Introduction](#)

[Structure](#)

[Objectives](#)

[Crypto API](#)

[CryptAcquireContext](#)

[phProv](#)

[szContainer](#)

[szProvider](#)

[dwProvType](#)

[dwFlags](#)

[CryptAcquireContext Example](#)

[CryptGenKey.](#)

[hProv](#)

[AlgId](#)

[dwFlags](#)
[phKey](#)
[CryptGenKeyExample](#)
[CryptGetUserKey](#)
[hProv](#)
[dwKeySpec](#)
[phUserKey](#)
[CryptGetUserKey Example](#)
[CryptExportKey](#)
[hKey](#)
[hExpKey](#)
[dwBlobType](#)
[dwFlags](#)
[pbData](#)
[pdwDataLen](#)
[CryptExportKey PlainTextBlob Example](#)
[CryptExportKey SimpleBlob Example](#)
[CryptImportKey](#)
[hProv](#)
[pbData](#)
[dwDataLen](#)
[hPubKey](#)
[dwFlags](#)
[phKey](#)
[CryptImportKey Example](#)
[Conclusion](#)

7. Portable Executable Insides

[Introduction](#)
[Structure](#)
[Objectives](#)
[Creating a custom binary](#)
[Structure of Portable Executable](#)
[DOS header](#)
[DOS Stub](#)
[PE header](#)
[File header](#)

[Optional header](#)
[SectionAlignment](#)
[FileAlignment](#)
[SizeOfImage](#)
[SizeOfHeaders](#)
[Subsystem](#)
[NumberOfRvaAndSizes](#)
[Data directory](#)
[Section table](#)
[Conclusion](#)

8. Portable Executable Sections

[Introduction](#)
[Structure](#)
[Objectives](#)
[Sections](#)
[.textbss section](#)
[.text section](#)
[.rsrc section](#)
[.rdata](#)
[.data](#)
[Export section](#)
[Import section](#)
[When PE file is loaded in memory](#)
[Conclusion](#)

Section III: Ransomware Assessment

9. Performing Static Analysis

[Introduction](#)
[Structure](#)
[Objectives](#)
[Static analysis](#)
[Phase 1 - Infect host](#)
[Phase 2 – Generate key](#)
[Phase 3 – Encrypt User Data](#)
[Phase 4 – Demand ransom](#)

[Static analysis tools](#)

[PE Studio](#)

[CFF explorer](#)

[Conclusion](#)

[10. Perform Dynamic Analysis](#)

[Introduction](#)

[Structure](#)

[Objectives](#)

[Dynamic analysis](#)

[Disassemblers](#)

[IDA](#)

[Ghidra](#)

[Cutter](#)

[Debuggers](#)

[x32dbg/x64dbg](#)

[Monitors](#)

[Process monitor](#)

[Process Explorer](#)

[Autoruns](#)

[Dependency Walker](#)

[Wireshark](#)

[Burp Suite](#)

[Conclusion](#)

[Section IV: Ransomware Forensics](#)

[11. What's in the Memory.](#)

[Introduction](#)

[Structure](#)

[Objectives](#)

[Static analysis](#)

[Dynamic analysis– Check in the memory.](#)

[Conclusion](#)

[12. LockCrypt 2.0 Ransomware Analysis](#)

[Introduction](#)

[Structure](#)
[Objectives](#)
[Static Analysis](#)
[Dynamic analysis](#)
[Conclusion](#)

13. Jigsaw Ransomware Analysis

[Introduction](#)
[Structure](#)
[Objectives](#)
[Ransomware analysis](#)
[Static analysis](#)
[Conclusion](#)

Section V: Ransomware Rescue

14. Experts Tips to Manage Attacks

[Introduction](#)
[Structure](#)
[Objectives](#)
[Ransomware incident response plan](#)
 [Identification](#)
 [Analysis](#)
 [Determining the strain of ransomware](#)
 [Determining the scope of the ransomware infection](#)
 [Assessing the impact of the ransomware attack](#)
 [Identifying the initial infection vector of the source of the infection](#)
 [Containment](#)
 [Eradication](#)
 [Communication](#)
 [Recovery](#)
 [Post-incident](#)
[Ransomware mitigation strategies](#)
[Endpoint security solutions](#)
 [Endpoint antivirus solutions](#)
 [Endpoint Detection and Response \(EDR\)](#)

Managed Detection and Response (MDR)
Extended Detection and Response (XDR)
Endpoint protection platform (EPP)
Endpoint operating system hardening
Display file extensions
Disable AutoPlay
Disable Remote Desktop protocol (RDP)
Software restriction policies (SRP) And AppLocker
Disable Windows Script Host
Disable Server Message Block (SMB)
Secure domain controllers (DCs)
Restricting the use of PowerShell

Network security solutions

Next Generation Firewall (NGFW)
Network sandboxing
Intrusion detection system (IDS)
Intrusion prevention system (IPS)
Data Loss Prevention
Honeypot
DNS (Domain Name System) Security
Security information and event management (SIEM)
Security Orchestration, Automation, and Response (SOAR)
Zero Trust Network Access (ZTNA)
Stop ransomware attacks
Least privilege account
Patch management
Environment hardening
Enterprise Mobility Management (EMM)
Components inside an EMM Solution

End user level security

Virtualization technology
Disable macros in Office files
Cyber security awareness training
E-mail security
Backup strategy
Auditing backup policies
Encrypting backup data

Immutable storage

Air gap backup

Use the 3-2-1 backup rule

Ascertain backup coverage

Test the backup plan

Conclusion

Index

Section I: Ransomware

Understanding

CHAPTER 1

Warning Signs, Am I Infected?

Introduction

Oh My God! Wired icon and different names.

I am not able to access my files. All my files are coming up with weird icons.

Why am I unable to open my important files?

This happens when you are hit by a ransomware. But what is this ransomware? It begins with the invention of computer virus or malware (new-age term for a family of viruses). Previously, computer viruses were built to disrupt another person's use of an application or system to take revenge, or just for fun. If we take a peek into the past, there are many versions of viruses or worms, like Morris Worm, ILOVEYOU, SQL Slammer, Stuxnet, and Blaster. All of these were developed to disrupt internet users, companies' or countries' computer networks or infrastructure.

With the advancement in software technologies, malware writers gradually realized that malware can be used to earn big bucks. With this, a new variant of malwares came forth, which the internet called ransomware. It is not only a malware but a real pain today for every individual and organization in the world.

Let's look at a simple example. Earlier, banks were robbed physically in a planned manner, but in today's age, banks are forced to send money to robbers over the wire because of ransomware attacks. A small file of few KBs can disrupt entire organizations, and the people behind the attack can earn millions of dollars in ransom. In this chapter, we will learn about the different types of ransomware and understand how we can protect ourselves from them.

Structure

In this chapter, we will discuss the following topics:

- Symptoms
- Proactive steps
- Immediate actions
 - Checking the scope of infection
 - Check which ransomware infected you
 - Plan for response

Objectives

The objective of this chapter is to make the user aware of the ransomware attack. The first and foremost point for the user is to stay calm and not panic. In this chapter, we will talk about the proactive steps to be taken in case you are hit by a ransomware attack. There are cases when a user wants to ensure that the attack on their computer is in fact a ransomware attack. So, we will talk about the symptoms of a ransomware attack, followed by immediate remedial actions that can be taken. After taking remedial actions, we will talk about the scope of infection, along with the steps required to identify the variant of ransomware. Toward the end of this chapter, we will talk about the next plan of action to eradicate the ransomware variant from your computer.

Proactive steps

If you find yourself facing this problem, what is the first step you should take to get yourself out of it? This is what we are going to discuss in this section. If you or your organization are hit by a ransomware, and you are not sure about the problem you are in, then look out for these symptoms to check if it really is a ransomware infection. In this section, we will first talk about the symptoms of a ransomware attack and then walk through the immediate action plan to prevent further infection in the network. Let's look at a few symptoms of ransomware infection.

Symptoms

If you are facing any of the following issues, then you are infected with ransomware:

- Clicking on any file leads to something like what is shown in [Figure 1.1](#):

Figure 1.1: Windows cannot open file

- Some ransomware window pops up on the screen, and you cannot close the window.
- You get an alarming message on the screen to pay ransomware, and that all your all files will be deleted if not paid.
- You get something like a counter, as shown in [Figure 1.2](#):

Figure 1.2: Ransomware screen

- All your files in a folder are not readable, and you see a file in the same folder named “**How To Restore Files.txt**”, as shown in [Figure 1.3](#):

Figure 1.3: How to restore files

All the mentioned situations are indicators of ransomware infections in your computer.

Immediate actions

Infected! What should I do immediately?

If you find that your personal or organization computer is showing any of the symptoms mentioned in the previous section, you have been hit by a ransomware. Following are some immediate remedial actions.

Disconnect the infected computer

The first step you should take is to immediately disconnect the computer from the network. If the computer is connected to the Ethernet network,

then unplug the network cable. If you are connected to a wireless network, switch off the computer wireless interface.

Once you are disconnected from the network, unplug any storage device (external hard drive or USB) connected to the computer. Do not delete any file from the computer. Additionally, don't change the name of any file, as this can harm a posterior tentative to recover the original file

Check the scope of infection

Once you have completely disconnected from the network, you will have to check the amount of damage caused. The damage can be partial or complete on important data. It can probably include devices connected to your computers like external hard disk drives or USB drives. To check the scope of infection caused due to a ransomware attack, we will check all devices in a step-by-step manner:

- First, check your infected computer. Dive into the computer drives and check whether the data in the drives is infected. If you have multiple drives, it might be possible that only your primary drive is infected.
- If you have any network drive mapped on the computer, check the data in those mapped computer drives.
- Check the data in the USB if it was connected to your computer.
- Check the data in the external disk drive if it was connected to your computer.
- If your computer data is in sync with any of the cloud-based storage (like Google drive, Dropbox, Microsoft OneDrive), check the corresponding cloud storage data for any type of encryption.

In the case of infection, our focus is to check the signs of encryption in our system. This will help us in planning further actions.

Check which ransomware infected you

Once we are able to evaluate and confirm the signs of encryptions and the damage caused, it is important to find the type of ransomware we are dealing with. This can be done by analyzing the patterns of infection on files by further doing some research on the internet to get the exact version

of the ransomware. What this means is that the name of ransomware can be identified by the file extension of the encrypted files. There is an easy way out to know the exact strain of ransomware: upload the ransomware note or the sample of encrypted file on the following website:

<https://id-ransomware.malwarehunterteam.com/>

As you can see in [Figure 1.4](#), we uploaded the ransomware note named “**How to restore Files.txt**” on the ID Ransomware website to know the exact strain of ransomware.

Figure 1.4: ID Ransomware

On uploading the ransomware note, we got what is shown in [Figure 1.5](#). In case you are unable to identify the ransomware variant, search for the file extension that the ransomware appended to the files on the internet. You can get some clue about the ransomware variant.

As we can see in [Figure 1.5](#), the ransomware identified is **LockCrypt 2.0**:

Figure 1.5: Ransomware Identified

In our case, we identified that the ransomware is of type **LockCrypt**, which uses AES256 for symmetric encryption and RSA-2048 for asymmetric encryption. We will talk about symmetric and asymmetric encryption in [Chapter 4, Ransomware Abuses Cryptography](#).

Plan for response

Now that we have identified the ransomware strain, it is time to get everything back to normal. Based on the ransomware variant, we will have to check on the internet for any decryptor for that ransomware. Ransomware decryptor is a tiny software or application that will help you to recover all your encrypted files. But before we get on to finding a decryptor, we will have to plan our course of action as listed here:

1. Check your data backup to restore data from the latest backup:
 - a. In this step, we should find all the possible sources where we have backed up our data. This will help us minimize the damage

caused to us, because at this point, we are unsure whether we will be able to remove ransomware to decrypt our data/files.

- b. Don't plug your backup into the alleged infected machine, as depending on the ransomware type, it can encrypt any other type of media (external HDD for example).

Most modern ransomware are programmed to delete the windows shadow files. Shadow files are nothing but the windows restore points. If you are lucky, then your shadow files are untouched by ransomware.

- c. If you have the latest data backup, then you are good to go; recover all your data from the backup. Once your data is restored, you can run multiple scans to remove the ransomware if possible.
2. Find your ransomware decryptor on the internet to decrypt the encrypted files.
- a. Once you know the ransomware variant, there are a couple of antivirus companies that offer free decryptor for ransomware.

i. **Trend Micro Ransomware File Decryptor**

[Figure 1.6](#) shows the Trend Micro Ransomware Decryptor interface:

Figure 1.6: Trend Micro Ransomware Decryptor

To use this ransomware decryptor, you have to select the ransomware from the **Select the ransomware name** list and then select the files or folders you want to decrypt. Trend micro ransomware decryptor can decrypt files encrypted with TeslaCrypt V1/V2/V3/V4, CryptXXX V1/V2/V3/V4/V5, XORBAT, CERBER V1, Stampado, SNSLocker, AutoLocky, BadBlock, 777, XORIST, Nemucod and Chimera.

ii. **McAfee**

[Figure 1.7](#) shows the McAfee Ransomware Decryptor:

Figure 1.7: McAfee Ransomware Decryptor

This tool by McAfee is a decryptor for Tesladecrypt ransomware. Along with this decryptor, McAfee provides other decryption tools for Shade and WildFire ransomware. In this command-line tool, the user will have to provide the directory to search for the encrypted Teslacrypt files. However, this can be quite tedious for a normal user.

McAfee also provides a framework called **McAfee Ransomware Recover (Mr2)**, which is also a command-line tool, but with a bunch of ransomware support, to download decryptor for them. This tool is shown in [Figure 1.8](#):

Figure 1.8: McAfee Ransomware Recovery

The framework is regularly updated by McAfee as the decryption logic and keys required to decrypt files become available.

iii. **Kaspersky ransomware decryptor**

When you search for Kaspersky ransomware decryptor, you will be redirected to the <https://noransom.kaspersky.com/> website, where you can see a list of ransomware decryptors available.

[Figure 1.9](#) shows the Kaspersky Ransomware Decryptor interface:

Figure 1.9: Kaspersky Ransomware Decryptors

These tools are easy to use as users only have to download the decryptor of the particular ransomware and click on **Start scan** in the Wildfire decryptor. This is illustrated in [Figure 1.10](#):

Figure 1.10: Kaspersky Wildfire Decryptor

iv. **ESET Ransomware Decryptor**

To download ESET ransomware decryptor, you have to visit <https://www.eset.com/int/download-utilities/> and find the *Malware Removal Tools* section. At the time of writing this book, the Malware removal tools link redirects you to <https://support.eset.com/en/kb2372-stand-alone-malware-removal-tools>, as shown in *Figure 1.11*:

Figure 1.11: ESET Malware removal tools

As you can see in the previous image, ESSET included ransomware decryptor for TeslaCrypt ransomware.

v. **AVG ransomware decryptor**

You can find the AVG ransomware decryptors on <https://www.avg.com/en-us/ransomware-decryption-tools>. Help provided on the website is pretty good from the end user point of view. AVG provides decryptor for Apocalypse, BadBlock, Bart, Crypt888, Legion, SZFLocker, and TeslaCrypt ransomware, as can be seen in *Figure 1.12*:

Figure 1.12: AVG Ransomware Removal

vi. **Emsisoft ransomware decryptor**

There are a couple of Emsisoft ransomware decryptors available free for download on <https://www.emsisoft.com/ransomware-decryption-tools/free-download>.

There are around more than 40 ransomware decryptor tools available for download like 777, Al-Namrood, Amnesia, Amnesia2, Apocalypse, ApocalypseVM, Aurora, AutoLocky, Avaddon, Avest, BadBlock, BigBobRoss, CheckMail7, ChernoLocker, Cry128, Cry9, CrypBoss, Crypt32, CryptInfinite, CryptoDefense, CryptON, CryptoPokemon, Cyborg, Damage, DeadBolt, Diavol, DMALocker, DMALocker2, Fabiansomware, FenixLocker, GalactiCrypter, GetCrypt, Globe, Globe2, Globe3, GlobeImposter, Gomasom, Hakbit, Harasom, HildaCrypt,

HKCrypt, HydraCrypt, Ims00rry, JavaLocker, Jigsaw, JSWorm 2.0, JSWorm 4.0, KeyBTC, KokoKrypt, LeChiffre, LooCipher, Marlboro, Maze / Sekhmet / Egregor, MegaLocker, MRCR, Muhstik, Nemucod, NemucodAES, NMoreira, NoWay, OpenToYou, OzozaLocker, Paradise, PClock, PewCrypt, Philadelphia, Planetary, Radamant, Ragnarok, Ransomwared, RedRum, SpartCrypt, Stampado, STOP Djvu, STOP Puma, SynAck, Syrk, TurkStatik, WannaCryFake, Xorist, ZeroFucks, Ziggy, Zorab, and ZQ.

[Figure 1.13](#) shows the Emsisoft Ransomware Decryptor interface:

Figure 1.13: Emsisoft Ransomware Decryptor

All these decryptors have great graphic user interfaces.

vii. **Avast ransomware decryptor**

Avast is known for its free antivirus solution for end users. It also provides ransomware decryptors on <https://www.avast.com/en-in/ransomware-decryption-tools>. Decryptor is shown in [Figure 1.14](#):

Figure 1.14: Avast Ransomware Decryptor

Avast provides ransomware decryptors for many ransomwares, like AES_NI, Alcatraz Locker, Apocalypse, AtomSilo & LockFile, Babuk, BadBlock, Bart, BigBobRoss, BTCWare, Crypt888, CryptoMix (Offline), CrySiS, EncrypTile, FindZip, Fonix, GandCrab, Globe, HermeticRansom, HiddenTear, Jigsaw, LambdaLocker, Legion, NoobCrypt, Prometheus, Stampado, SZFLocker, TargetCompany, TeslaCrypt, Trolldesh / Shade, and XData.

viii. **BitDefender ransomware decryptor**

To download ransomware decryptors from BitDefender, you can visit <https://www.bitdefender.com/blog/labs/tag/free-tools/>. For some ransomware, they provide a detailed

technical analysis along with the decryptor, as shown in [Figure 1.15](#):

Figure 1.15: BitDefender Ransomware Decryptors

The Decryptor comes with an easy-to-use graphical user interface, as shown for REvil ransomware decryptor from BitDefender in [Figure 1.16](#):

Figure 1.16: BitDefenderREvil Ransomware Decryptor

Now, from the list of ransomware decryptors, we will move on to a situation wherein you got hit by an unknown ransomware.

3. If you are unable to find the decryptor for your ransomware, there are three options:
 - a. Do not pay the ransomware and your all data will be lost.
 - b. Negotiate and pay the ransomware to retrieve your data.
 - c. Break the ransomware if possible. For this, you will have to understand the working of the ransomware and use reverse engineering techniques, which we will cover in the subsequent chapters.

Conclusion

In this chapter, we walked through the proactive steps to be taken in case you are hit by a ransomware attack. We also covered the symptoms of a ransomware attack, followed by some immediate remedial actions required in case you are affected. We learned about the different variants of ransomware and the steps followed to identify the variant of ransomware. Finally, we talked about ransomware eradication plan, wherein we saw that many antivirus companies are offering free decryptors for ransomware victims.

In the next chapter, we will cover ransomware and its building blocks in further detail. Also, we will understand the terms associated with

ransomware, from cryptocurrency and anonymity to a **Ransomware as a Service (RaaS)** model.

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

<https://discord.bpbonline.com>

CHAPTER 2

Ransomware Building Blocks

Introduction

In the previous chapter, we learned about the warning signs of ransomware. But what exactly is this ransomware, and how is it different from other malwares? It is not as complex as it sometimes seems. To understand any complex situation, it is always better to understand the building blocks and the way they communicate with each other. From the security point of view, if you need to find bugs or a hack in a complex system, it is necessary to have internal knowledge of the building blocks of a complex system and its internal working. This approach of breaking a complex system in small blocks really helps in finding the bugs or hacks in a system.

Your computer seem from the outside is somewhat complex, as it can do a whole range of functions. If you really want to understand how your computer works from a hardware perspective, you can disassemble it and look at the unitary pieces: Motherboard, Processor, RAM and Power Supply. Similarly, to understand ransomware and its working, we need to understand its building blocks and the internal working of these building blocks.

Structure

In this chapter, we will discuss the following topics:

- Defining ransomware
- Cryptocurrency
 - Bitcoin
 - Ethereum
- Cryptomining
- TOR (Anonymous Browsing)
- Ransomware as a Service (RaaS)

- How RaaS works
- RaaS business model
- Threat actors
- Vulnerability, Exploit and Payload
- Ransomware Attack Vectors
- Stages of ransomware

Objectives

The objective of this chapter is to understand the working of ransomware by breaking it down into different components. There are different concepts behind the workings of ransomware; we will talk about cryptocurrency and cryptomining, along with anonymous browsing. We will also talk about the concept behind **Ransomware as a Service (RaaS)** and get you familiarised with terms like vulnerability, exploit and payload. Additionally, we will understand these terms from the layman's point of view. Finally, towards the end of this chapter, we will cover ransomware attack vectors and the different stages of ransomware infection.

Defining ransomware

Any bad program that hinders the working of a computer is known as a virus in the early times. But with the evolving threat environment, several types of computer viruses were developed to perform specific tasks and target specific types of systems, companies or even persons. All these different types of bad behavior programs were put under one umbrella, known as malware.

Malware is a malicious program or software intentionally programmed to harm a computer, server or network. There are various types of malwares, ranging from computer viruses, and Trojan horses to worms, spyware, ransomware, adware and key logger. The following figure shows the different types of malware:

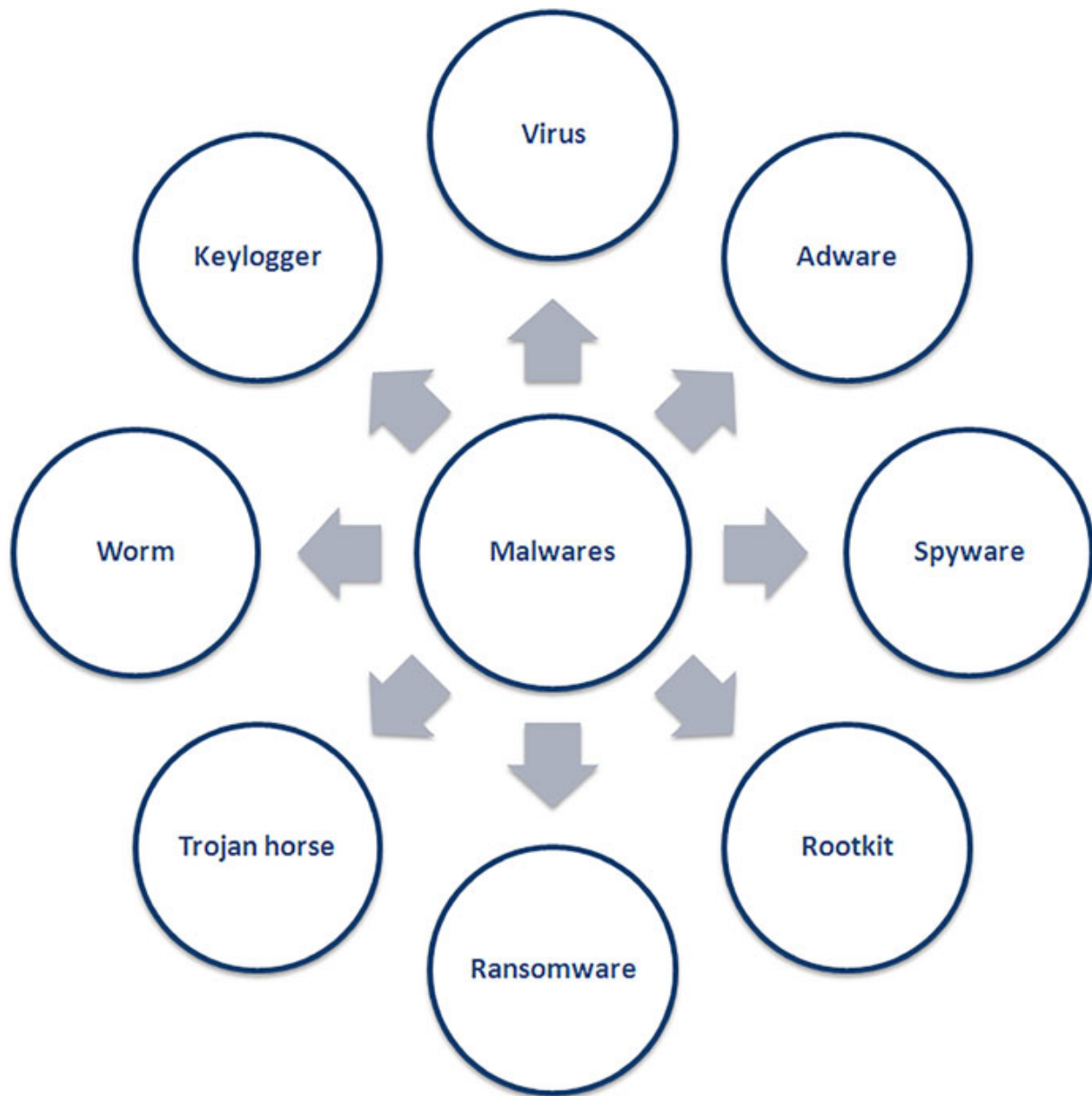


Figure 2.1: Types of Malwares

Ransomware is a kind of malware that is intentionally programmed to encrypt the victim's files or data and then demand a ransom to decrypt them. The problem with ransomware is so severe that if the victim does not pay the ransom on time, the victim's data is left encrypted forever or deleted. Moreover, in recent times, it is seen that the victim's data is sold in the underground forums or in the black market if the ransom is not paid on time. In this chapter, we will study the terms associated with ransomware and understand how ransomware works. The people who develop ransomware

use different vectors to infect victims' machines. They range from exploit targeted to unpatched machines, phishing emails, hacked or compromised websites, free software and poisoned advertisements. Once the ransomware infects a system, it encrypts all user data, including data on the network mapped drives.

Ransomware are programmed to display a screen to the victim, asking for instructions to pay ransom in cryptocurrency. It is also programmed to display a timer (like a timer on a bomb) for the victim to pay ransom on time. If the ransom is not paid on time, there are chances of the ransom amount going up. On the other hand, if the ransom is paid on time using the cryptocurrency account, ransomware developers provide a software called **decryptor**, which is used to decrypt all encrypted files on the computer or network. We will talk about each component of decryptor in the subsequent sections.

Cryptocurrency

Have you ever wondered what happens when you transfer some money from your bank account to your friends or family? Suppose person A transfers 100 dollars to person B. On receiving the money, person B can check with their bank whether person A has transferred 100 dollars or not. Person B, on the other side, can check their bank by visiting the bank branch office or by logging into their account using the bank's online portal. As we know, everything in this process is transparent to the bank and the sender/receiver. It is transparent from the side of the person who is sending the money and also from that of the person who is receiving the money in our traditional banking system, as shown in the following figure:



Figure 2.2: Traditional Banking

This process is completely anonymous when we use cryptocurrency or money. There are many cryptocurrencies available today, like Litecoin, Monero, Ripple, Ethereum and Bitcoin.

The popular among all cryptocurrencies is Bitcoin. In today's banking system, every user is assigned a bank account in which they keep their money. In case of any illegal fund transfer, a **law enforcement agency (LEA)** can request a bank to reveal the details of the bank account owner (like name, address, phone number and other past transactions). This process is completely anonymous when dealing with cryptocurrencies. Every cryptocurrency owner or user is assigned a unique identifier known as the cryptocurrency address. A cryptocurrency owner saves their cryptocurrencies in a cryptocurrency wallet. When person A transfers some cryptocurrencies from their wallet to person B's wallet, that transaction is completely anonymous. Being anonymous means there is no way of knowing about person A or person B's identity. In cryptocurrency, there is no physical bank to enquire about the user, as shown in the following figure:



Figure 2.3: Cryptocurrency Transaction

This complete anonymity feature is exploited by cybercriminals. With this, money can be transferred anywhere in the world anonymously. Ransomware developers use cryptocurrency to demand ransom from the victim. As we discussed earlier, ransomware, after hitting a target, flashes a message on the victim's computer, asking them to pay ransom to a specific address using cryptocurrency. This cryptocurrency address belongs to the hacker or a cybercriminal group. In the next section, we will discuss the two most common crypto currencies: Bitcoin and Ethereum.

Bitcoin (BTC)

Bitcoin is digital currency that is decentralized and can be transferred on the Bitcoin network. Bitcoin started in 2009 and rumored to be created by a mysterious person name *Satoshi Nakamoto*. Bitcoins are generated as a reward; this process is known as Bitcoin mining. Bitcoins are exchanged for goods and services and mostly used to perform illegal transactions.

Ethereum (ETH)

The second most popular digital currency is Ethereum. It started in 2015 by *Vitalik Buterin* and *Gavin Wood*. The core of Ethereum is its blockchain network. Blockchain is distributed public ledger and decentralized. It is a distributed public ledger, meaning that anyone participating in the Ethereum network gets the same copy of the ledger, enabling them to see all transactions. It is decentralized, meaning there is no central entity to manage network. Instead, it is managed by distributed ledger holders.

Cryptomining

As the name suggests, cryptomining is a process of mining cryptocurrency. This mining is done using software installed on your computer. Software generates cryptocurrency by participating in performing cryptographic calculations, as shown in the following figure:

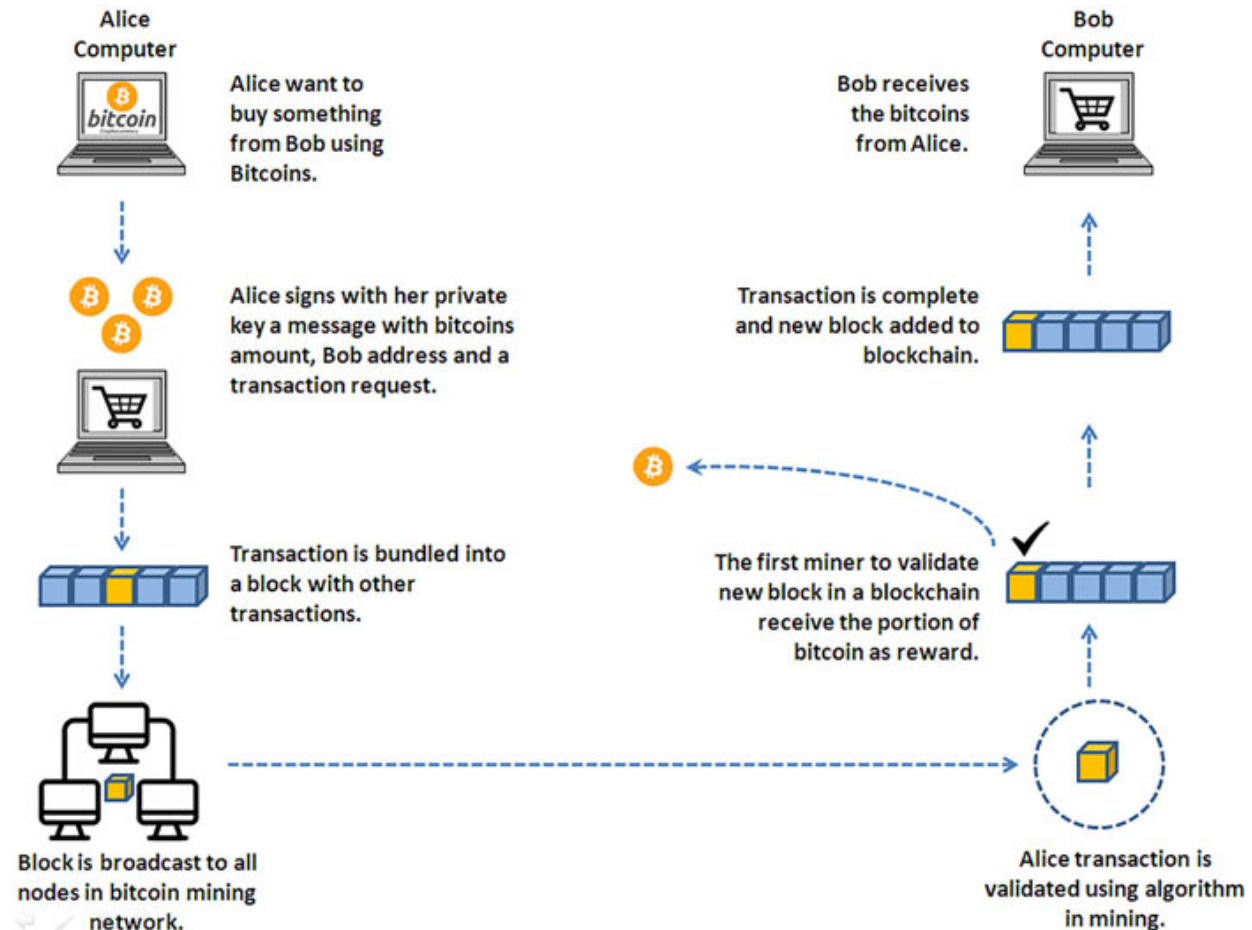


Figure 2.4: Cryptocurrency Mining

The software installed for crypto mining utilizes your computer's CPU and GPU to the fullest power and consumes a lot of electricity. It also slows down your computer. You will be surprised to know that cybercriminals are so smart in making money that they sometimes install crypt miner software as a backdoor with ransomware. Once the ransom is paid to remove the ransomware and decrypt your data, crypto miner software starts in the background to start cryptomining. This can be carried for days, weeks or even months.

TOR (Anonymous browsing)

TOR was originally developed by U.S Naval Research Laboratory and **Defense Advanced Research Projects Agency (DARPA)**. It stands for **The Onion Router (TOR)**. This is a free network to browse the internet anonymously. Many activists, journalists, and whistleblowers use this to communicate with each other for agendas like anti-government talks and many others. To use this, you will have to install a TOR browser from the <https://www.torproject.org/> website. Once the browser is installed, all your browsing traffic is masked to preserve your privacy and anonymity. Behind the scenes, all your browsing traffic is bounced from the chain of randomly chosen computers. These randomly chosen computers are called relay and are illustrated in the following figure:

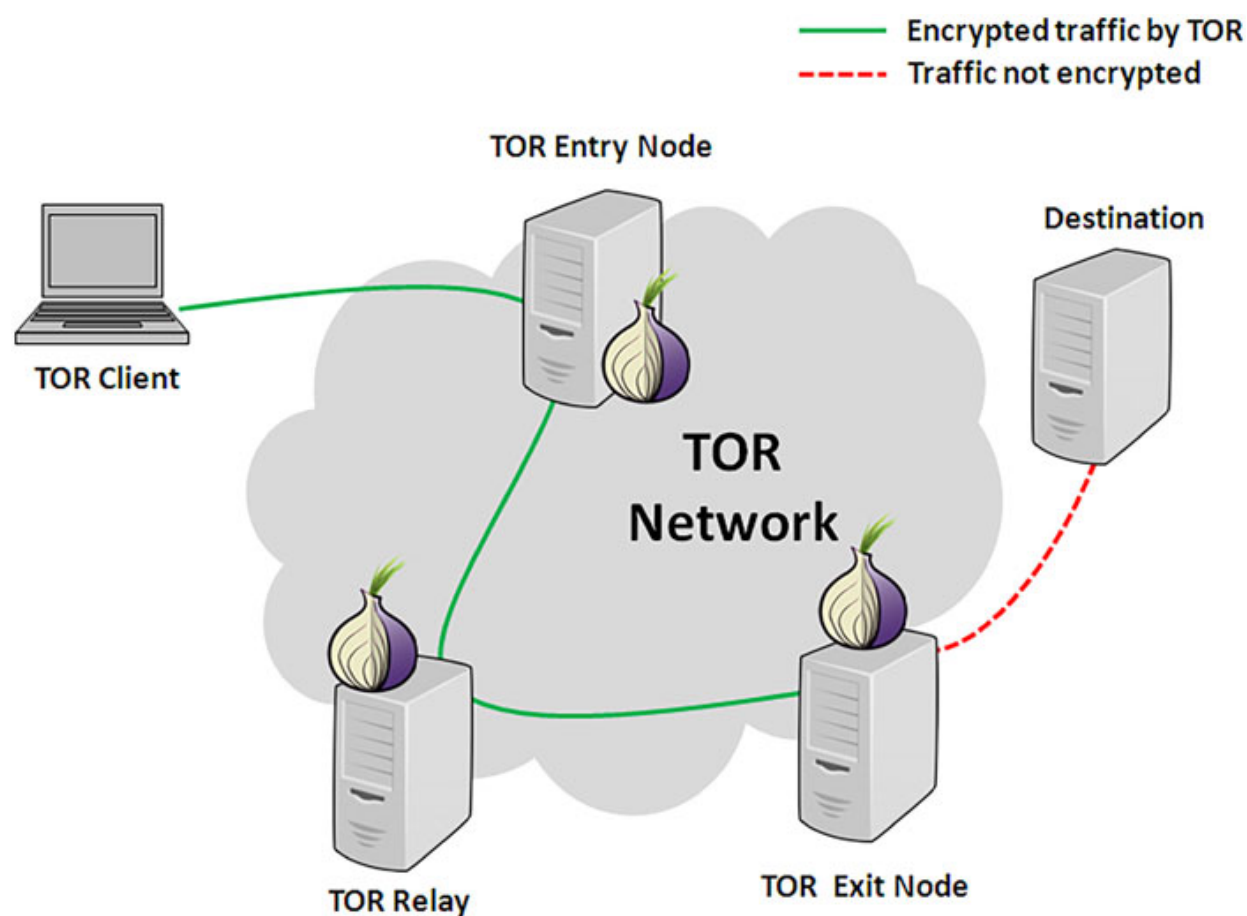


Figure 2.5: TOR (Anonymous browsing)

On the TOR network, it is easy for someone to run a website on a dedicated server without being known to anybody on the internet. But due to all these anonymity features, TOR is misused by ransomware mafias. All the communication between a victim's computer and the ransomware mafia's is

sometimes made anonymous using TOR network. This way, their activities cannot be easily tracked by law enforcement agencies. So, by using TOR, ransomware creators can anonymously communicate with the victim fearlessly.

Ransomware as a Service (RaaS)

Ransomware is new-age pain for internet users and organizations. The main reason behind the rapid growth of ransomware is the RaaS model. But what is RaaS?

To understand RaaS, we will first have to understand that it is very difficult for ransomware developer(s) to spread ransomware to earn more money until and unless it is self-spreading ransomware. As of today, every program has evolved to be in a subscription model - servers can be used in cloud providers, you can hear all your favorite music with a subscription, and even your photo software editor is now working as a service in cloud. Knowing this, even the bad guys use these tactics to spread and grow their business. So, in order to spread ransomware to target large victims, a business model is adopted, wherein renting and selling of ransomware to buyers, also known as affiliates, is done. This business model is known as **Ransomware as a Service (RaaS)**. With the help of this model, any threat vector with little technical know-how can deploy ransomware and earn a share of the profit. This is the primary reason for its rapid growth in ransomware.

How RaaS works

RaaS is based on the **software as service (SaaS)** model, in which ransomware can be rented or sold based on business models. We will be discussing different offerings in the next section. Before understanding the business model of RaaS, let us understand its working in the following figure:

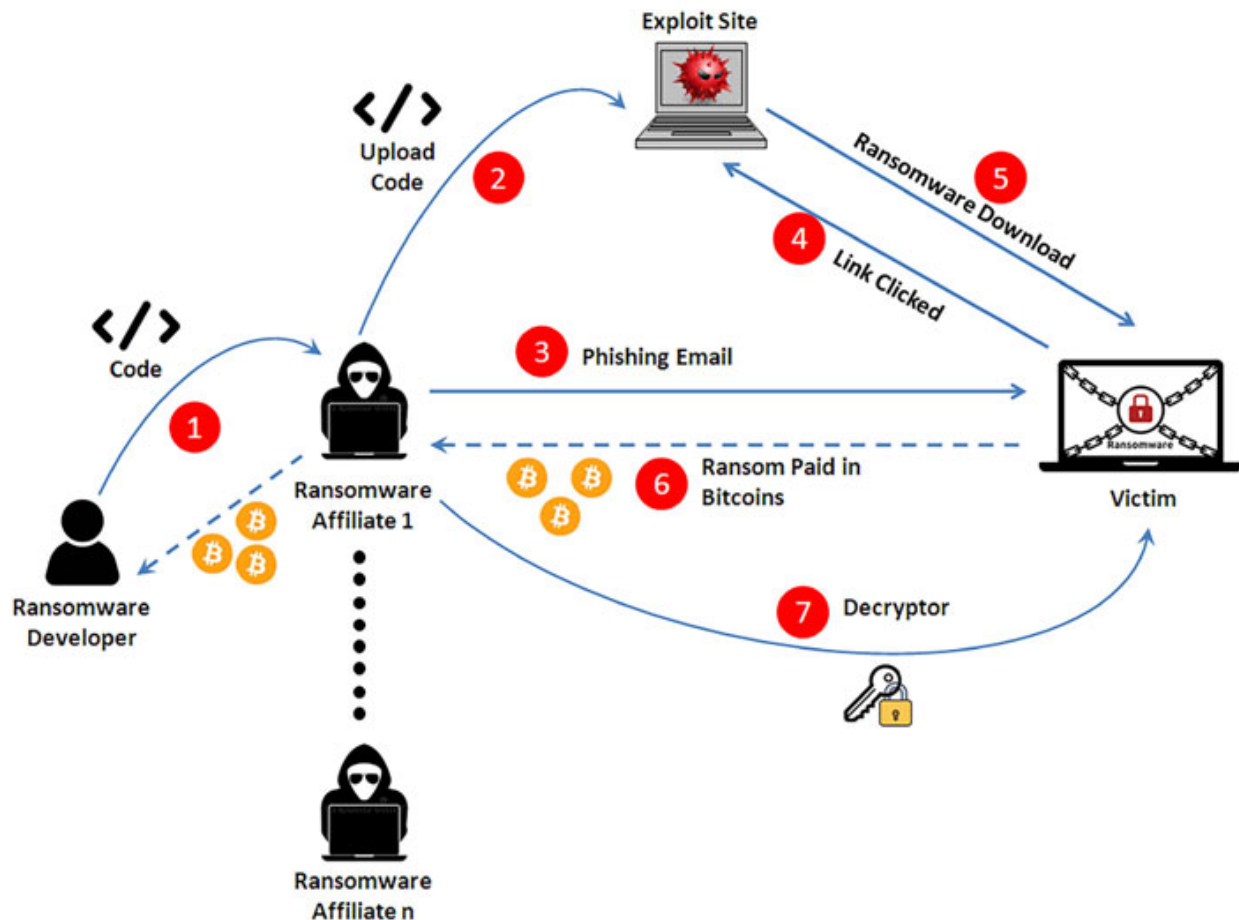


Figure 2.6: How RaaS works

1. The first step is when ransomware developers write a new ransomware. At this stage, ransomware is unknown to anybody on the internet. There are some groups of operators who work under the affiliate program to spread ransomware and earn a share of profit. We will call these operators ransomware affiliates.
2. On receiving the ransomware code from ransomware developers, ransomware affiliates run the affiliate program. Under this, they upload the code to the exploit site.
3. Then they run phishing email campaigns, with the link to exploit site in the email.
4. This email is so lucrative that the victim is lured into clicking on the link mentioned in the email.
5. This link points to the exploit site, where, on clicking on the link, ransomware is downloaded onto the victim's computer.

6. Once the victim is infected with ransomware, their computer or important data in the computer is encrypted. To get their data back, the victim is only left with an option to pay ransomware. Ransom is paid as per the instructions that flash on the screen of the victim's computer. Suppose the ransom is asked in Bitcoin. On paying the ransom in the form of Bitcoin, some share of the Bitcoin is transferred to the ransomware affiliate and remaining to the ransomware developer(s).
7. On receipt of its share of the profit, the ransomware affiliate sends the decryptor to the victim, which the victim can use to get their data back.

RaaS business models

Imagine the scale of infection that occurs when many operators participate in the ransomware affiliate program. Now, there are various business models in which RaaS works.

Affiliate-based

The affiliate business model is the same as the one in [Figure 2.6](#). Under this, operators are given a share of percentage from the profit. Operators are given support for successfully running the campaigns.

Subscription-based

Under this model, a flat fee is charged from the operators for using RaaS services. This flat fee can be charged monthly or yearly, based on the RaaS provider.

Licensing for lifetime

It involves selling a fully licensed ransomware toolkit with a fixed lifetime licensing fee. Buying a fully license toolkit is expensive when compared to the subscription model. On the ransomware operator's end, selling a fully-licensed toolkit is preferred over subscription or under affiliate model. It is always better in comparison to the passive income generated from the affiliates or subscribers.

Threat actors

We all know about hackers or attackers, as these are the common terms used to refer to individuals with technical skills and wrong intentions to breach into computers or networks. The term “*actors*” is a neutral term to refer to individual, groups of individuals or collections of groups. The term “*threat*” that is used as the prefix states the intention of the actors. The following figure shows some threat actors:

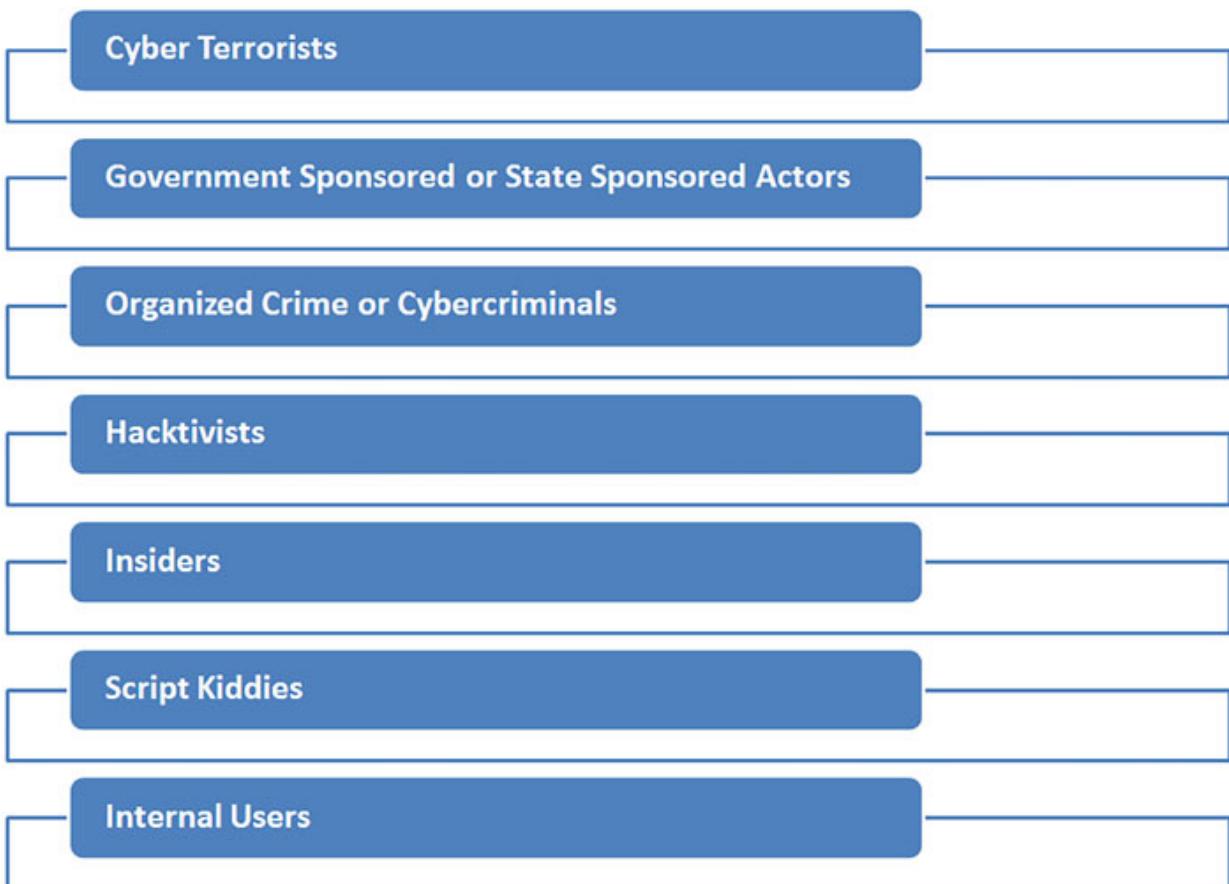


Figure 2.7: Threat Actors

Threat actor is a simple term used to refer to individuals who do not necessarily have technical skills, i.e., people who might or might not have technical skills, but they all have malicious intents to compromise a computer or network of computers. They can be people who just copy an organization’s confidential data onto a USB with an intention to destroy the organization data center.

[Vulnerability, exploit and payload](#)

To understand the concept behind a vulnerability, exploit and payload, let's walk through a simple example. Consider a house, as shown in the following figure:

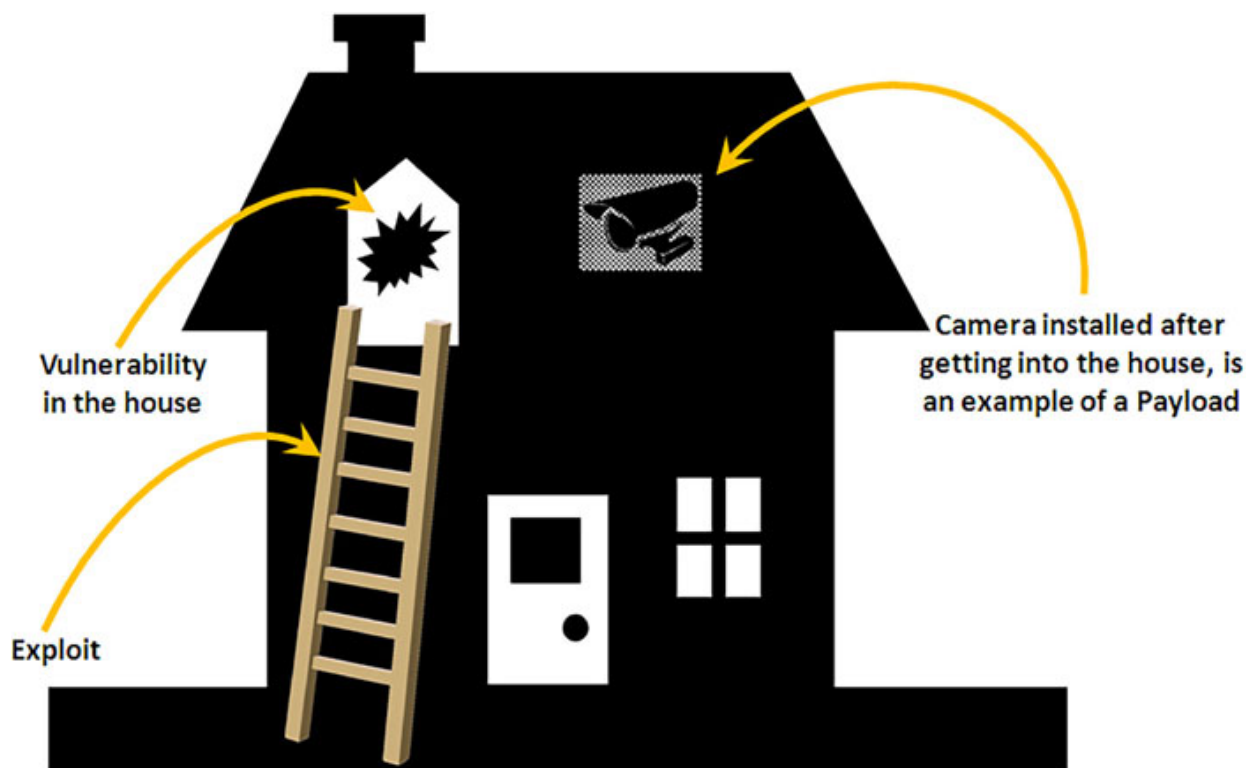


Figure 2.8: Vulnerability, Exploit and Payload

This house has two rooms: one on the ground floor and another on the first floor. The first floor has a broken window. As it is broken, it will catch the attention of any thief walking by the house. This broken window is the *vulnerability* in the house. Using this vulnerability or issue, any thief can get into the house. As this window is on the first floor, anybody who wants to get into the house will have to use ladder to reach there. Now, this ladder can be understood as the *exploit*. That means vulnerability can be exploited using this exploit.

To understand payload, we will consider a scenario wherein the thief's objective is not to rob the house but to keep a watch on the members in the house. To do that, the thief will use the ladder to get up to the broken window and put a camera inside the house. This camera can be controlled from the thief's mobile device. So, this camera can be compared to the term *payload*. Payload is what completes the actual intention of hacking. For any

malware, payload is the core component. Depending upon the hacker, the objective's payload is created. Payload can perform the following functions:

- It can steal the user's confidential data.
- It can be programmed to download other malware.
- It can be programmed to act as ransomware.

Ransomware attack vectors

Attack vector is a means or path by which an attacker or hacker gains access to computer or network in order to deliver payload for malicious activities. Similar to other malware, ransomware is distributed using different techniques. Ransomware authors are developing new variants to evade detection and infect as many systems as they can. To mitigate a ransomware attack, one needs to understand ransomware distribution methods.

Email

Email is the best way for hackers to spread ransomware, and it is done through spam and phishing email. Spam emails can be categorized based on the email content, and there are different types of spam based on that.

Phishing scam

In this process, spam emails are sent in bulk to many. These emails can have an attachment or an embedded URL for the user to click on. The objective of the phisher is to harvest user credentials like login details and credit card details.

Unsolicited advertisements

Such emails are also sent in bulk, but with an objective of promoting an organization's goods or services.

Email spoofing

In this type of phishing email, the sender's email address is spoofed to look as if it originated from a legitimate person. This provokes the receiver to open the attachment or the embedded URL in the email.

Spear phishing

Phishing emails are sent in bulk, but in spear phishing, specific individuals or organizations are targeted to steal confidential information or install malware, which can also be a ransomware. The process of how email service is used to spread malware and hack into user machines is shown in the following figure:

Email Hacking

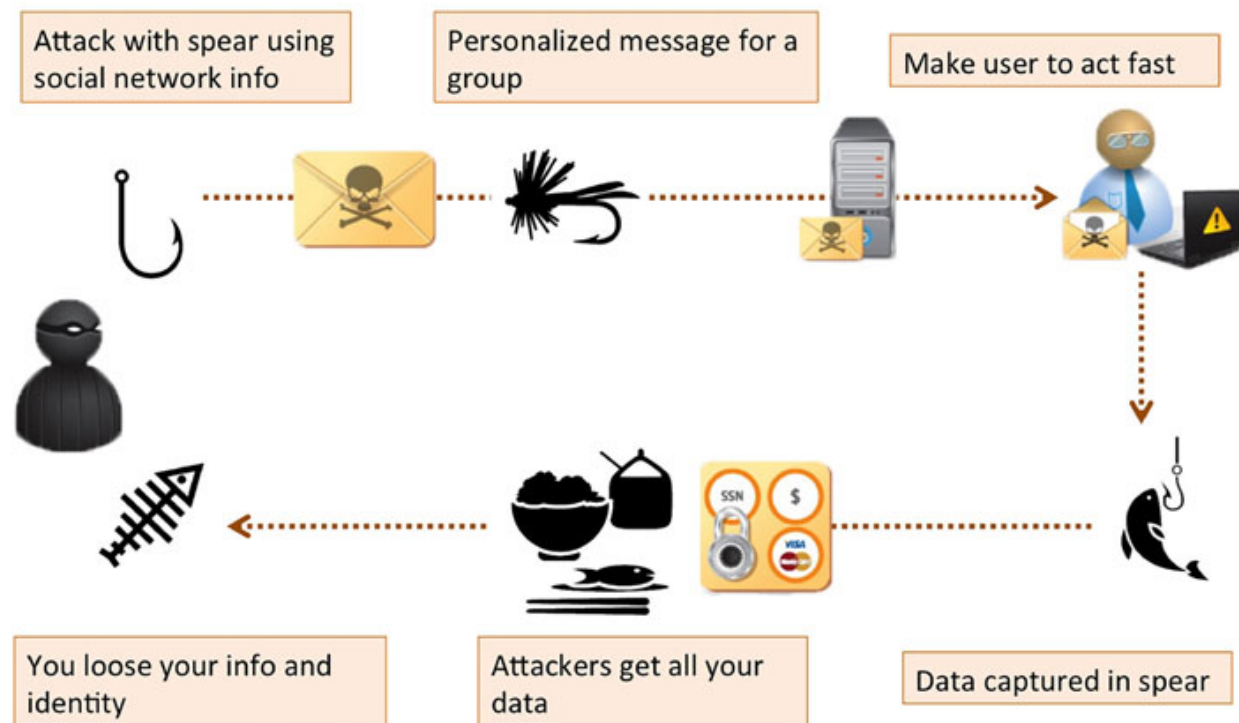


Figure 2.9: Hacking Through Spear Phishing Email

Exploit kits

Exploit kits are used by cybercriminals to deliver malware to the victims. The process of infection is quite simple: the victim visits the compromised website, and if the victim's computer has some vulnerability, then it is exploited. After successful exploitation, malware is installed on the victim's computer. We have already discussed the concept of vulnerability and exploit in the earlier section of this chapter.

Well-known applications are the common targets. There are a few popular software's with known vulnerabilities, like Adobe, Internet Explorer and

Oracle Java. Exploit kits target multiple vulnerabilities at the same time. As shown in the following figure, there are some stages at which attack is carried out using exploit kit:

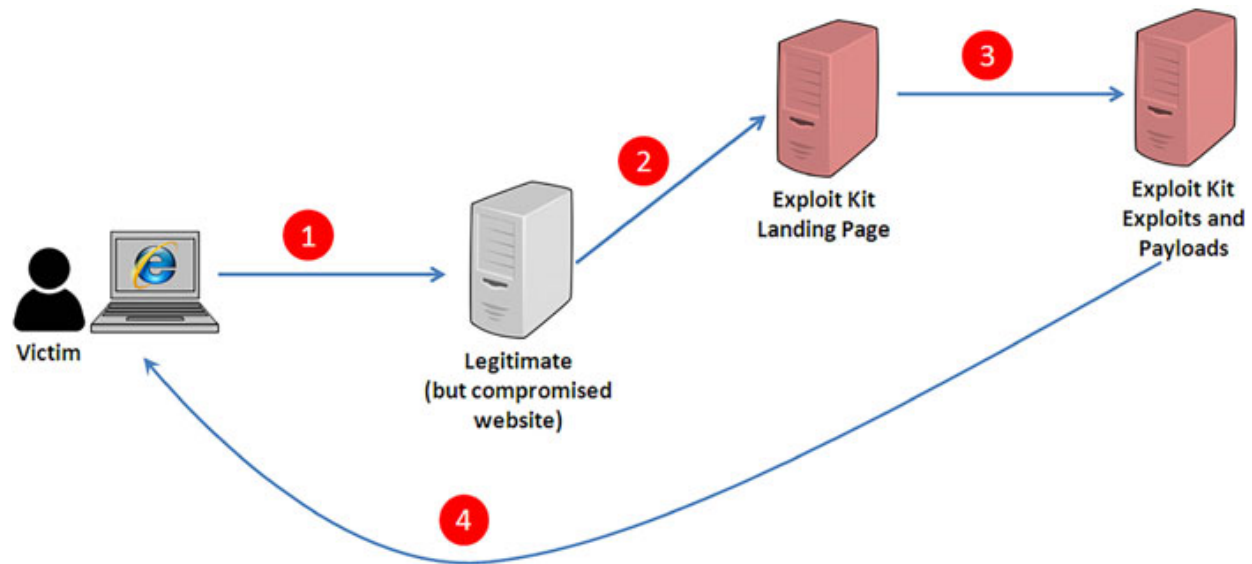


Figure 2.10: Exploit Kit Phases

Actual phases of the exploit kit's working may vary based on the sophistication of the attack:

1. Victim lands to the compromised website
2. They are redirected to the exploit kit landing page
3. The victim machine is exploited with the known exploits
4. The victim machine is infected with malicious payload

The main reason why exploit kits are successful is that the victim does not have to download anything on their computer. They only have to visit the compromised website and the malicious payload is downloaded on to the computer in the background.

USB and removable media

USB and removable media are major sources of infection. They fall under the local threat as the attackers require physical access to the victim device. But cybercriminals are smart and find ways to insert the USB into the victim's computer. Imagine yourself receiving a gold-plated USB in your office as a Christmas gift. On inserting the drive, it will execute a zero day

vulnerability to take complete control of your PC. If you are on your corporate network, it can compromise the entire network.

As part of security implementation in an organization, USB ports are blocked on the computers. Computers cannot access the USB data drives when the USB port is blocked, but on the same physical port, a keyboard with USB interface or USB mouse works perfectly fine. It means **Human Interface Device (HID)** works fine on blocked USB ports. So, to bypass this USB security control, USB data drives are programmed to work as **Human Interface Device (HID)**. Human interface devices are computer devices used to interact with humans, like keyboard and mouse.

In a road apple attack, a malware-infected storage device, such as a USB data drive or DVD, is left in a location in which it is bound to be discovered by an employee of the target company, for example, in the parking lot. Studies have shown that most people are more than happy to plug in their newly found USB drive into their work computer, and consequently, such attacks have the potential to be extremely effective.

Malvertising

As the name suggests, malvertising is a combination of two words: malware and advertising. It is a type of online attack wherein malware is hidden in the online ad and infects the user when the website is visited. Cybercriminals utilize the ad network to propagate their malicious ads along with the normal ads on the internet. The technology used in the background is advanced. A tiny piece of code is inserted in the advertisement, which makes your computer vulnerable to cybercriminals. To understand the processes behind malvertising, refer to the following figure:

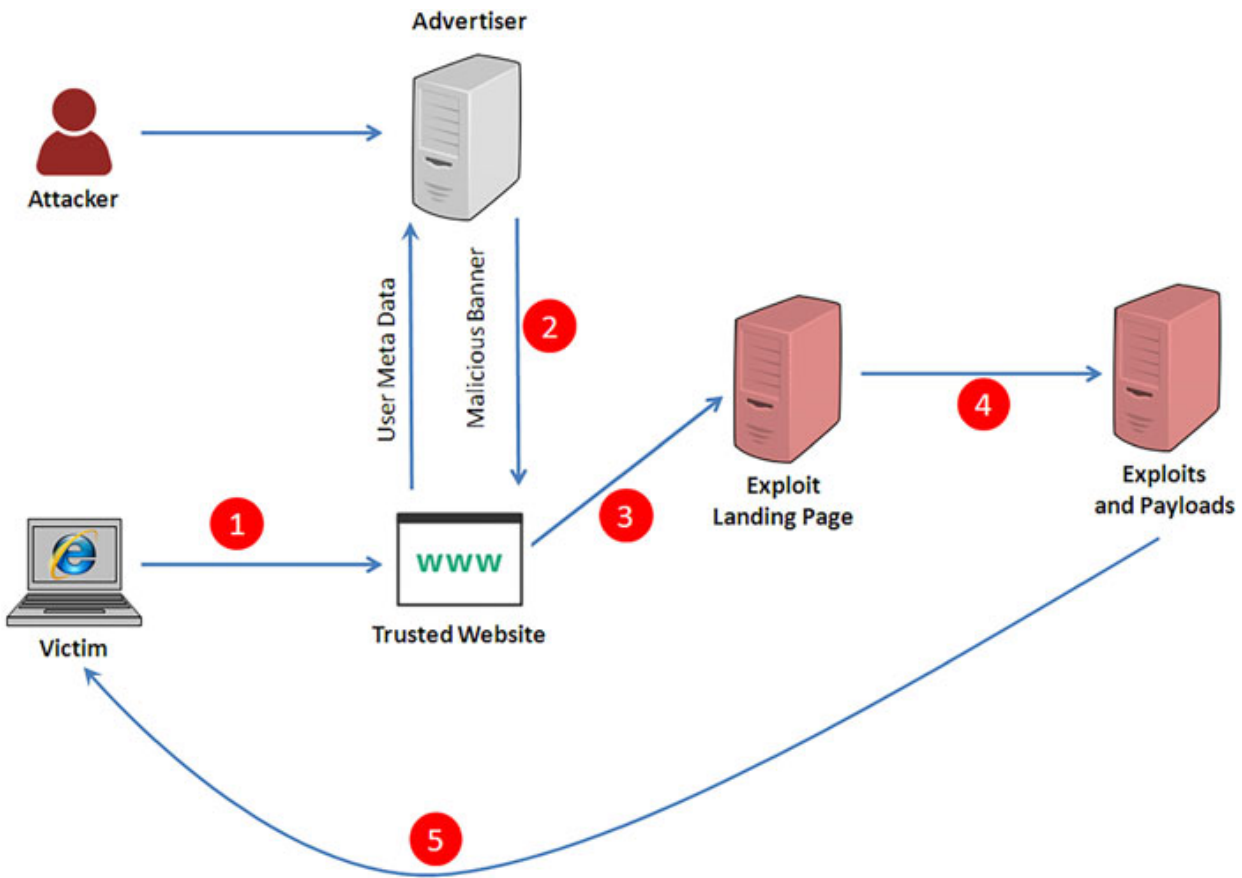


Figure 2.11: Malvertising

The attack works in the following phases:

1. Victim visits the trusted website. The website that the victim visits is neither compromised nor infected; it is programmed to show ads based on the victim's interest.
2. The attacker in the background programs the advertisement using a concept called *iFrames*, an invisible box that can secretly navigate to some web page. Advertisements with malicious code are propagated on the internet using advertisers like Google Ads.
3. Once this ad with malware is flashed on the victim's trusted website, the *iFrame* redirects the victim to the exploit landing page.
4. Exploit landing page queries the respective exploit of the vulnerability present in the in the victim's computer.
5. The exploit code attacks the victim computer and installs the backdoor.

Stages of ransomware

From the time ransomware is installed on the victim's machine and its presence is detected, there are five stages of a ransomware attack. Understanding these phases can help overcome the risk.

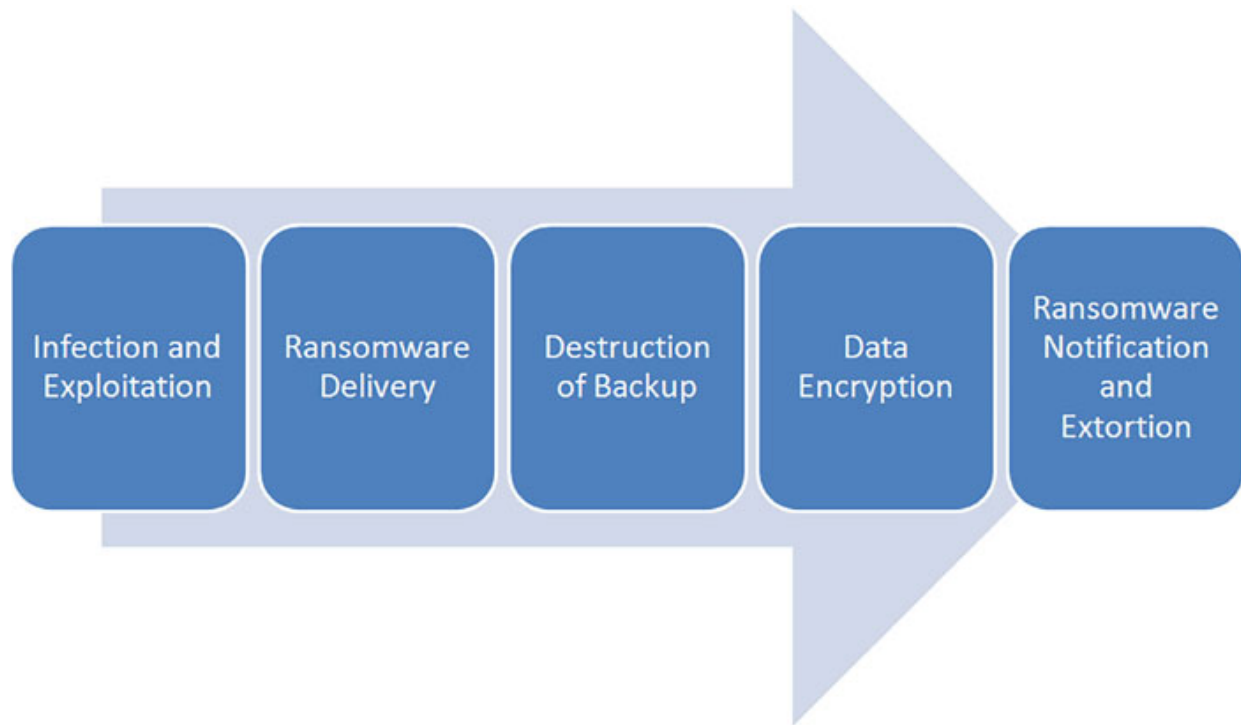


Figure 2.12: Stages of Ransomware Attack

1. **Infection and exploitation:** In this phase, the victim is introduced to the ransomware through phishing email, malvertising, exploit kit or any other medium.
2. **Ransomware delivery:** In this phase, ransomware is successfully delivered to and installed on the victim's machine.
3. **Destruction of backup:** Ransomware is programmed to delete all shadow copies of backup. Along with that, it can be programmed to delete the victim's backup elsewhere.
4. **Data Encryption:** After deleting the backup, it will start encrypting user data. Modern ransomware are programs used to generate keys to encrypt the data on the victim's computer. These keys are generated by making secure connections with the Command and Control server. Moreover, these keys can also be generated offline.

5. **Notification and extortion:** After encrypting the victim's data, ransomware displays a ransom note to the victim and demands a payment within a specific time frame. If the victim refuses to pay the ransom on time, then the ransom amount may increase, or it may lead to complete data deletion or the data being left encrypted forever.

Conclusion

In this chapter, we discussed the concept of ransomware by breaking it down into different components. We also talked about cryptocurrency and cryptomining, along with anonymous browsing. Cybercriminals are evolving with new technologies; among them is the **Ransomware as a Service (RaaS)**, which we covered in this chapter. We also discussed terms like vulnerability, exploit and payload. Finally, toward the end of this chapter, we covered ransomware attack vectors and its infection stages.

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

<https://discord.bpbonline.com>



CHAPTER 3

Current Defense in Place

Introduction

The first known ransomware attack happened in the year 1989; it targeted the hospital industry. It was initiated by *Joseph Popp*, who was a PhD and AIDS researcher. The way he distributed the malware (also called ransomware), was by distributing 20,000 floppy drives to different AIDS researchers across 90 countries. All these floppy drives were loaded with questionnaires, which claimed to be for research. For the AIDS researchers, the objective of the questionnaire was to analyze a person's risk of acquiring AIDS, but in reality, it was to distribute the ransomware. The floppy drive, when inserted into a system, installed a malware program that remained dormant. Malware was only activated after the computer was switched on 90 times; the names of the user's files would be encrypted and the ransom message would appear, asking victims to send US\$189 to a PO box in Panama. This ransomware became popular and was named PC Cyborg.

With time, ransomware is evolving and malware writers are becoming smarter. Years ago, people using the internet felt the need for systems that could combat ransomware attacks. There are many defense mechanisms available to combat ransomware attacks now. In this chapter, we will study some of the defense mechanisms.

Structure

In this chapter, we will discuss the following topics:

- Existing solutions in place
 - Backup solutions
 - Static or signature-based solutions
 - Dynamic behavior-based solutions
- User awareness trainings

- Vulnerability management tools
- Cryptographic interceptors
- Analysis of current solutions

Objectives

The objective of this chapter is to understand the existing solution that helps prevent a ransomware attack or recover data from them. There are a whole range of solutions available today as ransomware type's roars. But thankfully, we can count on the global security community to help mitigate and minimize impact of these kinds of attacks. Some are open-source, developed by big corporations or the cyber security global community, and others are available by the traditional payment method. Each solution is different from the other, and every solution has its own pros and cons. In this chapter, we will walk through backup solutions and then through solutions that offer a static approach to detect ransomware. Moving on, we will look at dynamic behavior-based solutions, and we will also cover vulnerability management tools and cryptographic interceptors. In the end, we will cover open-source and free tools available to combat ransomware attacks.

Existing solutions in place

There are many existing solutions that act as the counter measures against ransomware attacks. In this section, we will discuss these counter measures one by one.

Backup solutions

This is the best solution to combat ransomware attacks. It is also the simplest solution to protect you from these attacks. Data backup can be done online as well as offline. Online data backup solutions involve network-based storage solutions, and offline backup solutions cover external drives, which include external hard drives, tape drives and others. The differences between these two types are explained in [Table 3.1](#):

Offline backup solutions	Online backup solutions
They are kept locally or at some secure place.	They are available online, either on the network or in the cloud.

They do not require an internet connection.	They require internet connection for data backup and data retrieval.
Offline backups are fast as they do not require internet.	The speed for online backups depends on the speed of the internet connection.
Example: External hard drive, Tape drives, DVD, USB stick	Example: Network storage solutions, Cloud backup solution (Google Drive, OneDrive)

Table 3.1: Differences between offline and online backup solutions

The biggest problem with backup solution is the availability of backup at the time of the attack. It is very difficult for individuals to keep regular copies of the data backup. Moreover, it is also not feasible for organizations to regularly backup all their data. Ransomware does a denial of service attack on the victim system. It makes the data unavailable to the user and demands a ransom for the victim to be able to access their own data. If you have your backups up to date, the recovery process can be completed without any issues. You just have to reset the machine, reinstall the operating system and restore the data backup.

Nonetheless, modern ransomware is programmed to attack data backups. They are programmed to search for online data backups and encrypt them. Modern ransomware also delete the shadow files from the victim computer. Shadow files are the backup files maintained by the Windows operating system and are used to restore the system in the event of failure. [Figure 3.1](#) shows the reverse engineering code of a famous ransomware:

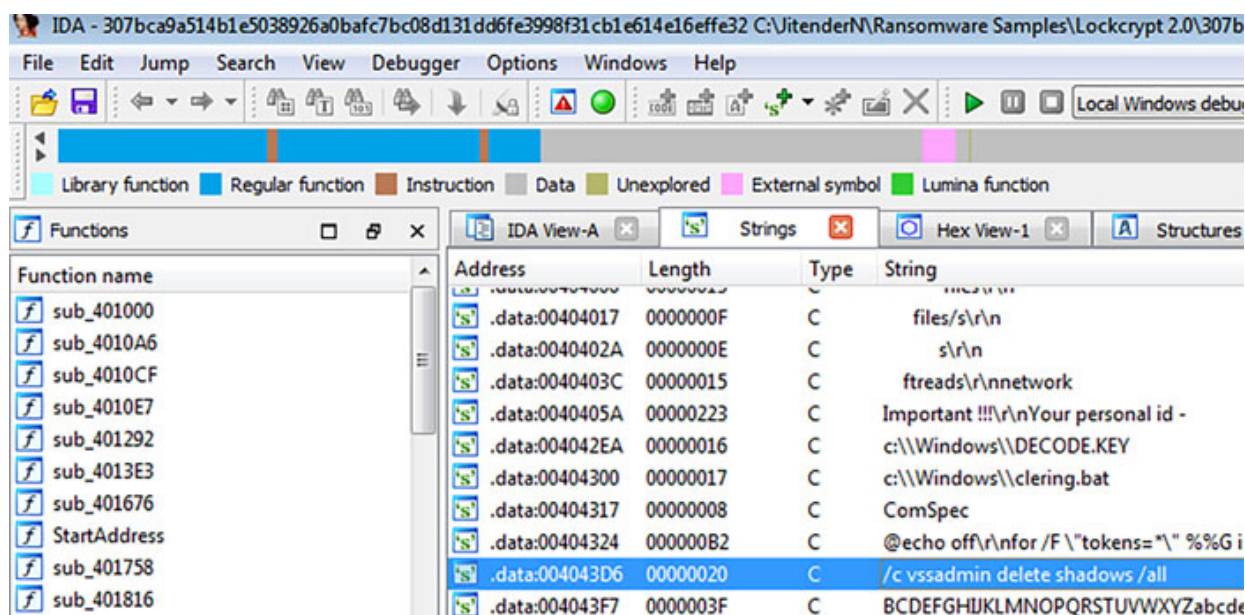


Figure 3.1: Ransomware deleting shadow files

The highlighted part in [Figure 3.1](#) shows how the ransomware deletes shadow files. To further elaborate this explanation, refer to the following code:

```
vssadmin delete shadows /all
vssadmin delete shadows = command to delete a specified
volume's shadow copies
/all = Deletes the all volume's shadow copies.
```

Thus, backing up your data is the best strategy to combat ransomware, but only if it is planned and implemented properly.

Static or signature-based solutions

Most antivirus use the signature-based approach to detect malware or ransomware. Signatures are created by vendors and researchers and are patterns that can be identified to take further action. Signature-based solutions can also use **Indicators of Compromise (IoCs)** to determine a certain type of malware.

Consider a case of a malware executable that has a signature that can be used to identify the executable. If a researcher is writing signature for the lock crypt ransomware, they can use the SHA256 value of lock crypt ransomware to identify whether the binary is a lock crypt ransomware. This value can be obtained by opening ransomware sample in pestudio that shows the SHA256 value, as shown in [Figure 3.2](#):

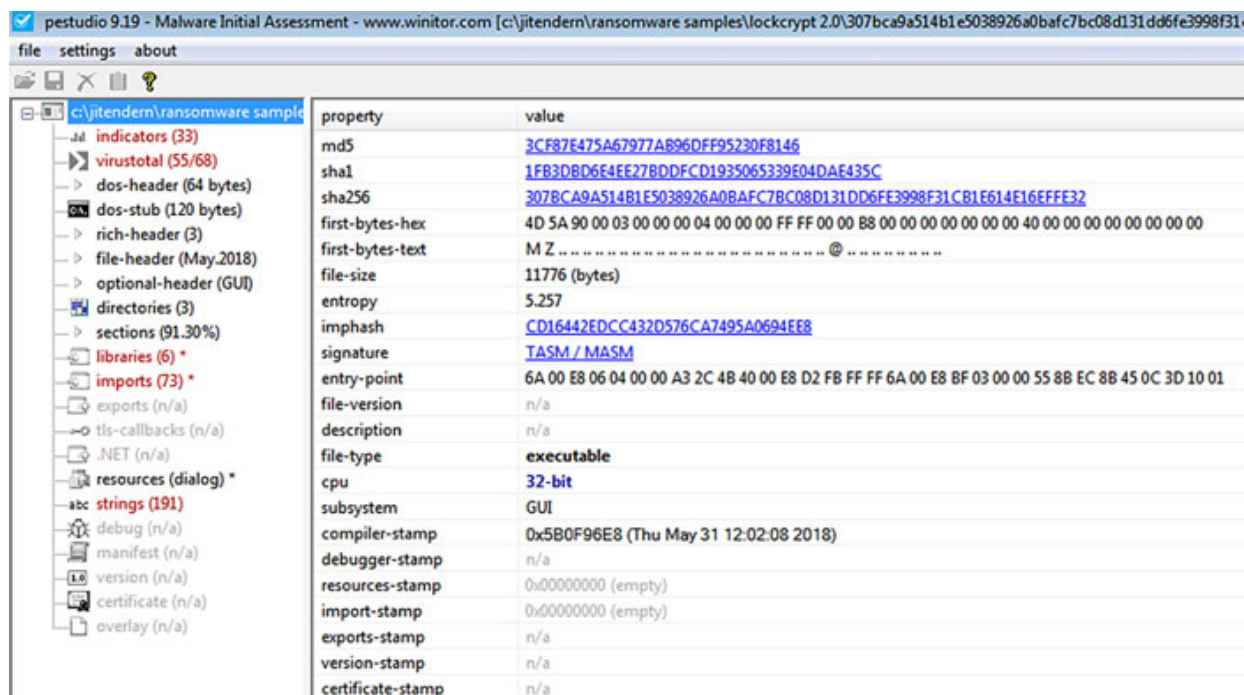


Figure 3.2: LockCrypt signature using pestudio

Products using the signature-based approach to detect malware work fine if the ransomware is known on the internet. However, these products are not effective to detect the latest or not yet known ransomwares. There is one tool called **YARA**, which is used to detect malware. YARA is a tool aimed at (but not limited to) helping malware researchers identify and classify (also detect) malware samples. With YARA, you can create descriptions of malware families (or whatever you want to describe) based on textual or binary patterns.

With YARA rules, you can detect when a rule is alerted. It uses a signature-based approach to detect malware, and rules are defined in YARA to detect it. YARA rules are defined by finding a pattern in the malware, and once the pattern is identified for a particular malware, a rule is written to detect that malware. Following is a sample YARA rule:

```
rule sample_rule : sample {
meta:
  author = "Atul Narula"
  description = "This is just a sample rule"
  threat_level = 3
strings:
```

```
$a = {6A 30 00 40 6A 14 8D 68 00 00 91}  
$b = {8D C1 4D B0 2B 6A 4E 83 C0 27 99 F7 F9 59}  
$c = "XXXXXXXXJITENDERNARULAXXXXXXXXX"  
condition:  
  all of them  
}
```

This rule will only match when all the strings, i.e., a, b and c, are in the content.

Ransomware writers are smart enough to bypass signature-based detection using different techniques. One among them is using packers to pack the malware and thus, obfuscate its code. Packing is the technique of compressing an executable and combining decompression code with the compressed data of executable in a single executable file.

The signature-based approach is good for known ransomware, but these products are ineffective against new families of ransomware.

Dynamic behavior-based solutions

When we talk about dynamic behavior-based solutions, we are dealing with products that detect malware during execution. When ransomware is executed in the victim computer, its core objective is to encrypt the user data. This is not done in one step but with a series of tasks. These sequences of tasks are uniquely identified to build a dynamic signature, which reflects ransomware behavior during execution. When the same ransomware variant is executed on another victim machine, it can be identified using the identified dynamic behavior.

Dynamic behavior-based products seem promising, but they generate a large number of false positives. To understand the concept behind generating false positives, consider a case of lock crypt ransomware. Lock crypt ransomware deletes the shadow files when executed on a victim machine using vsadmin call. If a dynamic signature that checks the call to vsadmin is created, it will no doubt detect ransomware execution. However, it will also detect execution of legitimate software or applications using vsadmin. So, dynamic behavior-based products perform well when used in a controlled environment. Dynamic solutions have several categories.

Entropy-based products

Before we talk about these segments of products, let's understand a term called entropy. Entropy is the degree of randomness in a system. To understand entropy from a layman's perspective, consider an example of broken glass and a jigsaw puzzle. It is very difficult to collect all pieces of broken glass to rejoin them, as they are completely disordered. So, we can say entropy is high for the broken glass. But with jigsaw puzzles, it is easy to join the pieces as they are easy to identify. So, we can say that entropy in the jigsaw puzzle is low as compared to broken glass.

Now, from a malware and ransomware code point of view, entropy-based products measure the randomness of data in ransomware or malware code. Encryption, data compression and code obfuscation techniques are used to increase the entropy. Measuring the entropy helps researchers determine whether the malware sample is obfuscated in any way. The most popular way to measure entropy in the code is by using Shannon's formula. With this formula, each executable is measured on the scale of 0 to 8. If the executable has lower code entropy, then there are fewer chances that the code is obfuscated or encrypted. If the executable has high entropy, then there are higher chances that the executable code is encrypted or obfuscated.

When talking about data access pattern products, these products measure the entropy of data on the system. When ransomware hits a victim machine, it starts the encryption of data, which is a high-entropy operation in itself. So, data access pattern products measure the entropy of data to detect a ransomware attack.

Machine learning products

Technology has evolved over the decades and today, we see machine learning leveraging every product that we can see. From self-driving cars to Amazon Alexa, we see machine learning everywhere. Keeping the current trend in view, many products are coming up with machine learning capabilities to detect and stop ransomware attacks. There is a process behavior of every ransomware variant, so when different combinations of process behaviors are fed to a machine learning system, they can be useful to detect ransomware. But what are these process behaviors?

These process behaviors can be differentiated in many dimensions, like the following:

- What types of file types is the ransomware targeting?
- How is the ransomware encrypting files, randomly or sequentially?
- Which type of encryption is the ransomware using?
- Is the ransomware programmed to encrypt only local file systems or network file systems?
- Is the ransomware connecting to the command and control server?
- If the ransomware is connecting to a server, in which country does the server reside? Or is it in cloud?
- Is ransomware connecting to several different IPs from different countries, calling TOR address or so on?

There are many ways in which ransomware variants can be bifurcated. All these dimensions help machine learning algorithms learn and detect ransomware. However, till machine learning products mature, they tend to generate many false positives. A slight change in ransomware architecture defeats the machine learning algorithms.

Honeyfile products

Honeyfile products are just like honeypot products, wherein we prepare a scapegoat machine to attract hackers to hack into your honeypot server. Once they are inside the honeypot, their behavioral activities are observed to prevent future attacks. The honeyfile concept is similar. Under this approach, bait files are deployed to catch the intrusion activities. For example, if a file named password.txt is created in the system, the hackers who are trying to intrude the system will be lured by this file name. So, when the file is opened, it will trigger all the alarms of intrusion.

This concept of catching intrusion is based on the work by *Cliff Stoll*, who deployed bait files to catch German hackers who penetrated his system. *JAGomez-Hernandez* developed a tool called R-Locker, based on the honeyfiles approach, to detect ransomware. This anti-ransomware tool is for Windows-based systems. It can be accessed on *JAGomez-Hernandez* GitHub repository (<https://github.com/JA-Gomez-Hernandez/R-Locker>), as shown in [Figure 3.3](#):

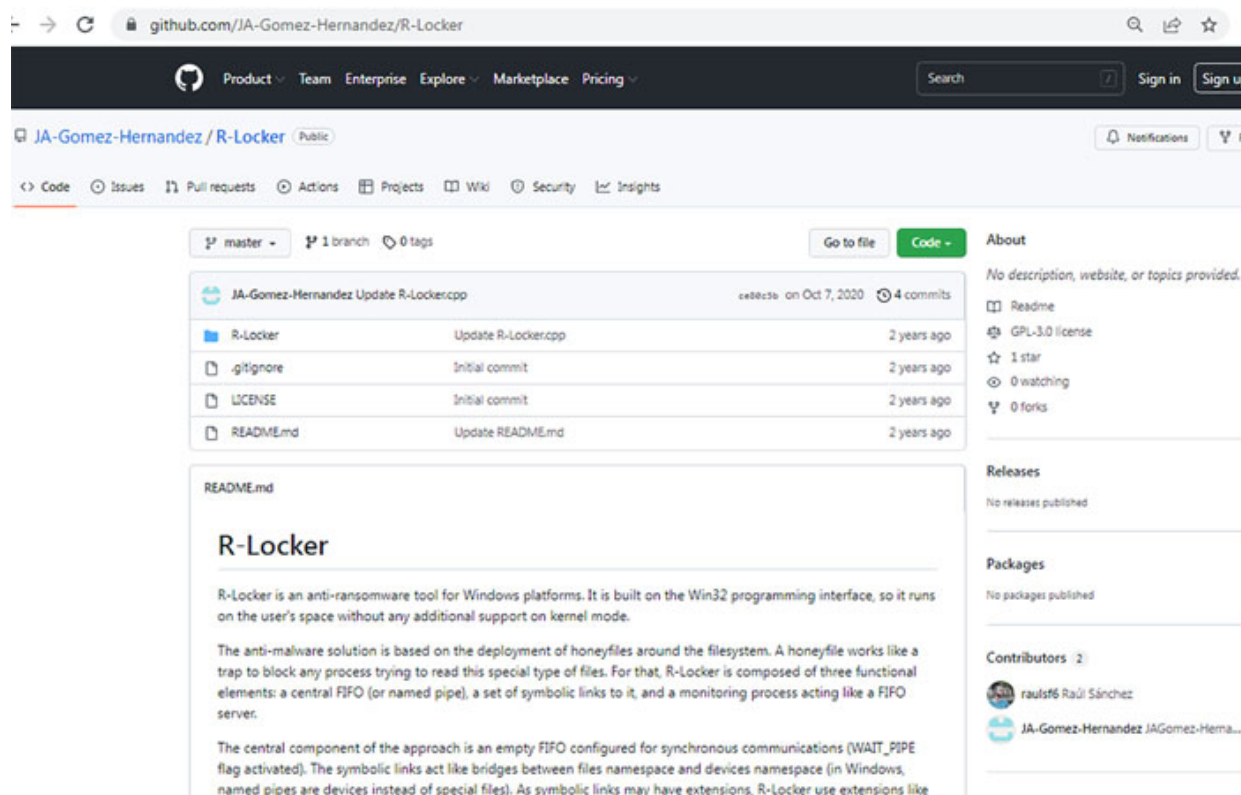


Figure 3.3: R-Locker Tool on GitHub

This solution is based on the deployment of honeyfile to stop a ransomware attack. It works like a trap to block any process that tries to access special type of files.

There are some limitations while deploying honeyfiles-based solutions. These limitations are leveraged by smart ransomware coders in the following ways:

- Planned ransomware attacks can avoid honeyfile to bypass detection. In this case, ransomware will not touch bait files during its encryption process.
- Some honeyfiles-based solutions deploy many bait files in different places in the system. This can cause unnecessary cluster of files in the system, causing inconvenience to the users.
- As we know, most operating systems nowadays index files and folders for fast and easy accessibility of data. These bait files or folders can create a lot of false positives, which will eventually cause issues while searching for data and can generate false positives.

User awareness trainings

Spending millions of dollars on security products and deploying expensive security resources cannot protect an organization unless the people in the organization are trained about security awareness. Employees are valuable assets but also the biggest risk for an organization. When we say valuable assets, we mean that employees are the ones who make and break an organization. But when we say that they are the biggest risk, it implies that even a small mistake of an employee can lead to a partial or even complete compromise of the organization's network.

Consider a scenario wherein an organization implements a very expensive solution to prevent attacks from cyber security threats, but an employee in the organization receives a phishing email. This email is so lucrative that the employee cannot resist opening it. Once that email is opened, and the employee clicks on the link provided in the email, which installs ransomware on the system.

Here, one click was enough to break the security of organization. Now the question that arises is, '*Why was the ransomware system undetected by the antivirus on the employee's system?*' Ransomware writers are smart to develop ransomware that go undetected by antivirus solutions.

Keeping all this in mind, organizations are focusing on developing a Human Firewall. Under this theory, humans are trained to act like a firewall. They are trained by providing cyber security awareness training. In cyber security awareness training, employees are trained on different aspects of cyber security, which include the following:

- Passwords and authentication
- Physical security
- Phishing attacks
- Social media use
- Mobile device security
- Cloud security
- Working remotely
- Internet and email use
- Public Wi-Fi

- Removable media
- Fair usage of company data and assets

Vulnerability management tools

Ransomware like Wannacry and Petya are known for the devastation they caused on the internet. Millions of dollars were lost in these ransomware attacks. The main reason behind propagation on the internet is the vulnerability they exploited to get into systems. Both these ransomware exploits are known as **Server Message Block (SMB)** vulnerabilities. Fixed vulnerabilities in a system greatly reduce the risk of ransomware attacks on the system. But the question arises, how does a vulnerability management tool help prevent ransomware attacks?

Vulnerability management is an ongoing process of identifying, assessing, reporting and remediating vulnerabilities across systems and endpoints. There are different phases of vulnerability management cycles, as shown in [Figure 3.4](#):

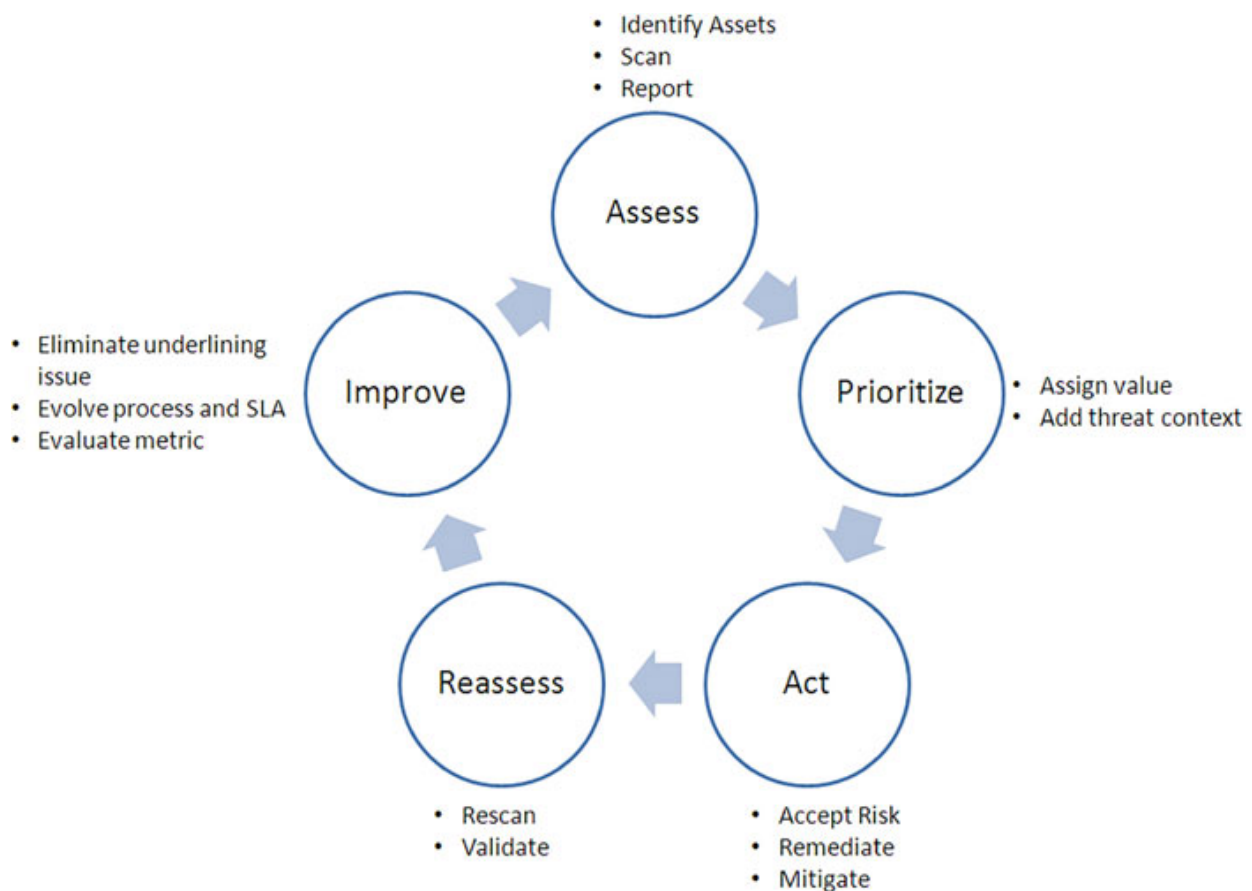


Figure 3.4: Vulnerability Management

Under these vulnerability management tools, systems are regularly updated for known vulnerabilities, and security patches are installed. This solution is not the final solution to fix ransomware attacks, but they add a layer of protection against cyber security threats.

Cryptographic interceptors

The primary reason for ransomware to become the biggest threat on the internet is its key generation algorithms. Ransomware generate keys on execution. There are some patented solutions on the internet that intercept the ransomware encryption process and extract the keys used by ransomware to encrypt victim data. They work by installing a tiny executable on the system, which continuously monitors the system for mass encryption. Once the signs of anomalous encryption are detected, they start intercepting the encryption process. From this encryption process, encryption keys are extracted.

One such patented product is Ransomware Reversal from Nubeva, which is a new solution to combat ransomware attacks. Nubeva Ransomware Reversal captures the encryption keys by taping the encryption process at first point in time. With the encryption process in place, keys are extracted, which makes decryption easy and fast.

There are many other solutions available that monitor the read/writes on the hard drive, since read/writes on the infected device tends to increase massively during the encryption process. Signs of anomalous read/write operations either on the operating system level or hardware levels can be used to trigger the ransomware operation in the system.

Analysis of current solutions

There are many open-source and free tools available to prevent ransomware attacks. In this section, we will look at such solutions.

R-Locker

We have already discussed about this tool in the *Honeyfile products* section. It is an anti-ransomware tool that works on the Windows platform.

PayBreak

Eugene Kolodenker introduced PayBreak, which is an open-source tool to monitor all generated keys by dynamic hooking cryptographic APIs. This solution works on the assumption of ransomware using cryptographic API. [Figure 3.5](#) shows the GitHub link of PayBreak:

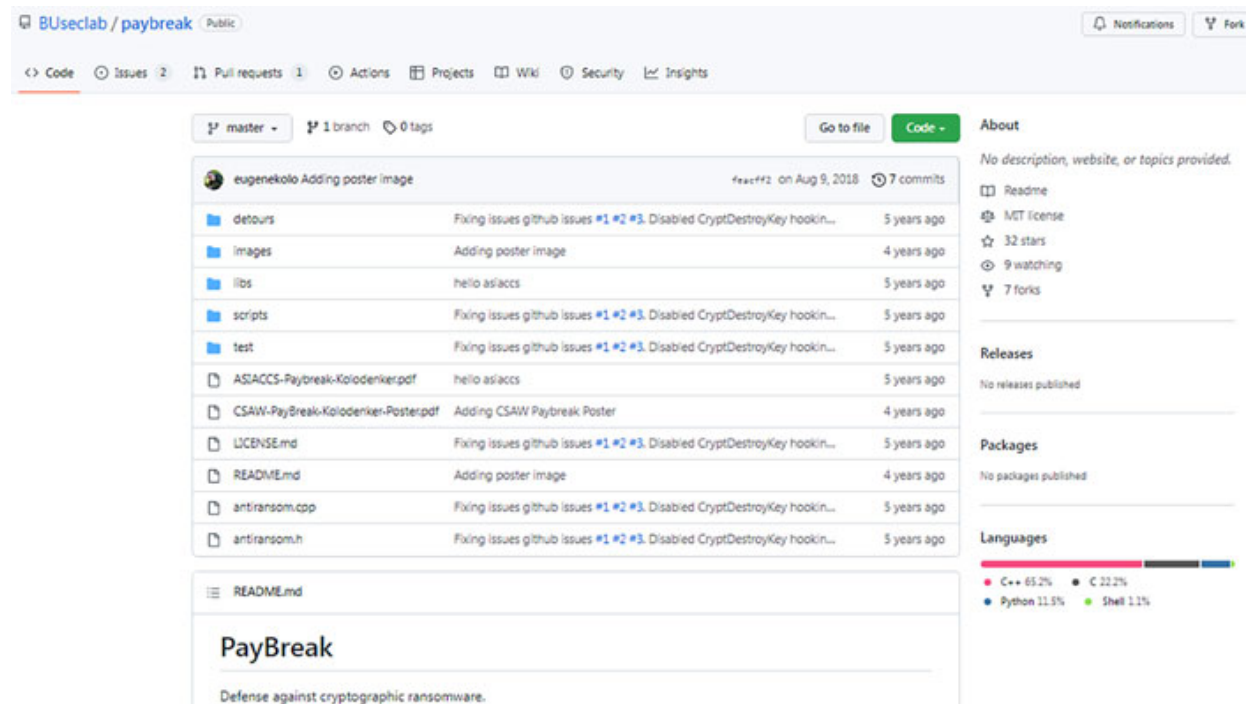


Figure 3.5: PayBreak

This solution is beaten by use of modified cryptographic functions in non-standard cryptographic APIs.

Redemption-anti-ransomware

This is introduced with the concept of creating honeypots all over the computer. It scans the files in the directories to create something called **Critical Zone Table (CZT)**. The content of CZT should be safe from massive amounts of changes, which can be encryption of files inside the directories. [Figure 3.6](#) shows the GitHub link of redemption-anti-ransomware:

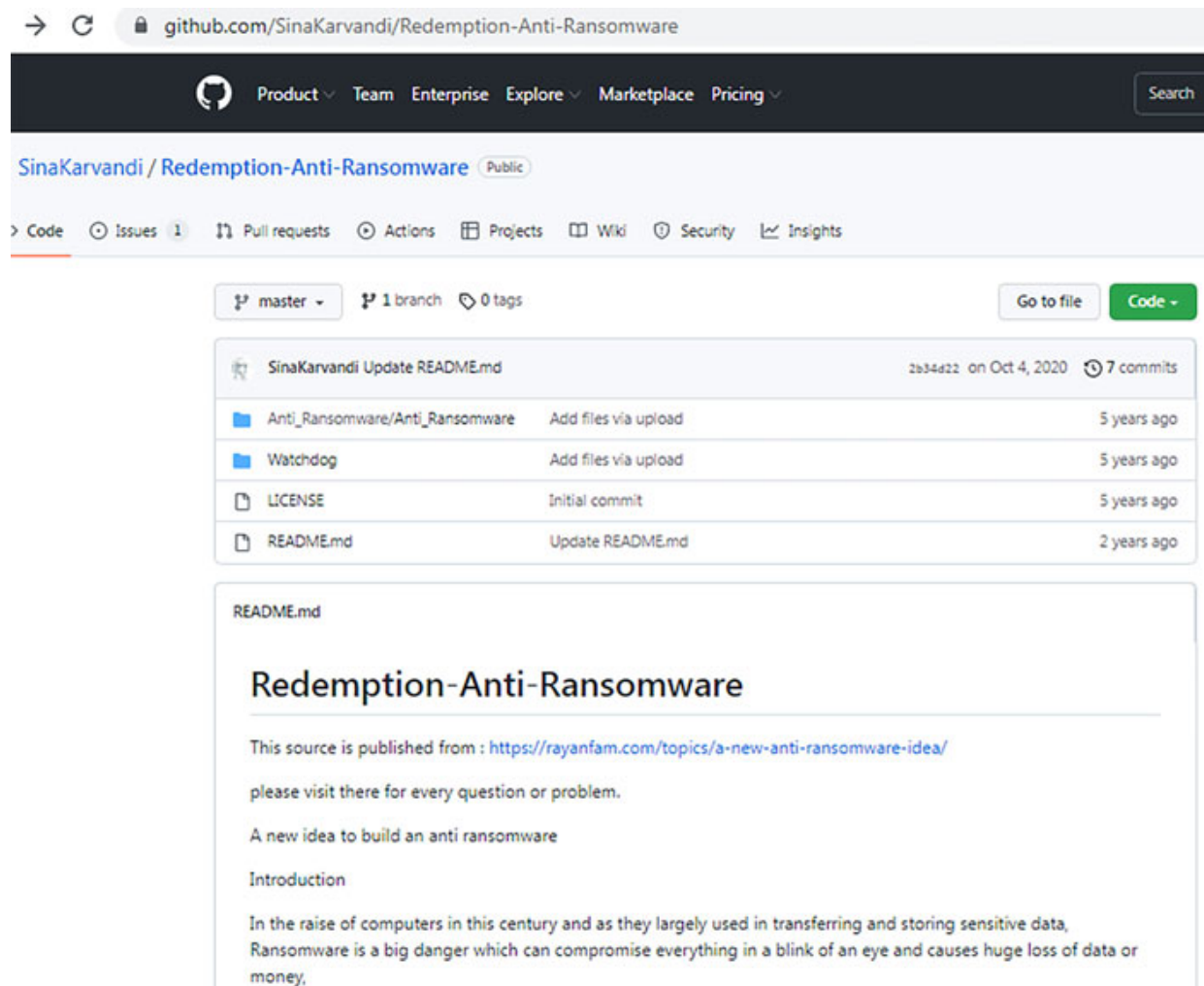


Figure 3.6: Redemption-Anti-Ransomware

If any process makes massive changes in CZT, it immediately puts the directory in Critical Mode. Watchdog does not kill or suspend the process; rather, it injects itself into the process for further mitigation.

Microsoft controlled folder access

The Microsoft control folder access allows you to protect important data from malicious activities like ransomware attack. All the applications are assessed by Microsoft for any malicious behavior. [Figure 3.7](#) shows the Controlled Folder Access:

Controlled Folder Access

Scenario description

Controlled Folder Access helps you protect valuable data from malicious apps and threats, such as ransomware. All apps (any executable file, including .exe, .scr, .dll files and others) are assessed by Microsoft Defender Antivirus, which then determines if the app is malicious or safe. If the app is determined to be malicious or suspicious, then it will not be allowed to make changes to any files in any protected folder.

Scenario requirements and setup

- Windows 10 1709 build 16273
- Microsoft Defender AV

Figure 3.7: Microsoft CFA

If any application is found to be malicious, it is not allowed to make any changes to any file in the protected folder.

Conclusion

In this chapter, we covered the existing solutions to prevent ransomware attacks or recover data from them. We also talked about the current solutions, backup solutions, and solutions that offer a static approach to detect ransomware. Then, we walked you through dynamic behavior-based solutions. Further on, we covered vulnerability management tools and cryptographic interceptors, and then we moved on to open-source and free tools available to combat ransomware attacks.

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

<https://discord.bpbonline.com>



CHAPTER 4

Ransomware Abuses Cryptography

Introduction

Data can be classified in two forms: data in transit and data at rest. Data in transit refers to the data that travels across endpoints in networks. When we talk about data at rest, we refer to the data stored in our computers, storage devices or in cloud. Security of the data is always a concern for organizations and individuals. While designing information security policies of organizations, the standard CIA model is adopted. The CIA triad standard is the very first law of the cybersecurity. Also known as the pillars of cybersecurity, CIA refers to the Confidentiality, Integrity and Availability.

Confidentiality in CIA refers to securing information or data from unauthorized viewing and access. This can be achieved using a method called **cryptography**. When cryptography is applied to the data at rest, data becomes unreadable. When it is applied to data in transit, it is prevented from eavesdropping and enables data to be transmitted even via untrusted channels.

Cryptography is not a new concept; in fact, it is a really ancient one. In olden times, cryptography was achieved by carving on wood or stone, which was then delivered to the intended individual for further deciphering. With the advent of internet, cryptography has come a long way. In this chapter, we will talk about the concepts behind cryptography and how hackers or malware writers are using this to develop ransomware.

Structure

In this chapter, we will discuss the following topics:

- Concept of cryptography
- Types of cryptography algorithms
- Difference between symmetric and asymmetric algorithms

- Block ciphers and stream ciphers
- Hybrid encryption method
- How ransomware abuses cryptographic algorithms

Objectives

The objective of this chapter is to understand the concept behind the working of ransomware and how ransomware is abusing different algorithms. Ransomware attacks are becoming sophisticated, and it is becoming more and more difficult for individuals or organizations to recover their data after a ransomware attack. The question therefore is, 'What makes ransomware attacks so different?'. The answer lies in the use of cryptographic algorithms.

To understand this concept in the sequential form, we will first look at the different types of cryptographic algorithms and cover the workings of symmetric and asymmetric algorithms. Moving further, we will talk about the hybrid encryption method. Toward the end, we will understand how current ransomwares are abusing cryptographic algorithms.

Concept of cryptography

Before we look at cryptography, we will understand the term encryption. Imagine that we want to transfer data from source computer to destination computer. The requirement is to transfer the data in such a manner that anybody sitting in between the source computer and the destination computer cannot read the data. In order to achieve this, the data is modified in such a manner that anybody eavesdropping on this data inbetween cannot read it. The method by which data is transformed from a human readable format to non-readable format is called **encryption**. In another words, encryption is a method of transforming plain text data into a secure and unreadable format. With the help of encryption, we can transfer the data from source computer to destination computer over any insecure channel. With encryption, we can secure data in both forms: when at rest or in transit.

[Figure 4.1](#) illustrates the encryption concept:

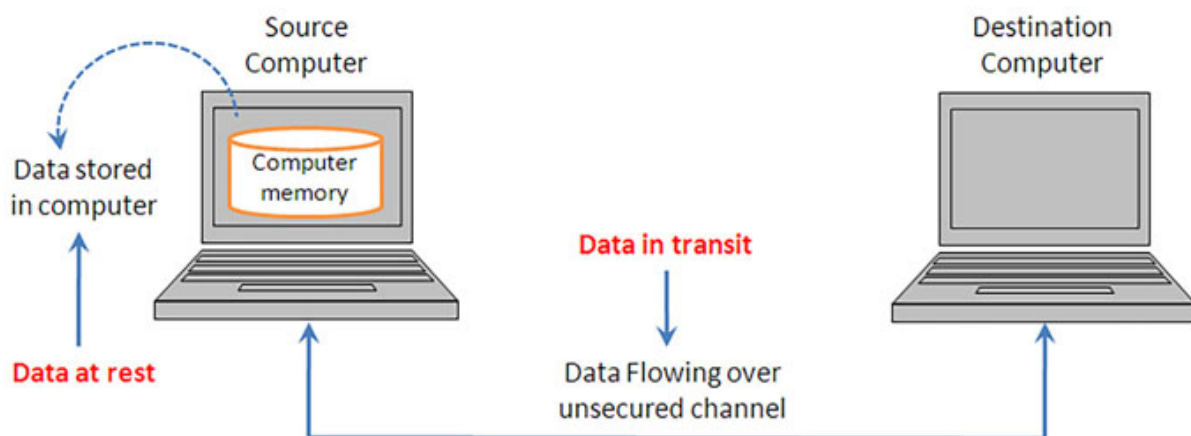


Figure 4.1: Encryption concept

Encryption and decryption can be achieved at software or hardware level. The systems that use encryption or decryption are called cryptosystems. All cryptosystems use complex mathematical formulas to achieve encryption or decryption. When these complex mathematical formulas are used in a sequence of activities, they make up the term *algorithm*. Algorithms are nothing but the set of rules defined to achieve encryption and decryption in cryptosystems. We also refer to an algorithm as *cipher* in a cryptography context.

There are various algorithms, ranging from proprietary to the ones that are publicly known. Proprietary algorithms are known to only a few subsets of individuals, but public ones are known to everyone. Publicly known algorithms are used across the globe for encryption and decryption across devices, software and applications. Internal complex mathematical formulas of public algorithms are known to everyone. So how do we achieve data confidentiality with algorithms known to everybody?

To understand this concept, imagine a hardware lock manufacturing company that manufactures hardware locks used in our day-to-day activities to lock our houses, cupboard, cars, and many more. Hardware locks of the same company differ from each other by the key they have. Every lock has a unique key that can unlock it. Using the same concept, every algorithm uses random bits called keys to encrypt and decrypt the data. Now, if the source computer wants to encrypt data, it will use an encryption algorithm with a key to encrypt the data. On the other end, the same key and algorithm is used

by the destination computer to decrypt the data. If we use the same key every time to encrypt and decrypt the data, it becomes very easy for hackers to break the key.

To prevent this, randomness is added to the key, wherein the key is generated from a range of values. The encryption algorithm generates a different random key every time to encrypt the data and the same is used by the destination computer to decrypt data.

To understand this concept of randomness, imagine we have 3 bits (nothing but a combination of 0's or 1's) and the algorithm is programmed to take 3 bits key. Now, from our range of 3 bits, different combinations of 3 bits key can be generated. The algorithm will randomly pick one key of 3 bits for encryption. To elaborate, let's understand the concept in details. Every X here represents one bit, and 3 'X' represent 3 bits. So, the total number of different combinations of 3 bits will be $2^3 = 8$. The range of bits from which random key is generated is called keyspace or keysize or key length. So, we will have 8 keys in total from our 3 bit length, as can be seen in [Table 4.1](#):

Key length/size/space (3 bits) ->	X	X	X
Key 1	0	0	0
Key 2	0	0	1
Key 3	0	1	0
Key 4	0	1	1
Key 5	1	0	0
Key 6	1	0	1
Key 7	1	1	0
Key 8	1	1	1

Table 4.1: Keys created from 3 bit length

Consider a case of an encryption algorithm known as AES, which stands for Advanced Encryption Standard. We will talk about this encryption algorithm further in this chapter. AES supports three key lengths: 128, 192, 256 bits. If we are using AES 128 bits for encryption and decryption, 128 bits will be the key size, and we will have 2^{128} different combinations of keys. If we are using AES 192, then we will have 2^{192} different combinations of keys. So the larger the key length or key space, the more are the number of keys we

have and the greater the possibility of the algorithm to randomly pick the unique key. The more number of unique keys we have, the more secure our encryption is and lower is the chance of an attacker finding out the key by chance.

In our earlier example, when the source computer uses publicly known encryption algorithm to talk to the destination computer, the encryption algorithm is known to everybody, but the key is only known to source and destination.

Types of cryptography algorithms

We discussed about the publicly known or proprietary cryptography algorithms. Proprietary algorithms are known to a subset of individuals, but the public ones are known to everyone. Publicly known algorithms are used across the globe for encryption and decryptions across devices, software and applications. A public v/s proprietary algorithm differs on the basis of the exposure of internal mathematical logic to the external world. There are chances that proprietary algorithms are tweaked a bit to achieve more security.

In general, there are two types of cryptography algorithms: symmetric and asymmetric algorithms. Symmetric are the ones that use same key to encrypt and decrypt the data. On the other hand, asymmetric algorithms are the ones that use different keys to encrypt and decrypt the data. We will talk about them in detail in the following sections.

Symmetric algorithms

In symmetric algorithms, the source computer uses a key to encrypt data and the same key is used by the destination computer to decrypt the data. [Figure 4.2](#) shows a diagram of symmetric algorithm being used:

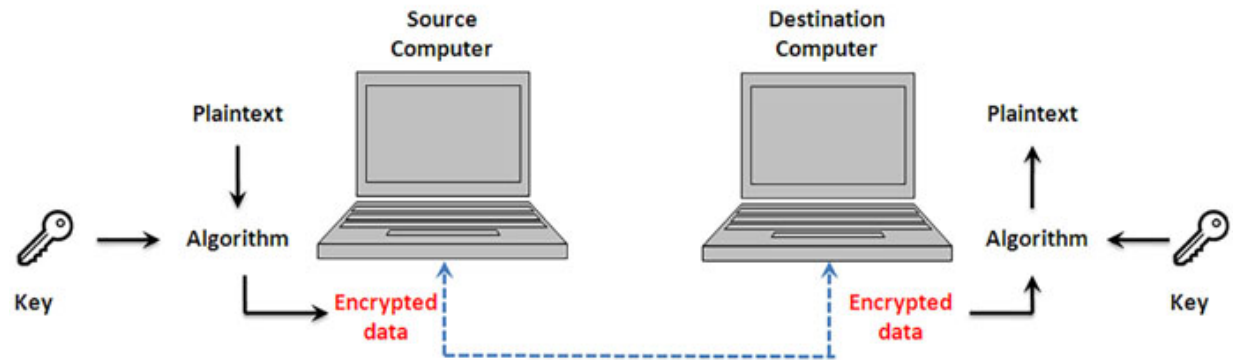


Figure 4.2: Symmetric algorithm

The key here is also referred as the secret key or symmetric key. The main problem with the symmetric key is the secrecy of the key. As the same key is used by source and destination, this key needs to be securely transmitted from source to the destination for further decryption.

Now, imagine the communication between three computers. In that case, the total number of symmetric keys will be 3. The formula to calculate the total number of symmetric keys with N number of computers is as follows:

$$N(N - 1) / 2 = \text{total number of symmetric keys}$$

The more the number of computers, the more is the key management required to manage keys in the network. Following are the well-known algorithms that fall under symmetric algorithms:

- Data encryption standard (DES)
- Triple data encryption standard (3DES)
- International data encryption algorithm (IDEA)
- Blowfish
- Advance encryption standard (AES)
- RC4, RC5 and RC6

Categories of symmetric encryption algorithm

There are two categories of symmetric ciphers: block ciphers and stream ciphers. **Block cipher** breaks the plaintext text message in blocks of fixed size, which are further passed through the encryption algorithm using a key.

To understand the block cipher concept, we will take a plain text message “How r u”; converting this to binary gives us the following:

01001000011011110111011100100000011100100010000001110101

If the block size is 8 bits (it can be 64 bits, 128 bits, or 256 bits, depending on the block ciphers), we break the binary value of plain text in the blocks on 8 bits. The first 8 bits are bold to demonstrate the first block, second block is represented by the next 8 bits (which are not in bold), and the third block is shown again shown in the bold and so on. Consider a case where the last block is only left with 3 bits; in that case, padding of 0 bits is added towards the end to ensure that the last block is of the same size as the others.

Now depending on the mode of operation in block cipher, blocks are encrypted.

Stream cipher encrypts the plaintext message one bit at a time instead of in blocks. We encrypt each bit by using the keystream, which is a stream of pseudo-random bits based on a nonce (random or semi-random number) and encryption key. This keystream is XOR with plaintext to generate cipher text.

To understand the stream cipher concept, we will again take a plain text message “How r u”; converting this to binary gives us the following:

01001000011011110111011100100000011100100010000001110101

Once we have the keystream, we XOR the plaintext bit with the keystream bits. If the first bit of keystream is 1, then XOR’ing it with first bit of plain text 0 gives 1. Now, if the second bit of keystream is also 1, then XOR’ing it with second bit of plain text 1 gives 0. This is how we get the cipher text.

Stream cipher is faster and better suited for devices that have fewer resources as encryption happens bit by bit, unlike block cipher.

Asymmetric algorithms

An asymmetric algorithm is also referred to as public key cryptography, wherein every entity (or in our case, computer) have two keys: a public key and a private key. The public key of every entity is known to everybody, and the private key of an entity is only known to the entity it belongs to. The way it works is different from a symmetric algorithm.

In asymmetric algorithms, if an encryption is done with a public key of an entity, then the data can only be decrypted by the entity's private key. The same holds the other way around: if an encryption is done with private key of an entity, then the data can only be decrypted by the entity's public key. Let's understand this with the help of [Figure 4.3](#):

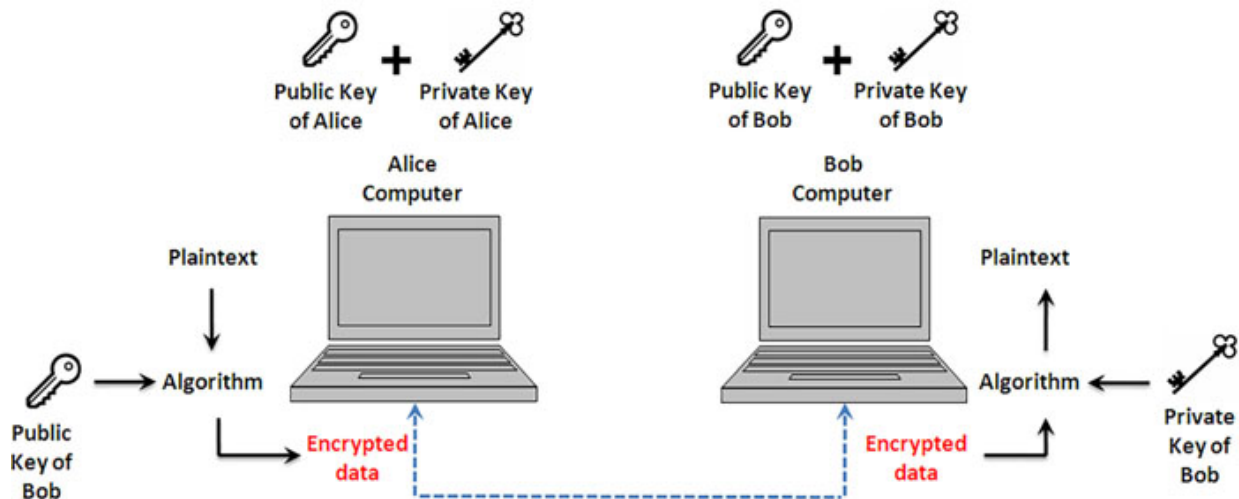


Figure 4.3: Asymmetric algorithm

In the preceding figure, both the entities, i.e., Alice computer and Bob computer, have public and private keys. Alice has its own public key and private key, and Bob has its own public key and private key. If Alice wants to send encrypted data to Bob, then Alice will use Bob's public key to encrypt data and send it across to Bob. As we discussed earlier, if an encryption is done with public key of an entity, then the data can only be decrypted by the entity's private key. In this case, on receiving encrypted data from Alice, Bob will use its own private key to decrypt the data.

With the help of public key, cryptography authenticity can also be achieved. Refer to [Figure 4.4](#) to understand this concept:

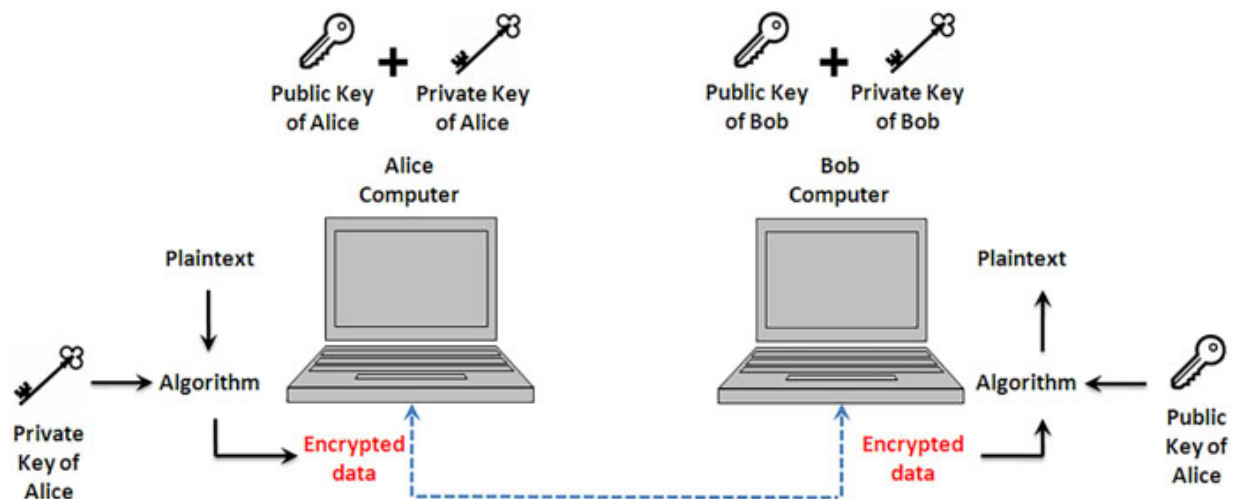


Figure 4.4: Authenticity in asymmetric algorithm

In the preceding scenario, if Alice wants to prove its authenticity, then Alice will use its private key to encrypt the data. Bob, on receiving the encrypted data from Alice, will use Alice's public key to decrypt data. On successful data decryption, Bob will confirm that the sender of data is Alice as Alice's private key was used to encrypt the data. But in this case, confidentiality of data is not ensured, as the public key of Alice is known to everyone and anyone who has the encrypted data can decrypt it using Alice public key.

Following are the well-known algorithms that fall under asymmetric algorithms:

- Rivest Shamir Adleman (RSA)
- Elliptic curve cryptosystem (ECC)
- Diffie Hellman
- El Gamal
- Digital Signature Algorithm (DSA)

Hybrid encryption method

In the previous section, we studied about symmetric algorithms and asymmetric algorithms. We learned that symmetric algorithms are fast, but they lack key management scalability and provide only confidentiality. Asymmetric algorithms, on the other hand, are slow, but key management is

easy when there are large numbers of entities involved. So to take advantage of both symmetric and asymmetric algorithms, we have a hybrid approach. Let's understand the concept behind the hybrid approach and learn how hackers use the hybrid approach to develop ransomware.

As we learned in the earlier section, in the symmetric algorithm, the same key is used to encrypt as well as decrypt data. On the other hand, in asymmetric algorithm, also referred as public key cryptography, we have two keys: public key and private key. The public key of every entity is known to everybody, and the private key of an entity is only known to the entity it belongs to. Encryption done with the public key of an entity can only be decrypted by the entity's private key. The same holds true the other way around: if an encryption is done with private key of an entity then the data can only be decrypted by the entity's public key. Hybrid approach takes the advantages of both algorithms. Let's understand the concept using [Figure 4.5](#):

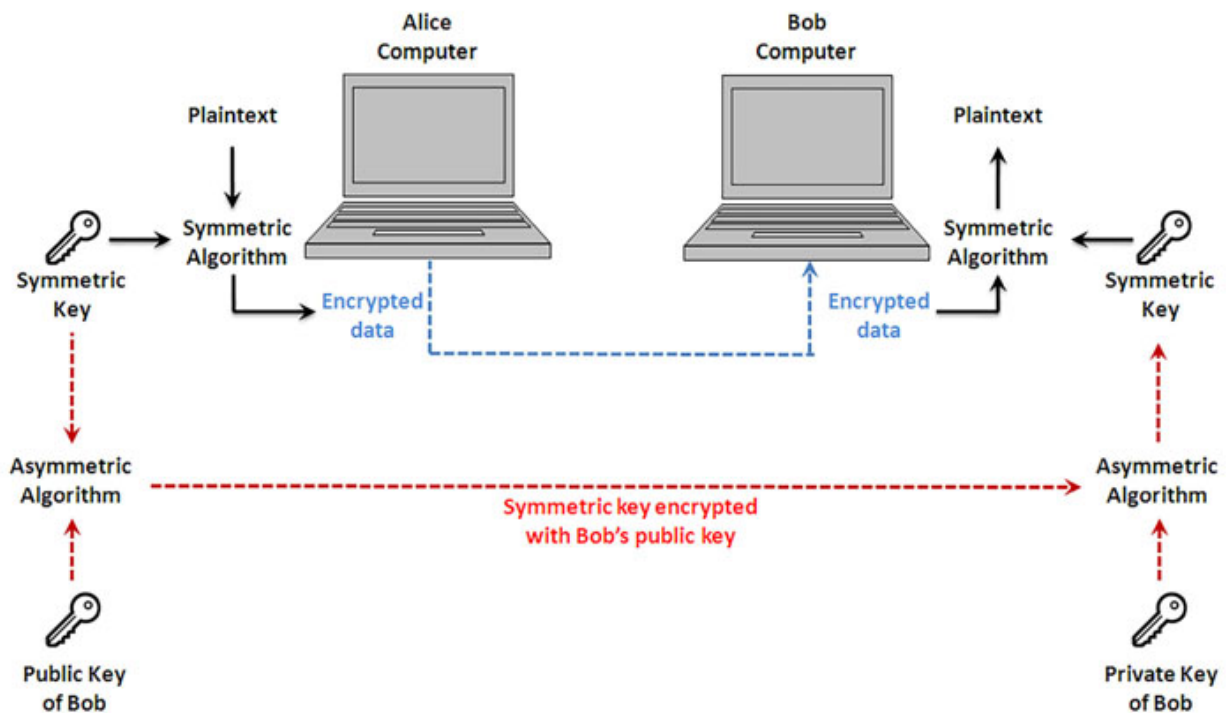


Figure 4.5: Hybrid Encryption Method

Let's suppose Alice needs to securely send data to Bob. As we learned in symmetric encryption, Alice and Bob both need to have the same symmetric keys for encryption and decryption. If Alice encrypts data with a symmetric key, then Bob will require the same key to decrypt the data received from

Alice. But how will Bob get the symmetric key from Alice? There are two ways to send symmetric key to Bob.

One is to directly send the symmetric key from Alice to Bob in an unencrypted form. In this case, any eavesdropper can get hold of the key. So, this method of transferring symmetric is not safe.

The second approach is to send the symmetric key from Alice to Bob in an encrypted form. Encryption of the symmetric key can be achieved using asymmetric algorithm. It means Alice will encrypt the symmetric key with the public key of Bob. On receipt of the encrypted symmetric key, Bob will use its private key to decrypt it.

Further on, Bob will use this symmetric key to decrypt the data. The question arises that why did Alice use Bob's public key to encrypt symmetric key instead of its own private key? This is because if Alice used its own private key to encrypt the symmetric key, then anyone with Alice's public key could decrypt and retrieve the symmetric key. This could lead anyone to decrypt the encrypted data using the retrieved symmetric key. In the next section, we will see how the hybrid approach is used to develop ransomware.

How ransomware abuses cryptographic algorithms

Encryption algorithms help maintain security of data at rest or in transit. There are numerous examples where we have seen how technology developed for benefiting mankind is being misused by the bad guys. The same goes true with encryption, wherein hybrid encryption is being misused to develop ransomware logics. [Figure 4.6](#) shows how hybrid approach is used in recent ransomware:

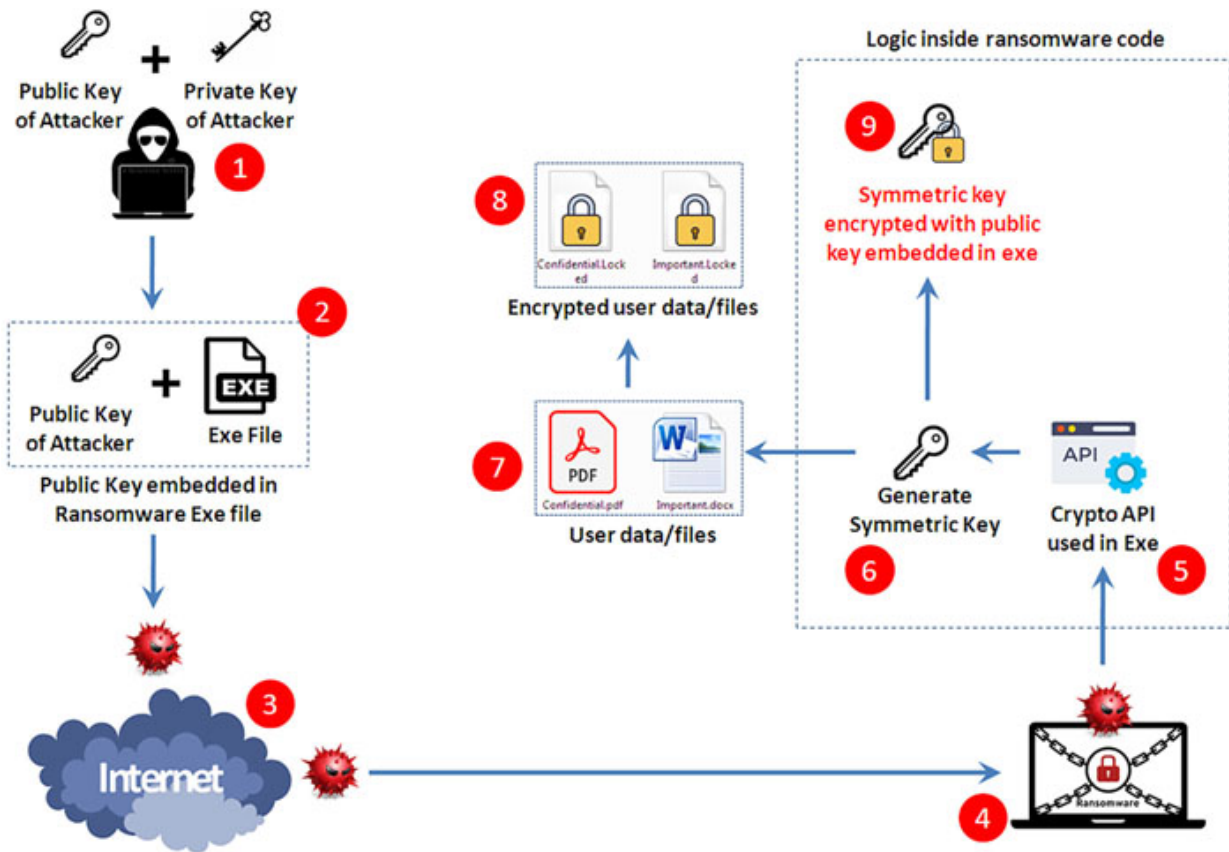


Figure 4.6: Ransomware flow concept

When ransomware uses the hybrid approach, it encrypts the user data or files with symmetric algorithm for speedy encryption of user/victim data. Furthermore, the symmetric key used for encryption of user data or files is encrypted with the attacker public key embedded in the ransomware code or binary. The process of ransomware is explained in the following steps:

1. Attacker generates public and private key pair.
2. Attacker develops ransomware using crypto APIs and embeds the public key generated in step 1 in the ransomware code. Now, on code compilation, ransomware binary will have the attacker's public key embedded.
3. Now, this ransomware is transported to the victim using different attack vectors on the internet. Attack vector is the method used by the hacker or attacker to illegally access the user's computer or network to deliver the malicious payload.

4. On commencing ransomware binary, the user or the victim's computer is infected with ransomware.
5. Inside the ransomware logic or code, crypto API is being used to perform ransomware operations on the computer.
6. Symmetric key is generated using crypto APIs programmed within ransomware code.
7. Symmetric key generated in the previous step is used to encrypt user data or files, as the symmetric algorithm is faster than the asymmetric algorithm.
8. All important data on the user's computer is encrypted and inaccessible to anyone. At this point, the user can see that their files are not accessible and have a different file extension.
9. Now in order to secure the symmetric key on the computer, in the ransomware code logic attacker encrypts the symmetric key with the attacker public key embedded in the ransomware code or binary. Thereafter, the symmetric key is securely deleted from the user's computer. At this point, the attacker is the only person who can retrieve the symmetric key using attacker's private key that is known only to them.

There are many deviations across different ransomware workings. Many ransomware store the encrypted symmetric key on the system, with a ransomware note. At the time of ransomware payment, this encrypted symmetric key is transferred to the attacker. On receipt of the payment, the attacker decrypts the encrypted symmetric key with his private key and transfers it to the user for data retrieval.

In order to perform a large-scale ransomware attack, modifications are done by ransomware writers in ransomware key generation logic. Ransomware attack and detection is like a cat and mouse game, as technology is evolving on both sides.

Conclusion

In this chapter, we talked about the concept of cryptography and its types. We covered the workings of symmetric and asymmetric algorithms and also discussed the different categories of symmetric encryption algorithms, which involve block ciphers and stream ciphers. Additionally, we learned how the

uses of cryptographic algorithms are making ransomware attacks more sophisticated. Towards the end, we learned how current ransomwares are abusing cryptographic algorithms.

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

<https://discord.bpbonline.com>



CHAPTER 5

Ransomware Key Management

Introduction

Ransomware has grown steadily over the years, with several worrying trends that point to effective and targeted attacks on organizations and individuals. These opportunistic attackers blindly target public and private sector entities for maximum profit. In this chapter, we will underline the criticality of key management in the ransomware cryptographic system to facilitate the construction of effective solutions against this threat.

The core component of ransomware is the key management component. Once the victim is infected with ransomware, the core objective of the malware or ransomware writer is to safeguard the key used to decrypt the victim's data. This key is the crucial factor through which malware mafias earn bounties. The way in which ransomware key management is done has changed over time. The more advanced the ransomware is, the more advanced the logic for key management. In the earlier chapters, we studied the different encryption algorithms and hybrid encryption method used by ransomware writers to protect the key generated during ransomware execution on a victim machine.

Structure

In this chapter, we will discuss the following topics:

- Key management techniques
- No key management or scareware
- Key on the victim machine
- Key on hacker or attacker machine

Objectives

The objective of this chapter is to help the reader understand the key management techniques used by ransomware writers. To break any techniques, we will have to understand the origination of key management process and how it evolved over time. In this chapter, we will talk about ransomware, which uses no key management technique to scare users and ask for ransom. Moving further, we will also talk about key management techniques, where key generation and handling is done on the victim machine. Further on, we will talk about the technique of key management on the hacker or attacker network. It means key management is in control of the attacker. There are many variants of ransomware that used key management on their network. But we will see how it evolved over time and talk about WannaCry ransomware key generation model.

Key management techniques

In the previous chapter, we discussed the hybrid encryption technique and learned how hackers are abusing algorithms to build ransomwares. The objective of ransomware writers is simple: somehow commence ransomware execution on the victim's computer, encrypt user data and demand ransom to decrypt it. To generalize, all of this can be divided in phases, as shown in [Figure 5.1](#):

Figure 5.1: Ransomware Phases

Symmetric key is the most important element of ransomware encryption method. Knowing the symmetric key is enough to decrypt the victim's data. Thus, safeguarding the symmetric key is the most important task for ransomware writers. This is where key management comes in. In the following sections, we will talk about some key management techniques used by ransomware writers. This will help us to understand the techniques followed by ransomware writers to safeguard the key used for encrypting user data.

No key management or scareware

This is the old technique used by malware writers to fool users into paying ransom. Under this technique, malicious software is developed and displayed to users using social engineering. This software can be a simple popup shown to users to have them believing that their computer data is encrypted. This type of malware is called scareware. Some scareware are designed to trick victims into believing that their security is compromised and their data is encrypted. These scam techniques provoke users into making wrong decisions, as shown in [Figure 5.2](#):

Figure 5.2: Scareware

These software do not encrypt user data but display a warning to the user saying that their data is encrypted. Some scareware obfuscate user data on the victim system itself and display a ransom note.

Key on the victim machine

Not all ransomware writers are smart; sometimes, they also tend to make mistakes. Here, we are talking about the ransomware that hide key on the victim machine in one or the other form. We are talking about ransomware to which the key can be extracted using reverse engineering techniques, either before or after ransomware execution. Some ransomware use static hard coded key and some encode or encrypt the key.

[Figure 5.3](#) illustrates that the key generation logic is stored in the ransomware code. Ransomware code can be easily reverse engineered to fetch the ransomware key generation logic and to further break the ransomware attack. Refer to the following figure:

Figure 5.3: Key on Victim Machine

If any ransomware of such kind is delivered to a victim, it follows the given set of steps:

1. Ransomware is delivered to victim using different attack vectors.
2. Once the ransomware is executed, it extracts the key either from some hard coded encoded variable or generates one using the logic mentioned in the ransomware code.
3. Once the ransomware code has the key, it starts encrypting user data with that key.
4. On successful execution, the key is deleted from the victim computer.
5. Victim is shown a message to pay ransom to decrypt their data.
6. On paying the ransom, the key that is already known to the hacker is now passed on to the victim for data decryption.

The biggest drawback of this method is that the same key, once received from hacker, is sufficient to decrypt data of other victims as well. Thus, there is no need to pay more ransom to the said hacker.

There were many variants of ransomware that followed a similar approach of storing key inside the ransomware code or logic. Some of them are RansomWarrior and Jigsaw. [Figure 5.4](#) displays the message that is shown to the victim when RansomWarrior attacks:

Figure 5.4: *RansomWarrior*

According to the ransom note shown in the preceding figure, RansomWarrior appears to have been developed by Indian hackers, who also appear to be inexperienced in malware development. The executable is not obfuscated, packed, or otherwise protected, indicating that those behind it are new to the game. In reality, the Ransomware's "encryption" is a stream cypher, with a key chosen at random from a list of 1000 hard-coded keys in RansomWarrior's binary code. To make the victim's files inaccessible, RansomWarrior 1.0 employs AES 256 encryption. The RansomWarrior 1.0 Ransomware identifies the encrypted files by appending the file extension. THBEC' to each file's name. The

RansomWarrior 1.0 Ransomware is distributed to victims via spam email attachments. It encrypts user-generated files once installed.

Key on hacker machine or attacker network

These variants of ransomware are the ones in which ransomware writers are smart and keep control over the ransomware key. There are many ways to protect the key from getting into the hands of the victim. In this section, we will talk about the theory behind these types of ransoms. One among them are ransoms that use public key cryptography, which means they use public key and private key for encryption and decryption. How they work is explained in the following steps:

1. Ransomware writer generates both public and private key.
2. The public key is embedded in the ransomware code and delivered to the victim using different attack vectors.
3. Once the ransomware is executed, it extracts the public key from the ransomware binary.
4. Once the ransomware has the public key, it starts encrypting user data with that public key.
5. Victim is shown the message that they need to pay ransom in order to decrypt victim data.
6. On paying the ransom, ransomware writer shares private key to victim to decrypt the data.

The major benefit of these types of ransomware is that the ransomware decryption key is a private key that never leaves the ransomware writer's domain. The major drawback of public key cryptography is that the same private key, once received from ransomware writer, is sufficient to decrypt data of other victims infected with the same ransomware variant. Thus, there is no need to pay more ransom to hacker.

To overcome this drawback, ransomware writers twisted the ransomware model by generating public as well as private key specific to a victim. One such example of ransomware that uses the **Command & Control (C&C)** server to overcome the drawback is CryptoLocker ransomware. It works as illustrated in [Figure 5.5](#):

Figure 5.5: *CryptoLocker Ransomware*

CryptoLocker ransomware uses the C&C server for its execution in the following manner:

1. After compromising the victim ransomware, it sends a notification to the C&C server, which is controlled by the hacker or attacker.
2. The C&C server acknowledges the victim and request ID from the victim.
3. Victim machine sends the unique ID and Campaign ID to the C&C server.
4. On receiving unique attributes of the victim machine, the C&C server generates the public and private key pair. This is where ransomware generates a unique key pair for every victim. C&C sends the public key to the victim.
5. On receipt of the public key, the victim acknowledges to C&C server about its receipt.
6. The victim data is encrypted with this public key.
7. On paying ransom to hacker/attacker, the private key to decrypt victim data is released from C&C server to the victim.

Communication from the victim computer to the C&C server may not be encrypted in the ransomware communication with the C&C server. In some variants of ransomware, the communication channel is encrypted with custom encryption, and new variants use the **Transport Layer Security (TLS)** encryption.

This communication between ransomware and the C&C server became the major reason for the failure of ransomwares using this approach. To protect organization or individuals from this ransomware, the following techniques were used:

1. Blocking the communication between ransomware and the C&C server.
2. As the ransomware grew, the C&C servers were blacklisted by organizations and individuals, to break the ransomware functioning

chain.

With these approaches, ransomware operations crumbled and led to a new variant of ransomware. The operation of this new variant is the same as the earlier one, but the key pair was generated on the victim computer, as shown in [Figure 5.6](#):

Figure 5.6: *Ransomware with C&C and key pair on victim*

CryptoDefence was one among various ransomware that generated keys on the victim end and further deleted the private key in the following manner:

1. Once the victim is compromised, the ransomware sends notification to C&C server, which is controlled by the hacker or attacker.
2. The C&C server acknowledges the victim and request ID from the victim.
3. The victim machine sends the unique ID and Campaign ID to the C&C server.
4. Ransomware uses crypto API to generate public and private key pair on the victim machine.
5. Ransomware uses the public key generated to encrypt victim data.
6. After encrypting victim data, ransomware sends the private key to the C&C server.
7. Then, ransomware destroys the private key on the victim's computer and displays a ransom note to the victim.
8. On paying the ransom to the hacker/attacker, the private key to decrypt victim data is released from the C&C server to the victim for decryption.

The advantage of this approach is that the key pair is generated on the victim machine and thus, there is no dependency of communication channel. But the major flaw in this approach is that of the restoration possibilities of the private key from the victim machine. Once the ransomware deletes the private key from the victim machine after encrypting user data, this key can be restored using forensics techniques in some cases.

All the ransomware variants we discussed till now require user involvement during infection, like clicking on a link, opening malicious attachment, or installing some software. Moreover, these variants had some or the other drawback. But with the birth of WannaCry ransomware, drawbacks in earlier variants were overcome, and threat vector changed to vulnerability exploitation.

WannaCry got special attention because of its distribution method. It does not necessitate active user involvement, like clicking on a malicious link. It spreads like a worm by exploiting an unpatched vulnerability on the victim machine. However, the encryption model is different from that of the previous ransomware models. [Figure 5.7](#) describes the encryption procedure in a WannaCry infection:

Figure 5.7: *WannaCry Ransomware*

1. The attacker generates the master public key and master private key.
2. The attacker embeds the master public key in WannaCry ransomware code.
3. WannaCry ransomware spreads on the internet like a worm and exploits vulnerability to get into victim machine.
4. The victim machine is then infected with WannaCry ransomware.
5. The ransomware runs on the victim machine using crypto APIs and embeds the master public key.
6. Ransomware code generates **Rivest Shamir Adleman (RSA)** key pair for victim, that is, the Victim public key and the Victim private key. Victim private key is encrypted with the Master public key, embedded in the ransomware code or binary.
7. AES key per file to be encrypted is generated using ransomware code. Since symmetric algorithm is fast, it was used to encrypt user data. The ransomware encrypts all AES keys with the Victim public key generated in step 6.
8. Before encryption, the user data was in readable form.
9. After encryption with AES key generated in step 7, the user data becomes inaccessible.

10. After encryption, WannaCry deleted all AES keys from victim machine that cannot be recovered.
11. WannaCry displayed ransom note to victim.

There were many advantages of the approach used in WannaCry ransomware. They are as follows:

1. Symmetric encryption (such as AES) is used to encrypt files, so encryption is done pretty quickly.
2. Only during the payment process does communication with an external entity, such as C&C server, occur.
3. If the victim can extract the symmetric key used to encrypt one file, then using it, the victim will only be able to decrypt that one file. The remaining files will continue to be kept as hostage, as the AES key per file is generated using ransomware code.

Conclusion

In this chapter, we learned about the key management techniques used by ransomware writers. We also covered the concept behind the origin of key management process and understood how it evolved over time.

Additionally, we covered ransomware that used no key management technique to scare users and demand a ransom. We then walked over the key management technique, where key generation and handling is done on the victim machine. Furthermore, we talked about the technique where key management happens on the hacker or attacker network. It means key management is within the control of the attacker. Toward the end of this chapter, we covered different variants of ransomware that used key management on their network and also spoke about WannaCry ransomware key generation model.

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

<https://discord.bpbonline.com>

Section II: Ransomware Internals

CHAPTER 6

Internal Secrets of Ransomware

Introduction

Most applications today are incorporated with the capabilities of encryption. So how do programmers add these encryption capabilities into applications? Microsoft started adding these capabilities as extensions in the Internet Explorer, which was the first released web browser of Microsoft. But with time, these capabilities became the core of the operating system. Now, advanced encryption capabilities are available for programmers through simple API calls. These capabilities were built to make the internet a safe place, but with good also comes a bad side. Today, malware writers are using these encryption capabilities to build ransomware. In this chapter, we will discuss Crypto API concepts.

Structure

In this chapter, we will discuss the following topics:

- Cryptographic Application Programming Interface (API)
- What crypto API provides
- Use of functions in cryptographic API
- Definition and declarations of crypto functions
- Example codes to understand crypto API functions
- Types of blob

Objectives

The objective of this chapter is to help you understand ransomware internals. When we talk about ransomware internals, we mean how malware writers develop and architect ransomware. In order to break ransomware, an understanding of its internals and working is necessary. Cryptographic API

is used to add encryption capabilities in the application. We will see how Cryptographic API is used to develop ransomware. In this chapter, we will walk through the Cryptographic API and the different functions it exposes. With the help of examples, we will understand the usage of different functions. Furthermore, we will use reverse engineering techniques to understand crypto API disassembled code. We will use x32dbg and IDA freeware for reverse engineering purposes. In this chapter, we will understand ransomware from the malware developer point of view.

Crypto API

Different applications may need different types of encryption strength. Taking this into consideration, Microsoft developed Crypto API to solve the needs of different types of encryption in different applications. This means that the Crypto API is shipped with some plug-in encryption engines, also called **Cryptographic Service Provider (CSP)**. Each CSP is developed to serve basic encryption algorithms. Looking from the application development point of view, a programmer needs to use specific CSP to implement a particular type of encryption in application. All these CSPs are programmed with the help of mathematicians, as they involve many complex mathematical formulas to implement encryption algorithms. Now, we will walk you through the process of implementing encryption in a simple application or program.

CryptAcquireContext

The first step to implementing encryption using Crypto API in an application is to open a handle to appropriate CSP. The handle can be opened using the **CryptAcquireContext** function. The syntax for this is as follows:

```
BOOL CryptAcquireContextA(  
    HCRYPTPROV *phProv,  
    LPCSTR     szContainer,  
    LPCSTR     szProvider,  
    DWORD      dwProvType,  
    DWORD      dwFlags  
);
```

phProv

The first parameter in the CryptAcquireContext function is **phProv**, which is the handle to **Cryptographic Service Provider (CSP)**. This variable is used thought out the application using encryption.

szContainer

The second parameter is the name of Key Container. Every **Cryptographic Service Provider (CSP)** maintains a database to store encryption keys, which are used through application sessions. Public and private keys are stored in the key container (also referred as key store), but session encryption keys are not stored in it. Some CSPs maintain their key store in registry, while some store the key internally in the hardware like smart cards. When this parameter is NULL, CSP will use the current user's login name as the name of key container. If an application is using private keys, then it is recommended to not use the default key store, irrespective of the CSP. Using the default key store will enable other applications to overwrite it.

szProvider

The third parameter is the name of CSP to be used. If NULL is specified, then the default CSP will be used. [Table 6.1](#) features a list of CSP defined in **wincrypt.h**:

Constant/value	Description
MS_DEF_DH_SCHANNEL_PROV	The Microsoft DSS and Diffie-Hellman/Schannel Cryptographic Provider
MS_DEF_DSS_DH_PROV	The Microsoft Base DSS and Diffie-Hellman Cryptographic Provider
MS_DEF_DSS_PROV	The Microsoft DSS Cryptographic Provider
MS_DEF_PROV	The Microsoft Base Cryptographic Provider
MS_DEF_RSA_SCHANNEL_PROV	The Microsoft RSA/Schannel Cryptographic Provider
MS_DEF_RSA_SIG_PROV	The Microsoft RSA Signature Cryptographic Provider is not supported
MS_ENH_DSS_DH_PROV	The Microsoft Enhanced DSS and Diffie-Hellman Cryptographic Provider

MS_ENH_RSA_AES_PROV	The Microsoft AES Cryptographic Provider
MS_ENHANCED_PROV	The Microsoft Enhanced Cryptographic Provider
MS_SCARD_PROV	The Microsoft Base Smart Card Cryptographic Service Provider
MS_STRONG_PROV	The Microsoft Strong Cry

Table 6.1: CSP defined in Wincrypt.h

dwProvType

The fourth parameter is the CSP type. The provider type should be chosen based on the support of provider selected in the third parameter. [Table 6.2](#) is a list of the CSP types:

Constant	Description
PROV_RSA_FULL	This general-purpose type supports both digital signatures and encryption.
PROV_RSA_SIG	A subset of the PROV_RSA_FULL type, this type is for hashes and digital signatures only.
PROV_DSS	This type implements the Digital Signature Algorithm (DSA) for hashes and digital signatures only.
PROV_FORTEZZA	This type implements a series of National Institute of Standards and Technology (NIST) algorithms.
PROV_MS_EXCHANGE	This provider type is designed for use with Microsoft Exchange and other MS Mail-compatible applications.
PROV_SSL	This provider type supports the Secure Sockets Layer (SSL) protocol.
PROV_RSA_SCHANNEL	This general-purpose type supports both the RSA and SChannel protocols.
PROV_DSS_DH	A superset of the PROV_DSS type, this type is for both digital signatures and encryption.

Table 6.2: CSP types

dwFlags

This is the last parameter of the **CryptAcquireContext** function. It specifies how the CSP handle needs to be opened. The options available are used to

create key containers and prevent applications from accessing private keys. When this parameter is set to NULL, **dwFlags** is set to **CRYPT_VERIFYCONTEXT**. [Table 6.3](#) features the available flags:

Flags	Description
CRYPT_VERIFYCONTEXT	This flag is used when the application does not require access to private keys. When this flag is used, the szContainer parameter should be NULL.
CRYPT_NEWKEYSET	With this flag, a new Key Container will be created with the name mentioned as the second parameter.
CRYPT_MACHINE_KEYSET	Key container can be user container or machine container. When it is user container, it means keys are stored as user keys and are stored in user profile. On the other hand, when this flag is used, key container is machine container and the application is running as a service.
CRYPT_DELETEKEYSET	When this flag is used, the key container specified using the second parameter is deleted. If the szContainer parameter is NULL, then default key container will be deleted.
CRYPT_SILENT	It is used when the application for which user interface cannot be displayed.
CRYPT_DEFAULT_CONTAINER_OPTIONAL	This flag is only with smart card CSP.

Table 6.3: Flags

The **CryptAcquireContext** function returns the Boolean value, and the resultant Boolean value is used to determine whether the function is executed successfully.

[CryptAcquireContext Example](#)

[Figure 6.1](#) shows an example of the *CryptAcquireContext* code:

```

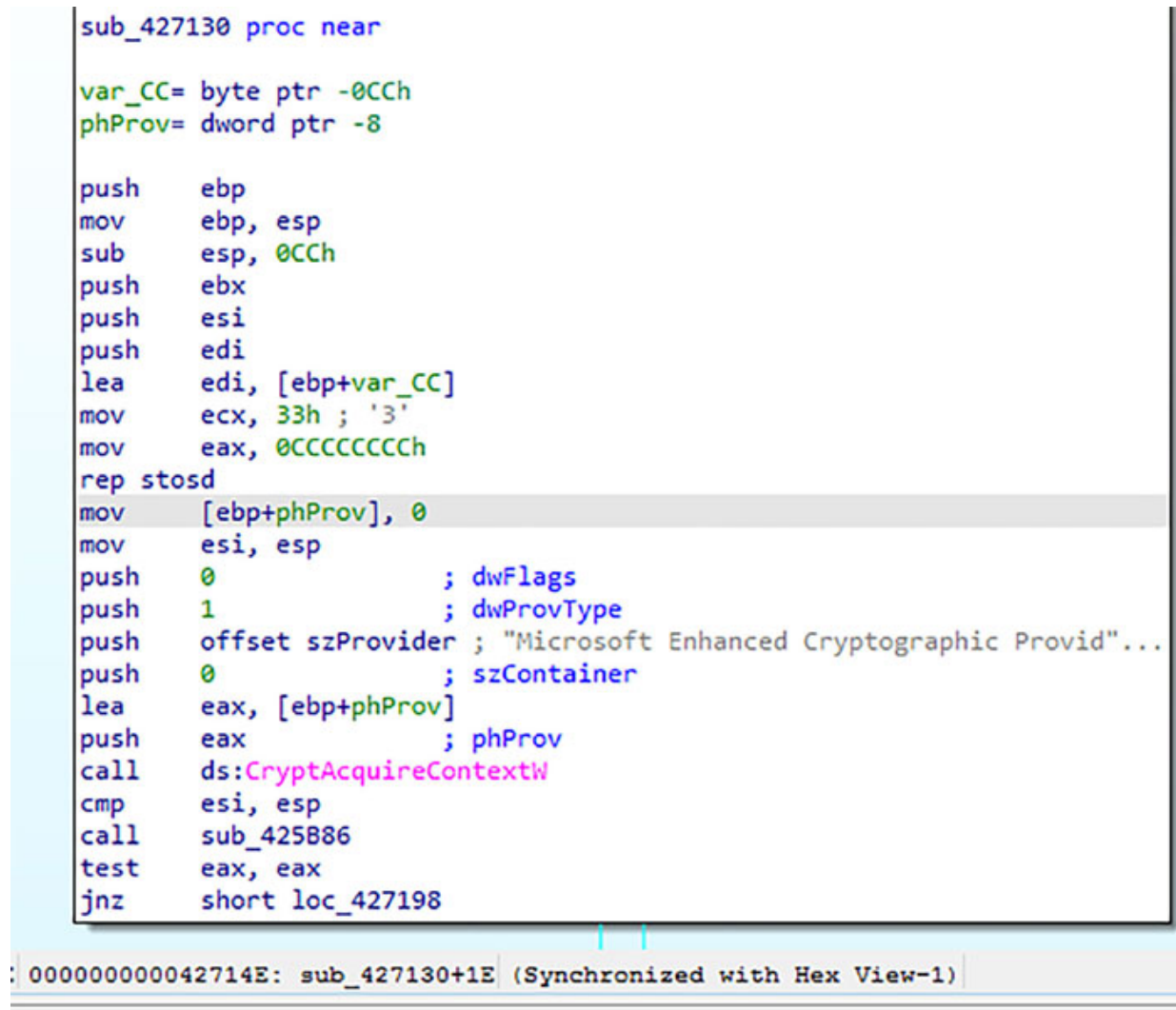
01. // CryptAcquireContext.cpp : Defines the entry point for the console application.
02. //
03.
04. #include "stdafx.h"
05. #include <windows.h>
06. #include <wincrypt.h>
07.
08. // Link with the Advapi32.Lib file.
09. #pragma comment (lib, "advapi32")
10.
11. int _tmain(int argc, _TCHAR* argv[])
12. {
13.     HCRYPTPROV hProv = NULL;
14.
15.     // Get handle to the Microsoft Enhanced Cryptographic Provider.
16.     if(!CryptAcquireContext(
17.         &hProv,
18.         NULL,
19.         MS_ENHANCED_PROV,
20.         PROV_RSA_FULL,
21.         0))
22.     {
23.         printf("Error %x during CryptAcquireContext!\n", GetLastError());
24.         return 0;
25.     }
26.     // Add cryptographic operation code.
27.
28.     // Release provider handle.
29.     if(!CryptReleaseContext(hProv, 0))
30.     {
31.         printf("Error %x during CryptReleaseContext!\n", GetLastError());
32.         return 0;
33.     }
34.     return 0;
35. }

```

Figure 6.1: CryptAcquireContext Code

In the preceding code, we are opening a handle to Microsoft Enhanced Cryptographic Provider CSP. We will compile this code to generate executable code, which will be reverse engineered using IDA freeware and x32dbg. We are compiling this code on Windows 7 32-bit environment using Microsoft Visual Studio. The resultant binary will have **Address Space Layout Randomization (ASLR)** enabled. So, to disable ASLR, we will open compiled binary in CFF Explorer software and go to *Optional Header | DLLCharacteristics* | *Click here* | Uncheck *DLL can move*.

Let's move on to [Figure 6.2](#) to understand the assembly listing of the *CryptAcquireContext* code using IDA freeware:



```
sub_427130 proc near
var_CC= byte ptr -0CCh
phProv= dword ptr -8

push    ebp
mov     ebp, esp
sub     esp, 0CCh
push    ebx
push    esi
push    edi
lea     edi, [ebp+var_CC]
mov     ecx, 33h ; '3'
mov     eax, 0CCCCCCCCh
rep stosd
mov     [ebp+phProv], 0
mov     esi, esp
push    0          ; dwFlags
push    1          ; dwProvType
push    offset szProvider ; "Microsoft Enhanced Cryptographic Provid"...
push    0          ; szContainer
lea     eax, [ebp+phProv]
push    eax        ; phProv
call    ds:CryptAcquireContextW
cmp     esi, esp
call    sub_425B86
test    eax, eax
jnz     short loc_427198

: 000000000042714E: sub_427130+1E (Synchronized with Hex View-1)
```

Figure 6.2: IDA Freeware

In the assembly listing, all the parameters to *CryptAcquireContext* are pushed on to the stack one by one in the order from right to left. As per our C/C++ code, at line 13, **phProv** is initialized to NULL; this can be observed in assembly listing **MOV [EBP+phProv], 0** instruction.

Moving further down our last parameter, *dwFlags* is pushed first on to the stack, followed by the other parameters. The last parameter to push before the call to the *CryptAcquireContext* function is **phProv**, which is the handle to **Cryptographic Service Provider (CSP)**. This variable is used throughout the application using encryption. To understand the memory state

of assembly listing, we will open the binary in x32dbg disassembler by putting a breakpoint at the **MOV [EBP+phProv]** instruction at 0x0042714E. This is obtained from IDA by hovering the cursor over the instruction, as shown in [Figure 6.2](#).

Once the breakpoint is hit, we can see that all the parameters are highlighted in x32dbg with the help of the *xAnalyzer* plug-in, as can be seen in [Figure 6.3](#):

```

00427130  > push ebp
00427131  . mov ebp,esp
00427133  . sub esp,CC
00427139  . push ebx
0042713A  . push esi
0042713B  . push edi
0042713C  . lea edi,dword ptr ss:[ebp-CC]
00427142  . mov ecx,33
00427147  . mov eax,CCCCCCC
0042714C  . rep stosd
0042714E  . mov dword ptr ss:[ebp-8],0
00427155  . mov esi,esp
00427157  . push 0
00427159  . push 1
0042715B  . push cryptacquirecontext.471CD0
0042715E  . push 0
00427160  . lea eax,dword ptr ss:[ebp-8]
00427165  . push eax
00427166  . call dword ptr ds:[<&CryptAcquireContextW>]

```

cryptacquirecontext.cpp:12
33: '3'

cryptacquirecontext.cpp:13
cryptacquirecontext.cpp:21
DWORD dwFlags = CRYPT_NEWKEYSET
DWORD dwProvType = PROV_RSA_FULL
LPCTSTR pszProvider = "Microsoft Base Cryptographic Provider v1.0"
LPCTSTR pszContainer = "Foo"

Figure 6.3: x32dbg with breakpoint

Furthermore, we will look at the instructions one by one until the last parameter to push. Before the call instruction, our stack state will be as shown in [Figure 6.4](#):

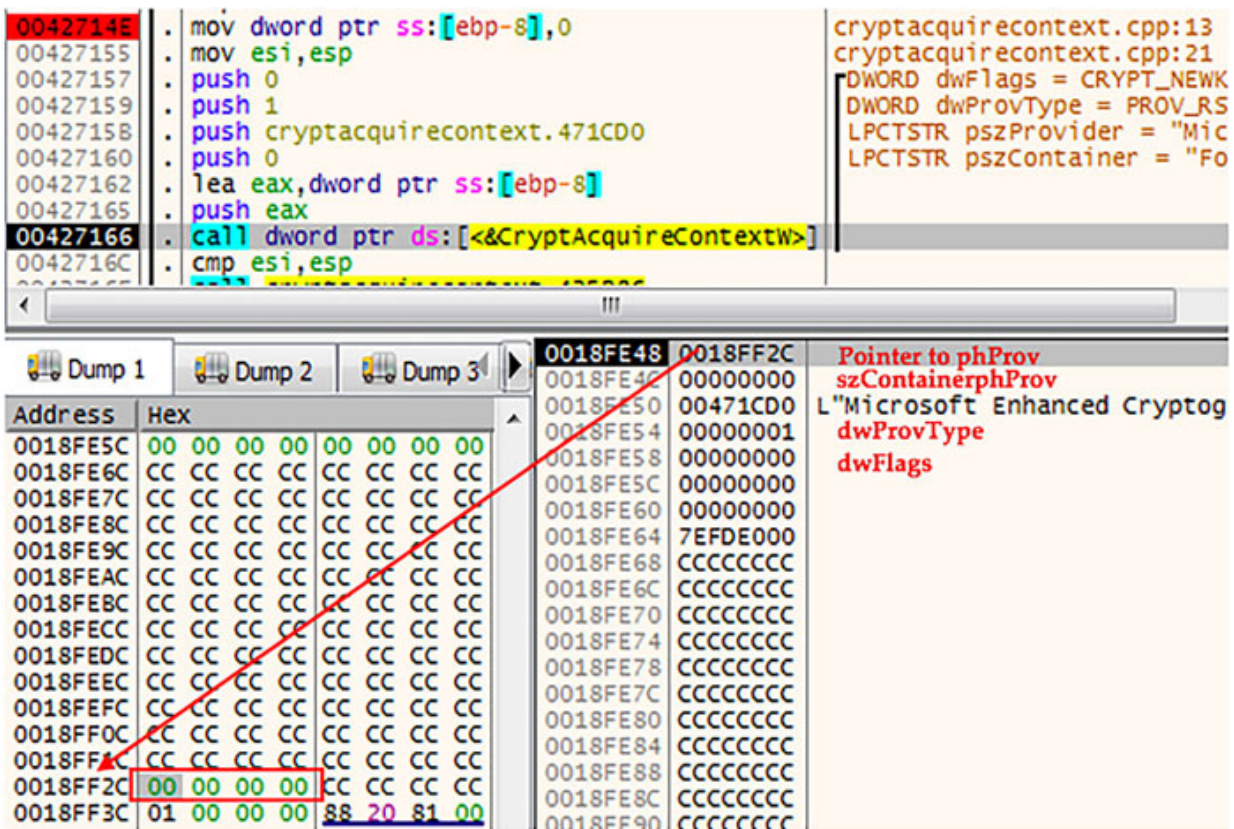


Figure 6.4: Stack state

On stepping over the call instruction, we will see the function return value in EAX as 0x01. This is the indication of the successful execution of the *CryptAcquireContext* function, with 1 as the Boolean return value.

The handle to CSP will be stored at 0x0018FF2C, which is the placeholder to store pointer to CSP. As we can see in [Figure 6.4](#), before the call instruction 0x0018FF2C memory location is NULL. On stepping over the function call instruction, 0x0018FF2C will hold pointer to CSP (0x00556F00), as shown in [Figure 6.5](#):

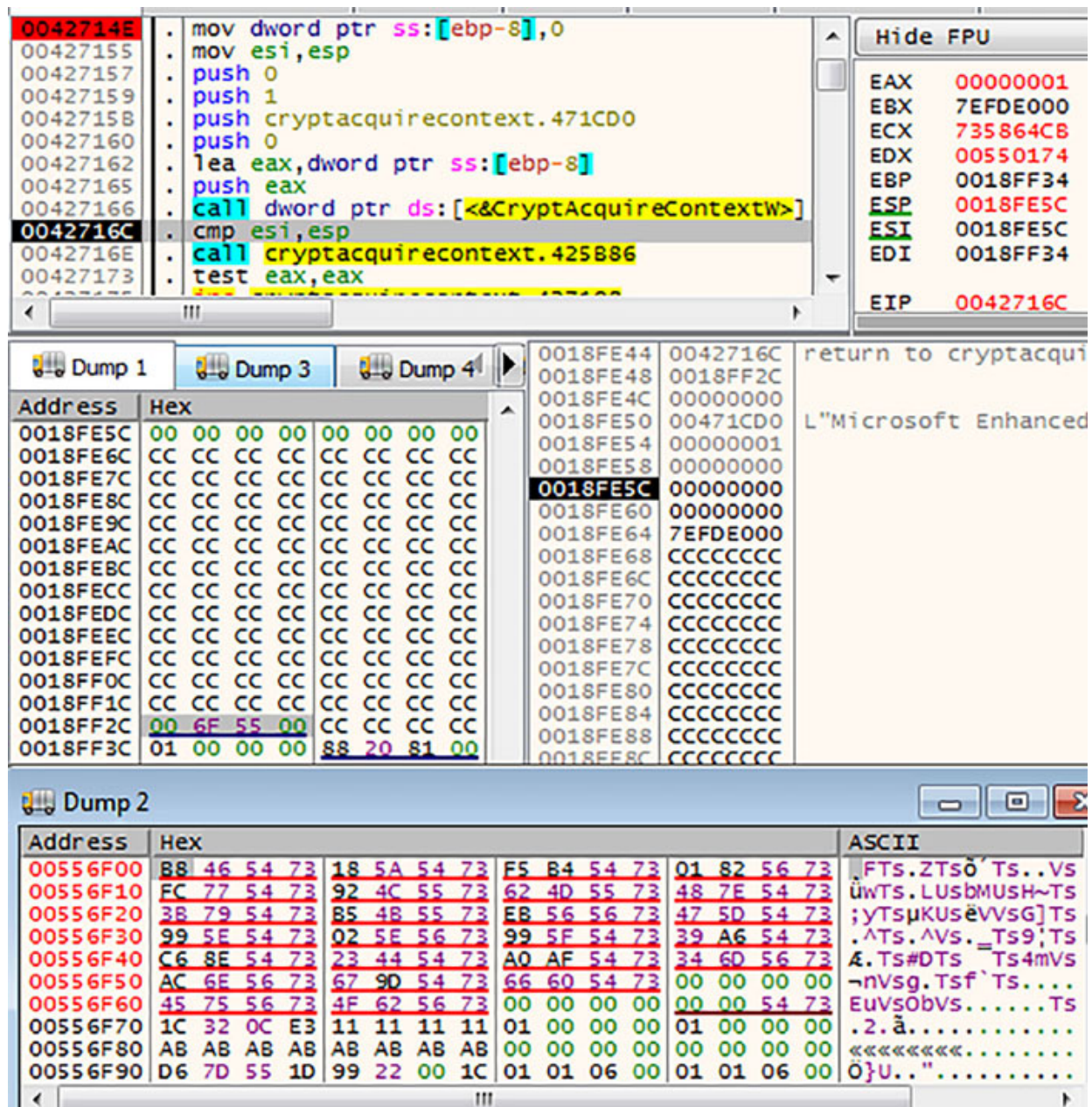


Figure 6.5: Pointer to CSP

CryptGenKey

The **CryptGenKey** function is used to generate encryption keys for symmetric encryption and public/private keys for asymmetric encryption. After getting the handle to CSP, we need encryption keys:

```
BOOL CryptGenKey(
    HCRYPTPROV hProv,
```



```

    ALG_ID      Algid,
    DWORD      dwFlags,
    HCRYPTKEY    *phKey
);

```

The handle to the key or key pair is returned in **phKey**. The parameters passed to the function are as follows:

hProv

The first parameter is the handle to CSP created with the **CryptAcquireContext** function.

Algid

The second parameter is the algorithm ID with which the key will be used. Depending on the CSP algorithm, ID will be used from [Table 6.4](#):

Algorithm ID	Description
CALG_RC2	RC2 block encryption algorithm
CALG_RC4	RC4 stream encryption algorithm
AT_KEYEXCHANGE	This Algorithm ID will be used to generate public/private key pair
AT_SIGNATURE	This Algorithm ID will be used to generate signature public/private key pair
CALG_DH_EPHEM	“Ephemeral” Diffie-Hellman key
CALG_DH_SF	“Store and Forward” Diffie-Hellman

Table 6.4: Algorithm ID

dwFlags

This parameter is used to specify the options used in generating keys. **KEYLENGTH** is the key size, the upper bits of 0x08000000 are 0x0800 or 2048 in decimal and denotes 2048 bit RSA. **KEYLENGTH** is used in combination with flags using OR. [Table 6.5](#) features the different flags used in dwFlags:

Options	Description
CRYPT_EXPORTABLE	This flag allows keys to be exportable using the CryptExportKey function. If this flag is not set, session key in the case of symmetric

	and public/private keys in the case of asymmetric generated using the CryptGenKey function are not exportable. If keys are not exportable, they can only be used during the duration of application session. However, if we want to use a key again or exchange it across computers or applications, then we will have to use this flag. So, when this flag is set, a key can be transferred out of CSP into key BLOB using the CryptExportKey function.
CRYPT_USER_PROTECTED	When an action is trying to use the derived keys, then this flag notifies the user with a dialog box. The exact behavior of this flag is governed by the CSP being used. If CSP is opened using the CRYPT_SILENT flag, this flag will result in an error.
CRYPT_CREATE_SALT	Random salt value is assigned to keys when this flag is used, and salt value of zero is assigned if flag is not set.
CRYPT_NO_SALT	With this flag, no salt value is allocated to keys.
CRYPT_PGEN	This flag is only applicable to Diffie-Hellman CSP.

Table 6.5: Flags used in dwFlags

phKey

This is the last parameter that is the handle to generated keys. This handle should not be deleted until all the encryption and decryption are done.

Function return value is nonzero if successful; otherwise, it will return zero.

CryptGenKeyExample

Figure 6.6 is an example of the CryptGenKey code:

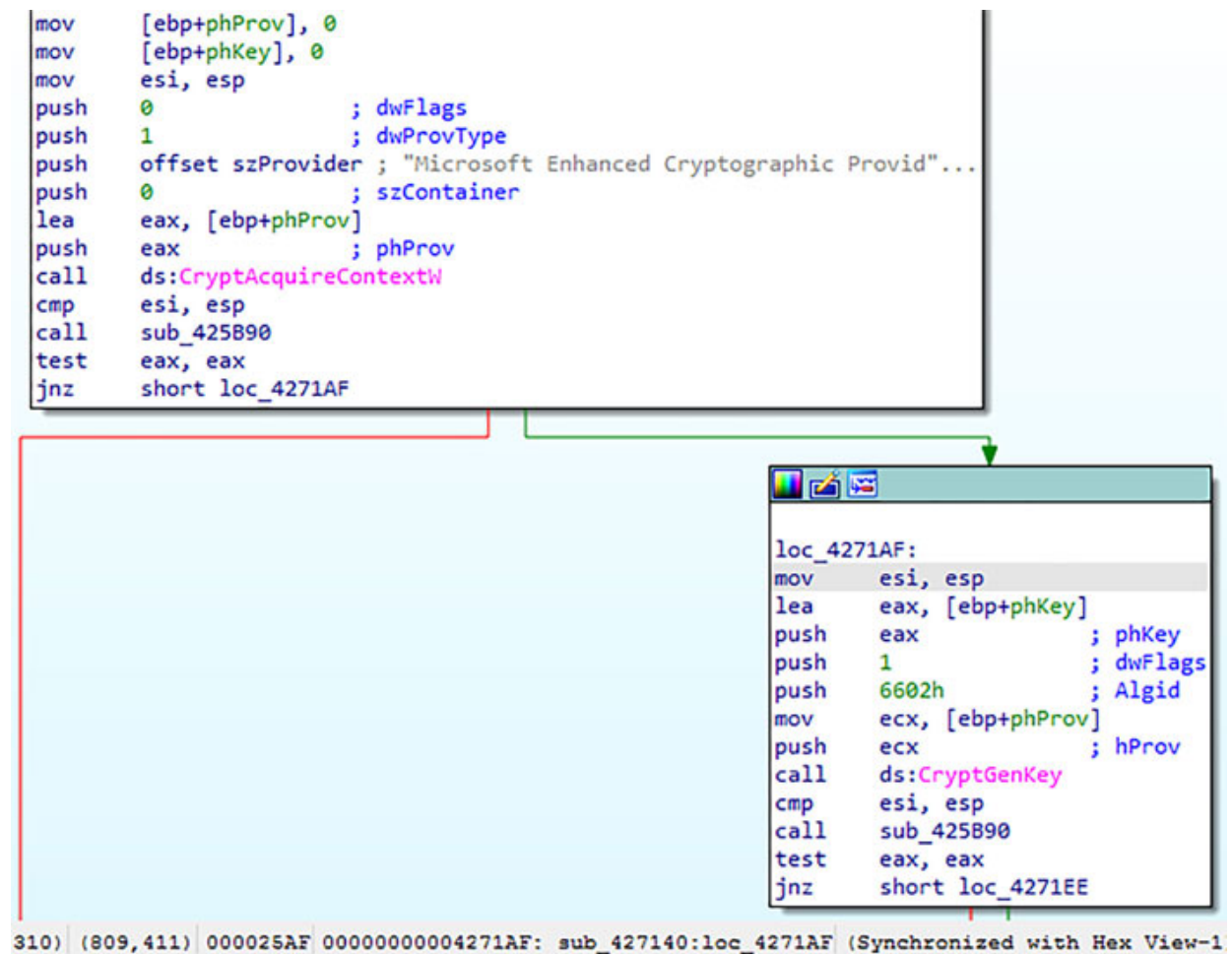
```

01. // CryptGenKey.cpp : Defines the entry point for the console application.
02. //
03.
04. #include "stdafx.h"
05. #include <windows.h>
06. #include <wincrypt.h>
07.
08. // Link with the Advapi32.lib file.
09. #pragma comment (lib, "advapi32")
10.
11. int _tmain(int argc, _TCHAR* argv[])
12. {
13.     HCRYPTPROV hProv = NULL;
14.     HCRYPTKEY hKey = NULL;
15.
16.     // Get handle to the Microsoft Enhanced Cryptographic Provider.
17.     if(!CryptAcquireContext(
18.         &hProv,
19.         NULL,
20.         MS_ENHANCED_PROV,
21.         PROV_RSA_FULL,
22.         0))
23.     {
24.         printf("Error %x during CryptAcquireContext!\n", GetLastError());
25.         goto exit;
26.     }
27.
28.     // Create block cipher session key.
29.     if(!CryptGenKey(
30.         hProv,
31.         CALG_RC2,
32.         CRYPT_EXPORTABLE,
33.         &hKey))
34.     {
35.         printf("Error %x during CryptGenKey!\n", GetLastError());
36.         goto exit;
37.     }
38.
39.     // 'hKey' is used to perform something.
40.
41.
42.     exit:
43.     // Destroy session key.
44.     if(hKey != 0) CryptDestroyKey(hKey);
45.     // Release provider handle.
46.     if(hProv != 0) CryptReleaseContext(hProv, 0);
47.     return 0;
48. }

```

Figure 6.6: *CryptGenKey Code*

In the preceding code, we are opening a handle to Microsoft Enhanced Cryptographic Provider CSP to be used as a parameter to *CryptGenKey*. Following the same steps as for **CryptAcquireContext** analysis, this code is compiled to generate executable, which will be reverse engineered using IDA freeware and x32dbg. We are compiling this code on Windows 7 32-bit environment using Microsoft Visual Studio. The resultant binary will have Address space layout randomization (ASLR) enabled. So, to disable ASLR, we will open compiled binary in CFF Explorer software and go to *Optional Header | DLLCharacteristics| Click here|* Uncheck *DLL can move*. As we have already covered *CryptAcquireContext* analysis, we will move on to *CryptGenKey* Code using IDA freeware. Refer to [Figure 6.7](#) for an IDA freeware interface:



```
mov     [ebp+phProv], 0
mov     [ebp+phKey], 0
mov     esi, esp
push    0                ; dwFlags
push    1                ; dwProvType
push    offset szProvider ; "Microsoft Enhanced Cryptographic Provid"...
push    0                ; szContainer
lea     eax, [ebp+phProv]
push    eax              ; phProv
call    ds:CryptAcquireContextW
cmp     esi, esp
call    sub_425890
test    eax, eax
jnz     short loc_4271AF

loc_4271AF:
mov     esi, esp
lea     eax, [ebp+phKey]
push    eax              ; phKey
push    1                ; dwFlags
push    6602h            ; Algid
mov     ecx, [ebp+phProv]
push    ecx              ; hProv
call    ds:CryptGenKey
cmp     esi, esp
call    sub_425890
test    eax, eax
jnz     short loc_4271EE
```

(310) (809,411) 000025AF 000000000004271AF: sub_427140:loc_4271AF (Synchronized with Hex View-1)

Figure 6.7: *IDA Freeware*

In the assembly listing, all the parameters to *CryptGenKey* are pushed on to the stack one by one in the order from right to left. As per our C/C++ code, at line 33, *phkey*, which is the handle to generated keys, is the last parameter. So, it will be the first to be pushed on to the stack in the assembly listing **push EAX** instruction. But before EAX is pushed on to the stack, it is filled with the memory location (which is a generated keys pointer placeholder at the **EBP+phKey** location).

Moving further down, the last parameter to push before the call to the **CryptGenKey** function is **hProv**, which is the handle to **Cryptographic Service Provider (CSP)**. To understand the memory state of assembly listing, we will open the binary in x32dbg disassembler by putting breakpoint at *MOV ESI, ESP* instruction at 0x0042714F. This is obtained from IDA by hovering the cursor over the instruction, as shown in [Figure 6.7](#).

Once the breakpoint is hit, we can see that all the parameters are highlighted in x32dbg with the help of the xAnalyzer plugin, as can be seen in [Figure 6.8](#):

Address	Assembly	C++ Code
0042715E	mov dword ptr ss:[ebp-8],0	cryptgenkey.cpp:13
00427165	mov dword ptr ss:[ebp-14],0	cryptgenkey.cpp:14
0042716C	mov esi,esp	cryptgenkey.cpp:22
0042716E	push 0	DWORD dwFlags = 0
00427170	push 1	DWORD dwProvType = PROV_RSA_FULL
00427172	push cryptgenkey.471CC0	LPCTSTR pszProvider = "Microsoft Enhanc
00427177	push 0	LPCTSTR pszContainer = NULL
00427179	lea eax,dword ptr ss:[ebp-8]	HCRYPTPROV* phProv
0042717C	push eax	CryptAcquireContextW
0042717D	call dword ptr ds:[<&CryptAcquireContextW>]	
00427183	cmp esi,esp	
00427185	call cryptgenkey.425B90	
0042718A	test eax,eax	
0042718C	jne cryptgenkey.4271AF	
0042718E	mov esi,esp	cryptgenkey.cpp:24
00427190	call dword ptr ds:[<&GetLastError>]	
00427196	cmp esi,esp	
00427198	call cryptgenkey.425B90	
0042719D	push eax	
0042719E	push cryptgenkey.471C90	471C90:"Error %x during CryptAcquireCont
004271A3	call cryptgenkey.425E00	
004271A8	add esp,8	
004271AB	jmp <cryptgenkey.exit>	cryptgenkey.cpp:25
004271AD	jmp <cryptgenkey.exit>	
004271AF	mov esi,esp	cryptgenkey.cpp:33
004271B1	lea eax,dword ptr ss:[ebp-14]	HCRYPTKEY* phKey
004271B4	push eax	DWORD dwFlags = 1
004271B5	push 1	unsigned int Algid = CALG_RC2
004271B7	push 6602	
004271B8	mov ecx,dword ptr ss:[ebp-8]	HCRYPTPROV hProv
004271BF	push ecx	CryptGenKey
004271C0	call dword ptr ds:[<&CryptGenKey>]	

Figure 6.8: x32dbg with breakpoint

Furthermore, we will look at the instructions one by one until the last parameter to push. Before the call instruction, our stack state will be as shown in [Figure 6.9](#):

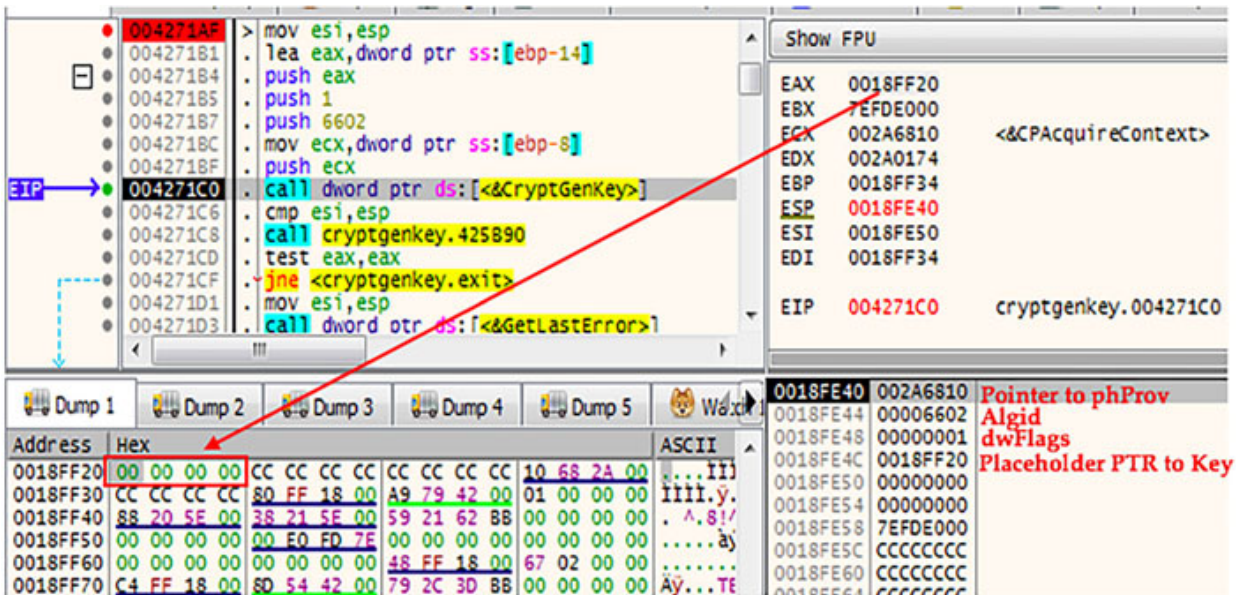


Figure 6.9: Stack state

On stepping over the call instruction, we will see the function return value in EAX as 0x01. This indicates the successful execution of the **CryptGenKey** function with 1 as the Boolean return value.

The handle to keys will be stored at 0x0018FF20, which is the placeholder to store the pointer to key. As we can see in [Figure 6.9](#), before the call instruction 0x0018FF20, memory location is NULL. On stepping over the function call instruction, 0x0018FF20 will hold pointer to key (0x002B1F90), as shown in [Figure 6.10](#):

Figure 6.10: Pointer to Keys

[CryptGetUserKey](#)

When we call the **CryptAcquireContext** function, key pairs are automatically stored in the key store. The **CryptGetUserKey** function is used to get public/private keys from key store.

```
BOOL CryptGetUserKey(
    HCRYPTPROV hProv,
    DWORD      dwKeySpec,
    HCRYPTKEY  *phUserKey
);
```

The **CryptGenKey** function is used to get public/private keys for asymmetric encryption from key store. The parameters passed to the function are as follows:

hProv

The first parameter is the handle to CSP created with the **CryptAcquireContext** function.

dwKeySpec

It specifies the key to be used further in code and accepts two values: **AT_KEYEXCHANGE** and **AT_SIGNATURE**.

phUserKey

This stores the handle to opened keys, which is the same as the handle we created using the **CryptGenKey** function.

CryptGetUserKey Example

[Figure 6.11](#) is an example of the **CryptGetUserKey** code:

Figure 6.11: CryptGenUserKey Code

As did in **CryptGenKey**, opening a handle to Microsoft Enhanced Cryptographic Provider CSP and then calling **CryptGenKey** function. We are following the same process in the **CryptGenUserKey** function example. Now, we will compile this code to generate an executable, which will be reverse engineered using IDA freeware and x32dbg. We are compiling this code on Windows 7 32-bit environment using Microsoft Visual Studio. The resultant binary will have Address space layout randomization (ASLR) enabled. So, to disable ASLR, we will open compiled binary in CFF Explorer software and go to *Optional Header | DLLCharacteristics | Click here* | Uncheck *DLL can move*. We will move on to **CryptGenUserKey** Code using IDA freeware, as can be seen in [Figure 6.12](#):

Figure 6.12: IDA Freeware

In the assembly listing, all the parameters to **CryptGenUserKey** are pushed on to the stack one by one in order from right to left. As per our C/C++ code, at line 33, **hXchgKey**, which is the handle to key, exchange key is the last parameter. So, it will be the first to be pushed on to the stack in assembly listing *push EAX* instruction. But before EAX is pushed on to the stack, it is filled with the memory location (which is a key exchange key pointer placeholder at the **EBP+phUserKey** location).

Moving further down, the last parameter to push before the call to the *CryptGenUserKey* function is **hProv**, which is the handle to CSP. To understand the memory state of assembly listing, we will open the binary in x32dbg disassembler by putting breakpoint at *MOV ESI, ESP* instruction at 0x004271AC. This is obtained from IDA by hovering the cursor over the instruction, as shown in [Figure 6.12](#).

Once the breakpoint is hit, we can see that all the parameters are highlighted in x32dbg with the help of the *xAnalyzer* plugin, as shown in [Figure 6.13](#):

Figure 6.13: x32dbg with breakpoint

Furthermore, we will look at the instructions one by one till the last parameter to push. Before the call instruction, our stack state will be as shown in [Figure 6.14](#):

Figure 6.14: Stack state

On making the call instruction, we will see the function return value in EAX as 0x01. This indicates the successful execution of the *CryptGenUserKey* function with 1 as the Boolean return value.

The handle to keys will be stored at 0x0018FF20, which is the placeholder to store the pointer to key. As we can see in [Figure 6.14](#), before the call instruction 0x0018FF20, memory location is NULL. On making the function call instruction, 0x0018FF20 will hold pointer to keys (0x008E1F90), as shown in [Figure 6.15](#):

Figure 6.15: Pointer to Keys

On looking at the call instruction, we will see the function return value in EAX as 0x01. This indicates the successful execution of the **CryptGenUserKey** function with 1 as the Boolean return value.

CryptExportKey

While discussing the **CryptExportKey** function, we will understand the concept behind keys generation, saving of keys and retrieval of keys. While studying the **CryptGenKey** function, we learned about key pair's generation using the **AT_KEYEXCHANGE** and **AT_SIGNATURE** key algorithms. **AT_KEYEXCHANGE** is used to generate public/private key pair that uses algorithm for encrypting data. On the other hand, **AT_SIGNATURE** is used to generate public/private key pair that uses algorithm for signing data.

To understand this conceptually, when the **CryptAcquireContext** function is called to get handle to appropriate CSP, every CSP maintains a database to store encryption keys, which are used through application sessions. When the keys are generated using the **CryptGenKey** function, they are automatically stored by the CSP in key store opened by the **CryptAcquireContext** function.

Now we have understood the concept behind keys generation and how keys are saved, but what about retrieval of keys?

The **CryptGetUserKey** function is used to get keys from the key store, as was discussed earlier.

To export cryptographic keys in a secure manner, we can use the **CryptExportKey** function.

To understand the concept behind exporting keys, we will consider two scenarios:

1. Suppose our cryptographic application is encrypting confidential files and at a later point, we want to use the same keys to decrypt our encrypted files. As CSP does not preserve keys from session to session, we need a mechanism to export keys to be used later.
2. Consider another scenario in which the sender is encrypting data using session key and then transmitting it to the receiver. In order for the receiver to decrypt data, they need the session key from sender. Using the **CryptExportKey** function, the sender will export the session key

from CSP and transmit it from the sender application to the receiver application; it is then imported by the receiver CSP.

When keys are exported, they are converted into series ASCII characters, which are then passed on to other users; when keys are imported, they are converted back into binary format to make the original keys. Exported keys are encrypted with the recipient public key to ensure secure transmission of keys. On the recipient's end, an encrypted key is decrypted using the recipient private key and then imported. The **CryptExportKey** function is declared as follows:

```
BOOL CryptExportKey(  
    HCRYPTKEY hKey,  
    HCRYPTKEY hExpKey,  
    DWORD     dwBlobType,  
    DWORD     dwFlags,  
    BYTE      *pbData,  
    DWORD     *pdwDataLen  
);
```

[hKey](#)

The first parameter to the **CryptExportKey** function is the handle to the key to be exported, which can be public/private key pair or a session key. So, the handle to the key to be exported is passed to the function, which returns a key blob. Exported keys are stored in the encrypted data structure, which is known as key blob.

[hExpKey](#)

The second parameter to the **CryptExportKey** function is the handle to the key that is used to encrypt the exported key. Key data within key blob is encrypted using this key. Now, the value of this parameter varies based on the scenario:

- This parameter is zero (0) if we are exporting public key.
- If we are exporting private key, then this parameter will be a handle to a session key that will be used to encrypt the private key. This pertains to a situation in which the CSP supports.

- Usually it is the handle to recipient public key which is used to encrypt the session key.

dwBlobType

This parameter defines the type of key blob we will be creating. This must be one among the following constants:

- **SIMPLEBLOB**
 - This simple key blob has session key encrypted with the public key of the recipient. This is used when we need to be securing transmitted session key to the recipient. The structure of SIMPLEBLOB is as follows:

```
PUBLICKEYSTRUC publickeystruc;
ALG_ID algid;
BYTE encryptedkey[rsapubkey.bitlen/8];
```

Publickeystruc, also known as blobheader and structure of **Publickeystruc**, is as follows:

```
typedef struct _PUBLICKEYSTRUC {
    BYTE    bType;
    BYTE    bVersion;
    WORD    reserved;
    ALG_ID  aiKeyAlg;
} BLOBHEADER, PUBLICKEYSTRUC
```

bType

It defines the blob type and has predefined values, as shown in [Table 6.6](#):

BLOB TYPE	VALUE	MEANING
KEYSTATEBLOB	0xC	The BLOB is a key state BLOB.
OPAQUEKEYBLOB	0x9	The key is a session key.
PLAINTEXTKEYBLOB	0x8	The key is a session key.
PRIVATEKEYBLOB	0x7	The key is a public/private key pair.
PUBLICKEYBLOB	0x6	The key is a public key.
PUBLICKEYBLOBEX	0xA	The key is a public key.

SIMPLEBLOB	0x1	The key is a session key.
SYMMETRICWRAPKEYBLOB	0xB	The key is a symmetric key.

Table 6.6: Blob type

bVersion

It contains the version number of blob format. If blob is DSS version, its value is 3.

Reserved

This field is reserved for the future and is set to zero.

aiKeyAlg

This identifies the algorithm of the key in key blob. It also has predefined values.

Now, let's go back to the second element of SIMPLEBLOB structure.

AlgId

It defines the encryption algorithm used to encrypt the session key data. For AlgId value, refer to

<https://docs.microsoft.com/en-us/windows/win32/seccrypto/alg-id>

Encryptedkey

This is a BYTE sequence of encrypted session key data in the form of a PKCS #1, type 2 encryption block. This data is always the same size as the public key's modulus. If the size of the public key generated by Microsoft RSA Base provider is 512 bits (64 bytes), then the encrypted session key data will also be of 64 bytes.

• **PUBLICKEYBLOB**

- The public key blob has a public key part of public/private key pair. They are not encrypted, so they contain plain text format of public key. Format of PUBLICKEYBLOB is as follows:

```
PUBLICKEYSTRUC publickeystruc;
RSAPUBKEY rsapubkey;
```

```
BYTE modulus[rsapubkey.bitlen/8];
```

Publickeystruc

This is the same as explained in SIMPLEBLOB.

Rsapubkey

Structure of RSAPUBKEY is as under:

```
Typedef struct _RSAPUBKEY {  
    DWORD magic;  
    DWORD bitlen;  
    DWORD pubexp;  
} RSAPUBKEY;
```

Magic

For public keys, it is set to (0x31415352), which is the ASCII representation of RSA1; for private keys, it is set to (0x32415352), which is the ASCII representation of RSA2.

Bitlen

It is a multiple of 8 bits and defines the number of bits in modulus.

Pubexp

The public exponent.

Modulus

This size of this varies as per the size of public key. The number of bytes can be calculated by dividing *RSAPUBKEY's Bitlen* by 8.

- **PRIVATEKEYBLOB**

- The private key blob has a private key part of public/private key pair. As it is confidential, it is encrypted with symmetric cipher. The format of PRIVATEKEYBLOB is as follows:

```
PUBLICKEYSTRUC publickeystruc;  
RSAPUBKEY rsapubkey;  
BYTE modulus[rsapubkey.bitlen/8];  
BYTE prime1[rsapubkey.bitlen/16];  
BYTE prime2[rsapubkey.bitlen/16];  
BYTE exponent1[rsapubkey.bitlen/16];
```



```
BYTE exponent2[rsapubkey.bitlen/16];
BYTE coefficient[rsapubkey.bitlen/16];
BYTE privateExponent[rsapubkey.bitlen/8];
```

As we can see, this format does not have encryption key parameters and encryption algorithm. **Publickeystruc** and **rsapubkey** we have already discussed earlier in **SIMPLEBLOB** and **PUBLICKEYBLOB**.

modulus

This has a value of prime1 X prime2 and is often known as “n”.

prime1

Prime number 1, often known as “p”.

prime2

Prime number 2, often known as “q”.

exponent1

This has a numeric value of “d mod (p - 1)”.

exponent2

This has a numeric value of “d mod (q - 1)”.

coefficient

This has a numeric value of “(inverse of q) mod p”.

privateExponent

Private exponent, often known as “d”.

- **PLAINTEXTKEYBLOB**

- This blob contains plaintext key and the blob header. The structure of *plaintextblob* is as follows:

```
typedef struct _PLAINTEXTKEYBLOB {
    BLOBHEADER hdr;
    DWORD      dwKeySize;
    BYTE       rgbKeyData[];
} PLAINTEXTKEYBLOB, *PPLAINTEXTKEYBLOB;
```

Hdr

This **Publickeystruc** is the same as the one explained in SIMPLEBLOB.

dwKeySize

The size of the key material in bytes.

rgbKeyData

The key material.

dwFlags

This field specifies additional options. This can be zero or a combination of predefined values.

pbData

This is the pointer to the buffer that holds the key blob data. The size of the buffer is calculated by calling the **CryptExportKey** function with this parameter set to NULL. When the **CryptExportKey** function is called with the **pbData** parameter as NULL, it returns buffer size in the value pointed to by the **pdwDataLen** parameter.

pdwDataLen

It is the pointer to the DWORD value that specifies the size of key blob data in bytes.

With this, we have completed our discussion on the structure of the **CryptExportKey** function; let's move on to the example.

CryptExportKey PlainTextBlob Example

To better understand the concept of **CryptExportKey**, we will take up the **CryptExportKey** example with plaintext blob, as shown in [Figures 6.16](#) and [6.17](#):

Figure 6.16: PLAINTEXTKEYBLOB Code

Figure 6.17: PLAINTEXTKEYBLOB Code

In the preceding code, we are opening a handle to Microsoft Enhanced Cryptographic Provider CSP and then calling the **CryptGenKey** function to generate a random session key. Now, to export keys in plaintext blob, we are calling the **CryptExportKey** function. The first call to the **CryptExportKey** function determines the size of the key BLOB, and allocates memory and second call to export the session key into a Plain Text blob. We compiled this code to generate an executable, which will be reverse engineered using IDA freeware and x32dbg. We are compiling this code on Windows 7 32-bit environment using Microsoft Visual Studio. The resultant binary will have Address space layout randomization (ASLR) enabled. So, to disable ASLR, we will open compiled binary in CFF Explorer software and go to *Optional Header | DLLCharacteristics | Click here | Uncheck DLL can move*. We will move on to *CryptExportKey* Code using IDA freeware, as shown in [Figure 6.18](#):

Figure 6.18: IDA

The first **CryptExportKey** assembly listing is what determines the size of the key BLOB. In the assembly listing, all the parameters to **CryptExportKey** are pushed on to the stack, one by one in order from right to left. On **CryptExportKey** function return, as per our C/C++ code at line 42, data length of key blob will be stored in the **pdwDataLen**. **[EBP+pdwDataLen]** holds the memory location that will have the data length of key blob. Memory location of **[EBP+pdwDataLen]** is moved to EAX register, which is further pushed on to the stack first.

Moving further down, the last parameter to push before the call to the **CryptExportKey** function is **hkey**, which is the handle to session key. Next, the assembly listing shows the second call to the **CryptExportKey** function, as can be seen in [Figure 6.19](#):

```

CryptExportKey4PLAINTEXTKEYBLOB:
mov     esi, esp
lea     eax, [ebp+pdwDataLen]
push    eax                ; pdwDataLen
mov     ecx, [ebp+pbData]
push    ecx                ; pbData
push    0                  ; dwFlags
push    8                  ; dwBlobType
push    0                  ; hExpKey
mov     edx, [ebp+phKey]
push    edx                ; hKey
call    ds:CryptExportKey
cmp     esi, esp
call    sub_425895
test    eax, eax
jz      short loc_4272F9

push    offset aTheKeyHasBeenE ; "The key has been exported. \n"
call    sub_425EE2
add     esp, 4
jmp     short loc_42731A

```

0,102) 000026C5 000000000004272C5: sub_427150:CryptExportKey4PLAINTEXTKEYBLOB (Synchronizer

Figure 6.19: IDA

In assembly listing of the **CryptExportKey** function, all the parameters to **CryptExportKey** are pushed on to the stack one by one in order from right to left. The main parameter that is pushed on to the stack is **dwBlobType** with value of 8. This value represents that the key will be exported in plain text blob. The second parameter of our interest is the *pbData*, which is the pointer to the buffer that holds the key blob data.

To understand the structure of plain text blob, we will open the binary in x32dbg disassembler by putting breakpoint at *MOV ESI, ESP* instruction at 0x004272C5. This is obtained from IDA by hovering the cursor over the instruction, as shown in [Figure 6.19](#).

Once the breakpoint is hit, we can see that all the parameters are highlighted in x32dbg with the help of the xAnalyzer plugin, as shown in [Figure 6.20](#):

```

004272C5 > mov esi, esp
004272C7 . lea eax, dword ptr ss:[ebp-2C]
004272CA . push eax
004272CB . mov ecx, dword ptr ss:[ebp-20]
004272CE . push ecx
004272CF . push 0
004272D1 . push 8
004272D3 . push 0
004272D5 . mov edx, dword ptr ss:[ebp-14]
004272D8 . push edx
004272D9 . call dword ptr ds:[<&CryptExportKey>]
004272DF . cmp esi, esp

```

plaintextkeyblob.cpp:63

```

DWORD* pdwDataLen
BYTE* pbData
DWORD dwFlags = 0
DWORD dwBlobType = PLAINTEXTKEYBLOB
HCRYPTKEY hExpKey = NULL
edx: _iob+20
HCRYPTKEY hKey = edx: __enc$textbss$end+7A8
CryptExportKey

```

Figure 6.20: x32dbg with breakpoint

Further, we will look at the instructions one by one until the last parameter to push. Before the call instruction, our stack state will be as shown in [Figure 6.21](#):

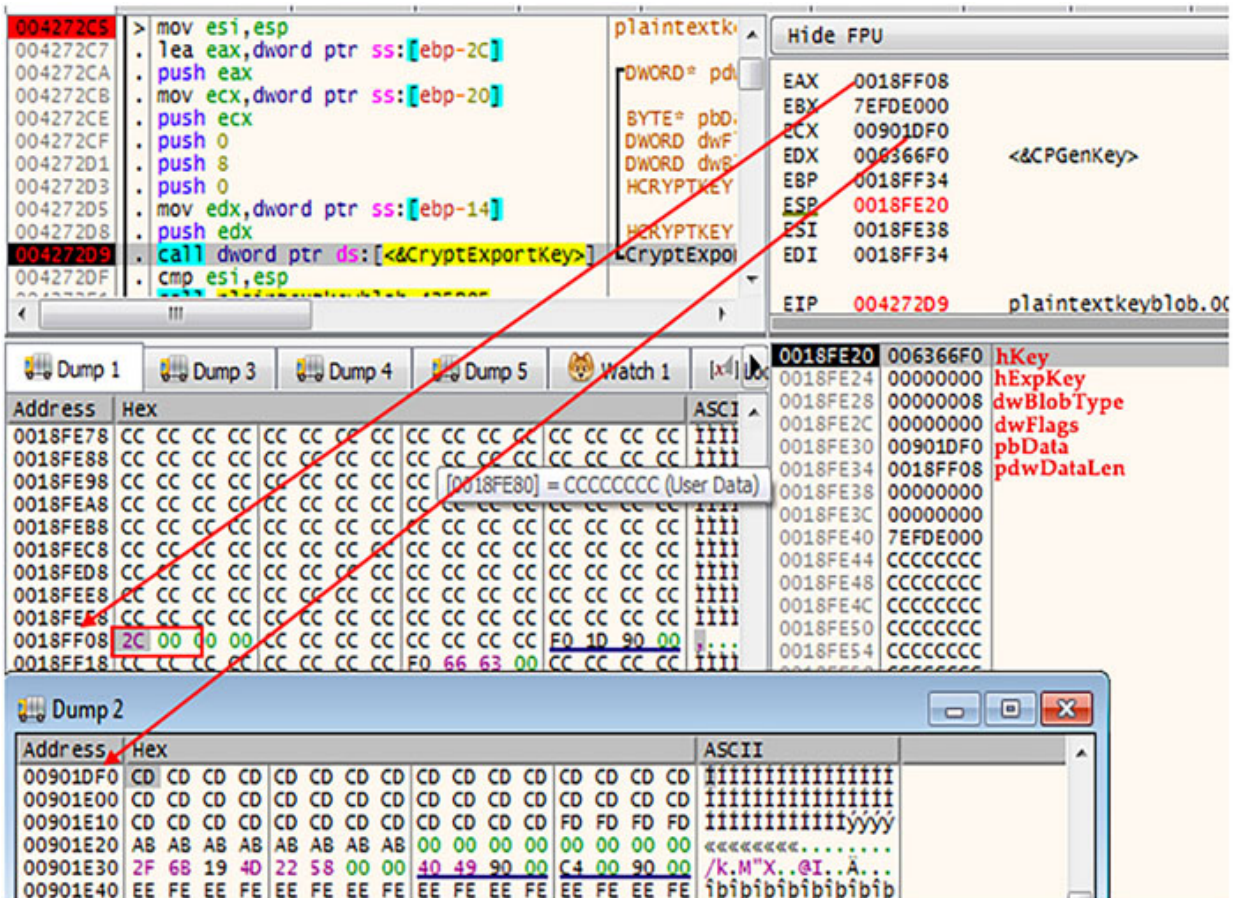


Figure 6.21: Stack state

EAX holds the memory location to the length of key blob. As we can see in [Figure 6.21](#), 0x0018FF08 holds the length of key blob as 0x2C = 44 in decimal. So, the length of plain text key blob returned is 44 bytes. This is the size of whole plain text key blob, which includes the key blob header length.

The length of blob header is 12 bytes, and the remaining $44 - 12 = 32$ bytes is the plain text key data. 32 bytes when multiplied by 8 gives 256 bits. This key length is the same as the one we used in our C/C++ code encryption algorithm CALG_AES_256.

If we use CALG_AES_128 as the encryption algorithm, then the length of the key returned using the **CryptExportKey** function will be 28 bytes (12

bytes blob header + 16 byte key data). On stepping over the call instruction, we will see the function return value in EAX as 0x01. This indicates the successful execution of the **CryptExportKey** function, with 1 as the Boolean return value.

The handle to keys will be stored at 0x00901DF0, which is the placeholder to plain text key blob. As we can see in [Figure 6.21](#), before the call instruction 0x00901DF0, memory location is JUNK. On stepping over the function call instruction, 0x00901DF0 will hold the key blob data, as shown in [Figure 6.22](#):

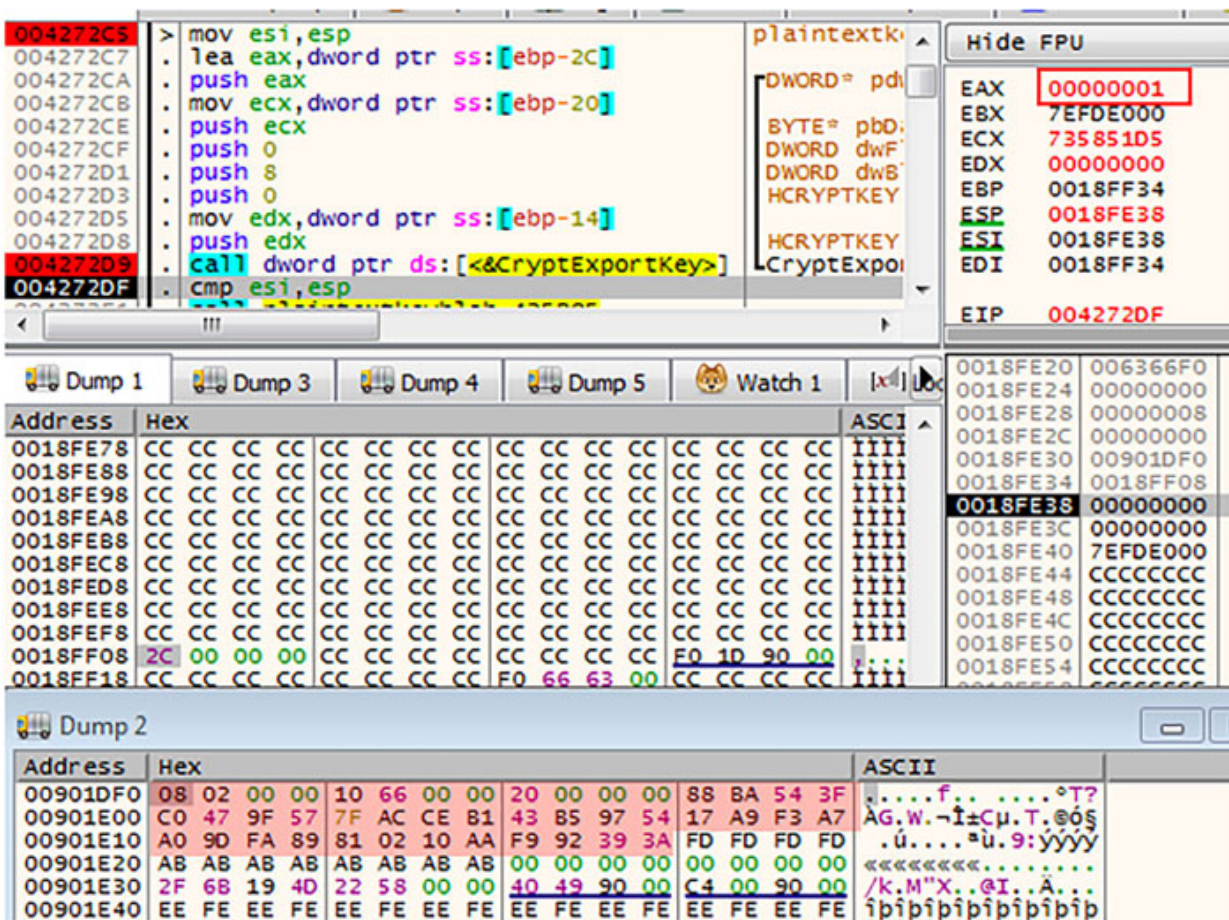


Figure 6.22: Pointer to Keys

As highlighted in the preceding image, the first 12 Bytes of the plain text key blob is the blob header, and the remaining 32 bytes are the key blob data.

[CryptExportKey SimpleBlob Example](#)

To better understand the concept of *CryptExportKey*, we will take up the *CryptExportKey* example with Simple blob, as shown in [Figures 6.23 to 6.26](#):

```
01. // SIMPLEBLOB.cpp : Defines the entry point for the console application.
02. //
03.
04. #include "stdafx.h"
05. #include <windows.h>
06. #include <wincrypt.h>
07.
08. // Link with the Advapi32.lib file.
09. #pragma comment (lib, "advapi32")
10.
11. int _tmain(int argc, _TCHAR* argv[])
12. {
13.     //-----
14.     // Declare and initialize local variables.
15.     HCRYPTPROV hCryptProv = NULL;
16.     HCRYPTKEY hKey = NULL;
17.     HCRYPTKEY hXchgKey = NULL;
18.
19.     PBYTE pbKeyBlob = NULL;
20.     DWORD dwKeyBlobLen;
21.
22.     // Get the handle to the default provider.
23.     if(CryptAcquireContext(&hCryptProv, NULL, MS_ENH_RSA_AES_PROV, PROV_RSA_AES, 0))
24.     {
25.         printf("A cryptographic provider has been acquired. \n");
26.     }
27.     else
28.     {
29.         printf("Error %x during CryptAcquireContext!\n", GetLastError());
30.         goto Exit;
31.     }
32.
33.     // Create a random session key.
34.     if(CryptGenKey(hCryptProv, CALG_AES_256, CRYPT_EXPORTABLE, &hKey))
35.     {
36.         printf("A session key has been created. \n");
37.     }
```

Figure 6.23: SIMPLEBLOB Code

```

37.     }
38.     else
39.     {
40.         printf("Error %x during CryptGenKey. \n", GetLastError());
41.         goto Exit;
42.     }
43.
44.     // Get the handle to the exchange public key.
45.     if(CryptGetUserKey(hCryptProv, AT_KEYEXCHANGE, &hXchgKey))
46.     {
47.         printf("The user public key has been retrieved. \n");
48.     }
49.     else
50.     {
51.         if(NTE_NO_KEY == GetLastError())
52.         {
53.             // No exchange key exists. Try to create one.
54.             if(!CryptGenKey(hCryptProv, AT_KEYEXCHANGE, CRYPT_EXPORTABLE, &hXchgKey))
55.             {
56.                 printf("Error %x Could not create a user public key.\n", GetLastError());
57.                 goto Exit;
58.             }
59.         }
60.         else
61.         {
62.             printf("Error %x User public key is not available and may not exist.\n", GetLastError());
63.             goto Exit;
64.         }
65.     }
66.
67.     // Determine size of the key BLOB, and allocate memory.
68.     if(CryptExportKey(hKey, hXchgKey, SIMPLEBLOB, 0, NULL, &dwKeyBlobLen))
69.     {
70.         printf("The key BLOB is %d bytes long. \n", dwKeyBlobLen);
71.     }
72.     else

```

Figure 6.24: SIMPLEBLOB Code


```

73.     {
74.         printf("Error %x computing BLOB length! \n", GetLastError());
75.         goto Exit;
76.     }
77.
78.     if(pbKeyBlob = (BYTE *)malloc(dwKeyBlobLen))
79.     {
80.         printf("Memory is allocated for the key BLOB. \n");
81.     }
82.     else
83.     {
84.         printf("Out of memory. \n");
85.         goto Exit;
86.     }
87.
88.     // Encrypt and export the session key into a simple key BLOB.
89.     if(CryptExportKey(hKey, hXchgKey, SIMPLEBLOB, 0, pbKeyBlob, &dwKeyBlobLen))
90.     {
91.         printf("The key has been exported. \n");
92.     }
93.     else
94.     {
95.         printf("Error %x during CryptExportKey!\n", GetLastError());
96.         goto Exit;
97.     }
98.
99.     // Release the key exchange key handle.
100.    if(hXchgKey)
101.    {
102.        if(!(CryptDestroyKey(hXchgKey)))
103.        {
104.            printf("Error %x during CryptDestroyKey.\n", GetLastError());
105.            goto Exit;
106.        }
107.

```

Figure 6.25: SIMPLEBLOB Code

```

108.     hXchgKey = 0;
109. }
110.
111. // Free memory.
112. free(pbKeyBlob);
113.
114. Exit:
115. // Release the session key.
116. if(hKey)
117. {
118.     if(!(CryptDestroyKey(hKey)))
119.     {
120.         printf("Error %x during CryptDestroyKey!\n", GetLastError());
121.     }
122. }
123.
124. // Release the provider handle.
125. if(hCryptProv)
126. {
127.     if(!(CryptReleaseContext(hCryptProv, 0)))
128.     {
129.         printf("Error %x during CryptReleaseContext!\n", GetLastError());
130.     }
131. }
132.
133. return 0;

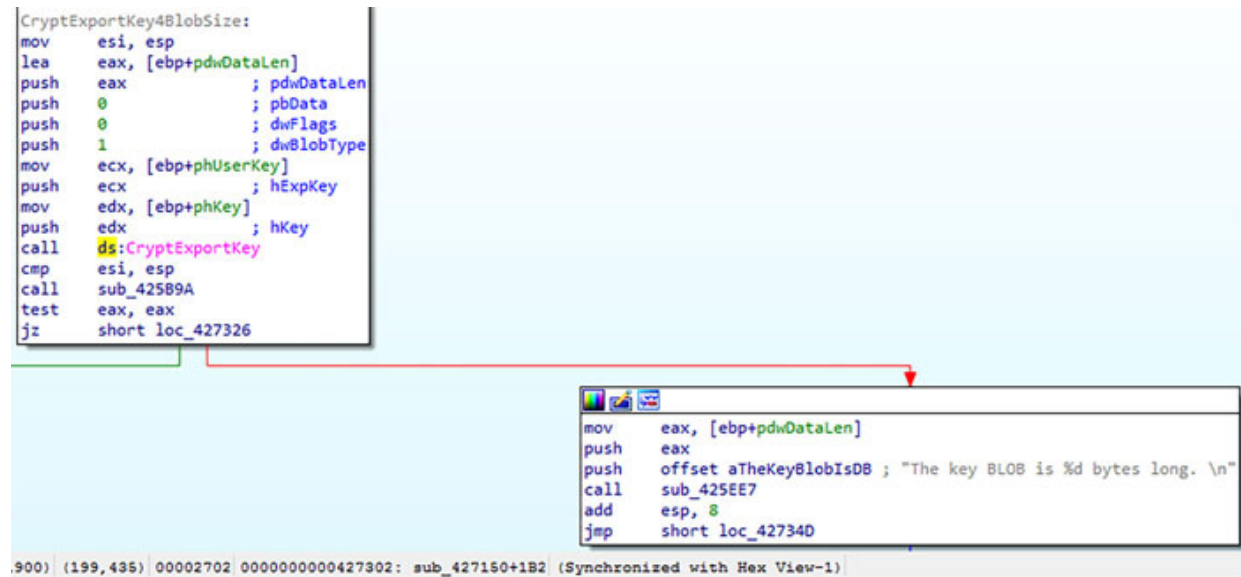
```

Figure 6.26: SIMPLEBLOB Code

In the preceding code, we are opening a handle to Microsoft Enhanced Cryptographic Provider CSP and then calling the **CryptGenKey** function to generate a random session key. Then, to export keys in Simple blob, we are calling the **CryptExportKey** function. The first call to the **CryptExportKey** function determines the size of the key BLOB and allocates memory. The second call exports the session key into a Simple blob and this time, the session key is encrypted with public key. Key data within key blob is encrypted using public key.

We compiled this code to generate an executable, which will be reverse engineered using IDA freeware and x32dbg. We are compiling this code on Windows 7 32-bit environment using Microsoft Visual Studio. The resultant binary will have Address space layout randomization (ASLR) enabled. So, to disable ASLR, we will open compiled binary in CFF Explorer software and go to *Optional Header | DLLCharacteristics | Click here | Uncheck DLL can move*.

We will move on to **CryptExportKey** Code using IDA freeware, as can be seen in [Figure 6.27](#):



The screenshot shows the IDA Pro interface. The main window displays assembly code for a function labeled `CryptExportKey4BlobSize`. The code is as follows:

```
CryptExportKey4BlobSize:
mov     esi, esp
lea     eax, [ebp+pdwDataLen]
push    eax                ; pdwDataLen
push    0                  ; pbData
push    0                  ; dwFlags
push    1                  ; dwBlobType
mov     ecx, [ebp+phUserKey]
push    ecx                ; hExpKey
mov     edx, [ebp+phKey]
push    edx                ; hKey
call    ds:CryptExportKey
cmp     esi, esp
call    sub_42589A
test    eax, eax
jz      short loc_427326
```

A red arrow points from the `call ds:CryptExportKey` instruction to a secondary window showing the code for `sub_427340`:

```
mov     eax, [ebp+pdwDataLen]
push    eax
push    offset aTheKeyBlobIsDB ; "The key BLOB is %d bytes long. \n"
call    sub_425EE7
add     esp, 8
jmp     short loc_42734D
```

The status bar at the bottom indicates the current address is `900: (199,435) 00002702 00000000000427302: sub_427150+1B2 (Synchronized with Hex View-1)`.

Figure 6.27: IDA

The first **CryptExportKey** assembly listing is what determines the size of the key BLOB. In the assembly listing, all the parameters to **CryptExportKey** are pushed on to the stack one by one in order from right to left. On **CryptExportKey** function return, as per our C/C++ code at line 68 data length of key blob will be stored in the **pdwDataLen**. `[EBP+pdwDataLen]` holds the memory location that will have the data length of key blob. Memory location of `[EBP+pdwDataLen]` is moved to EAX register, which is pushed on to the stack first.

Moving further down, the last parameter to push before the call to the **CryptExportKey** function is **hkey**, which is the handle to session key. Next, assembly listing shows the second call to the **CryptExportKey** function. Refer to [Figure 6.28](#):

```

CryptExportKey4SIMPLEBLOB:
mov     esi, esp
lea     eax, [ebp+pdwDataLen]
push    eax                ; pdwDataLen
mov     ecx, [ebp+pbData]
push    ecx                ; pbData
push    0                  ; dwFlags
push    1                  ; dwBlobType
mov     edx, [ebp+phUserKey]
push    edx                ; hExpKey
mov     eax, [ebp+phKey]
push    eax                ; hKey
call    ds:CryptExportKey
cmp     esi, esp
call    sub_425B9A
test    eax, eax
jz      short loc_42738E

push    offset aTheKeyHasBeenE ; "The key has been exported. \n"
call    sub_425EE7
add     esp, 4
jmp     short loc_4273DF
  
```

(274,118) 00002788 00000000000427388: sub_427150:CryptExportKey4SIMPLEBLOB (Synchronized with Hex View-1)

Figure 6.28: IDA

In assembly listing of the **CryptExportKey** function, all the parameters to **CryptExportKey** are pushed on to the stack one by one in order from right to left. The main parameter that is pushed on to the stack is **dwBlobType**, with the value of 1. This value represents that the key will be exported in Simple blob. The second parameter of our interest is *pbData*, which is the pointer to the buffer that holds the encrypted key blob data.

To understand the structure of plain text blob, we will open the binary in x32dbg disassembler by putting a breakpoint at *MOV ESI, ESP* instruction at 0x00427388, as shown in [Figure 6.29](#). This is obtained from IDA by hovering the cursor over the instruction, as shown in [Figure 6.28](#):

Address	Disassembly	Comment
00427388	> mov esi, esp	simpleblob.cpp:89
0042738A	. lea eax, dword ptr ss:[ebp-38]	DWORD= pdwDataLen
0042738D	. push eax	BYTE= pbData
0042738E	. mov ecx, dword ptr ss:[ebp-2C]	DWORD dwFlags = 0
00427391	. push ecx	DWORD dwBlobType = SIMPLEBLOB
00427392	. push 0	edx:_iob+20
00427394	. push 1	HCRYPTKEY hExpKey = edx: __enc\$textbss\$end+6F5
00427396	. mov edx, dword ptr ss:[ebp-20]	HCRYPTKEY hKey
00427399	. push edx	CryptExportKey
0042739A	. mov eax, dword ptr ss:[ebp-14]	
0042739D	. push eax	
0042739E	. call dword ptr ds:[<&CryptExportKey>]	
004273A4	. cmp esi, esp	

Figure 6.29: x32dbg with breakpoint

Once the breakpoint is hit, we can see that all the parameters are highlighted in x32dbg with the help of the xAnalyzer plugin. Furthermore, we will look at the instructions one by one till the last parameter to push. Before the call instruction, our stack state will be as shown in [Figure 6.30](#):

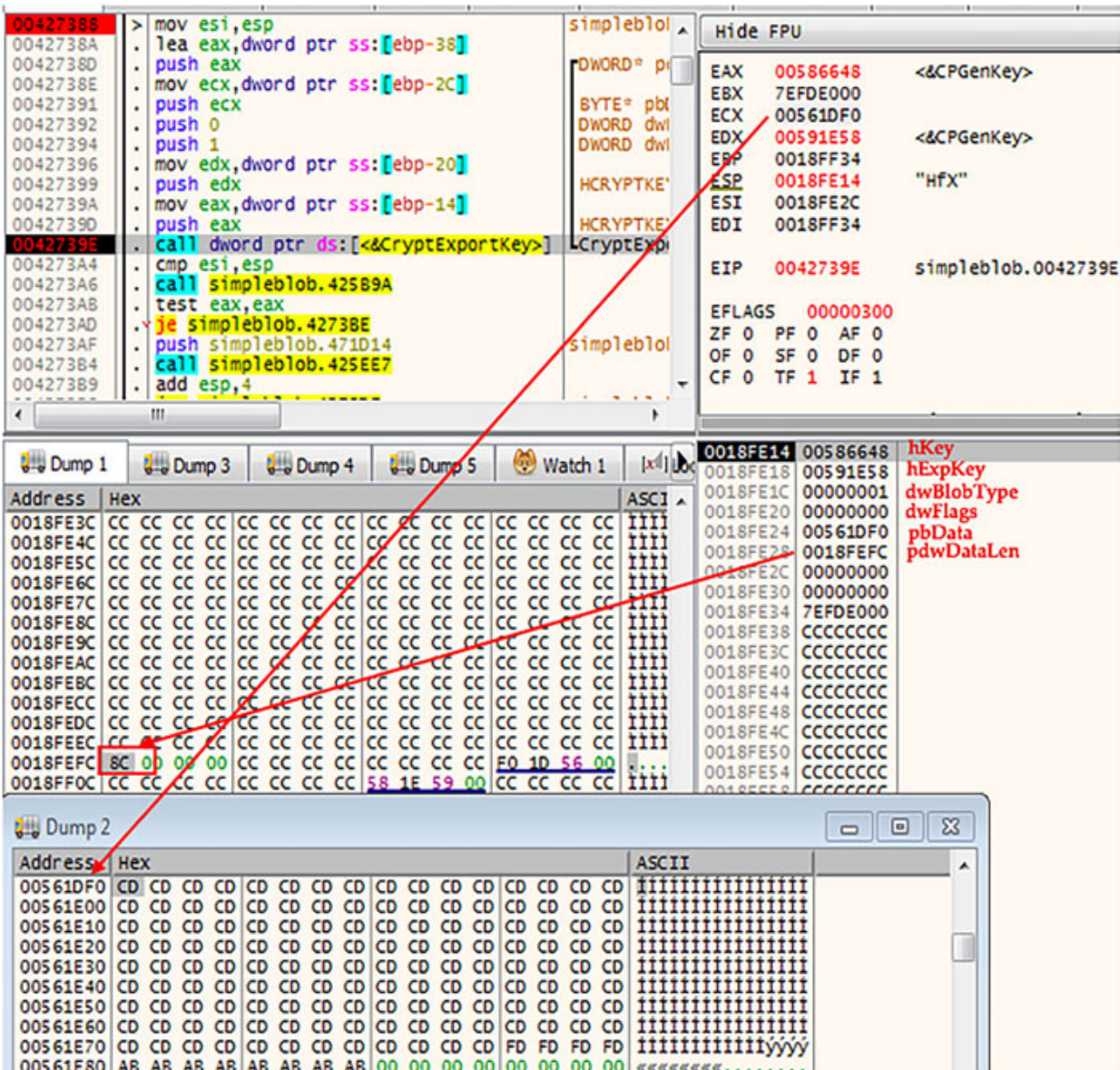


Figure 6.30: Stack state

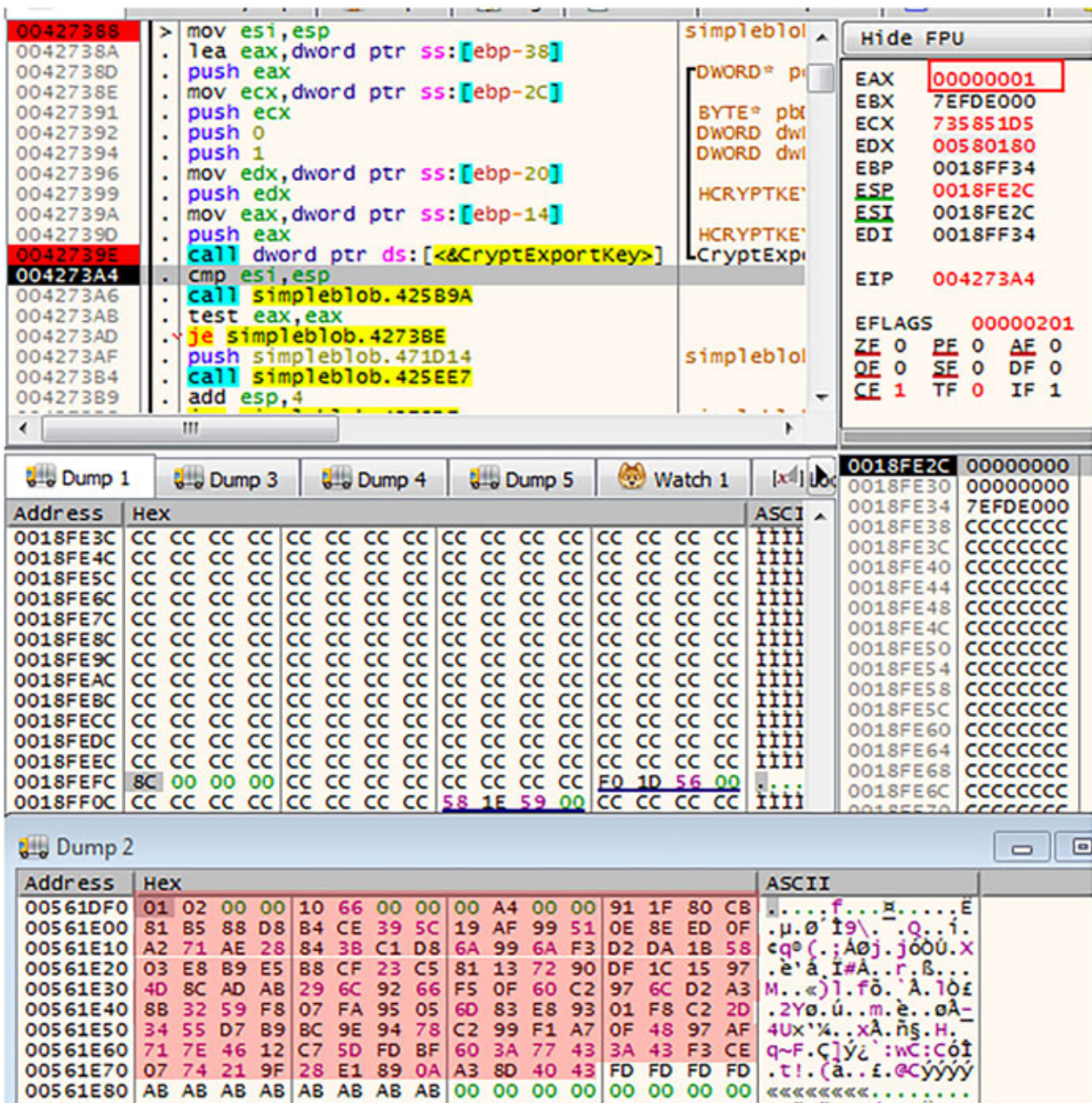
The first parameter that is pushed on to the stack is the memory location that holds the length of key blob. As we can see in [Figure 6.30](#), 0x0018FEFC holds the length of key blob as 0x8C = 140 in decimal. So, the length of Simple key blob returned is 140 bytes. This is the size of whole encrypted Simple key blob, which includes the key blob header length.

The length of blob header is 12 bytes in size, and the remaining $140 - 12 = 128$ bytes is the Simple blob key data. 128 bytes when multiplied by 8 gives 1024 bits. This key length is the same as that of public key.

As we discussed in SIMPLEBLOB, the encrypted session key data is in the form of a PKCS #1, type 2 encryption block. This data is always the same size as the public key's modulus. On stepping over the call instruction we will see the function return value in EAX as 0x01. This indicates the successful execution of the **CryptExportKey** function, with 1 as the Boolean return value.

The handle to keys will be stored at 0x00561DF0, which is the placeholder to Simple key blob. As we can see in [Figure 6.30](#), before the call instruction 0x00561DF0, memory location is JUNK.

On stepping over the function call instruction, 0x00561DF0 will hold the key blob data, as shown in [Figure 6.31](#):



As highlighted in the preceding image, the first 12 Bytes of the plain text key blob is the blob header, and the remaining 128 bytes are the key blob data.

CryptImportKey

There is also a provision to import keys from key BLOB to CSP. The **CryptImportKey** function is used to import keys from key BLOB to CSP.

The syntax of the function is as follows:

```
BOOL CryptImportKey(  
    HCRYPTPROV hProv,  
    const BYTE *pbData,  
    DWORD      dwDataLen,  
    HCRYPTKEY   hPubKey,  
    DWORD      dwFlags,  
    HCRYPTKEY   *phKey  
);
```

hProv

This is the first parameter is the handle to CSP created with the **CryptAcquireContext** function.

pbData

This is the buffer containing the key string or key BLOB. It is the same buffer we discussed in the **CryptExportKey** function, which holds the key BLOB data.

dwDataLen

It is the size of key string or key BLOB in bytes.

hPubKey

This is the handle to the key that is used to decrypt the key BLOB stored in **pbData**. So, the value of this parameter depends on the CSP type and the key BLOB being imported:

- If the key BLOB being imported is PAINTEXTBLOB, PUBLICKEYBLOB or OPAQUEKEYBLOB, which is not encrypted, then this parameter is zero.
- If the key BLOB is encrypted like SIMPLEKEYBLOB, where session key is encrypted with the recipient public key, then this parameter is the handle to the recipient's public/private key pair, which will be used to decrypt the session key.

- If the key BLOB to be imported is PRIVATEKEYBLOB, then this parameter is the handle to session key.

dwFlags

This parameter is used to specify options used in importing keys. This is only used when public/private key pair is imported into CSP, which is in the form of PRIVATEKEYBLOB. It is set to CRYPT_EXPORTABLE when the imported key needs to be re-exported from CSP.

phKey

This parameter will have the handle or pointer to the imported key.

On successful execution, the function returns the value as non zero. Otherwise, it will return zero.

CryptImportKey Example

To understand the concept of **CryptImportKey**, we will take up an example where we will take a buffer that will hold PLAINTEXTKEY. Using the **CryptImportKey** function, we will import plain text key into the CSP. Once the key is imported, we will use the **CryptExportKey** function to print the imported key (PLAINTEXTKEY) on the console for verification. Refer to [Figures 6.32](#) and [6.33](#):

```

01. // CryptImportKey.cpp : Defines the entry point for the console application.
02.
03. #include "stdafx.h"
04. #include <windows.h>
05. #include <wincrypt.h>
06.
07. // Link with the Advapi32.Lib file.
08. #pragma comment (lib, "advapi32")
09.
10. //PlainTextBlob is 44 bytes where 12 bytes are blob header and reaming 32 bytes key length.
11. BYTE PlainTextBlob[] = {
12.     0x08,0x02,0x00,0x00,0x10,0x66,0x00,0x00, // BLOB header, 8 bytes
13.     0x20,0x00,0x00,0x00, // key length, in bytes (0x20 = 32 bytes = 256 bit key length)
14.     0x65,0x4F,0xC2,0xC4,0x1B,0x95,0x1B,0x8C,0x70, // CALG_AES_256 key
15.     0xC4,0xB7,0xCF,0x37,0x90,0xAD,0xFF,0x00,0x2D,
16.     0xC6,0x3A,0xED,0x4C,0xD4,0x7B,0x4F,0x9A,0xE4,
17.     0xED,0x14,0x17,0x0D,0xB9,0xFD,0xFD,0xFD,0xFD
18. };
19.
20. int _tmain(int argc, _TCHAR* argv[])
21. {
22.     // Declare and initialize local variables.
23.     HCRYPTPROV hCryptProv = NULL;
24.     HCRYPTKEY hKey = NULL;
25.
26.     PBYTE pbKeyBlob = NULL;
27.     DWORD dwKeyBlobLen;
28.
29.     // Get the handle to the default provider.
30.     if(CryptAcquireContext(&hCryptProv, NULL, MS_ENH_RSA_AES_PROV, PROV_RSA_AES, 0))
31.     {
32.         printf("A cryptographic provider has been acquired. \n");
33.     }
34.     else
35.     {
36.         printf("Error %x during CryptAcquireContext!\n", GetLastError());
37.         goto Exit;
38.     }
39.
40.     // Import PLAINTEXTBLOB into CSP.
41.     if (!CryptImportKey(hCryptProv, PlainTextBlob, sizeof(PlainTextBlob), 0, CRYPT_EXPORTABLE, &hKey ))
42.     {
43.         printf("Error 0x%08x in importing the Des key \n",
44.             GetLastError());
45.     }
46.
47.     // Determine size of the key BLOB, and allocate memory.
48.     if(CryptExportKey(hKey, NULL, PLAINTEXTKEYBLOB, 0, NULL, &dwKeyBlobLen))
49.     {
50.         printf("The key BLOB is %d bytes long. \n", dwKeyBlobLen);
51.     }
52.     else
53.     {
54.         printf("Error %x computing BLOB length! \n", GetLastError());
55.         goto Exit;
56.     }
57.
58.     if(pbKeyBlob = (BYTE *)malloc(dwKeyBlobLen))
59.     {
60.         printf("Memory is allocated for the key BLOB. \n");

```

Figure 6.32: CryptImportKey Code

```

61.     }
62.     else
63.     {
64.         printf("Out of memory. \n");
65.         goto Exit;
66.     }
67.
68.     // Encrypt and export the session key into a simple key BLOB.
69.     if(CryptExportKey(hKey, NULL, PLAINTEXTKEYBLOB, 0, pbKeyBlob, &dwKeyBlobLen))
70.     {
71.         printf("The key has been exported. \n");
72.     }
73.     else
74.     {
75.         printf("Error %x during CryptExportKey!\n", GetLastError());
76.         goto Exit;
77.     }
78.
79.     //Printing Key BLOB for verification
80.     DWORD Num = 0;
81.     printf("For verification printing Key BLOB: \n");
82.     for(Num=0; Num < dwKeyBlobLen;)
83.     {
84.         printf("%02x ", pbKeyBlob[Num]);
85.         Num ++;
86.     }
87.
88.     // Free memory.
89.     free(pbKeyBlob);
90.
91. Exit:
92.     // Release the session key.
93.     if(hKey)
94.     {
95.         if(!(CryptDestroyKey(hKey)))
96.         {
97.             printf("Error %x during CryptDestroyKey!\n", GetLastError());
98.         }
99.     }
100.
101.     // Release the provider handle.
102.     if(hCryptProv)
103.     {
104.         if(!(CryptReleaseContext(hCryptProv, 0)))
105.         {
106.             printf("Error %x during CryptReleaseContext!\n", GetLastError());
107.         }
108.     }
109.
110.     return 0;
111. }

```

Figure 6.33: *CryptImportKey Code*

On execution of this code, we will see Plain text blob on the console, as shown in [Figure 6.34](#):

```
C:\JitenderN\Book\CryptImportKey\Debug>CryptImportKey.exe
A cryptographic provider has been acquired.
The key BLOB is 44 bytes long.
Memory is allocated for the key BLOB.
The key has been exported.
For verification printing Key BLOB:
08 02 00 00 10 66 00 00 20 00 00 00 65 4f c2 c4 1b 95 1b 8c 70 c4 b7 cf 37 90 ad ff 00 2d c6 3a ed 4c d4
7b 4f 9a e4 ed 14 17 0d b9
C:\JitenderN\Book\CryptImportKey\Debug>_
```

Figure 6.34: CryptImportKey Code Output.png

The Blob shown in [Figure 6.34](#) is the same PlainTextBlob buffer we imported into CSP.

Conclusion

In this chapter, we talked about the ransomware internals, which covers ransomware from the malware writers' point of view. We also learned about cryptographic API and how Cryptographic API is used to write ransomware. With the help of sample examples, we understood the usage of different crypto functions. Further on, we used x32dbg and IDA freeware for reverse engineering and understood ransomware from malware developers' point of view.

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

<https://discord.bpbonline.com>



CHAPTER 7

Portable Executable Insides

Introduction

The Portable Executable, also known as PE format, is the native file format for Windows 32-bit and Windows 64-bit version. It is the file format of files like EXE, DLLs, OCX Controls, COM files, **Control Panel Applet** (.CPL files) and .NET executable. Throughout this chapter, we will be using a custom binary created by compiling basic Win32 example to understand the structure of portable executable files.

Structure

In this chapter, we will discuss the following topics:

- Creating custom binary
- Structure of Portable Executable (PE)
- DOS header
- DOS stub
- Portable Executable (PE) header
- Section tables

Objectives

The objective of this chapter is to help you understand the binary file internals. We will start by creating a binary file, which is also referred to as **Portable Executable (PE)** or executable file. We will then cover the internal details of the PE file, which includes a discussion on the DOS Header and the DOS stub. We will then discuss the PE header, which will cover File header, Optional header and Data directories. In order to understand ransomware, we will have to understand the DLL imported or

exported while writing that executable. This will involve a deeper understanding of the binary. To understand PE files in further detail, we will talk about the different sections of a PE file.

Creating a custom binary

In order to understand the PE structure, we will create an empty Win32 project using Microsoft Visual Studio 2010 Professional. To create an empty project, follow the given steps:

1. Go to **File** menu | **New** project.
2. Select **Win32 project** and enter the name of project. For our example, we have entered **HelloWorld**.
3. Select the **Location** of the project as required.
4. Click on **OK** to proceed to the **Win32 Application Wizard**.
5. Select **Next** to continue.
6. Select **Application type** as **Windows application** and **Additional options** as **Empty project**.
7. Click on **Finish** to close the wizard.
8. Under **Source Files**, add new item with extension of the **.cpp** file with the name **HelloWorld**.
9. After adding the file, add the following code to compile and generate our custom binary for PE analysis:

Figure 7.1: *HelloWorld program*

Before compilation, change **Character Set** to the **Not Set** option. Go to project **Properties** | **Configuration Properties** | **General** | **Character Set** | **Not Set**).

During the compilation, you might get an error:

```
LINK: fatal error LNK1123: failure during conversion to COFF:
file invalid or corrupt
```

The `cvtres.exe` file in the bin directory of Visual studio (`c:\Program Files\Microsoft Visual Studio 10.0\VC\bin`) is lower in version than that in the .NET directory (`c:\Windows\Microsoft.NET\Framework\v4.0.30319`). To solve this problem, replace the file in the bin directory of Visual studio with that in the .NET directory.

The output of the mentioned custom binary is as shown in [Figure 7.2](#):

Figure 7.2: *HelloWorld program output*

Structure of Portable Executable

In the previous section, we got the custom binary `HelloWorld.exe`; as this is a **Portable Executable (PE)** file, going forward in this chapter, we will study the structure of portable executable.

Before diving deeper into the PE file structure, let us talk about the origin and importance of PE structure. Portable executable format was introduced by Microsoft in July 1993 for Windows NT v3.1. It is used by Windows executable and **Dynamic Link Libraries (DLL)**. We all know about executables in windows; for example, for running calculator in Windows, we have an executable for calculator executable in the `system32` folder. But when we talk about DLL, we must understand that they are shared libraries used by multiple processes. Malware also commonly use these DLL. With the understanding of the PE format, we will get valuable information about an exe file or executable, and the structure of the operating system and environment without even executing it.

The structure of a PE file is the combination of different headers, as shown in [Figure 7.3](#):

Figure 7.3: *PE Structure*

A PE file should have a minimum of two sections: a code section and a data section. We will be discussing these sections later on. The most common sections present in an executable are as follows:

- **.text section:** Contains actual binary executable code
- **.bss section:** Represents uninitialized data
- **.rdata section:** Contains read-only data, such as constants and string
- **.rsrc section:** Resource section, where resource information is stored
- **.edata section:** Export data section that contains information about exported functions
- **.idata section:** Import data section that contains information about imported functions, along with Import directory and Import address table
- **.debug section:** This section was initially used for placing debug information. A PE file also supports separate debug files with **.dbg** extension.

The names of these sections are not used by operating systems. They are present only for ease of understanding.

When we talk about a PE file, it is assumed to be either on the disk or loaded in memory or RAM. In [Figure 7.4](#), we can see how a PE file looks when lying on the disk and in contrast, how it looks when loaded in memory or RAM. A PE file contains the data in the linear stream, so a PE file on the disk is exactly the same when loaded in memory.

Figure 7.4: *PE file on disk and in memory*

When sections are loaded in the memory, they are aligned to fit in the 4Kb of memory pages. As the default page size in Windows is 4096 bytes (4Kb or 0x1000), it means that a new section starts on the new page. However, when the PE file is on the disk, sections are aligned to the hard disk sector size, which is 512 bytes (0x200 in hex). This alignment of sections on the disk and memory is referred to as *section alignment* and *file alignment*. Thus, *section alignment* is how sections are aligned in the memory, and

File alignment is how sections are aligned on the disk. We will talk about this in the upcoming sections.

Now, we will start discussing the different headers and sections in a standard PE file and check in our custom created binary or executable above.

DOS header

All PE files start with the DOS header, and the size of DOS header is 64 bytes. The structure of the DOS header is defined in the `winNT.h` file, as shown in [Figure 7.5](#):

Figure 7.5: *Structure of DOS header*

4 and 10 in the red brackets indicate 4 WORDs and 10 WORDs, respectively. The total size of the DOS header is calculated as follows:

Total number of WORDS = 30 = 60 bytes (1 WORD = 2 bytes)

Total number of DWORDS = 1 = 4 bytes (1 DWORD = 4 bytes)

Total size of DOS header = 64 bytes

Two members in the DOS structure are of interest to us: `e_magic` and `e_lfanew`. Let's look at them in detail:

- `e_magic` is called the magic number, and it signifies whether the file is a valid PE file. If the file is a valid PE file, then the value of this field in hexadecimal is 4D 5A (MZ, which is the initials of architect of MS-DOS, Mark Zbikowski).
- `e_lfanew` is the last field of DOS header and specifies the offset of PE header, relative to the beginning of PE file. The Windows loader uses this value to skip the DOS header and goes directly to the PE header.

DOS Stub

After the DOS header, is the DOS stub that is executed when the binary file is executed in MS DOS. When the program is not compatible with Windows, the stub prints “*This program cannot be run in DOS mode*”. Thus, if we run a Win32 program in an environment that does not support Win32 environment, we will get this message.

While we build an application or binary, we can mention the /STUB option. This will attach the MS DOS program to a Win32 program instead of the default stub program, and this job is performed by a linker. A Linker attaches the default stub program with our binary or executable.

Let us now look at the binary file opened in a Hex Editor. When we are opening this file in Hex Editor, we are analyzing PE on disk. Refer to [Figure 7.6](#):

Figure 7.6: *Headers in HelloWorld*

The DOS header occupied 64 bytes of the HelloWorld PE file, and the last DWORD of DOS header is E8 00 00 00. When reversed, this gives us 00 00 00 E8 (because of little endian format). This is the memory offset from the start of a PE file, where PE header starts with the signature 50 45 00 00 (which is PE in ASCII, followed by two terminating 00). This value is also defined as IMAGE_NT_SIGNATURE in the WinNT.h file, as shown in [Figure 7.7](#):

Figure 7.7: *PE signature*

If we see NE or LE signature instead of PE in the PE header, then it refers to the executable for OS2 operating system.

We can also see something interesting in [Figure 7.6](#), marked in the blue box. We refer to this as a *Rich* signature. *Rich*, as you see, is sandwiched

between encrypted texts. This is the undocumented feature of Microsoft linker, wherein *Rich* signature is inserted in Microsoft linked executable to identify the machine and compiler on which the binary is built. This fingerprinting information can be used to determine whether a particular machine belongs to a malware writer.

PE header

The structure of the PE header is defined in the `winNT.h` file, as shown in [Figure 7.8](#), as `IMAGE_NT_HEADERS`:

Figure 7.8: *PE header*

This structure has 3 members and the first one is the signature of PE header. It is the same as we discussed previously; the PE header starts with signature 50 45 00 00 (which is PE in ASCII followed by two terminating 00). The second member is the File header and the other one is the Optional header, which we will discuss in the succeeding sections.

File header

File Header structure is 20 bytes in size, and it informs us about the physical layout (when file is on the disk) and the properties of a file. It is defined in the `winNT.h` file, as shown in [Figure 7.9](#):

Figure 7.9: *File header*

Machine here defines the CPU the file is intended for. In our case, the value is 0x014C, which defines that the file is for Intel 386 platform.

NumberOfSections defines the number of sections in the PE file. For ease of understanding, we will open our file in PE view and can see that it has a value 0x0007, which means seven sections, as shown in [Figure 7.10](#):

Figure 7.10: *Number of sections*

The next field of interest is **Characteristics**, which indicates whether the file is DLL or an executable. Let us now move on to the next member of PE header, which is optional header.

Optional header

This structure lets us know about the logical layout (when the file is loaded in memory) of a PE file, and its size is mentioned in the File header with the field named **SizeOfOptionalHeader**. It forms the next 224 bytes in the PE file, as the value of **SizeOfOptionalHeader** in File header is 0x00E0 (224 bytes in decimal). The structure of Optional header in the **winNT.h** file is shown in [Figure 7.11](#):

Figure 7.11: *Optional header structure*

This header starts with the **Magic** field, which specifies whether the PE file is 32-bit or 64-bit. The next field of interest is **AddressOfEntryPoint**, which is the **Relative Virtual Address (RVA)** of the instruction to be executed when PE loader loads the file in memory.

Note: We came across the terms RVA, which is defined as relative virtual address, and VA, which is Virtual Address. Virtual address is simply the memory location of a field in a file when loaded in the memory, or we can call it the virtual address space. There are many fields in the PE file, where RVA is referred. To understand the concept of RVA, let's take an example where an EXE or DLL file is loaded in memory with the starting address of 0x00400000. This starting address is the Image base address in virtual address space. RVA is the offset of the field, relative to where the file is loaded in memory.

Virtual Address = Image Base Address + Relative Virtual Address

Sometimes, the executable files are packed to bypass the antivirus detection mechanism. Executable packed files usually have **AddressOfEntryPoint** pointing to the decompression stub, after which execution is jumped to the original executable code.

ImageBase is the address in the virtual address space, where file is mapped in the memory location. The default value for an executable is 0x00400000 and that for DLLs is 0x10000000.

In our **HelloWorld** PE file case, **AddressOfEntryPoint** is 0x00011140, which is RVA, and **ImageBase** is 0x00400000, as shown in the PEview [Figure 7.12](#). Thus, the Virtual Address of **AddressOfEntryPoint** will be $0x00400000 + 0x00011140 = 0x00411140$, which can be observed in x32dbg when the file is loaded in memory, as shown in [Figure 7.13](#):

Figure 7.12: *RVA of AddressOfEntryPoint*

Figure 7.13: *VA of AddressOfEntryPoint*

SectionAlignment

When a file is loaded in memory, each section in memory starts with the multiple of this value. The value of this field is 0x00001000, as shown in [Figure 7.14](#). This value defines the logical value when the file is in memory. Sections loaded in the memory are aligned to fit in the 4Kb of memory pages. The default page size in Windows is 4096 bytes (4Kb or 0x1000). To understand this concept, let us take an example of a file in a memory with two sections. If the first section loads at 0x00401000, with size of 100 bytes, then the second section will not start at 0x00401100; it will start from 0x00402000. The rest of the space in between will be unused.

Figure 7.14: *Alignment and other fields of Optional header*

FileAlignment

When the file is on, the disk sections are aligned to the hard disk sector size, which is 512 bytes (0x200 in hex). This is the same as the value of **FileAlignment**, as we could see in [Figure 7.14](#). Thus, when the file is on the disk, sections in the file always start with a multiple of this value. Let's take the same example, where we have a file with two sections on the disk. If the first section starts at 0x00000200 with a size of 100 bytes, then the second section will start at 0x00000400. The rest of the space in between will be unused.

SizeOfImage

This defines the total size of a PE file when it is loaded in the memory. It includes all headers and sections. It starts from the image base to the last section in memory. The size of the last section in memory is round up to the *section alignment*. We can observe the **SizeOfImage** in PEview and x32dbg; the value of **SizeOfImage** in PEview is 0x0001B000, and this size can be calculated as shown in [Figure 7.15](#):

Figure 7.15: *SizeOfImage in memory*

PE Image is loaded at 0x00400000, which is **ImageBase**, and the last section of the image is **.reloc**, which starts at 0x0041A000. Till this point in the memory, the size is $0x0041A000 - 0x00400000 = 0x0001A000$. Now add 0x00001000 (which is the size of **.reloc** section, as per the **SectionAlignment** we discussed earlier) to this size, $0x0001A000 + 0x00001000 = 0x0001B000$. This value is the same as that of **SizeOfImage** we saw in the PEview.

SizeOfHeaders

This defines the size of all *headers* and *section table* when the file is on the disk. This value is used to find the offset to the first section in the file on disk. The value of **SizeOfHeaders** in PEview is 0x00000400, and if we check the offset of the first section, which is the **.text** section that starts at 0x00000400, it is the same as the value of **SizeOfHeaders**. The offset of the **.text** section can be seen in [Figure 7.16](#):

Figure 7.16: *Offset of .text section in HelloWorld.exe*

Now, by looking at the preceding figure, one question comes to mind: why is the first section as per section table, **.textbss**, not reflecting in the PEview?

This is because **.textbss** is empty in the file on disk. It is used in the virtual memory for non-initialized global variables.

Subsystem

It defines the target subsystem for portable executable files. This defines the type of subsystem, and a portable executable uses it for user interface. The value of our file is 0x0002 as it is a GUI application. Value subsystem is defined in **winNT.h**, as shown in [Figure 7.17](#):

Figure 7.17: *Subsystem Values*

NumberOfRvaAndSizes

It defines the number of elements in the Data Directories, which we will be discussing next. The value is 0x00000010, which means that the Data Directory has 16 entries.

Data directory

To understand the concept of data directories, suppose the loader needs to find the list of imported function table or list of exported function table in a PE file (it can be executable or DLL). In such a scenario, it has to iterate through each image section to find the respective details. With the help of the data directory, the loader can quickly find the list of imported functions table, exported function tables, or any another details defined in the data directory. Data directory is an array of 16 `IMAGE_DATA_DIRECTORY` structures, each of which is 8 bytes in size. That makes a total of 128 bytes (16 x 8 bytes) of the Optional header. Each `IMAGE_DATA_DIRECTORY` structure has two members, as defined in `winNT.h`, which is shown in [Figure 7.18](#):

Figure 7.18: *Data directory structure*

The first member of the `IMAGE_DATA_DIRECTORY` structure is `VirtualAddress`, which is the **Relative Virtual Address (RVA)** of the data structure. The second member is the `Size`, which defines the size of the data structure in bytes. There are a total of 16 Directory entries, as defined in `winNT.h`. Refer to [Figure 7.19](#):

Figure 7.19: *Data directory entries*

- The first entry in the Data Directory is the `IMAGE_DIRECTORY_ENTRY_EXPORT`, which defines the RVA and size of Export Directory for the exported functions' details (if exported functions are present).
- The second entry in the Data Directory is `IMAGE_DIRECTORY_ENTRY_IMPORT`, which defines the RVA and size of Import Directory, with imported function details.
- The third Directory entry defines the RVA and size of resource directory used in the PE file. Many Directory entries draw an entire section's data, as detailed in the following calculation for Import Directory in our file.

To understand the relevance of Data Directory, we will go through the process followed to locate a particular directory (export directory is used to find the list of exported function, import directory is used to find the list of imported function, and so on). In order to determine the list of imported functions in an application, we will have to locate the Import Directory.

In order to locate the Import Directory in a PE file, we need the RVA to Import Directory and its size in the PE file. As discussed earlier, the second entry in the Data Directory is `IMAGE_DIRECTORY_ENTRY_IMPORT`, which defines the RVA to Import Directory and the size of Import Directory, with the imported function details. The position of Data Directories is always the same in a PE file, as the RVA to Import Directory is 80 bytes from the beginning of the PE header. Once we have the RVA, we can get the VA (Virtual Address = Image base Address + Relative virtual address) to determine which section the directory is in. With the section, the corresponding section header can be used to determine the offset to a particular directory. Another way to find the offset is to use the following formula:

$$\text{Raw Offset} = \text{RVAToConvert} - \text{RVAOfSection} + \text{PointertoRawSection}$$

Raw Offset = Offset to particular directory

RVAToConvert = RVA that need to converted to Offset

RVAOfSection = we can get this from corresponding section header

PointertoRawSection = we can also get this from corresponding section header

With the understanding of the data directory traceability, let us check our Import Directory in *HelloWorld* PE file. Refer to [Figure 7.20](#):

Figure 7.20: *Data directory entries in HelloWorld*

As shown in [Figure 7.20](#), the Relative Virtual Address of Import Directory is 0x00018000. Thus, to calculate the Virtual Address, we will add Image base onto it, $0x00018000 + 0x00400000 = 0x00418000$. This address, when viewed in x32dbg, describes an entire *.idata*, as shown in [Figure 7.21](#):

Figure 7.21: *Import Data directory in x32dbg*

Data Directory is the final member of `IMAGE_NT_HEADERS` or PE header. Till this point, let us recap the file in Hex Editor, as shown in [Figure 7.22](#):

Figure 7.22: *PE header in hex editor*

Section table

The Section table comes after the PE header. Remember, in File Header, we have a field called **NumberOfSections** that defines the number of sections in the file. The Section table is an array of `IMAGE_SECTION_HEADER` structures, which define the attributes of sections in a PE file. Each section is 40 bytes in size, and there is no padding between them. In our file, we have 7 sections, as indicated by **NumberOfSections** field. Thus, we have 7 duplicate `IMAGE_SECTION_HEADER` structures in the section table, as shown in [Figure 7.23](#):

Figure 7.23: *Section table*

We will discuss the important fields in the `IMAGE_SECTION_HEADER` structure defined in `winnt.h`. Refer to [Figure 7.24](#):

Figure 7.24: *IMAGE_SECTION_HEADER* structure

Name: This is 8 bytes in size, and it is just a label that can be left blank. This is not zero terminated ASCII string.

VirtualSize: This defines the actual size of the section in the memory, in bytes. In our case, the value of **VirtualSize** is 0x00010000, as shown in [Figure 7.25](#), so it is the same when looked in the x32dbg.

Figure 7.25: *Virtual size of .textbss section*

VirtualAddress: This is the RVA of the section, and the loader uses this value to map the section in the memory. So, to calculate the address of section in the memory, we have to add this RVA with Image base address, $0x00400000 + 0x00001000 = 0x00401000$. We can see, in the preceding figure, that the **.textbss** section is mapped on the 0x00401000 address.

SizeOfRawData: It defines the size of section data when the file is on disk. It is rounded to the multiple of file alignment, which is 512 bytes (0x200 in hex). As discussed earlier, **textbss** is empty when file is on disk. It is used for non-initialized global variables when the file is loaded in memory. So, the value of **SizeOfRawData** is 0x00000000, as shown in [Figure 7.25](#).

PointerToRawData: By name, this means that it is the pointer to the raw section data; raw means when the file is on the disk. This is useful in finding the section data in the file on disk, so it is the offset of section data from the file beginning.

To understand this concept of sections more clearly, we will take the **.data** section and view it when the file is on disk and then when the file is loaded in the memory. Refer to [Figure 7.26](#):

Figure 7.26: *Value of PointerToRawData of .data section*

The **PointerToRawData** field of the **.data** section header is 0x00005800. Thus, it is the offset of the **.data** section data from the file beginning. [Figure 7.27](#) shows the **.data** section data of our file on disk:

Figure 7.27: *Offset of raw .data section*

To view the **.data** section in memory, we will first have to check the **VirtualAddress** value in the **.data** section header, which is 0x00017000. This is the RVA, and to get the address in memory, we will add Image base to it, $0x00017000 + 0x00400000 = 0x00417000$.

[Figure 7.28](#) shows the dumped view of the **.data** section in memory at address 0x00417000, and we can see the same section data in the file on disk and when the file loaded in memory:

Figure 7.28: *Section .data in memory*

Characteristics: This is the set of attributes that define the section attribute, like whether Section is writeable, readable, or executable, or any other value defined in **winNT.h**. Refer to [Figure 7.29](#):

Figure 7.29: *Section characteristics*

In [Figure 7.30](#), we can see 7 section headers in hex editor marked with different colors. There is no padding between the section headers, and after the section headers, we will see the sections themselves. Between the section headers and sections, a data padding of 0x00 bytes (marked with green in the following figure) is present.

When the file is on disk, each section starts with the multiple of the **FileAlignment** field in **OptionalHeader**. When the file is loaded in memory, all sections start with the multiple of the **SectionAlignment** field in **OptionalHeader**. On x86 platform, sections are aligned to the 4KB page

alignment, and on x64, they are aligned to 8KB of pages when file is loaded in memory.

Figure 7.30: *Section table in hexeditor*

Conclusion

In this chapter, we learned the binary file internals. We started by creating a binary file, which is also referred to as PE or executable file. We then covered the internal details of the PE file, which includes a discussion on DOS Header and DOS stub. Then we covered PE header, which also included File header, Optional header and Data directories. For more details about the PE file, we talked about different section in the PE file.

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

<https://discord.bpbonline.com>

CHAPTER 8

Portable Executable Sections

Introduction

In [Chapter 7, *Portable Executable Insides*](#), we took a brief look at the DOS and PE headers. We will take an in-depth look at the sections in portable executable files in this chapter. Sections form a majority of an executable, which contains the programmer's code. Sections are used to logically and physically separate the components of a program. Additionally, they assist with memory loading of the executable file during execution, since separate sections facilitate access and page loading. In this chapter, we will look at various sections in detail.

Structure

In this chapter, we will discuss the following topics:

- Portable executable sections:
 - Executable code section (commonly referred as .text)
 - Data sections (of which .data, .rdata, .bss are types)
 - Resources section (commonly referred as .rsrc)
 - Export data section (commonly referred as .edata)
 - Import data section (commonly referred as .idata)
 - Debug information section (commonly referred as .debug)

Objectives

The objective of this chapter is to help you understand the binary file internals section. Most executable files are divided into sections. Knowing about the various kinds and the significance of each is crucial. At least two sections—the code section and the data section—are present in every PE

file. Some applications might require different kind of sections, but not all applications require all the sections. In this chapter, we will talk about the various sections one by one.

Sections

This is where we can find the content of the file. This content can be code, resources, data and other information useful for the working of a PE file. Sections in the PE file are like the organs in the human body: section head and section body. By section head, we mean section header and by section body, we mean section raw data. We saw in the section table that section headers are organized in the systematic manner, but when we talk about section's data, it is organized the way the linker is programmed. All the information required to read a section can be found in the respective section header. In our file, we have seven sections: `.textbss`, `.text`, `.rdata`, `.data`, `.idata`, `.rsrc` and `.reloc`.

To better understand the sections, we will see a tabular figure of section table (taken from CFF Explorer tool) in [Figure 8.1](#), which will help us draw how our file looks when on the disk:

Figure 8.1: *Sections table view in CFF Explorer*

With the help of this, we can visualize our PE file on disk versus memory, as shown in [Figure 8.2](#):

Figure 8.2: *Sections on disk vs in memory*

We will now talk about these sections in detail to understand the data inside them.

.textbss section

As stated while discussing Optional header, **.textbss** is empty in the file on disk. It is used in virtual memory for non-initialized global variables. This section is present in our binary file (**HelloWorld.exe**) as default. Incremental Linking is enabled in Microsoft Visual Studio. If we disable Incremental Linking (by going to *Solution Properties | Linker | General | Enable Incremental Linking | No*) while compilation, the **.textbss** section will not be present in our executable.

In the earlier times, Linking was done in the batch process, wherein all object modules were merged into a single executable. However, this process was time-consuming when limited numbers of object modules were changed. So, to speed up the process, Microsoft enabled Incremental Linking by default.

.text section

This section is where the main application code resides, or we can also say our **HelloWorld.exe** file's executable code resides. The entry point of the first instruction to be executed falls inside this section. [*Figure 8.3*](#) is just a view of this section on the disk and in memory:

Figure 8.3: *.text section on disk vs in memory*

.rsrc section

This section contains the resource information of the file. It is structured into resources tree (which includes file icon, menu, dialog and other resource types) that can be viewed using resource editor. We will use Resource Hacker tool as it is freeware. You can download this from

<https://web.archive.org/web/20060115122347/http://www.angusj.com/resourcehacker/>.

However, before we open our file with resource editor, we will add icon resource to it. For that, we will create another application named **HelloWorldIconResource** and add resource to it. In order to understand the **.rsrc** segment, we will create an empty Win32 project using Microsoft Visual Studio 2010 Professional or any other version you have installed.

To create an empty project, go to **File** menu | **New** project, select **Win32 project** and enter the name of the project; we entered **HelloWorldIconResource**. Then, select the **Location** of the project as required and click on **OK** to proceed to the **Win32 Application Wizard**; select **Next** to continue. Select **Application type** as **Windows application** and **Additional options** as **Empty project**. Click on **Finish** to close the wizard.

Under **Source Files**, add new **item** with extension of **.cpp** file with the name **HelloWorldIconResource**.

After adding the file, add [Figure 8.4](#) to compile and generate our custom binary for PE analysis:

Figure 8.4: *HelloWorldIconResource program*

Before compilation, change **Character Set** to the **Not Set** option (go to project **Properties** | **Configuration Properties** | **General** | **Character Set** | **Not Set**).

In Solution explorer, right-click on the Resource Files **Add** | **Resource**, as shown in [Figure 8.5](#):

Figure 8.5: *Add Resource*

From the **Resource type**, select **Icon** and click on **New**. You will see files under both Resource Files and Header Files: **resource.h**, **HelloWorldIconResource.rc** and **icon1.ico**. Now, we will compile the

code to get our PE file with icon resource. [Figure 8.6](#) shows our compiled executable with icon and Resource Hacker tool with our executable:

Figure 8.6: *Resource Hacker*

Of all the sections in the PE file, **.rsrc** is the most complex to navigate. All the resources are found in this section. In *Data Directory*, we saw the **IMAGE_DIRECTORY_ENTRY_RESOURCE** entry, which defines the RVA and the size of resources in the PE file. Resources are organized in the tree structure. The pointer from Data Directory points to a structure of the **IMAGE_RESOURCE_DIRECTORY** type, as defined in the **winNT.h** header file. Refer to [Figure 8.7](#):

Figure 8.7: *IMAGE_RESOURCE_DIRECTORY structure*

The sum of **NumberOfNamedEntries** and **NumberOfIdEntries** defines the total number of elements in the **IMAGE_RESOURCE_DIRECTORY_ENTRY** array, as shown in the PView of resource section (PE file opened twice in PView for understanding). Refer to [Figure 8.8](#):

Figure 8.8: *Elements in IMAGE_RESOURCE_DIRECTORY_ENTRY array*

[Figure 8.9](#) provides a view of this section on the disk and in memory:

Figure 8.9: *.rsrc section on disk vs in memory*

.rdata

The section holds the read-only data and serves two purposes:

- Executable files produced using Microsoft linker has the Debug Directory. This directory holds the array of the **IMAGE_DEBUG_DIRECTORY** structure, which has the details about the size, type and the memory location of other types of debug information in executable. Debug directory in the **.rdata** section does not necessarily start from the beginning of the **.rdata** section, as shown in [Figure 8.10](#):

Figure 8.10: *Debug directory in .rdata section*

Debug directory is located in the **.rdata** section with the RVA value of the **IMAGE_DIRECTORY_ENTRY_DEBUG** structure, which is the one of the structures present in the array of 16 **IMAGE_DATA_DIRECTORY** structures.

- The second purpose of this section is that it stores the string global constants. [Figure 8.11](#) shows **.rdata** section on disk and in memory for understanding.

Figure 8.11: *.rdata section on disk vs in memory*

.data

Our initialized data goes under this section. Static and global variables initialized at compile time go under this section.

Export section

The export section is relevant to the **Dynamic Link Libraries (DLL)**. DLL files are not like executable files; you cannot run a DLL file in Windows by double-clicking on it. DLL files are library files that contain code (functions) and data to carry out certain tasks. With the help of DLL, code can be reused and modularized. Developers do not have to spend time

writing code from scratch for common functions. If an executable or process is using function(s) from a DLL, then the DLL file is loaded at the runtime and mapped to the process address space. In our **HelloWorld** binary file, we can list the DLL files loaded in the process address space. Refer to [Figure 8.12](#):

Figure 8.12: *DLL files loaded*

Functions are exported using **Ordinal** or **Name**, as shown in [Figure 8.12](#), for user32.dll. Ordinal is a 16-bit number that uniquely identifies a function in a DLL, and Name is a simple ASCII string that represents the function name.

Now, to understand the export section concept, we will create a DLL file using Microsoft Visual Studio 2010 or the latest version of Visual studio you have. Create a **New project**, select **win32 Console Application**, and then on the **Advance** tab, select **DLL** and check the **empty project** option.

We named our DLL file **SampleDLL**; creating a DLL file is not as difficult as some might believe. First, make a header file **SampleDLL.h**, which has our function prototypes. Refer to [Figure 8.13](#):

Figure 8.13: *SampleDLL Header file*

There are two ways of exporting functions in VC++:

- Using **__declspec**, which is a Microsoft-specific keyword
- By creating a **Module-Definition File (.DEF)**

The first way is a bit easier to do, so we used that in our header file.

Now, make the DLL source file and name it **SampleDLL.cpp**. Refer to [Figure 8.14](#):

Figure 8.14: *SampleDLL source file*

This is where you define all the functions. We have defined four functions to add, subtract, multiply and divide the parameters passed to functions. Now, compile code to generate **SampleDLL.dll**. We will be using this DLL to understand the export section by opening the DLL in PEview. In [Figure 8.15](#), we can see the mapping of DLL with respect to the PE structure:

Figure 8.15: *SampleDLL.dll PE structure mapping*

To find the offset of export directory in our PE file, the Export table in *Optional header* is referred to; it contains the *RVA of Export Directory*, as shown in [Figure 8.16](#):

Figure 8.16: *Export table of SampleDLL.dll*

The RVA of export directory is 0x00017D50, and to calculate raw offset, we will have to find the section in which this RVA is falling, as shown in [Figure 8.17](#):

Figure 8.17: *RVA is falling in .text section*

This RVA is falling in the **.rdata** section, so we use the following formula to calculate Raw Offset:

$$\text{Raw Offset} = \text{RVAtoConvert} - \text{RVAofSection} + \text{PointertoRawSection}$$
$$0x00006F50 = 0x00017D50 - 0x00016000 + 0x00005200$$

Raw Offset of export directory is 0x00006F50, as shown in [Figure 8.18](#). This offset is falling in the **.rdata** section. Under the **.rdata** section, we have **IMAGE_EXPORT_DIRECTORY** structure.

Figure 8.18: *Export directory is falling in .rdata section*

Export directory is structured using the **IMAGE_EXPORT_DIRECTORY** structure, as defined in **WinNT.h**. Refer to [Figure 8.19](#):

Figure 8.19: *IMAGE_EXPORT_DIRECTORY structure*

Characteristics: This field is unused and set to 0x000000.

Name: This is the internal name of the module. If the name of the file is changed by the user, PE loader will use the internal name.

Base: It is the starting ordinal number of the exported functions. Ordinal uniquely identifies a function in a DLL by a 16-bit number. If the ordinal value of three functions is 2, 4, and 5, then 2 is the Base value. In our SampleDLL.dll case, 1 is the *Base* value.

NumberOfFunctions: It is the number of exported functions either by name or by ordinal. This defines the number of elements of **AddressOfFunctions**. In our SampleDLL.dll, we have 4 functions defined, so the value of **NumberOfFunctions** is 4.

NumberOfNames: This value defines the number of functions exported by names. If this value is 0, it means that all the functions in the module are exported by ordinal. This defines the number of elements of **AddressOfNames**. In our SampleDLL.dll, we have 4 functions defined, so the value of **NumberOfNames** is 4.

AddressOfFunctions: This is an RVA that points to the **Export Address Table (EAT)**. EAT is the array of the RVAs of functions exported by the module, so **AddressOfFunctions** points to the initial element of the array.

AddressOfNames: This is an RVA that points to the **Export Name table (ENT)**. ENT is the array of RVAs of the function names in the module, so **AddressOfNames** points to the initial element of the array.

AddressOfNameOrdinals: This is an RVA that points to the **Export Ordinal table (EOT)**. EOT is the 16-bit array that holds ordinals of named functions.

Refer to [Figure 8.20](#) for an illustration of the Image export directory:

Figure 8.20: *Image export directory*

To understand this concept better, we will examine the **SampleDLL.dll** in x32dbg. But before opening this DLL, change the **DllCharacteristics** value to 0x0100 to disable ASLR on the DLL using *CFF Explorer*. To debug and analyze **SampleDLL.dll** in x32dbg, the first step is to load **C:\Windows\system32\rundll32.exe** in x32dbg. Once it is loaded, change the command line to enter the path of the DLL with the exported function ordinal:

```
"C:\Windows\System32\rundll32.exe"
```

```
"C:\JitenderN\Book\SampleDLL\Debug\SampleDLL.dll", #1
```

In x32dbg, remember to change the **Options|Preferences** in x32dbg to check the DLL entry point. Restart the process in x32dbg and hit **Run** until you see **Module: sampledll.dll** in the title of x32dbg and SampleDLL.dll load in the Memory map, as shown in [Figure 8.21](#):

Figure 8.21: *SampleDLL.dll loaded in process memory*

Our DLL is loaded at the 0x10000000 base addresses; using this base address, we will calculate Virtual Address to examine **Export Address Table (EAT)**, **Export Name table (ENT)** and **Export Ordinal table (EOT)**:

Virtual Address = Image base Address + Relative virtual address

Now, we will examine the EAT, ENT, and EOT in memory dump using x32dbg, as shown in [Figure 8.22](#):

Figure 8.22: *EAT, ENT, EOT in memory dump*

Now, to find the address of the function from function name, the operating system first gets **NumberOfFunctions** and **NumberOfNames** from the **Export Directory**. Then, it walks through the array pointed by **AddressOfNames** (**ENT**) and **AddressOfNameOrdinals** (**EOT**) in parallel. If the function name is found in the *ENT*, the associated element of *EOT* is extracted. This value is then used as an index into *EAT* to find the address of the function. The ordinals of the function change when the DLL is upgraded, so we cannot depend on the function ordinals to find the address of function. If a program is using ordinals to extract the function address, that program will break on the DLL upgrade.

Import section

This section can be viewed as the **.idata** section in portable executable editors. It contains information about the list of functions imported by an executable from **Dynamic Link Libraries (DLL)**. Windows loader is responsible for loading and mapping all DLLs used by an application onto the process address space. It also finds the memory addresses of imported functions from DLLs in use and makes them available to the executable.

To understand the Import Directory, we will create an application to use function from our **SampleDLL.dll** created in the export section. There are two ways to load DLL: one is using Implicit linking, and the other is using explicit linking. We will use Implicit linking to understand the import section concept. So, create a new **Win32 console project** and use the following source code to load **SampleDLL.dll**.

Add DLL in the Debug directory and link **SampleDLL.lib** in **Project properties | Linker | Input | Additional Dependencies**. Also, put the **SampleDLL.h** header file with other project header files or add a header file in the project. Refer to [Figure 8.23](#):

Figure 8.23: ImportSampleDLL code

The resulting executable (**ImportSampleDLL.exe**) will list our **SampleDLL.dll** in Import Directory.

Import Directory is an array of the **IMAGE_IMPORT_DESCRIPTOR** structure. Each structure is 20 bytes in size and contains information about a DLL imported in our application. For every DLL imported, we will have the **IMAGE_IMPORT_DESCRIPTOR** structure. Our application is importing functions from four different DLLs, so we have four **IMAGE_IMPORT_DESCRIPTOR** structures in array. There is no field that indicates the length of an array or the number of **IMAGE_IMPORT_DESCRIPTOR** structures in an array. Instead, the final **IMAGE_IMPORT_DESCRIPTOR** structure in an array is filled with zeros.

The structure of the **IMAGE_IMPORT_DESCRIPTOR** structure is defined in **WinNT.h** is shown in [Figure 8.24](#):

Figure 8.24: *IMAGE_IMPORT_DESCRIPTOR* structure

OriginalFirstThunk: Union means either of **DWORD (Characteristics** or **OriginalFirstThunk**). **Characteristics** were earlier used by Microsoft as a set of flags, but it is now the RVA of an array of **IMAGE_THUNK_DATA** structures.

TimeStamp: It is set to 0 if not bound. It is -1 if bound, and real date/time stamp.

ForwarderChain: This is the first forwarder chain reference's index. It is the component in charge of DLL forwarding. When a DLL transfers part of its exported functions to another DLL, it is known as DLL forwarding.

Name: It contains the RVA to ASCII name of DLL. We can also refer to RVA as pointer.

FirstThunk: It contains the RVA of an array of **IMAGE_THUNK_DATA** structures, which is a duplicate of arrays defined in **OriginalFirstThunk**.

When a file is on disk, the array pointed by **OriginalFirstThunk** and **FirstThunk** are terminated by NULL DWORD.

The structure of **IMAGE_THUNK_DATA** as defined in **winNT.h** is shown in [Figure 8.25](#):

Figure 8.25: *IMAGE_THUNK_DATA structure*

This structure is DWORD union of one of the two values. When the file is on disk, it contains the ordinal of imported functions or RVA to an **IMAGE_IMPORT_BY_NAME** structure. The structure of **IMAGE_IMPORT_BY_NAME** as defined in **winNT.h** is shown in [Figure 8.26](#):

Figure 8.26: *IMAGE_IMPORT_BY_NAME structure*

Hint: Loader uses the value as an index into **Export Address Table (EAT)** of the DLL to look up the function. Some linkers set this field to zero.

Name1: It contains the name of the imported function. As the function name length can vary, it is a variable size field that is NULL terminated ASCII string. As there is no way to define variable size field in structure, it is defined as BYTE.

For better understanding, let us analyze the import directory on our **ImportSampleDLL** executable file. Refer to [Figure 8.27](#):

Figure 8.27: *ImportSampleDLL.exe Import Directory*

To understand the import directory structure, we use PEview. Open **ImportSampleDLL** portable executable file in PEview. To get the pointer or

RVA to Import Directory, go to **IMAGE_NT_HEADERS** | **IMAGE_OPTIONAL_HEADER** | **Import Table** | **RVA**, which is 0x0001A000, as shown in [Figure 8.28](#).

As we are analyzing the file on the disk, we will have to convert this RVA to file offset to navigate to the Import Directory. To avoid this process of converting RVA to file offset again and again during our analysis, we will use an option in PEvent to convert file offset addresses to RVA for ease of analysis. To do this, we will modify the visual look of address column in PEvent by navigating to the **View** option in the menu bar, and then going to **Address** and changing it from **File Offset** to **Relative Virtual Address**.

[Figure 8.28](#) shows the **ImportSampleDLL.exe** Import Directory RVA:

Figure 8.28: *ImportSampleDLL.exe* Import Directory RVA

As defined earlier, the Import Directory is an array of **IMAGE_IMPORT_DESCRIPTOR** structure, and for every DLL imported by **ImportSampleDLL.exe**, we will have the **IMAGE_IMPORT_DESCRIPTOR** structure. This can be seen in the PEvent, as shown in [Figure 8.29](#). Our application is importing functions from four different DLLs, so we have four **IMAGE_IMPORT_DESCRIPTOR** structures in the array. The final **IMAGE_IMPORT_DESCRIPTOR** structure is an array filled with zeros.

Figure 8.29: *List of DLL Imported by application*

In PEvent, **OriginalFirstThunk** and **FirstThunk** are represented by **Import Name table RVA** and **Import Address Table RVA**. **OriginalFirstThunk** (value is 0x0001A230) and **FirstThunk** (value is 0x0001A440) contain the RVA of an array of **IMAGE_THUNK_DATA** structures. Each **IMAGE_THUNK_DATA** structure represents one imported function from the DLL. As our application imports two functions from *SampleDLL*, we have two **IMAGE_THUNK_DATA** structures. As we can see in [Figure 8.30](#), the

array pointed by **OriginalFirstThunk** and **FirstThunk** are NULL
DWORD terminated:

Figure 8.30: *Pointer to INT IAT*

The array pointed by **OriginalFirstThunk** is called **Import Name Table (INT)**, and the one pointed by **FirstThunk** is called **Import Address Table (IAT)**. RVA in **IMAGE_THUNK_DATA** points to the **IMAGE_IMPORT_BY_NAME** structure, as shown in [Figure 8.31](#):

Figure 8.31: *RVA to IMAGE_IMPORT_BY_NAME*

As discussed earlier, the **IMAGE_IMPORT_BY_NAME** structure comprises of Hint and the name of imported function from DLL. We can notice in the hex dump of **IMAGE_IMPORT_BY_NAME** structure, 00 03 is the hint value and name of the imported function is ?

Subtract@MySampleDLLFuncs@SampleDLLFuncs@@SANNN@Z which is null terminated ASCII value.

You must be thinking why the function name is a combination of special characters. This is because during compilation process, function names are encoded to append calling convention, function parameters, function return type and other information with the function name. This process is also referred to as name decoration and helps a linker find the correct function when linking an executable. This process is also known as name mangling.

When PE file is loaded in memory.

Now, we will analyze our executable when it is loaded in memory. We will use x32dbg to load our portable executable. As per [Figure 8.32](#), import directory starts from 0x0041A000 virtual address ($VA = Image\ Base\ Address\ (0x00400000) + RVA\ (0x0001A000)$), and this virtual address falls in the **.idata** section, as shown in the memory map in [Figure 8.32](#):

Figure 8.32: .idata section

In the .idata section memory dump, we can see start of import directory with the array of **IMAGE_IMPORT_DESCRIPTOR**, each of size 20 bytes. As discussed earlier, each **IMAGE_IMPORT_DESCRIPTOR** represents imported DLL, and this array of **IMAGE_IMPORT_DESCRIPTOR** is terminated by zeros of 20 bytes. Each group of 5 DWORD marked in different colors represents **IMAGE_IMPORT_DESCRIPTOR**. As per the **IMAGE_IMPORT_DESCRIPTOR** structure, the first and last DWORD represent the **OriginalFirstThunk** and **FirstThunk** fields.

OriginalFirstThunk contains RVA 0x0001A238 (because of little endian format, bytes are reversed, so VA = 0x0041A238) to IAT, and **FirstThunk** contains RVA 0x0001A440 (VA = 0x0041A440) to IAT. Now, let us dump IAT and INT in the memory dump by right-clicking in the hex dump to get **Context menu | Go to | Expression**. The dump at virtual address 0x0041A238 and 0x0001A440 shows INT and IAT, as shown in [Figure 8.33](#):

Figure 8.33: INT and IAT

The main point to note here is that when the PE file is loaded in memory, IAT is overwritten by loader with the address of the following functions in **SampleDLL.dll**:

- **?Add@MySampleDLLFuncs@SampleDLLFuncs@@SANNN@Z** is at address 0x1001107D.
- **?Subtract@MySampleDLLFuncs@SampleDLLFuncs@@SANNN@Z** is at address 0x100110E6.

Refer to [Figure 8.34](#):

Figure 8.34: *ImportSampleDLL.exe Import Directory in memory*

If we dump the values of IAT in CPU window, we see the code of the preceding functions, as illustrated in [Figure 8.35](#):

Figure 8.35: *IAT Dumped in memory*

We can also dump the entire IAT in dump windows by right-clicking in it to get the **Context** window and selecting **Address** to list all imported functions in the executable. Refer to [Figure 8.36](#):

Figure 8.36: *List of Functions Imported in Executable*

Conclusion

In this chapter, we learned the binary file section internals. We saw that most executable files are divided into sections, and knowing about the various kinds of sections and the significance of each is crucial for understanding any portable executable. We also saw that two sections—the code section and the data section—are present in every PE file. Further on, we understood that some executables might require different kinds of section, but all executables do not require all sections.

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

<https://discord.bpbonline.com>

Section III: Ransomware **Assessment**

CHAPTER 9

Performing Static Analysis

Introduction

Every element of life benefits greatly from the use of tools. In our daily lives, we frequently use mobile calculators to complete simple math. We used to carry hardware calculators to perform math on the job earlier. This calculator is a basic illustration of a tool we might employ to carry out specific activities. Simple computations can be made verbally, but tools are required for complex ones. Similar tools are readily accessible on the market for reverse engineering. Some are open source or free to use, while others are commercial. Conceptual knowledge about the subject is crucial for tool selection.

Imagine how good it will be to analyse any ransomware before executing it. This is only possible with static analysis. Performing static analysis with the help of tools greatly helps us avoid many mathematical calculations. In this chapter, we will talk about some open-source commercial tools for static analysis and will perform static analysis on a ransomware sample. Manually unpacking samples that were packed purposely to resist static analysis may be the most difficult aspect of the process.

Structure

In this chapter, we will discuss the following topics:

- Ransomware static analysis
- Decompile ransomware sample
- Look at all the phases of ransomware infection
- Static analysis tools like PE studio and CFF Explorer

Objectives

The objective of this chapter is to help you understand static analysis. In this chapter, we will take a ransomware sample and perform static analysis over it. We will decompile the ransomware, go through the ransomware decompiled code, and map it with different phases of ransomware in general. We will also cover how we can break ransomware using only static analysis. The inexperienced the ransomware writer is, the easier it is to break the ransomware. The reader will also learn about the tools required to perform static analysis.

Static analysis

Static analysis refers to the process of analysing malicious application without launching them and gathering data about them. Analysts researching ransomware do not have the luxury of analysing the source code of ransomware, as they do with a majority of malware. The next best choice, however, is provided by interpreted languages like .NET because ransomware built in them can be decompiled. In the case of interpreted languages, the outcome of decompilation is substantially nearer to the real source code. As a result, even if these ransomware programmers still use some form of obfuscation, analysis is made simpler when the infection is built in an interpreted language. Malware created in compiled languages, on the other hand, can only be statically disassembled once the ransomware has been unpacked to reveal the relevant assembly code. However, ransomware typically comes tightly packaged to prevent reverse engineering.

Ransomware reverse engineering is used for static analysis to examine its functionality. Decompilers like ILSpy generate source code in high level-languages that are similar to the original source code. Decompilers are unable to offer the source code directly, but some of their efforts are successful in producing high-level code that is readable to analyse than assembly code.

The objective of ransomware writers is simple: somehow commence ransomware execution on victim computer, encrypt user data and demand ransom to decrypt it. To generalize, all this can be bifurcated in phases, as shown in [Figure 9.1](#):

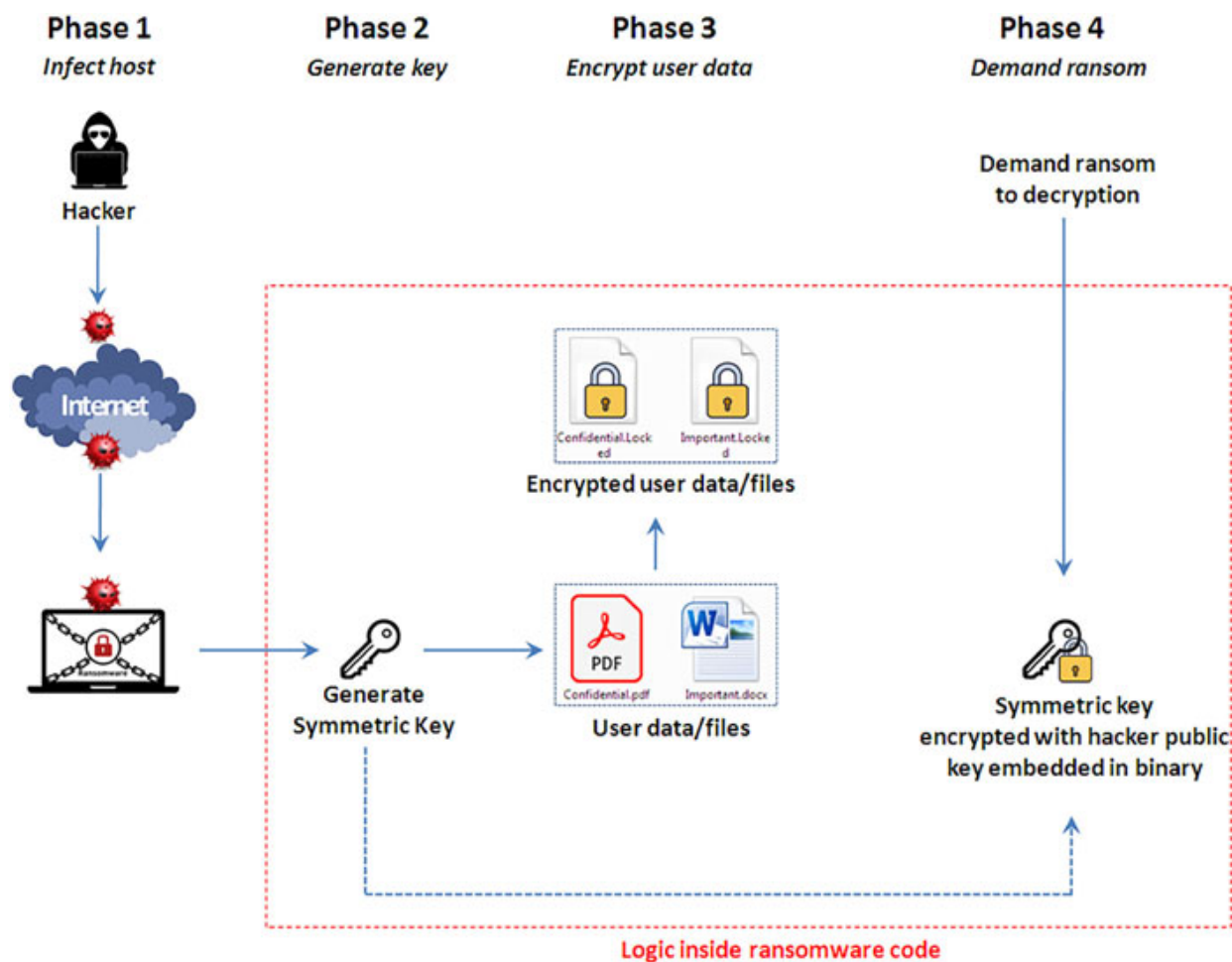


Figure 9.1: Ransomware Phases

In the next section, we will cover the various stages of ransomware infection and see how static analysis is enough to break ransomware sometimes.

Phase 1 - Infect host

Before we start with the static analysis, you must make sure you are testing in a dedicated and isolated environment when undertaking malware analysis. Malware analysis and testing in live systems is never a good idea. You must create your own malware analysis lab and sandbox for this reason. The simplest method is to set up isolated virtual machines (Linux and Windows). There are free or open-source tools within each category that serve this purpose, with just a few minor feature differences. Tools with a graphical user interface that are free and open source will be our main focus. All these tools are used throughout the analysis of ransomware.

Ransomware is delivered with an executable named “*A Big Present.exe*” that contains the ransomware executable code. When ransomware is executed, it encrypts files with the “.THBEC” extension. Upon ransomware encryption, a window alert is presented to the victim. The attackers promise to decrypt two files for free but warn the victim that if they do not pay, they will not get their files back. The ransom note includes a series of instructions to visit a darknet site to pay the ransom, along with the phrase, “*Have a lovely day with the love from India.*” To understand how static analysis can help understand the ransomware phases, we will take a ransomware sample and run it through a tool called PE studio. Refer to [Figure 9.2](#):



Figure 9.2: *RansomWarrior*

Most antivirus solutions use some of these fields like MD5 and SHA256 of the ransomware sample to detect and prevent ransomware attacks. There are also other fields that are of some interest to a ransomware analyst.

Signature of the sample shows that it is .NET executable. Malware researchers can tell whether an executable or malware sample is encrypted or obfuscated using entropy analysis. Entropy analysis works on the well-known Shannon's Formula and measures the entropy of executable on the scale of 0-8. Lower entropy values reduce the likelihood of code obfuscation, while higher entropy increases the likelihood of executable obfuscation. Entropy of a typical executable is usually around 3, indicating that the binary is not compressed, encrypted, or disguised. The RansomWarrior sample, in our case, has an entropy value of 3.1.34, indicating that binary is not obfuscated, compressed, or encrypted.

We can also confirm from [Figure 9.2](#) that the sample is portable executable of 32 bit. Compiler-stamp shows that the executable is compiled on Nov 06, 2017. RansomWarrior ransomware came after this, but we should not heavily rely on this value as it can be modified.

The next field of our interest is the strings in the PE studio. It lists all the text strings present in the ransomware sample, as shown in [Figure 9.3](#):

pestudio 9.19 - Malware Initial Assessment - www.wintor.com [c:\jiten\randomware samples\ransomwarrior\big present.exe.bin]

file settings about

encoding (2)	size (bytes)	file-offset	blacklist (2)	hint (74)	group (3)	value (1713)
ascii	5	0x00000872	-	utility	-	Write
ascii	7	0x00000C30	-	utility	-	Program
ascii	7	0x0000E20E	-	utility	-	Process
ascii	5	0x0000E302	-	utility	-	Start
unicode	6	0x0000E654	-	utility	-	Delete
unicode	7	0x0000E6EE	-	utility	-	Decrypt
unicode	7	0x0000E770	-	utility	-	cmd.exe
unicode	34	0x0000E780	-	utility	-	/C choice /C Y /N /D Y /T 3 & Del
unicode	13	0x0000786A	-	utility	-	Program Files
unicode	19	0x00007886	-	utility	-	Program Files (x86)
unicode	11	0x00007982	-	utility	-	Tor Browser
unicode	6	0x00007B82	-	utility	-	Chrome
ascii	7	0x0000B1D7	-	url-pattern	-	4.0.0.0
ascii	8	0x0000B230	-	url-pattern	-	11.0.0.0
ascii	11	0x0000B96A	-	import	-	_CorExeMain
ascii	21	0x0000DA78	-	file	-	RansomWarrior 1.0.exe
ascii	98	0x0000B830	-	file	-	C:\Users\lenovo\Documents\dd\RansomWarrior 1.0\
ascii	11	0x0000B976	-	file	-	mscoree.dll
unicode	12	0x000077F4	-	file	-	\$Recycle.Bin
unicode	4	0x00007CA4	-	file	-	.ASM
unicode	4	0x00007CAE	-	file	-	.ASP
unicode	4	0x00007CB8	-	file	-	.AVI
unicode	4	0x00007CC2	-	file	-	.BAK
unicode	4	0x00007CCC	-	file	-	.BAS
unicode	4	0x00007CD6	-	file	-	.BAT
unicode	4	0x00007CEA	-	file	-	.BMP
unicode	6	0x00007CFE	-	file	-	.CLASS
unicode	4	0x00007D16	-	file	-	.CMD
unicode	4	0x00007D2A	-	file	-	.COM
unicode	4	0x00007D34	-	file	-	.CPP
unicode	4	0x00007D48	-	file	-	.DAT
unicode	4	0x00007D52	-	file	-	.DBF
unicode	4	0x00007D5C	-	file	-	.DIF

sha256: 825A42A32624644933CF5513064CE3E299505F0DD0E1E524C5C8950D240F2945A cpu: 32-bit file-type: executable subsystem: GUI entry-point: 0x0000D786

Figure 9.3: Strings in RansomWarrior Sample

If we can see the highlighted lines in [Figure 9.3](#), when combined, these strings form a command that is coded to make an application delete itself completely, using just two lines of code. The two lines of code when combined are as follows:

```
cmd.exe /C choice /C Y /N /D Y /T 3 & Del
```

The definition of the various parameters is given in the following section:

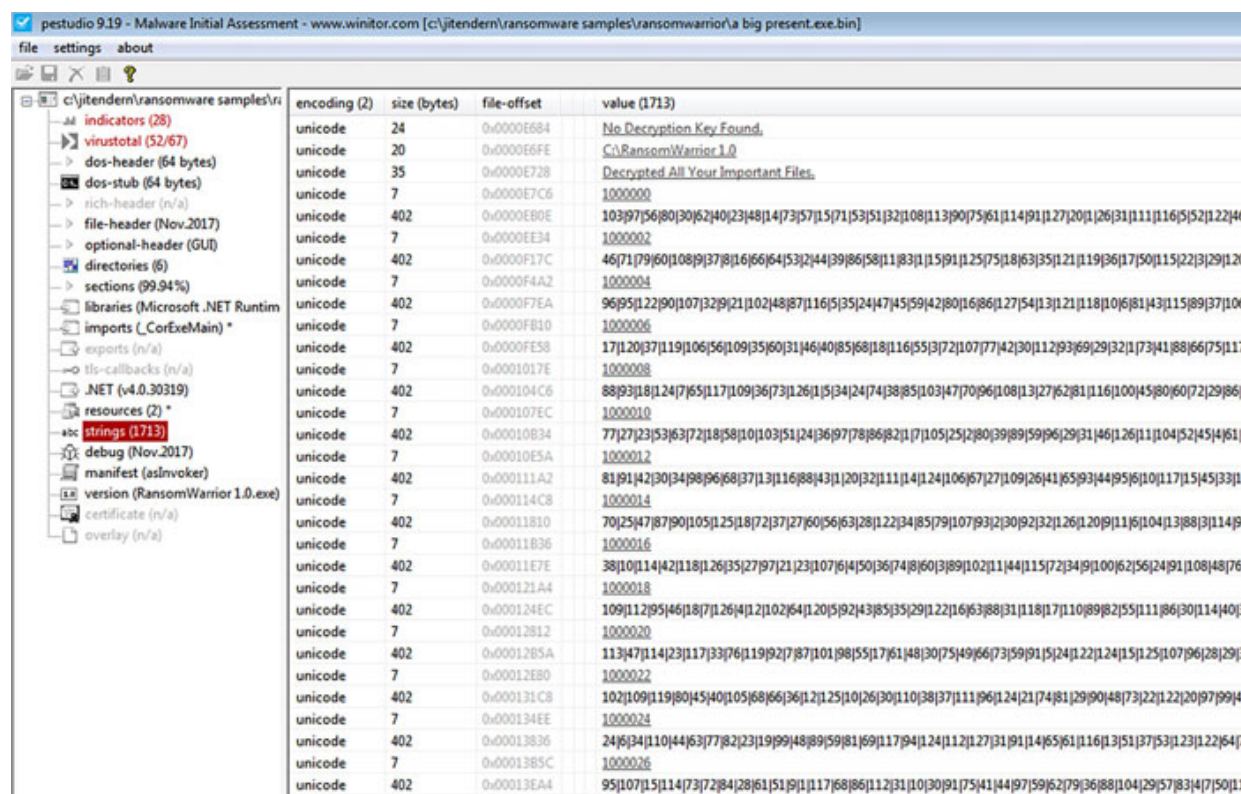
- When `cmd.exe /C` is used, a command window opens and executes the supplied command before closing itself.
- The flashing blank prompt is displayed by `choice /C Y /N /D Y /T 3`. However, `/T 3` instructs the prompt to choose the default option `Y (/D Y)` after 3 seconds.
- `&` is used in batch files to group related commands together on a single line.
- You probably have an idea of what `Del <Application.ExecutablePath>` does. Anything you wish to occur after the 3-second delay can be substituted for everything after the `&`

sign. You could instruct the command to remove each file from the directory.

While scrolling down the strings in the ransomware, something of interest can be seen, as shown in [Figure 9.4](#):

Figure 9.4: File Extensions

The file extensions marked in the red box are the ones that are encrypted by the ransomware, so these are the file extensions targeted by RansomWarrior ransomware. On moving further down the strings, we can see some abrupt long numbers, as shown in [Figure 9.5](#):



encoding (2)	size (bytes)	file-offset	value (1713)
unicode	24	0x0000E684	No Decryption Key Found.
unicode	20	0x0000E6FE	C:\RansomWarrior 1.0
unicode	35	0x0000E728	Decrypted All Your Important Files.
unicode	7	0x0000E7C6	1000000
unicode	402	0x0000E80E	1039715680130624023481473571571535132108113907561114911272011263111111655212246
unicode	7	0x0000EE34	1000002
unicode	402	0x0000F17C	4671796010893781666645324443986581183115911257518633512111936175011522329120
unicode	7	0x0000F4A2	1000004
unicode	402	0x0000F7EA	9695122901073292110248871165352447455942801686127541312111810681431158937106
unicode	7	0x0000FB10	1000006
unicode	402	0x0000FE58	1712037119106561093560314640856818116553721077742301129369293217341886675117
unicode	7	0x0001017E	1000008
unicode	402	0x000104C6	88931812476511710936731261153424743885103477096108132762811161004580607229861
unicode	7	0x000107EC	1000010
unicode	402	0x00010B34	772723536372185810103512436977886821710525280398959962931461261110452454611
unicode	7	0x00010E5A	1000012
unicode	402	0x000111A2	8191423034989668371311688431203211114124106672710926416593449561011715453312
unicode	7	0x000114C8	1000014
unicode	402	0x00011810	7012547879010512518723727605663281223485791079323092321261209116104113883114491
unicode	7	0x00011B36	1000016
unicode	402	0x00011E7E	381011442118126352797212310764503674860389102114411572349100625624911084876
unicode	7	0x000121A4	1000018
unicode	402	0x000124EC	10911295461871264121026541205924385352912216638831118171108982551118630114403
unicode	7	0x00012812	1000020
unicode	402	0x00012B5A	11347711423117337611992787101985517614830754966735991524122124151251079628293
unicode	7	0x00012E80	1000022
unicode	402	0x000131C8	1021091198045401056866361212510263011038371119612421748129904873221220979941
unicode	7	0x000134EE	1000024
unicode	402	0x00013836	246341104463778231999488959816911794124112127319114656111613513753123122647
unicode	7	0x00013B5C	1000026
unicode	402	0x00013EA4	951071511473728428615191117688611231103091754144975962793688104295783475011

Figure 9.5: Combination of Numbers

For the time being, we will keep note of this and assume that it is something that belongs to the ransomware key. Moving further down the strings list, we can see that the ransom note prompted to the victim is shown in [Figure 9.6](#):

Figure 9.6: Ransom Note

This ransom note also lists the Bitcoin address where the attacker wants the ransom to be sent. We can also see that the payment asked is in Bitcoin.

Now, we will move to the Key generation phase and see how it is being handled in the RansomWarrior ransomware.

Phase 2 – Generate key

After the ransomware infection phase is finished, the ransomware starts encrypting victim machine using cryptographic secret. Key generation is an essential component of the infection model, wherein ransomware needs to obtain unique keys to infect the victim. In this section, we will try to extract key generation login to RansomWarrior using reverse engineering techniques.

As we have extracted using PE studio, the ransomware sample of RansomWarrior is .NET executable. We will use Decompilers like ILSpy, which make an effort to produce source code files in high-level languages that are similar to the original source code. [Figure 9.7](#) shows RansomWarrior sample in ILSpy:

Figure 9.7: ILSpy decompile RansomWarrior

To open the ransomware sample in ILSpy, go to **File** in menu and open ransomware sample. As we can see, the ILSpy Decompiler produces source code files that are similar to the original source code.

Now, to extract the key generation login, we will go through the decompiled code and see how a ransomware writer has written the key generation logic. The objective is to check the key generation logic and find ways to break it. Refer to [Figure 9.8](#):

Figure 9.8: RansomWarrior Decompiling

When you click on **Crypt** in the left panel, it will decompile the code, as shown in [Figure 9.8](#). Looking inside the **Encryption** function, we notice that the Ransomware's "encryption" is a stream cipher with a key chosen at random from a list of 1000 hard-coded keys in RansomWarrior's binary code. Refer to [Figure 9.9](#):

Figure 9.9: RansomWarrior Keys

To make the victim's files inaccessible, RansomWarrior 1.0 employs AES 256 encryption.

Phase 3 – Encrypt User Data

This is the phase where ransomware performs the main job of encrypting the victim data. As you can see in [Figure 9.10](#), the bytes of the file to be encrypted are contained in the first parameter (Bytes List) of the encryption function:

Figure 9.10: RansomWarrior Encryption Logic

Now if we scroll down in the code, we see that RansomWarrior 1.0 Ransomware identifies the encrypted files by appending the file extension '.THBEC' to each file's name. Refer to [Figure 9.11](#):

Figure 9.11: Encrypt User Data

Phase 4 – Demand ransom

This is phase where the ransomware demands ransom from the victim by popping up a window or locking the whole system with an error message. If we look at the decompiled code of RansomWarrior in [Figure 9.12](#), we see who developed this ransomware and how the ransom is being demanded:

Figure 9.12: RansomWarrior Ransom Demand

So, we say that while analysing the key generation logic, we were able to break the ransomware by the extracted hardcoded keys used by ransomware writer.

We saw how static analysis sometimes can help us break ransomware logic. However, all this depends on the ransomware writer's skills and strategy. Some ransomware variants are very hard to break, and some are nearly

impossible, unless you have taken a memory snapshot at the time of ransomware infection or some other means.

Static analysis tools

We saw how static analysis plays a key role in breaking ransomware logic sometimes. When conducting ransomware analysis, you must ensure that you are testing in a dedicated and isolated environment. It is never a good idea to test and analyse ransomware on live systems. For this reason, you must establish your own ransomware analysis facility and sandbox. The simplest approach is to create a separate isolated virtual image (Linux and Windows). There are many categories of tools available for static analysis, and within each category, there are free or open-source programs that accomplish the same task with a few minor feature variations. We will concentrate on free and open-source tools having a graphical user interface; all these tools are utilised when analysing ransomwares.

PE Studio

With the help of the free program PE Studio, you can quickly evaluate ransomware without even infecting a system or examining its source code. The PE Studio can be seen in [Figure 9.13](#):

Figure 9.13: PE Studio

CFF explorer

The PE editing process was made as simple as possible with CFF Explorer. This tool is for programmers as well as reverse engineers. Refer to [Figure 9.14](#):

Figure 9.14: CFF Explorer

Conclusion

In this chapter, we covered the static analysis process. We worked on the ransomware sample and run static analysis over it. We also decompiled the

ransomware, went through the ransomware decompiled code and mapped it with different phases of ransomware in general. Further on, we broke ransomware sample key generation logic using static analysis and looked at how a weak ransomware code can easily be defeated. We also covered the tools required to do static analysis.

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

<https://discord.bpbonline.com>

CHAPTER 10

Perform Dynamic Analysis

Introduction

Analysis of ransomware is essential for preventing and comprehending the cyber attacks of today. When incident response teams are called to a ransomware outbreak, they usually collect and analyze one or more samples to learn more about the capabilities of the attacker and to help them with their investigation. Analysts are constantly searching for methods to triage and comprehend samples quickly and effectively, as firms cope with a growing number of attacks and breaches. In the previous chapters, we talked about ransomware static analysis, in which we perform ransomware analysis without running the ransomware binary or code. In this chapter, we will look at tools used for dynamic analysis, which covers analysis while ransomware is executing in the memory. We will also examine different ransomware dynamic analysis tools currently in use by researchers.

Structure

In this chapter, we will discuss the following topics:

- Dynamic analysis
- Dynamic analysis tools
- Disassemblers
- Debuggers
- Monitors

Objectives

The objective of this chapter is to help the user understand dynamic analysis. This chapter will walk you through the different types of tools required for dynamic analysis. We will start with the basic concept of ransomware dynamic analysis and then go over the basics of disassembler and different

disassemblers. Further on, we will cover different debuggers used for dynamic analysis of ransomware. Toward the end, we will cover different tools used to monitor the ransomware process.

Dynamic analysis

A program written in a high-level language like C or C++ is translated into a collection of bytes that a CPU can comprehend during compilation. This is machine code, which is challenging for humans to comprehend. With the aid of a programme called disassembler, we can make this code simpler to comprehend. This disassembler converts machine code into a form that is readable for humans. This format, which uses assembly language grammar, is known as assembly code. Refer to [*Figure 10.1*](#) to better comprehend the idea:

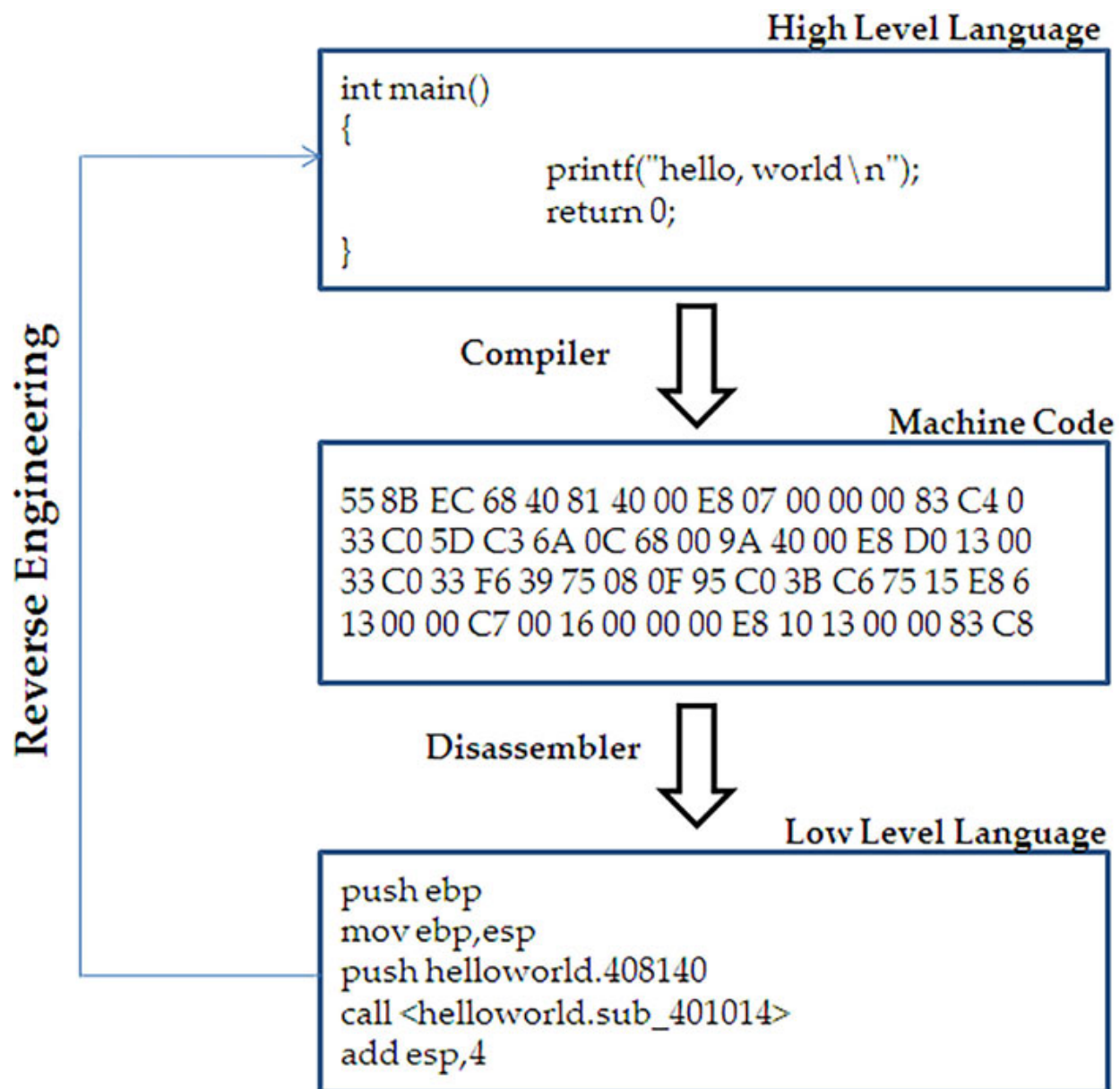


Figure 10.1: Machine Code Concept

Without access to an application or program's source code, reverse engineering aids in the regeneration of the application logic. Once we have the application or ransomware logic, we can walk through the assembly code using disassemblers and debuggers. This idea is used by ransomware researchers to perform ransomware dynamic analysis.

We will talk about some of the tools needed for reverse engineering now. These resources will also be used in this book. All the tools are free or have a free community edition, and using them to see whether they meet your

needs is a great way to save money. Where paid versions are offered, we strongly advise using them for business purposes.

Disassemblers

We need some tools to translate machine code into a human-readable format. Disassemblers come into play in this situation. They are tools that transform machine code into assembly code that can be read by humans. Some of the disassemblers used in this book are listed as follows:

IDA

Interactive Disassembler (IDA) comes in two different versions:

- IDA Starter
- IDA Professional

Hex-Rays, the organization that creates IDA, also provide the freeware version of IDA and the IDA Evaluation Version, a constrained disassembler (free for non-commercial use). [Figure 10.2](#) features IDA Pro:

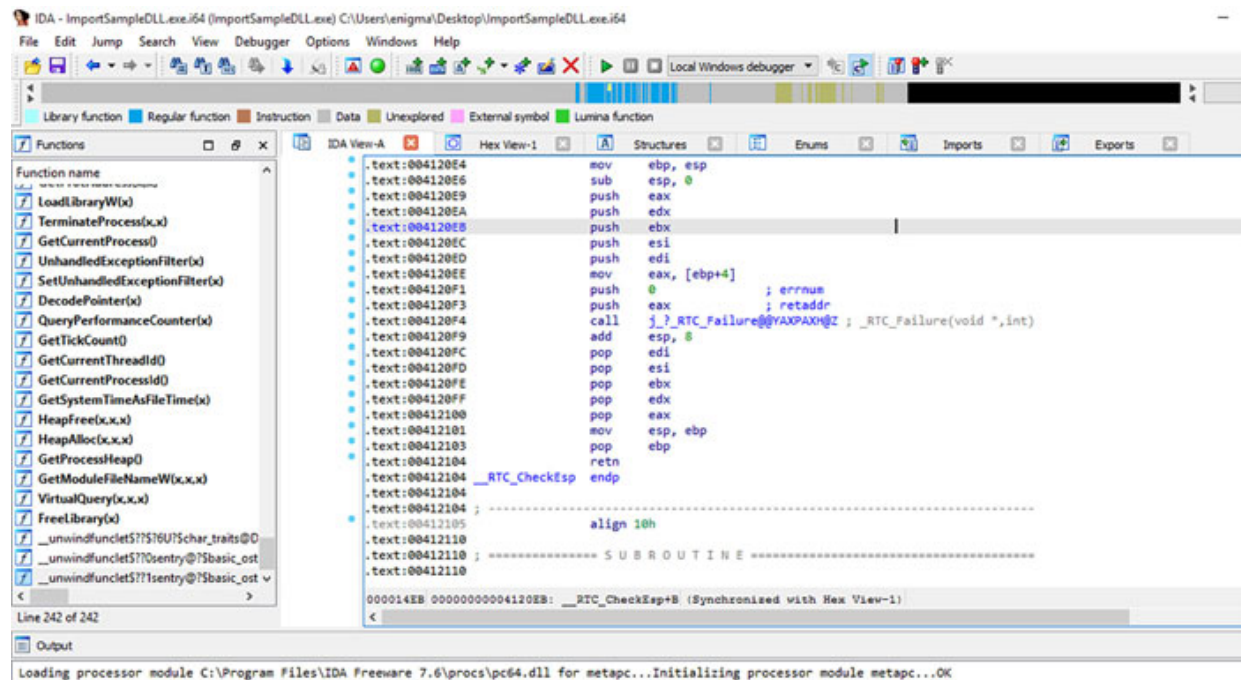


Figure 10.2: IDA Pro

One of the many reasons it is a wonderful tool for malware analysis is its capacity to extract a lot of data, including text, imports, exports, and graph flows. One platform can do the tasks of a disassembler, debugger, and decompiler all at once. Based on cross-references between code sections, knowledge of the arguments of API calls, and other information, IDA Pro can run an automatic code analysis. However, not everything is automated; human intervention is required to adjust the normally organic process of disassembly. The interactive features of IDA Pro are designed for this.

IDA Pro can operate in text and graph modes while performing the disassembly function. The graph mode displays a single function at a time by singling it out to display it as an interconnected block of code, while the text mode displays the full disassembled program as if it had been mapped into memory.

IDA Pro examines the whole executable file when a new file is opened for analysis, producing an **.idb** database archive. Four files, as listed here, are included in the **.idb** archive:

- **name.id0**: Contains B-tree style database
- **name.id1**: Contains flags for each program byte
- **name.nam**: Contains index information for program locations
- **name.til**: Contains details of local type definitions

These file types are all exclusive to IDA and are all proprietary. When a specific executable's **.idb** database is built, IDA does not have to re-analyze the program when we load it later. Additionally, IDA no longer needs the executable; instead, we can operate with just the **.idb** file. This feature can be used to distribute **.idb** files to other researchers without the malicious executable, which is a helpful feature. As a result, IDA may analyze an executable using only the database archive file and not the actual executable.

Ghidra

The **National Security Agency (NSA)** produced this open-source program. It is used for reverse engineering and is free. On April 4, 2019, the source code was made available. Researchers who study malware and reverse engineers use this tool to examine malware and identify app vulnerabilities.

[Figure 10.3](#) features the Ghidra interface:

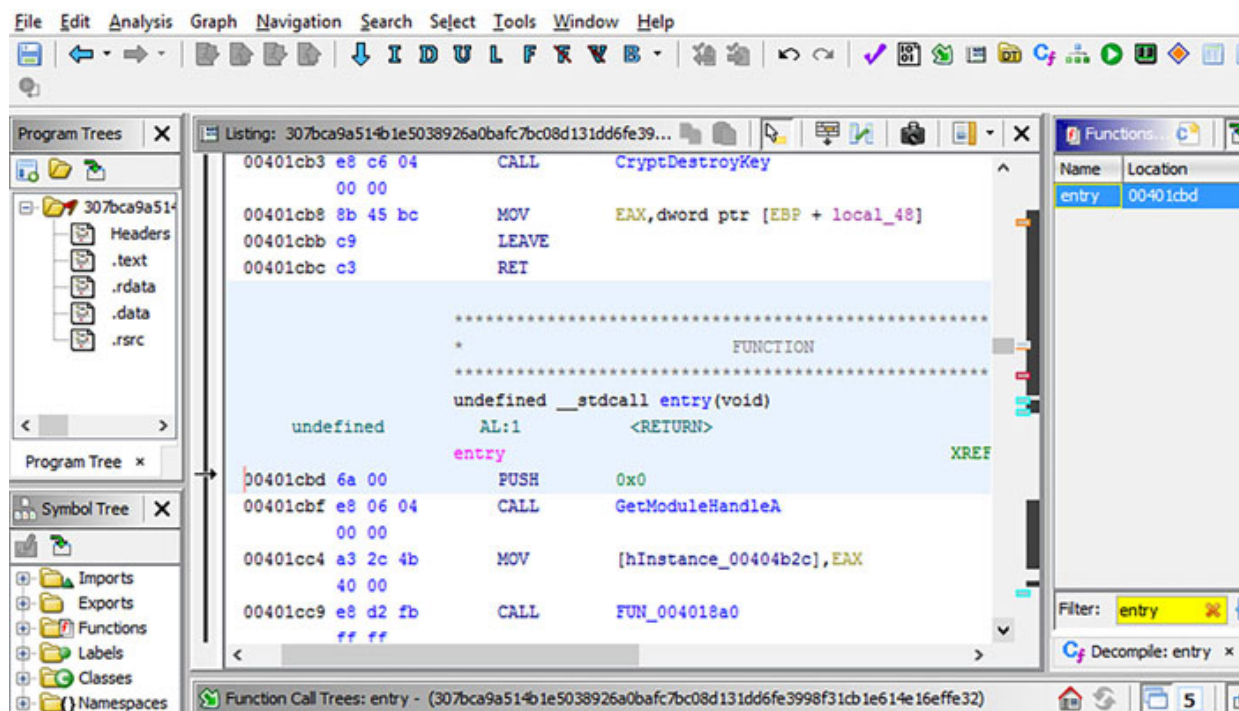


Figure 10.3: Ghidra

Ghidra can be modified by creating Python or Java plugins or scripts. This enables a malware analyst to examine a malware sample's functioning without running it. The analysts can then look through the malware code and determine what it is doing, which is very helpful.

Ghidra and tools like x64dbg are different since x64dbg is a debugger and will execute the malware as you examine the code. Therefore, if you discover a useful function in x64dbg, such as one that encrypts all files, it will execute that code and encrypt all the files on the computer you are using to analyze the malware.

A disassembly tool like Ghidra does not execute the code. Instead, it maps out the malware's assembly code and enables the user to go forward and backward through the code without affecting the analysis device's file system. Since Ghidra can map out functions that may be of additional interest to a malware analyst, it is the perfect tool for doing so.

Cutter

Cutter is an open-source interface to the Radare2 reverse engineering platform. Radare2 is a command-line program for reverse engineering that is used to analyze binary formats, both statically and dynamically, across many

platforms and architectures. Radare2's graphical user interface is Cutter. [Figure 10.4](#) features the cutter interface:

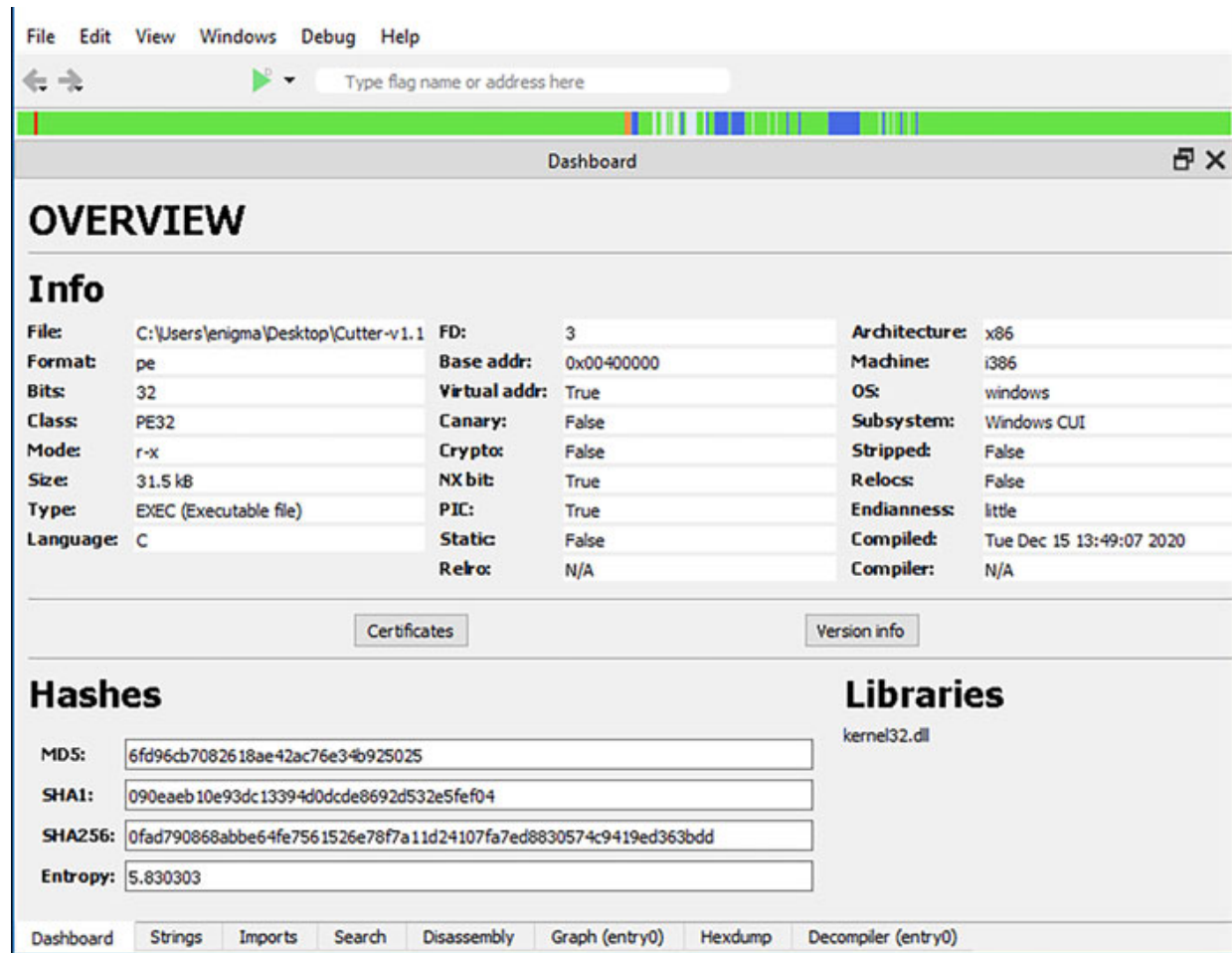


Figure 10.4: Cutter

Debuggers

We examine a running program's status when we run an application or binary. When a binary is performed in memory, a debugger displays its dynamic state. While writing or running a piece of code, some vulnerabilities are missed by the developer. Debuggers are used at this point to execute the code and keep an eye on the registers, memory locations, and other variables. In this book, the following debuggers will be used frequently.

x32dbg/x64dbg

An open-source Windows binary debugger called x32dbg/x64dbg is designed for malware investigation and executable reverse engineering. There are numerous features accessible, and a robust plugin system is included. On the wiki, there are numerous plugins for this debugger.

This debugger comes for 32 bit and 64 bit. For x86 (32 bit) binaries, x32dbg are utilized; and for x64dbg, x64 (64-bit) binaries are utilized. [Figure 10.5](#) illustrates the division of x32dbg into four sections:

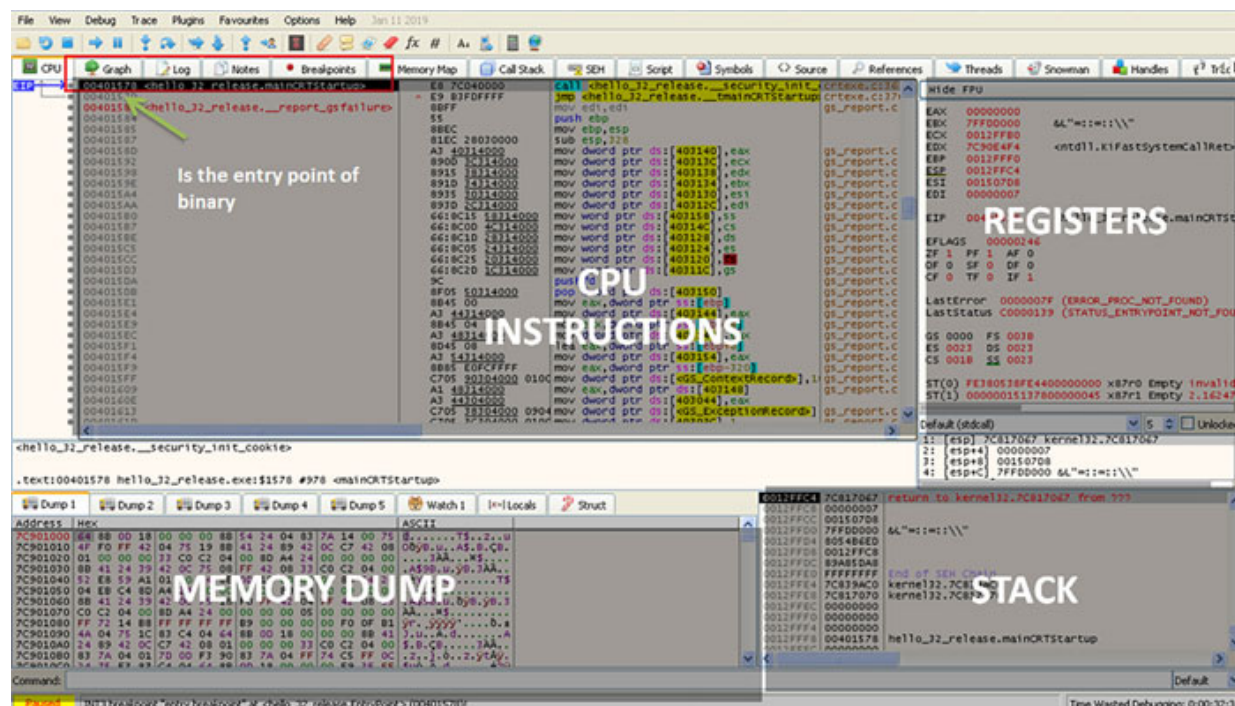


Figure 10.5: x32dbg

Monitors

Monitors play an important role in dynamically analyzing ransomware. With the help of these tools, we can examine the behavior of a process and its interaction with system resources like registry and read/writes on disk.

Process monitor

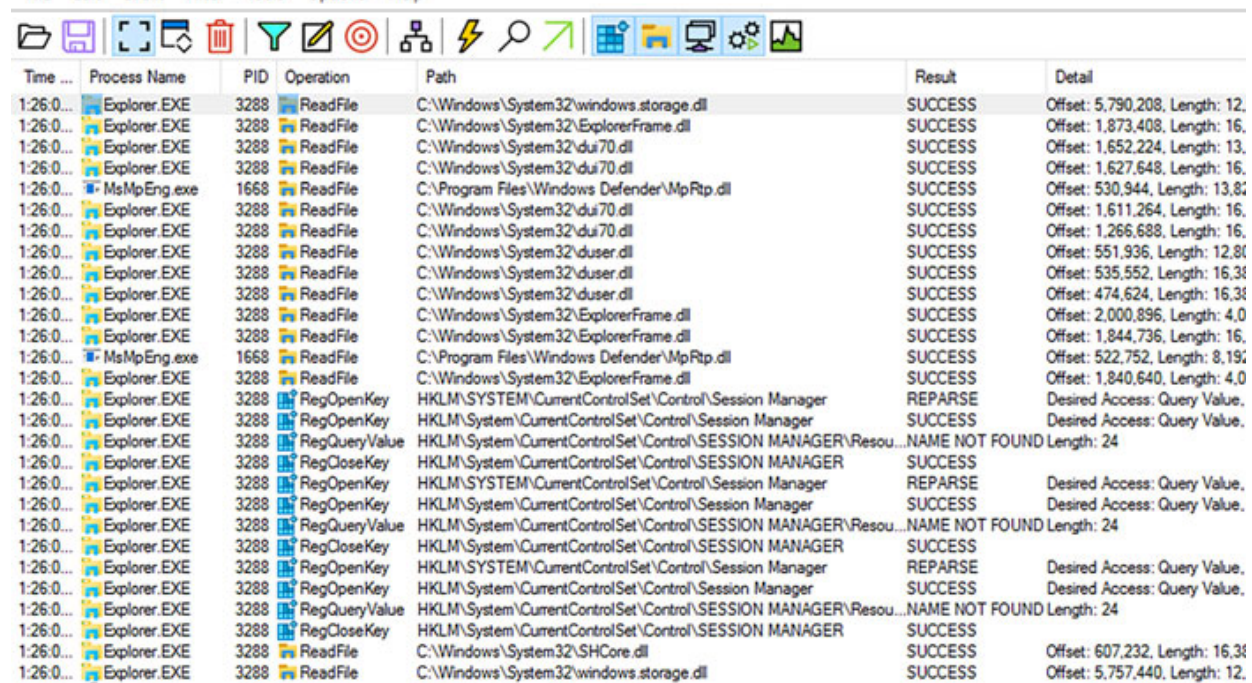
Process monitor is a sophisticated monitoring application for Windows that displays process and thread activity as well as file system, registry, and real-time activities. It offers reliable capture of process details in addition to strong monitoring and filtering features, boot-time logging of all activities,

and data for operation input and output. The process tree tool displays relationships between all processes referenced in a trace in addition to many other things.

[Figure 10.6](#) features a process monitor:

Process Monitor - Sysinternals: www.sysinternals.com

File Edit Event Filter Tools Options Help



Time ...	Process Name	PID	Operation	Path	Result	Detail
1:26:0...	Explorer.EXE	3288	ReadFile	C:\Windows\System32\windows.storage.dll	SUCCESS	Offset: 5,790,208, Length: 12,
1:26:0...	Explorer.EXE	3288	ReadFile	C:\Windows\System32\ExplorerFrame.dll	SUCCESS	Offset: 1,873,408, Length: 16,
1:26:0...	Explorer.EXE	3288	ReadFile	C:\Windows\System32\dui70.dll	SUCCESS	Offset: 1,652,224, Length: 13,
1:26:0...	Explorer.EXE	3288	ReadFile	C:\Windows\System32\dui70.dll	SUCCESS	Offset: 1,627,648, Length: 16,
1:26:0...	MsMpEng.exe	1668	ReadFile	C:\Program Files\Windows Defender\MpRtp.dll	SUCCESS	Offset: 530,944, Length: 13,8,
1:26:0...	Explorer.EXE	3288	ReadFile	C:\Windows\System32\dui70.dll	SUCCESS	Offset: 1,611,264, Length: 16,
1:26:0...	Explorer.EXE	3288	ReadFile	C:\Windows\System32\dui70.dll	SUCCESS	Offset: 1,266,688, Length: 16,
1:26:0...	Explorer.EXE	3288	ReadFile	C:\Windows\System32\duser.dll	SUCCESS	Offset: 551,936, Length: 12,8,
1:26:0...	Explorer.EXE	3288	ReadFile	C:\Windows\System32\duser.dll	SUCCESS	Offset: 535,552, Length: 16,3,
1:26:0...	Explorer.EXE	3288	ReadFile	C:\Windows\System32\duser.dll	SUCCESS	Offset: 474,624, Length: 16,3,
1:26:0...	Explorer.EXE	3288	ReadFile	C:\Windows\System32\ExplorerFrame.dll	SUCCESS	Offset: 2,000,896, Length: 4,0,
1:26:0...	Explorer.EXE	3288	ReadFile	C:\Windows\System32\ExplorerFrame.dll	SUCCESS	Offset: 1,844,736, Length: 16,
1:26:0...	MsMpEng.exe	1668	ReadFile	C:\Program Files\Windows Defender\MpRtp.dll	SUCCESS	Offset: 522,752, Length: 8,19,
1:26:0...	Explorer.EXE	3288	ReadFile	C:\Windows\System32\ExplorerFrame.dll	SUCCESS	Offset: 1,840,640, Length: 4,0,
1:26:0...	Explorer.EXE	3288	RegOpenKey	HKLM\SYSTEM\CurrentControlSet\Control\Session Manager	REPARSE	Desired Access: Query Value,
1:26:0...	Explorer.EXE	3288	RegOpenKey	HKLM\SYSTEM\CurrentControlSet\Control\Session Manager	SUCCESS	Desired Access: Query Value,
1:26:0...	Explorer.EXE	3288	RegQueryValue	HKLM\SYSTEM\CurrentControlSet\Control\SESSION MANAGER\Resou...	NAME NOT FOUND	Length: 24
1:26:0...	Explorer.EXE	3288	RegCloseKey	HKLM\SYSTEM\CurrentControlSet\Control\SESSION MANAGER	SUCCESS	
1:26:0...	Explorer.EXE	3288	RegOpenKey	HKLM\SYSTEM\CurrentControlSet\Control\Session Manager	REPARSE	Desired Access: Query Value,
1:26:0...	Explorer.EXE	3288	RegOpenKey	HKLM\SYSTEM\CurrentControlSet\Control\Session Manager	SUCCESS	Desired Access: Query Value,
1:26:0...	Explorer.EXE	3288	RegQueryValue	HKLM\SYSTEM\CurrentControlSet\Control\SESSION MANAGER\Resou...	NAME NOT FOUND	Length: 24
1:26:0...	Explorer.EXE	3288	RegCloseKey	HKLM\SYSTEM\CurrentControlSet\Control\SESSION MANAGER	SUCCESS	
1:26:0...	Explorer.EXE	3288	RegOpenKey	HKLM\SYSTEM\CurrentControlSet\Control\Session Manager	REPARSE	Desired Access: Query Value,
1:26:0...	Explorer.EXE	3288	RegOpenKey	HKLM\SYSTEM\CurrentControlSet\Control\Session Manager	SUCCESS	Desired Access: Query Value,
1:26:0...	Explorer.EXE	3288	RegQueryValue	HKLM\SYSTEM\CurrentControlSet\Control\SESSION MANAGER\Resou...	NAME NOT FOUND	Length: 24
1:26:0...	Explorer.EXE	3288	RegCloseKey	HKLM\SYSTEM\CurrentControlSet\Control\SESSION MANAGER	SUCCESS	
1:26:0...	Explorer.EXE	3288	ReadFile	C:\Windows\System32\SHCore.dll	SUCCESS	Offset: 607,232, Length: 16,3,
1:26:0...	Explorer.EXE	3288	ReadFile	C:\Windows\System32\windows.storage.dll	SUCCESS	Offset: 5,757,440, Length: 12,

Figure 10.6: Process Monitor

Procmon starts recording various Windows events as soon as you launch it. The Operation column in [Figure 10.6](#) shows various icons, each of which stands for a particular class of Windows events. Procmon records activities from five distinct classes:

- Registry
- Processes
- File system
- Network
- Profiling events

A single list pane with seven columns contains a representation of each event across all classes:

- **Time of day:** This tells the time event occurred.

- **Process name:** It lists the name of the process that triggered the event.
- **PID:** It stands for **Process Identifier**.
- **Operation:** It is the type of event, like changed a registry key value, process opened a file, or such.
- **Path:** It is the path to the object, like a file path or registry path.
- **Result:** This column indicates the result of the event, such as BUFFER OVERFLOW, REPARSE, and NAME NOT FOUND.
- **Detail:** This column contains all other details about an event.

You need to comprehend the idea of event filters to work on Procmon. To distinguish the signal from the noise, use event filters. All the events that you are not interested in are hidden via event filters. You can apply event filters to the events in the preceding example by using the full event class, or you can get more specific.

Process Explorer

Process Explorer is a free Windows task manager and system monitoring tool that shows which applications on a user's computer are currently viewing a certain file or location.

The tool is available from Microsoft for free download. Compared to Windows Task Manager, Process Explorer offers more detailed visual reporting. It is a component of the Sysinternals Process Utilities collection of tools, which includes several choices and controls for better Windows performance. [Figure 10.7](#) features the Process Explorer:

Process Explorer - Sysinternals: www.sysinternals.com [WIN10/enigma]

File Options View Process Find Users Help

Process	CPU	Private Bytes	Working Set	PID	Description	Company Name	Session	Proce
System Idle Process		0 K	4 K	0				
System	0.74	912 K	308,348 K	4				
smss.exe	2.04	0 K	0 K	n/a	Hardware Interrupts and DPCs		0	
csrss.exe		348 K	936 K	288				
wininit.exe		1,164 K	3,340 K	368				
services.exe		816 K	3,976 K	440				
svchost.exe		2,468 K	5,880 K	520				
svchost.exe	< 0.01	5,560 K	16,620 K	612	Host Process for Windows S...	Microsoft Corporation		
RuntimeBroker.exe		18,416 K	39,236 K	3324	Runtime Broker	Microsoft Corporation	1	
ShellExperienceHost.exe	Susp...	17,020 K	55,736 K	3724	Windows Shell Experience H...	Microsoft Corporation	1	
ApplicationFrameHost.exe		6,588 K	19,548 K	4552	Application Frame Host	Microsoft Corporation	1	
SearchUI.exe	Susp...	103,132 K	56,652 K	5936	Search and Cortana applicati...	Microsoft Corporation	1	
Microsoft.Photos.exe	Susp...	32,136 K	44,420 K	4049	Microsoft Photos		1	
explorer.exe		29,800 K	50,812 K	4852	Windows Explorer	Microsoft Corporation	1	
WmiPrvSE.exe		2,260 K	7,852 K	7024				
svchost.exe		4,312 K	9,448 K	660	Host Process for Windows S...	Microsoft Corporation		
svchost.exe		9,912 K	18,688 K	852	Host Process for Windows S...	Microsoft Corporation		
svchost.exe		15,364 K	23,132 K	860	Host Process for Windows S...	Microsoft Corporation		
audiodg.exe		6,588 K	10,624 K	5444			0	
svchost.exe	< 0.01	55,840 K	65,244 K	904	Host Process for Windows S...	Microsoft Corporation		
WUDFHost.exe		1,472 K	5,756 K	308				
dashHost.exe		4,640 K	13,084 K	1680				
svchost.exe		3,780 K	10,024 K	936	Host Process for Windows S...	Microsoft Corporation		
svchost.exe		27,272 K	50,476 K	988	Host Process for Windows S...	Microsoft Corporation		
shost.exe		5,468 K	25,572 K	2296	Shell Infrastructure Host	Microsoft Corporation	1	
taskhostw.exe		6,104 K	15,320 K	2224	Host Process for Windows T...	Microsoft Corporation	1	
taskeng.exe		1,256 K	5,868 K	5460				
sedlauncher.exe		5,240 K	540 K	1860				
conhost.exe		696 K	88 K	1816				
taskhostw.exe		4,828 K	13,060 K	5704			1	
svchost.exe		10,092 K	25,628 K	996	Host Process for Windows S...	Microsoft Corporation		
spoolsv.exe		5,016 K	10,952 K	1252	Spooler SubSystem App	Microsoft Corporation		
svchost.exe		12,652 K	21,692 K	1312	Host Process for Windows S...	Microsoft Corporation		
svchost.exe		1,352 K	5,632 K	1528	Host Process for Windows S...	Microsoft Corporation		

Figure 10.7: Process Explorer

Whenever you execute the software for the first time, a window resembling Windows Task Manager appears and lists all the processes that are currently active on your computer. Two sub-windows can be seen on the interface. In the top pane, information like the names of the owner accounts and currently active processes is presented.

There are two colorful panes of information in Process Explorer. The top window displays a list of all currently running processes, a description of what each process is doing, how much CPU and memory each process is using, and the name of the software provider.

Red is used to highlight the active, critical system programs, and the rest are depicted in blue. Green denotes the CPU, yellow stands for the system commits, and orange-red denotes the RAM or physical memory.

Autoruns

This tool, which has the most thorough understanding of auto-starting locations of any startup monitor, also displays the programs that are set to launch at system boot up or login when you launch other built-in Windows

programs, such as Internet Explorer, Explorer, and media players. The whole list of Registry and file system locations accessible for auto-start configuration is displayed by Autoruns, along with the apps currently setup for automatic startup. [Figure 10.8](#) features Autoruns:

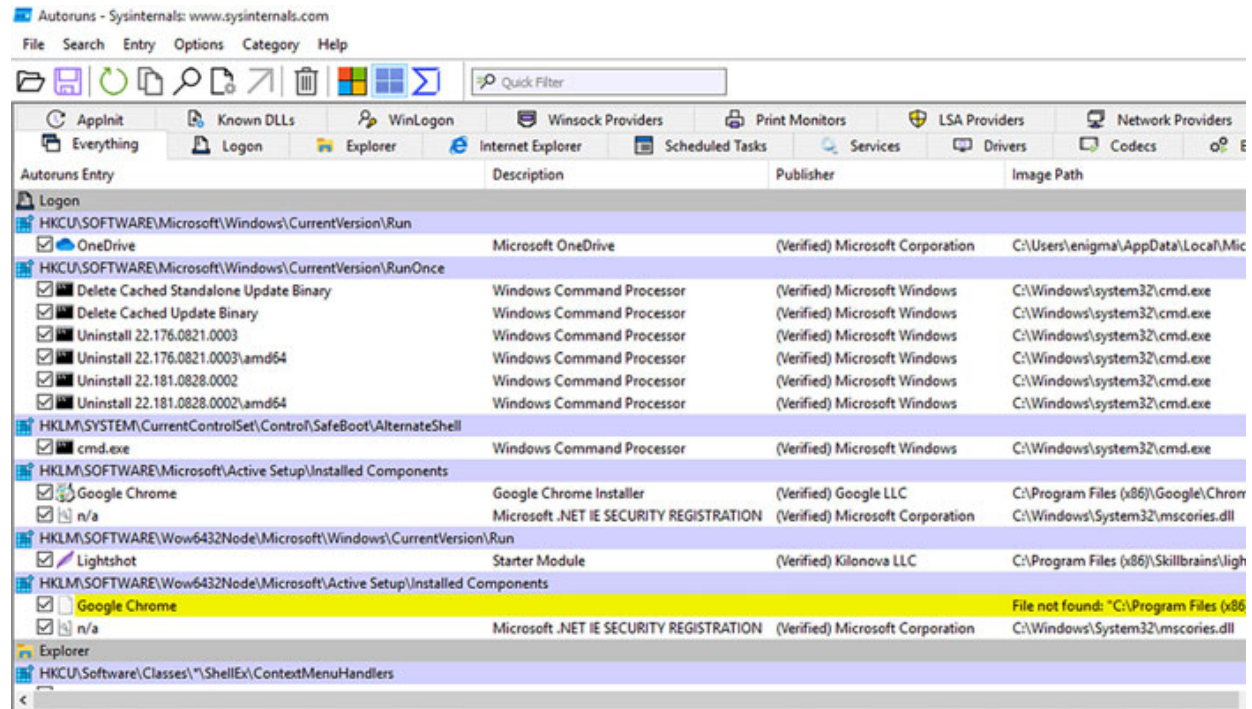


Figure 10.8: Autoruns

Dependency Walker

A free tool called Dependency Walker reads every 32-bit or 64-bit Windows module (such as an exe, dll, ocx, or sys) and creates a hierarchical tree diagram of every dependent module. It can assist you in resolving memory access violations, invalid page faults, and application errors, file registration errors, and file access errors. [Figure 10.9](#) features the Dependency Walker:

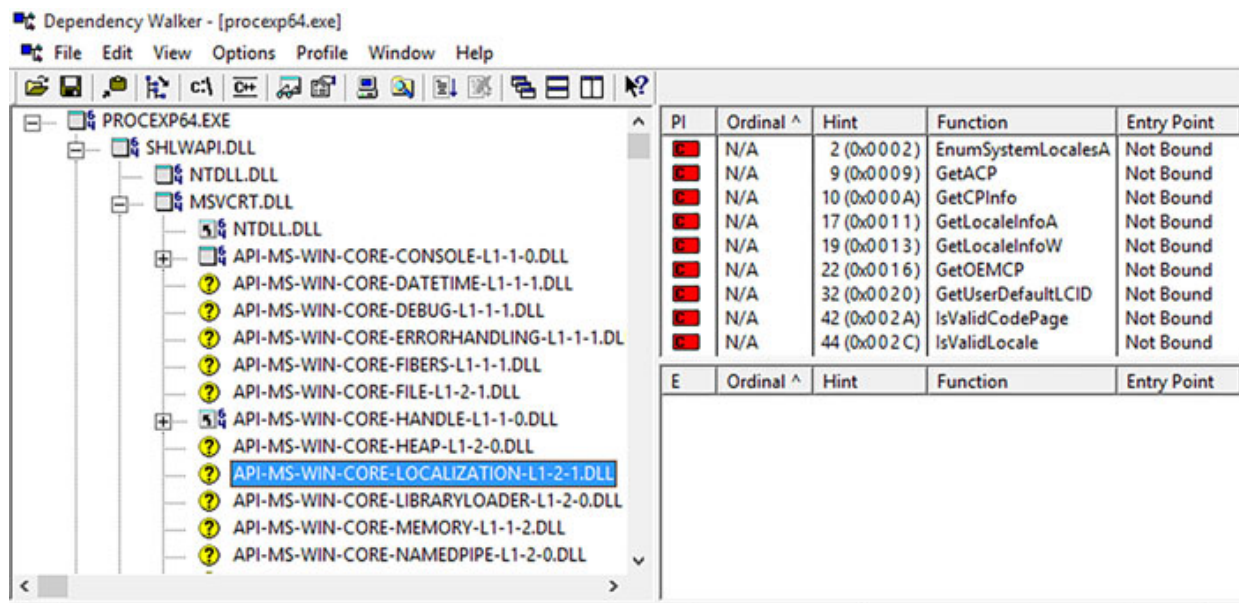


Figure 10.9: Dependency Walker

If a certain program refuses to load or a service fails to launch with an error referring to a specific dll, Dependency Walker can be especially useful. In these situations, you can load the program or dll in Dependency Walker to identify the problematic module or file and then fix it.

The dependency walker can be used to list the module's exported and imported functions. It can also be used to display the file dependencies, which minimizes the number of files needed. Plus, it can be used to display the data in these files, such as file path and version number. This program is free to use.

[Wireshark](#)

Wireshark is, without a doubt, the best tool for helping you analyze your network traffic. Using the deep inspection capabilities of Wireshark, you may analyze network protocols for free. You can use it to perform offline analysis or live packet capture. Numerous operating systems are supported, including Windows, Linux, MacOS, and FreeBSD. [Figure 10.10](#) features Wireshark:

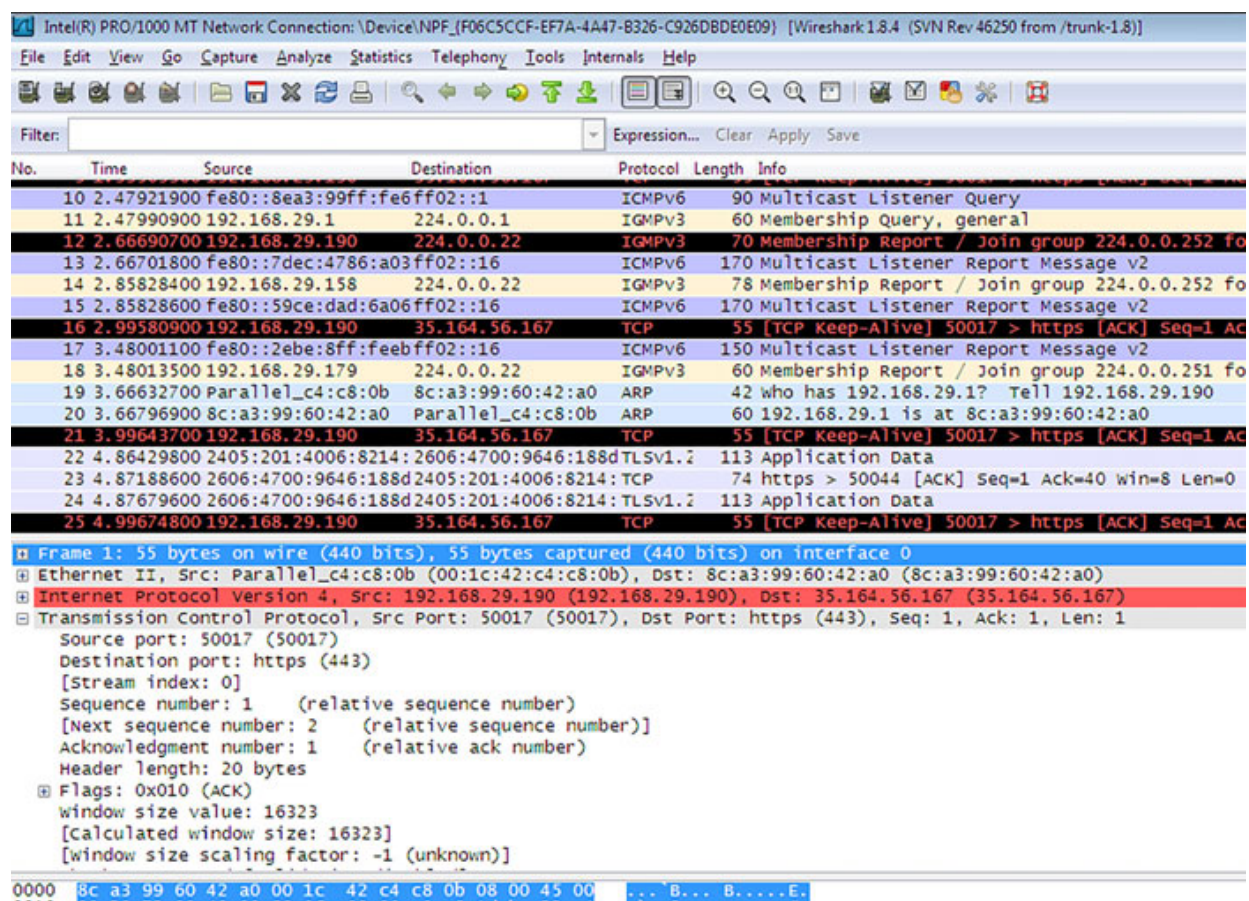


Figure 10.10: Wireshark

Wireshark is a well-known tool for analyzing network traffic and protocol. It can dump the captured traffic for subsequent offline analysis in addition to carrying out real-time capture and analysis. It can comprehend and parse a wide variety of network protocols because it integrates many dissectors and includes strong filters. Last but not least, if we provide the private key linked to the server certificate, it can decode SSL/TLS traffic. This is particularly helpful for analyzing malware traffic because it allows us to intercept and decrypt the traffic of the C&C server to examine how the virus communicates with it.

Burp Suite

One of the most well-liked penetration testing tools available today is Burp Suite Professional, which is especially useful if you want to utilize burp to intercept SSL traffic. When malware encrypts SSL traffic, this will be

useful. Different types of traffic, and more, can be captured with the help of Burp Suite. [Figure 10.11](#) features Burp Suite:

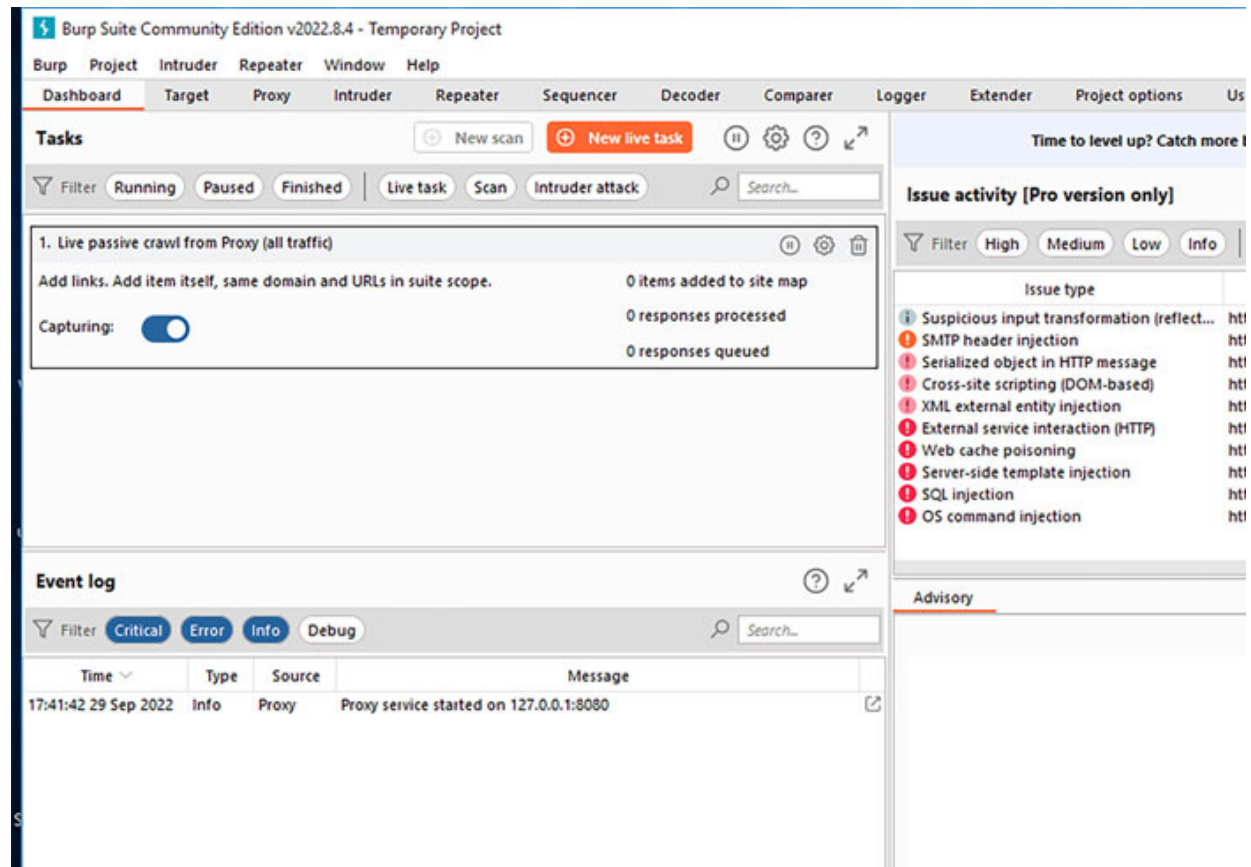


Figure 10.11: Burp Suite

Imagine a scenario where malware is communicating to the command-and-control server using HTTP or HTTPS. In this case, burp suite will help intercept the requests and further can be examined to find loophole in ransomware communication.

Conclusion

In this chapter, we covered dynamic analysis in depth. We walked through the different types of tools required for dynamic analysis. We covered the basic concepts of ransomware dynamic analysis and then looked at the basics of disassembler and different disassemblers. Moving further, we covered different debuggers used for dynamic analysis of ransomware. Toward the end, we identified different tools used to monitor the ransomware process.

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

<https://discord.bpbonline.com>



Section IV: Ransomware Forensics

CHAPTER 11

What's in the Memory

Introduction

It is crucial to protect your company from all sorts of ransomware assaults. For **Small and Medium Enterprises (SME)** firms, the average ransom price is \$50,000, and the sophistication and intelligence of cybercriminals are only increasing. You might wonder how these attacks continue to occur; one of the factors in these attacks' success is phishing. This emphasizes how crucial cyber education is for every firm. The secret to being secure is layering your cyber security strategy while putting it into practice. But what if a ransomware attack targets you or your company?

We discussed a few key Crypto API functions that are used to address the demand for various encryption kinds in various applications. You will be shocked to learn that malware authors leverage this encryption feature to create ransomware. To better understand this, we will look at a ransomware sample and reverse engineer it to see how a ransomware flaw might be exploited to defeat its goal.

Structure

In this chapter, we will discuss the following topics:

- Static analysis
 - Take a sample ransomware for analysis
 - Perform static analysis on the ransomware sample
- Dynamic analysis
 - Perform dynamic analysis using disassemblers and debuggers
 - Break the ransomware by extracting key from memory

Objectives

The objective of this chapter is to help you understand what was covered in the earlier chapters. In this chapter, we will take a ransomware sample to perform static and dynamic analysis using disassembler and debuggers. During static analysis, we will be using tools that we discussed in [Chapter 9, Performing Static Analysis](#), and identify points that can help us break the ransomware. After the static analysis, we will disassemble the ransomware to understand its code flow and logic. After identifying relevant functions in ransomware disassembled logic, we will use a debugger to set the breakpoint and understand its dynamic behavior. We will walk over the code in memory to understand different phases of ransomware infection, from infecting the host to generating key(s), encrypting user data and demanding ransom. It is during the key management phase that we will try to extract the key from the ransomware logic and then use that key to decrypt our locked data from ransomware.

Static analysis

Conducting a static analysis is the first step toward ransomware analysis. In our case, we will be taking a generic ransomware sample. Before performing ransomware analysis, be cautious about your testing machine. It should be done in a sandbox environment.

The details of the ransomware sample will be as follows:

File Type: PE32

Hashes:

md5: DE1C76810F1B55A169BEDD84033816FF

sha1: 84C0ABC186F556FF0779FE52763C67B1D3A8758A

sha256:

9A4F0F3240A84EFB9DF9A66284DA70079CBCE648C1E415D34B9D1491016B80FB

Now we will open the file in PEStudio for static analysis, as shown in [Figure 11.1](#):

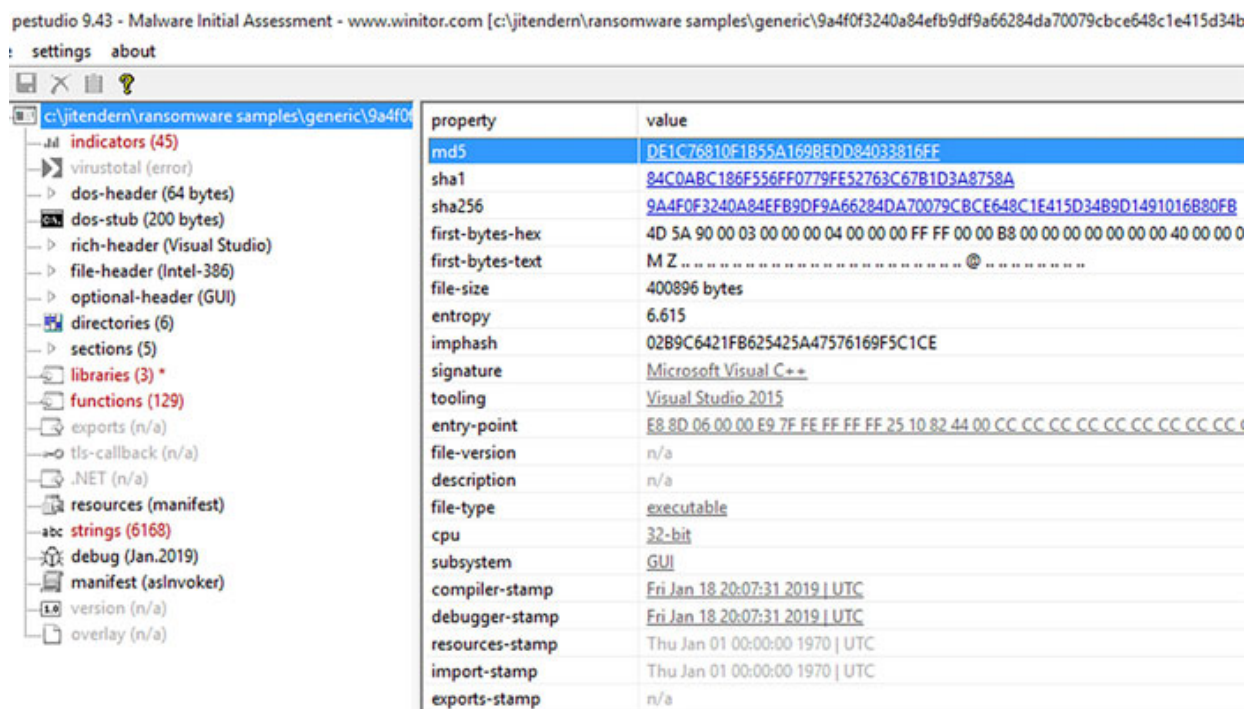


Figure 11.1: Generic Ransomware Sample

The **First-bytes-hex** value shows that the file is portable executable, with initial hex value of 0x4D5A equivalent to MZ in text form. For more details, refer to [Chapter 7, Portable Executable Insides](#).

We shall first comprehend the entropy analysis for the ransomware before moving on to the following value. Malware is created with the intention of being destructive; however, antivirus software is known to play a significant role in keeping our systems safe. Malware authors use obfuscation techniques to get around **antivirus (AV)** protection so that they will not be discovered. The original executable is packed into something else using various encryption methods, compression packages, and encoding conversions. It is challenging to reverse engineer malware when the executable is obfuscated.

Malware researchers can tell whether an executable or malware sample is encrypted or obfuscated using entropy analysis. The well-known Shannon's Formula is the foundation for entropy analysis. On a scale of 0 to 8, it calculates the executable's entropy. Lower entropy values reduce the likelihood of code obfuscation, while higher entropy increases the likelihood of executable obfuscation. Entropy of a typical executable is usually around 5, indicating that the binary is not compressed, encrypted, or disguised.

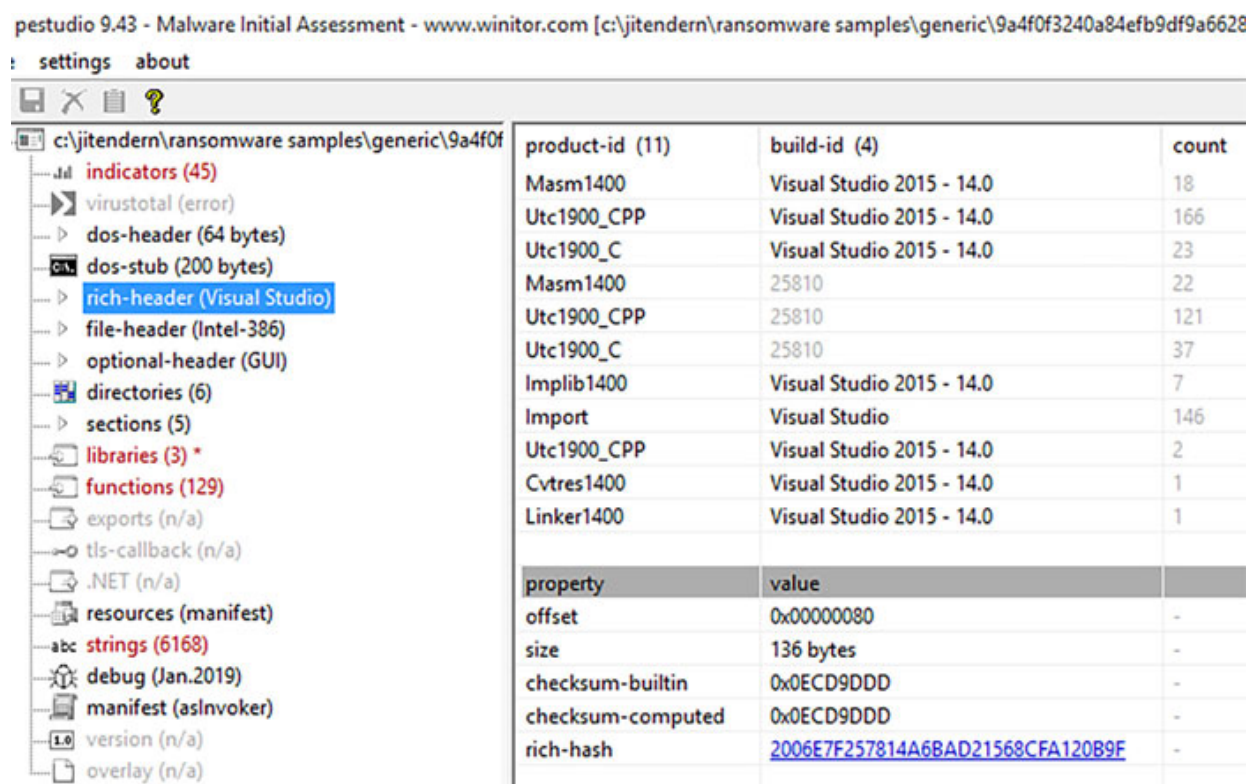
Ransomware sample in our case has an entropy value of 6.615, indicating that the binary is neither obscured or compressed nor encrypted.

We can also confirm, as shown in [Figure 11.1](#), that the sample is portable executable of 32 bit. Compiler-stamp shows that the executable is compiled on January 18, 2019. This ransomware came around the same time, but we should not heavily rely on this value as it can be modified.

If you remember, we studied the Rich signature in the structure of PE file. An undocumented feature of Microsoft linker wherein Rich signature is inserted in Microsoft linked executable, to identify back the machine and compiler on which binary is build. [Figure 11.2](#) illustrates the rich-header, which shows that the executable is compiled on Visual Studio.

pestudio 9.43 - Malware Initial Assessment - www.winitor.com [c:\jitendern\ransomware samples\generic\9a4f0f3240a84efb9df9a6628]

settings about

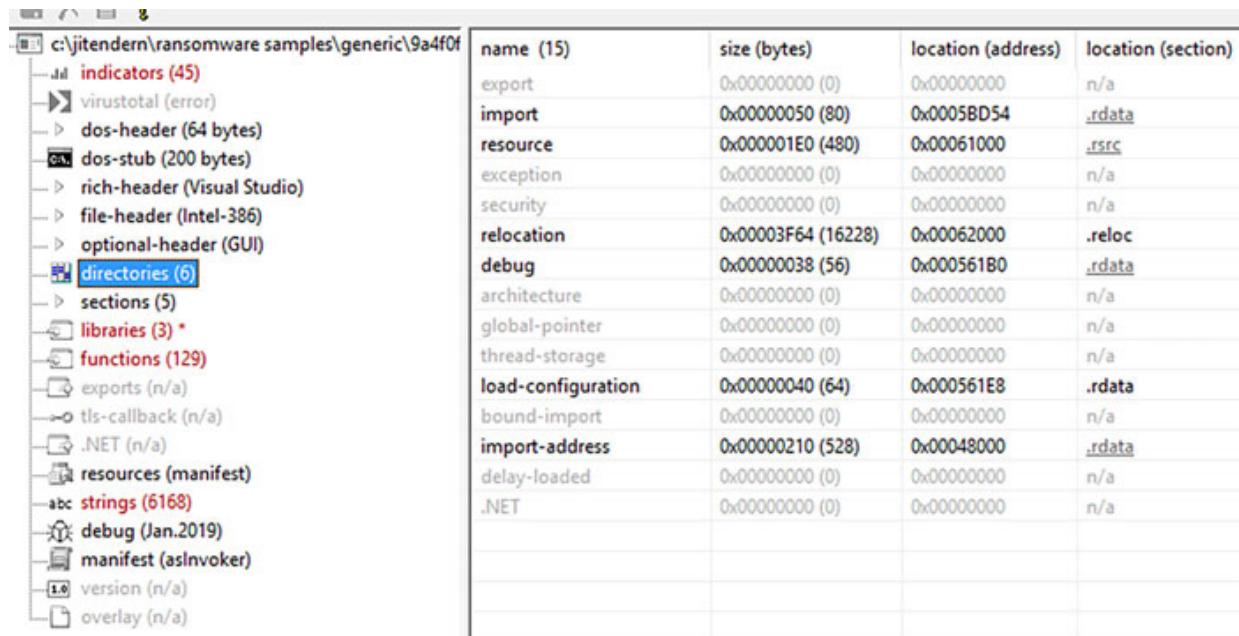


product-id (11)	build-id (4)	count
Masm1400	Visual Studio 2015 - 14.0	18
Utc1900_CPP	Visual Studio 2015 - 14.0	166
Utc1900_C	Visual Studio 2015 - 14.0	23
Masm1400	25810	22
Utc1900_CPP	25810	121
Utc1900_C	25810	37
Implib1400	Visual Studio 2015 - 14.0	7
Import	Visual Studio	146
Utc1900_CPP	Visual Studio 2015 - 14.0	2
Cvtres1400	Visual Studio 2015 - 14.0	1
Linker1400	Visual Studio 2015 - 14.0	1

property	value	
offset	0x00000080	-
size	136 bytes	-
checksum-builtin	0x0ECD9DDD	-
checksum-computed	0x0ECD9DDD	-
rich-hash	2006E7F257814A6BAD21568CFA120B9F	-

Figure 11.2: Generic Ransomware Rich Header

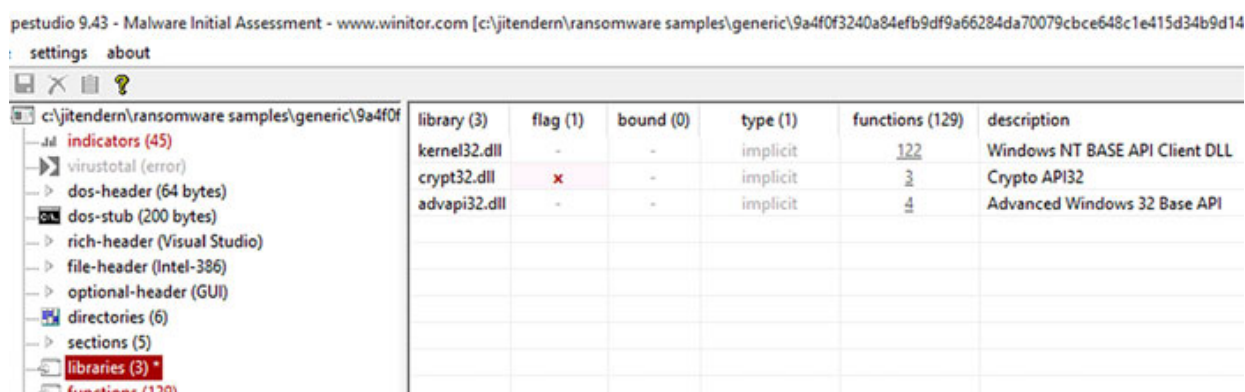
As shown in [Figure 11.3](#), the main directories of our interest are import and resource:



name (15)	size (bytes)	location (address)	location (section)
export	0x00000000 (0)	0x00000000	n/a
import	0x00000050 (80)	0x00058D54	.rdata
resource	0x000001E0 (480)	0x00061000	.rsrc
exception	0x00000000 (0)	0x00000000	n/a
security	0x00000000 (0)	0x00000000	n/a
relocation	0x00003F64 (16228)	0x00062000	.reloc
debug	0x00000038 (56)	0x000561B0	.rdata
architecture	0x00000000 (0)	0x00000000	n/a
global-pointer	0x00000000 (0)	0x00000000	n/a
thread-storage	0x00000000 (0)	0x00000000	n/a
load-configuration	0x00000040 (64)	0x000561E8	.rdata
bound-import	0x00000000 (0)	0x00000000	n/a
import-address	0x00000210 (528)	0x00048000	.rdata
delay-loaded	0x00000000 (0)	0x00000000	n/a
.NET	0x00000000 (0)	0x00000000	n/a

Figure 11.3: Ransomware directories

[Figure 11.4](#) lists imported libraries in our sample. The main libraries of our interest are **crypt32** and **advapi32**, which confirm the use of crypto function in the development of ransomware. We will talk about them in the later sections.



library (3)	flag (1)	bound (0)	type (1)	functions (129)	description
kernel32.dll	-	-	implicit	122	Windows NT BASE API Client DLL
crypt32.dll	x	-	implicit	3	Crypto API32
advapi32.dll	-	-	implicit	4	Advanced Windows 32 Base API

Figure 11.4: Generic Ransomware Libraries

The main imported library to notice is the **advapi32.dll**. This is the library used to add encryption capabilities into an application. [Figure 11.5](#) lists the imported functions used for cryptographic functionalities:

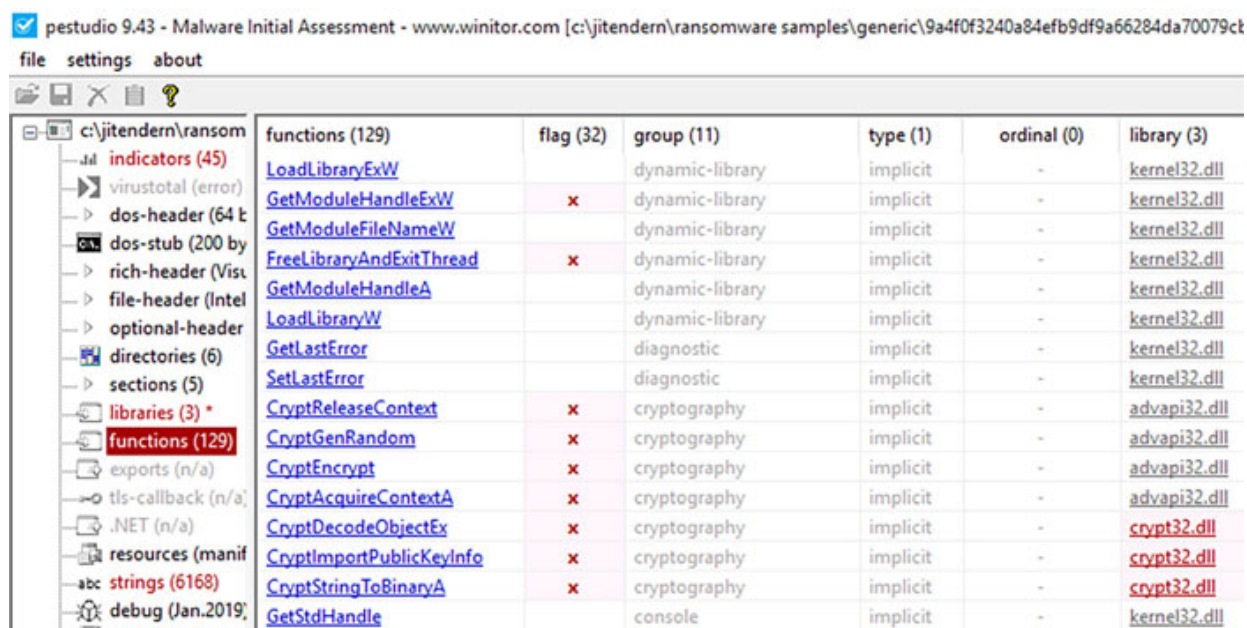


Figure 11.5: Import Functions

We will analyze these functions in our dynamic analysis and see what is running in the memory. Lastly, in our static analysis, we see some interesting clear text strings in our executable, as shown in [Figure 11.6](#):

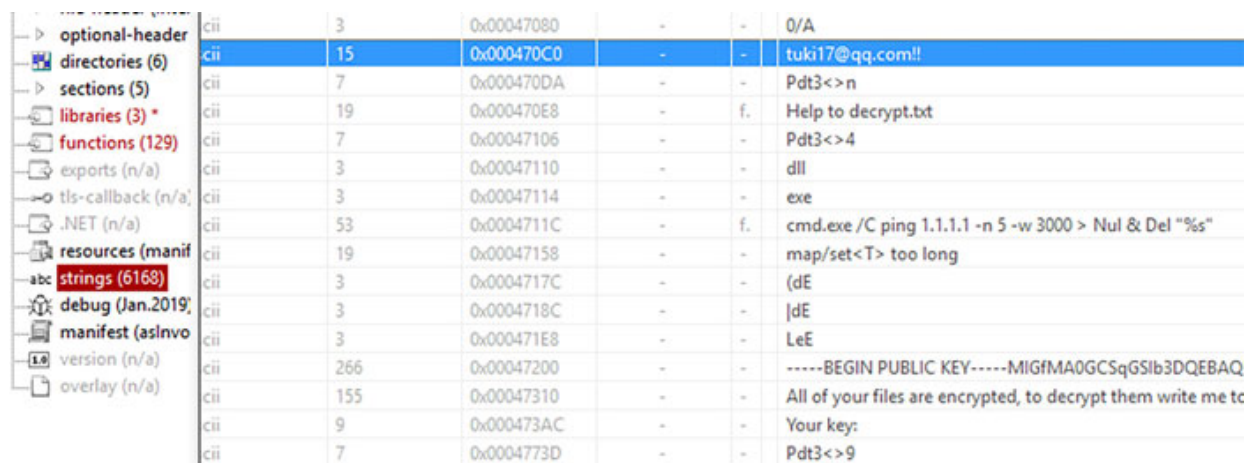


Figure 11.6: Strings

In string analysis, we see some Windows binaries being called and some command being executed. There is one command with string starting `cmd.exe /C ping`; it pings the public DNS 1.1.1.1.

The public DNS resolver 1.1.1.1 is run by Cloudflare and provides a quick and private way to access the Internet. Most DNS resolvers sell user

information to advertising, but 1.1.1.1 does not. Furthermore, 1.1.1.1 has been found to be the fastest DNS resolver currently in use.

Dynamic analysis– Check in the memory

Now, after static analysis, we will move on to see how ransomware behaves in memory. We have learned in an earlier chapter that if a ransomware is using Crypto API, then all that happens while running can be captured in the memory. We will be using Ghidra for performing dynamic analysis.

We have previously learned that Ghidra was programmed by the National Security Agency, and it is used for reverse engineering and is free. Researchers who study malware and reverse engineers use this tool to examine malware and identify app vulnerabilities. With the help of Ghidra, we will disassemble the ransomware binary and understand the code logic. Once we get an understanding of the code flow, we will set the breakpoint on the relevant functions to extract flaws in the ransomware logic.

The first step is to install Ghidra on your machine, and then open it and create a new project by navigating to **File | New Project**.

Select **Non-Shared Project** and then click on **Next>>**, as shown in [Figure 11.7](#):

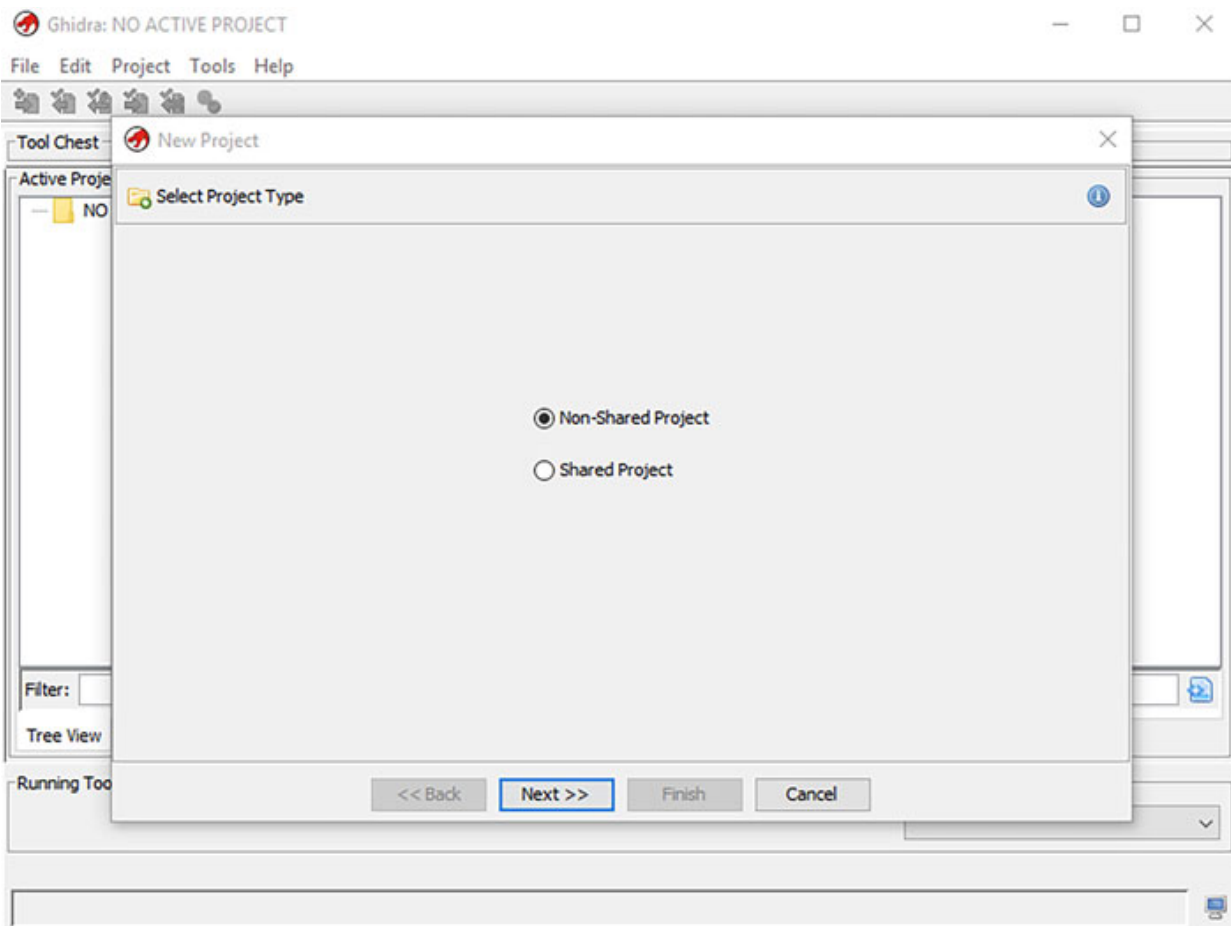


Figure 11.7: Ghidra new project

In the new dialog box that opens, enter **Project Directory** and **Project Name** as shown in [Figure 11.8](#); we named the project **RansomwareGeneric**. Then, click on **Finish** to continue to the next step.

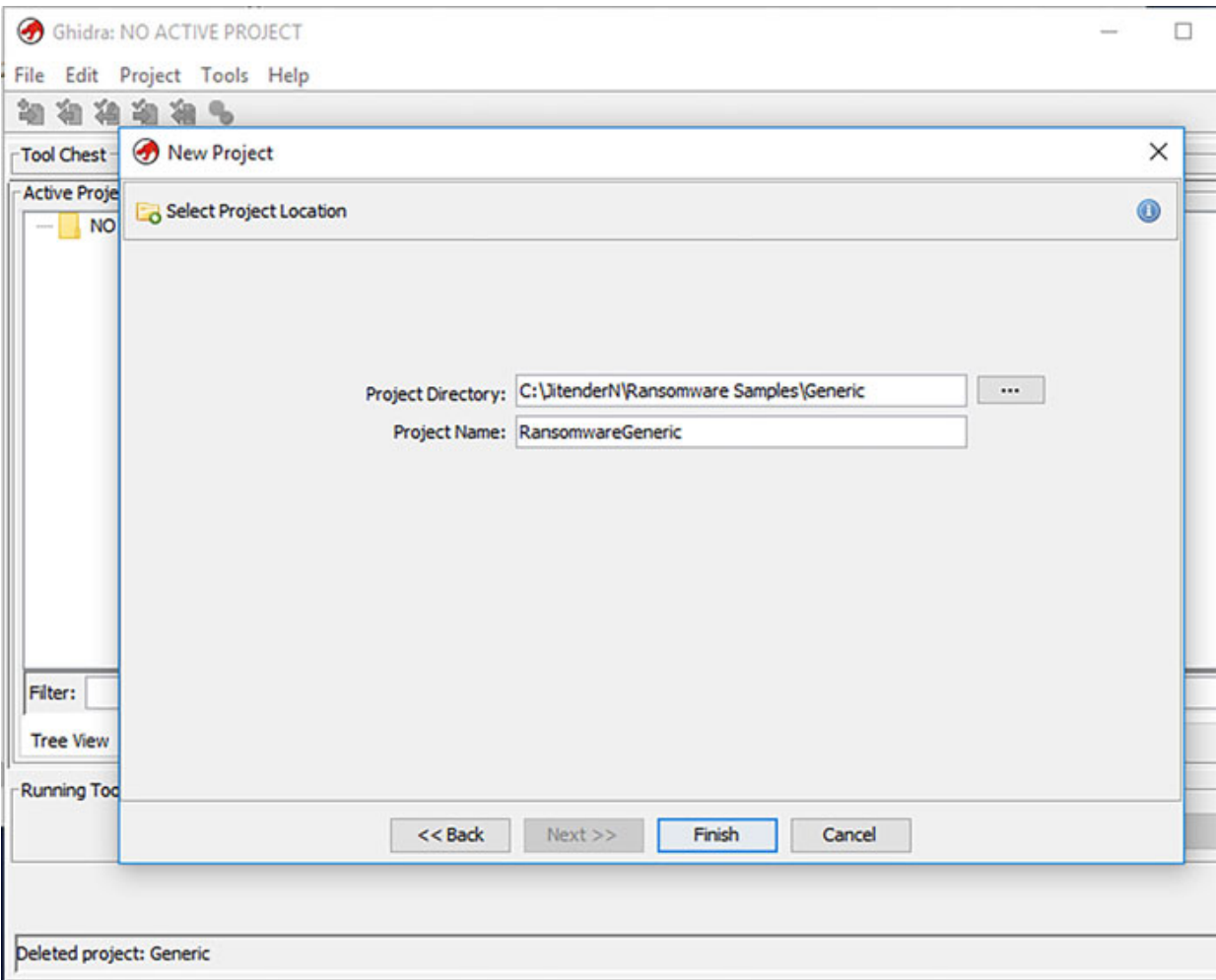


Figure 11.8: Ghidra New project

Now, drag and drop the ransomware sample in the *Active Project Window*. This will import the Ransomware PE sample. Once the import is done, *Import Result Summary* will pop up with binary format, Compiler ID, Endianness, Processor and many more, as shown in [Figure 11.9](#):

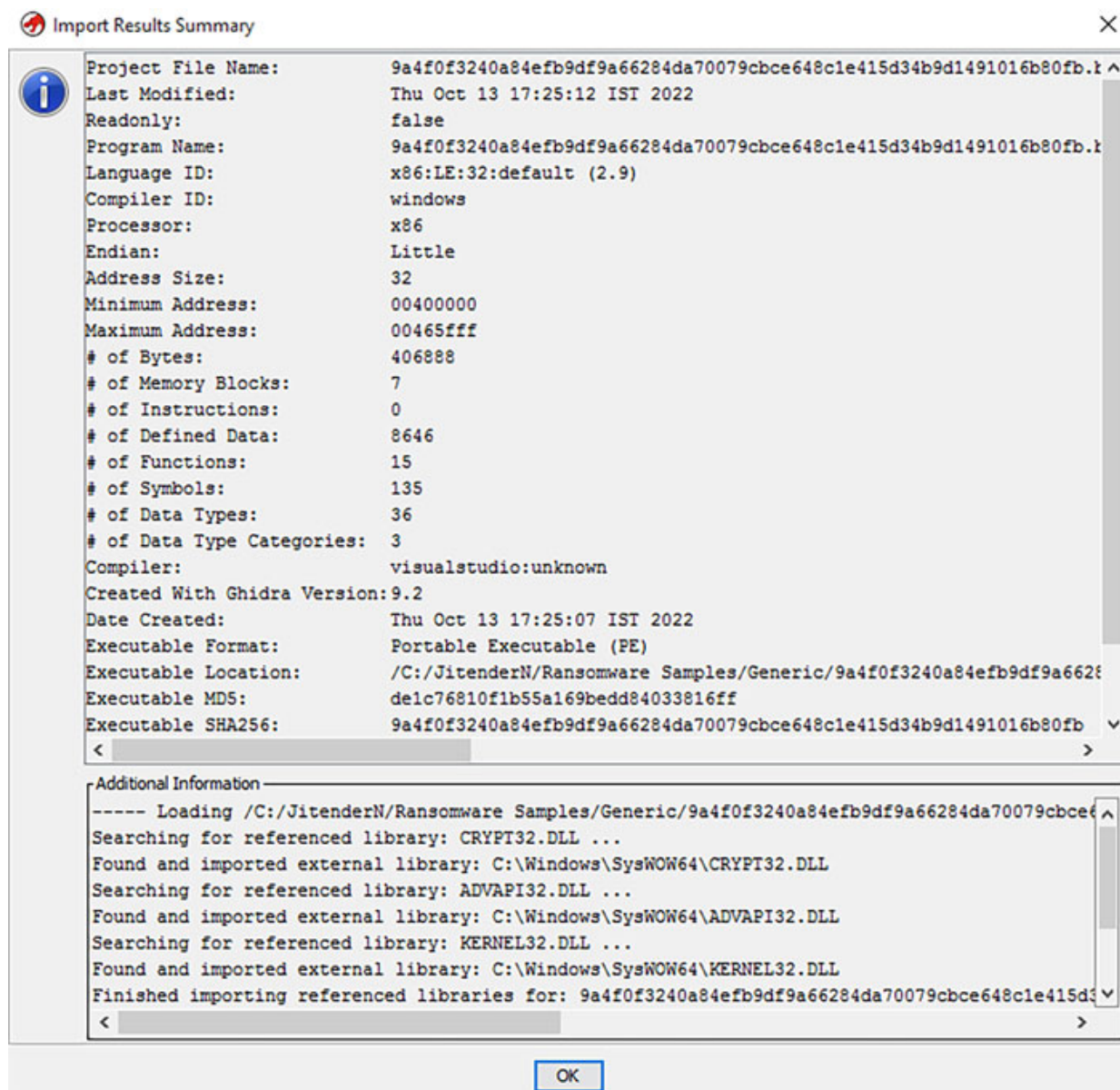


Figure 11.9: Ransomware Import Summary

Click on **OK** to return to the *Active Window* screen, and double-click on the imported ransomware sample file to open *CodeBrowser*. On opening it, you will be prompted to analyze the sample. Click on **Yes** to analyze the sample. You will be prompted to select the *Analysis Options*. Keeping the default options checked, enable a few more analysis options, such as **WindowsPE x86 Propagate External Parameters** and **Decompiler Parameter ID**, as shown in [Figure 11.10](#):

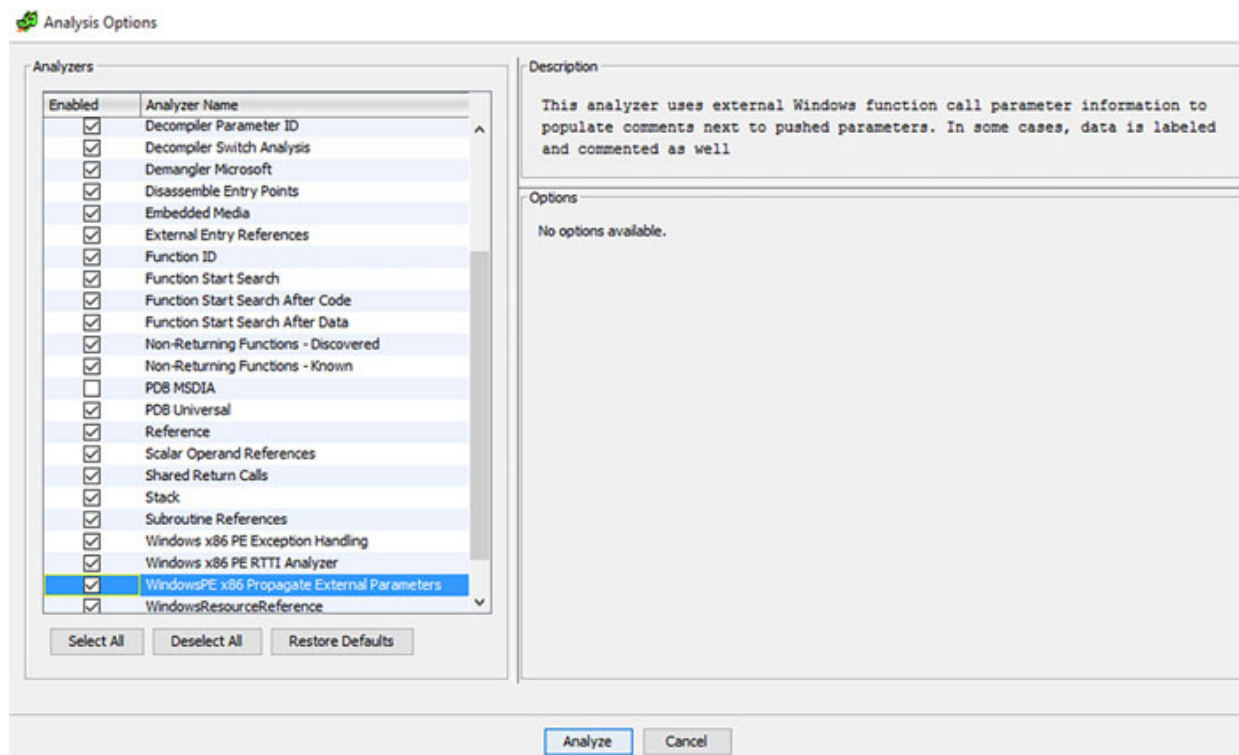


Figure 11.10: Enable Analysis options

Click on **Analyze** and ignore any warning message. This will load the ransomware sample for further analyses. On the top right, PE sections are displayed in the *Program Trees* window. Symbols are displayed in the *Symbol Tree* window. A disassembled view of the ransomware sample is shown in the middle window. To check the entry point of this ransomware, type *entry* in the *functions* window **Filter** option. Double-click on the entry function to move to the entry point of ransomware disassembled code; this is like the `main()` or `WinMain()` function. Refer to [Figure 11.11](#):

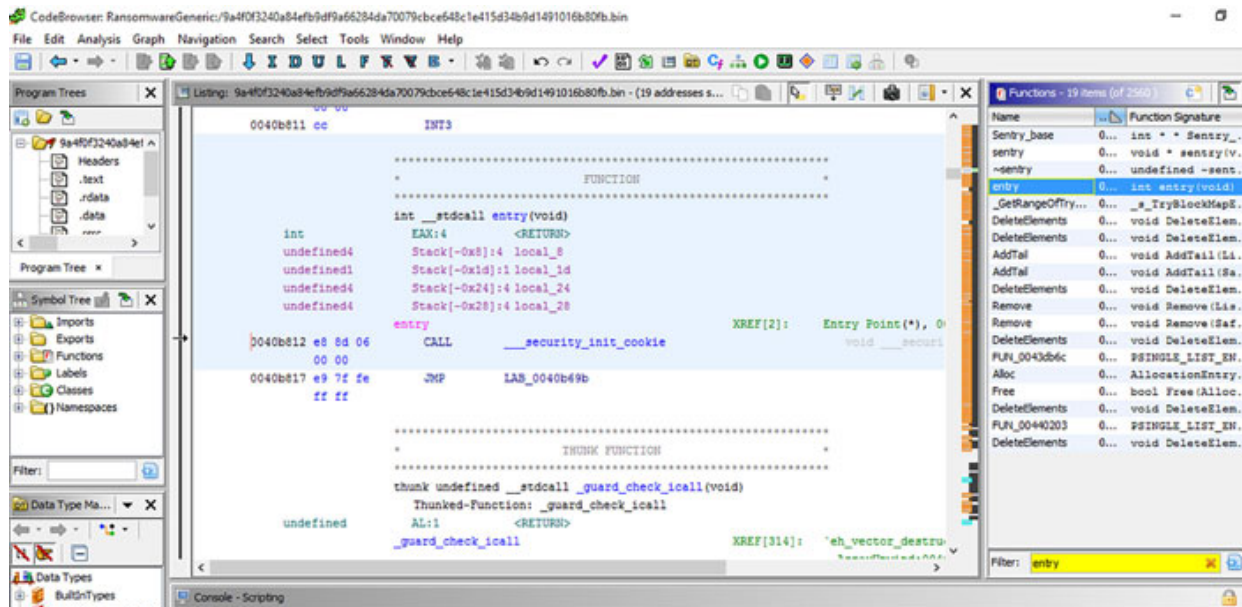


Figure 11.11: Ransomware Entry Point

Now we will drill down the ransomware code flow from the entry point and find out the list of functions called from the entry point. To get the list of functions called from the entry point, go to **Windows | Function** call tree. In the filter section, search for Crypto API functions. We will see the **FUN_0040af3d** function called from the entry point. This is then calling **FUN_004082b5** and subsequently, Crypto API. On further expanding on the list of functions, we see Windows cryptographic function calls like **CryptAcquireContextA**, **CryptGenRandom**, **CryptReleaseContext**, and **CryptEncrypt**, as shown in [Figure 11.12](#):

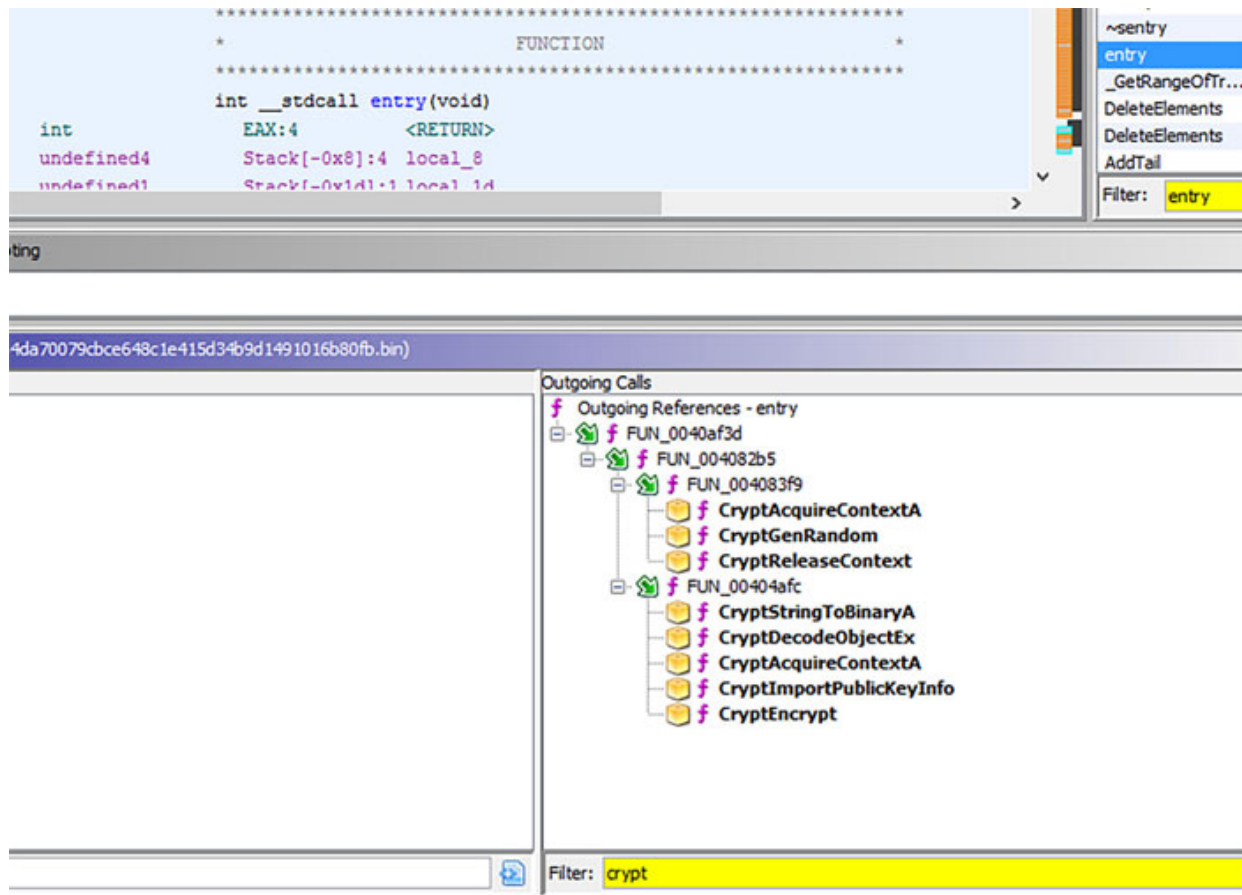


Figure 11.12: Crypto Functions Calls

As we read in [Chapter 6, Internal Secrets of Ransomware](#), the first step to implement encryption is to use Crypto API in the application, which is implemented by opening a handle to appropriate CSP. This handle is opened using the **CryptAcquireContextA** function.

As we can see in the ransomware assembly listing, the **CryptAcquireContextA** function is called in within the **FUN_004083f9** function. The **CryptAcquireContextA** function is followed by the **CryptGenRandom** function, which is normally used to generate some seed value required to make the ransomware key unique every time the ransomware is run. This is achieved using the **CryptGenRandom** function from Crypto API, as shown in [Figure 11.13](#) of Ghidra assembly listing:

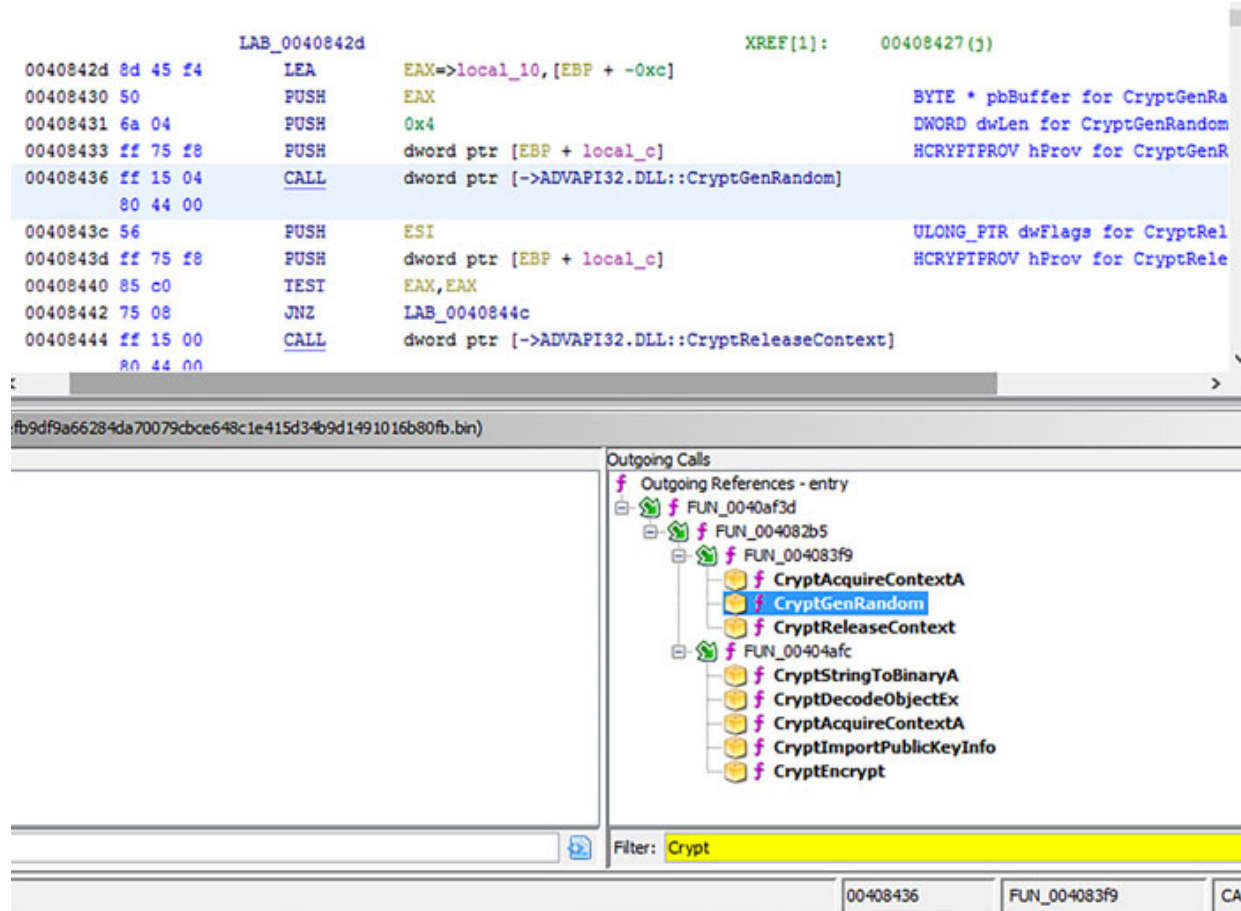


Figure 11.13: CryptGenRandom

If we note, the parameters preceding the **CryptGenRandom** function are pushed from right to left as per the function definition:

```
BOOL CRYPTFUNC CryptGenRandom(HCRYPTPROV hProv, DWORD dwLen, BYTE* pbBuffer);
```

hProv: This is the last parameter pushed before the call to the **CryptImportKey** function. This is the handle to CSP created with the **CryptAcquireContext** function before the **CryptGenRandom** function.

dwLen: This is the length(in bytes) of the random data to generate. [Figure 11.13](#) shows the ransomware generating 4 bytes of random data by pushing 0x04 bytes at memory location 0x00408431.

pbBuffer: This is the pointer to the buffer containing the random data. This buffer must be at least dwLenbytes in length.

Return Value: Success is indicated by TRUE, and failure is indicated by FALSE.

We have now seen how ransomware is generating random seed using **CryptGenRandom**. When we trace back on the function calling the **CryptGenRandom** function, we can see that **CryptAcquireContextA** is called before the **CryptGenRandom** function, as shown in [Figure 11.14](#):

01.	004083F9	55	push ebp	
02.	004083FA	8B	mov ebp,esp	
03.	004083FC	83	sub esp,C	
04.	004083FF	A1	mov eax,dword ptr ds:[45D010]	eax:&"ALLUSERSPROFILE=C:\\ProgramData"
05.	00408404	33	xor eax,ebp	
06.	00408406	89	mov dword ptr ss:[ebp-4],eax	
07.	00408409	56	push esi	
08.	0040840A	57	push edi	
09.	0040840B	68	push F0000040	DWORD dwFlags = CRYPT_VERIFYCONTEXT CRYPT_SILENT
10.	00408410	33	xor edi,edi	
11.	00408412	8D	lea eax,dword ptr ss:[ebp-8]	
12.	00408415	33	xor esi,esi	
13.	00408417	47	inc edi	
14.	00408418	57	push edi	DWORD dwProvType
15.	00408419	56	push esi	LPCTSTR pszProvider
16.	0040841A	56	push esi	LPCTSTR pszContainer
17.	0040841B	50	push eax	HCRYPTPROV* phProv
18.	0040841C	89	mov dword ptr ss:[ebp-8],esi	
19.	0040841F	FF	call dword ptr ds:[<&CryptAcquireCo	CryptAcquireContextA
20.	00408425	85	test eax,eax	eax:&"ALLUSERSPROFILE=C:\\ProgramData"
21.	00408427	75	jne 9a4f0f3240a84efb9df9a66284da700	
22.	00408429	8B	mov eax,edi	eax:&"ALLUSERSPROFILE=C:\\ProgramData"
23.	0040842B	EB	jmp 9a4f0f3240a84efb9df9a66284da700	
24.	0040842D	8D	lea eax,dword ptr ss:[ebp-C]	[ebp-C]:sub_411660
25.	00408430	50	push eax	BYTE* pBuffer
26.	00408431	6A	push 4	DWORD dwLen = 4
27.	00408433	FF	push dword ptr ss:[ebp-8]	HCRYPTPROV hProv
28.	00408436	FF	call dword ptr ds:[<&CryptGenRandom	CryptGenRandom
29.	0040843C	56	push esi	
30.	0040843D	FF	push dword ptr ss:[ebp-8]	
31.	00408440	85	test eax,eax	eax:&"ALLUSERSPROFILE=C:\\ProgramData"
32.	00408442	75	jne 9a4f0f3240a84efb9df9a66284da700	
33.	00408444	FF	call dword ptr ds:[<&CryptReleaseCo	
34.	0040844A	EB	jmp 9a4f0f3240a84efb9df9a66284da700	
35.	0040844C	FF	call dword ptr ds:[<&CryptReleaseCo	
36.	00408452	8B	mov ecx,dword ptr ss:[ebp-C]	[ebp-C]:sub_411660
37.	00408455	85	test eax,eax	eax:&"ALLUSERSPROFILE=C:\\ProgramData"
38.	00408457	0F	cmove ecx,edi	
39.	0040845A	8B	mov eax,ecx	eax:&"ALLUSERSPROFILE=C:\\ProgramData"
40.	0040845C	8B	mov ecx,dword ptr ss:[ebp-4]	
41.	0040845F	5F	pop edi	
42.	00408460	33	xor ecx,ebp	
43.	00408462	5E	pop esi	
44.	00408463	E8	call <9a4f0f3240a84efb9df9a66284da7	
45.	00408468	8B	mov esp,ebp	
46.	0040846A	5D	pop ebp	
47.	0040846B	C3	ret	

Figure 11.14: Function Calling CryptGenRandom

As we observed in [Figure 11.14](#), both the Crypto functions are called from **Func_4083F9**. If we check the references of **Func_4083F9**, we will observe that it is called within a function (**Func_4082B5**), and the **Func_4083F9**

function is called in a loop by observing JNZ instruction at 0x0040830C, as shown in [Figure 11.15](#):

004082e8	PUSH	0x10
004082ea	POP	EBX
004082eb	MOV	param_1,EDI
004082ed	CALL	FUN_004083f9
004082f2	PUSH	0x19
004082f4	POP	param_1
004082f5	XOR	EDX,EDX
004082f7	DIV	param_1
004082f9	MOV	param_1,ESI
004082fb	ADD	DL,0x61
004082fe	MOV	byte ptr [EBP + local_14],DL
00408301	PUSH	dword ptr [EBP + local_14]
00408304	CALL	FUN_0040a561
00408309	SUB	EBX,0x1
0040830c	JNZ	LAB_004082eb

Figure 11.15: Sub_4083F9 Reference

In the last two lines in [Figure 11.15](#), we see that the ransomware is subtracting **EBX** by 1 and that it is checking the jump condition for the loop to run. Every time the loop is executed, **EBX** is decremented by 1. Now the question is, ‘What is **EBX** set to before the call to **sub_4083F9**?’

EBX is set to 16, as shown in [Figure 11.15](#), at the **0x004082E8** location by pushing 0x10 and popping it to **EBX**. So, this loop runs 16 times (in decimal), and every time it runs, it generates 4 random bytes; these random bytes further give ASCII value. This 16-time loop will generate 16 bytes or a 16 char key, which we will talk about in the later sections. All these instructions are called within **Func_4082B5**, as shown in [Figure 11.16](#):

undefined __fastcall FUN_004082b5(void * param_1)		
004082b5	PUSH	0x34
004082b7	MOV	EAX, LAB_0044668e
004082bc	CALL	__EH_prolog3
004082c1	MOV	EDI, param_1
004082c3	PUSH	0x0
004082c5	CALL	FID_conflict: __time64
004082ca	PUSH	EAX
004082cb	CALL	FUN_0041618a
004082d0	POP	param_1
004082d1	POP	param_1
004082d2	LEA	ESI, [EDI + 0xc0]
004082d8	PUSH	DAT_004483a4
004082dd	MOV	param_1, ESI
004082df	CALL	FUN_004057a0
004082e4	TEST	AL, AL
004082e6	JZ	LAB_0040830e
004082e8	PUSH	0x10
004082ea	POP	EBX
004082eb	MOV	param_1, EDI
004082ed	CALL	FUN_004083f9
004082f2	PUSH	0x19
004082f4	POP	param_1
004082f5	XOR	EDX, EDX
004082f7	DIV	param_1
004082f9	MOV	param_1, ESI
004082fb	ADD	DL, 0x61
004082fe	MOV	byte ptr [EBP + local_14], DL
00408301	PUSH	dword ptr [EBP + local_14]
00408304	CALL	FUN_0040a561
00408309	SUB	EBX, 0x1
0040830c	JNZ	LAB_004082eb

Figure 11.16: Function Calling Sub_4083F9

There are a couple more Crypto functions that we will be analyzing in our dynamic analysis. One of them is **CryptEncrypt**; as per the Microsoft documentation, data is encrypted using this function. The key held by the

CSP module, which is referred to by the **hKey** parameter, determines the algorithm used to encrypt the data (in our ransomware sample, it might be the encrypting key we generated using the **Func_4082B5** function). We will figure out what kind of data it is encrypting in the ransomware code. We will take this in our dynamic analysis part by setting breakpoint on the **CryptEncrypt** function at **0x404CC3**.

Refer to [Figure 11.17](#):

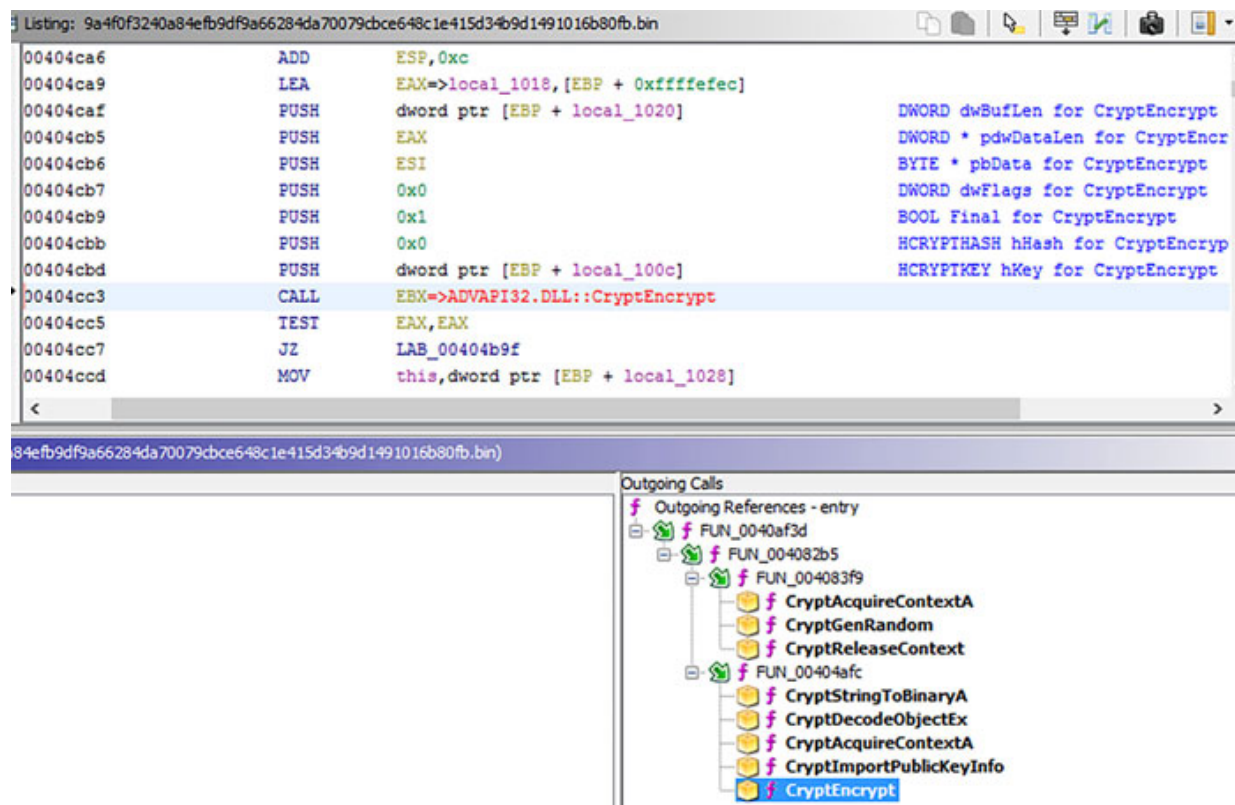


Figure 11.17: CryptEncrypt

Now, in order to find the encryption used for encrypting user data, we will use a tool called PEiD. For PE files, PEiD can identify the most popular packers, cryptors, and compilers. PEiD detects 470 or more distinct signatures, and it can be used to detect encryption algorithm in our ransomware PE file.

Opening our ransomware sample in PEiD and using Krypto Analyzer in PEiD plugins result in showing the encryption algorithm used in the ransomware sample, as shown in [Figure 11.18](#):

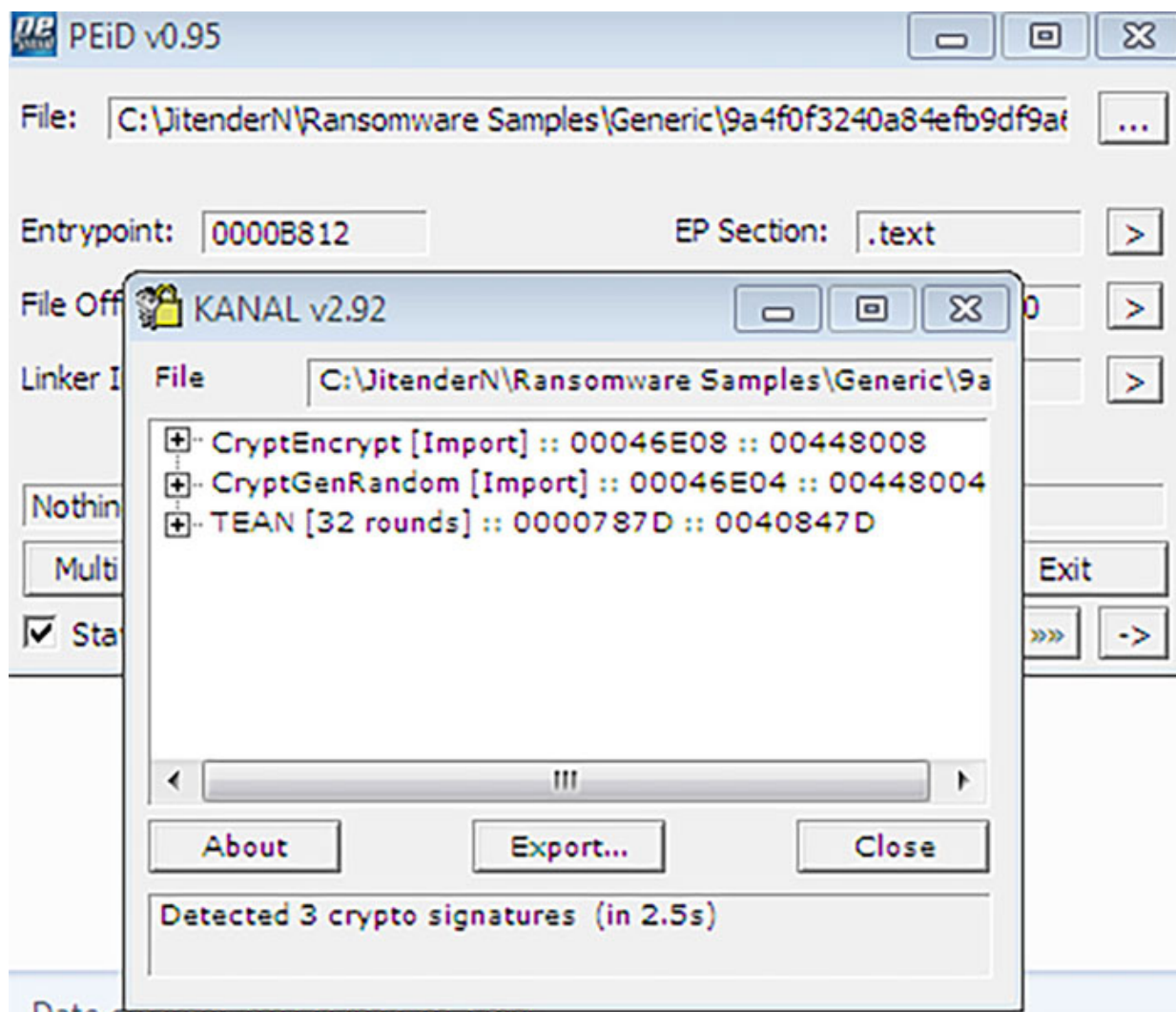


Figure 11.18: PEiD

PEiD shows that the ransomware is using the TEAN encryption algorithm to encrypt data.

Now, let us see what this ransomware is doing in the memory by running it in x32dbg and control environment. Before we start analyzing ransomware in a sandbox environment, create a sample file containing some important information in the **Documents** folder, as ransomware are programmed to encrypt data in the **Document** folder files by default. This is shown in [Figure 11.19](#):

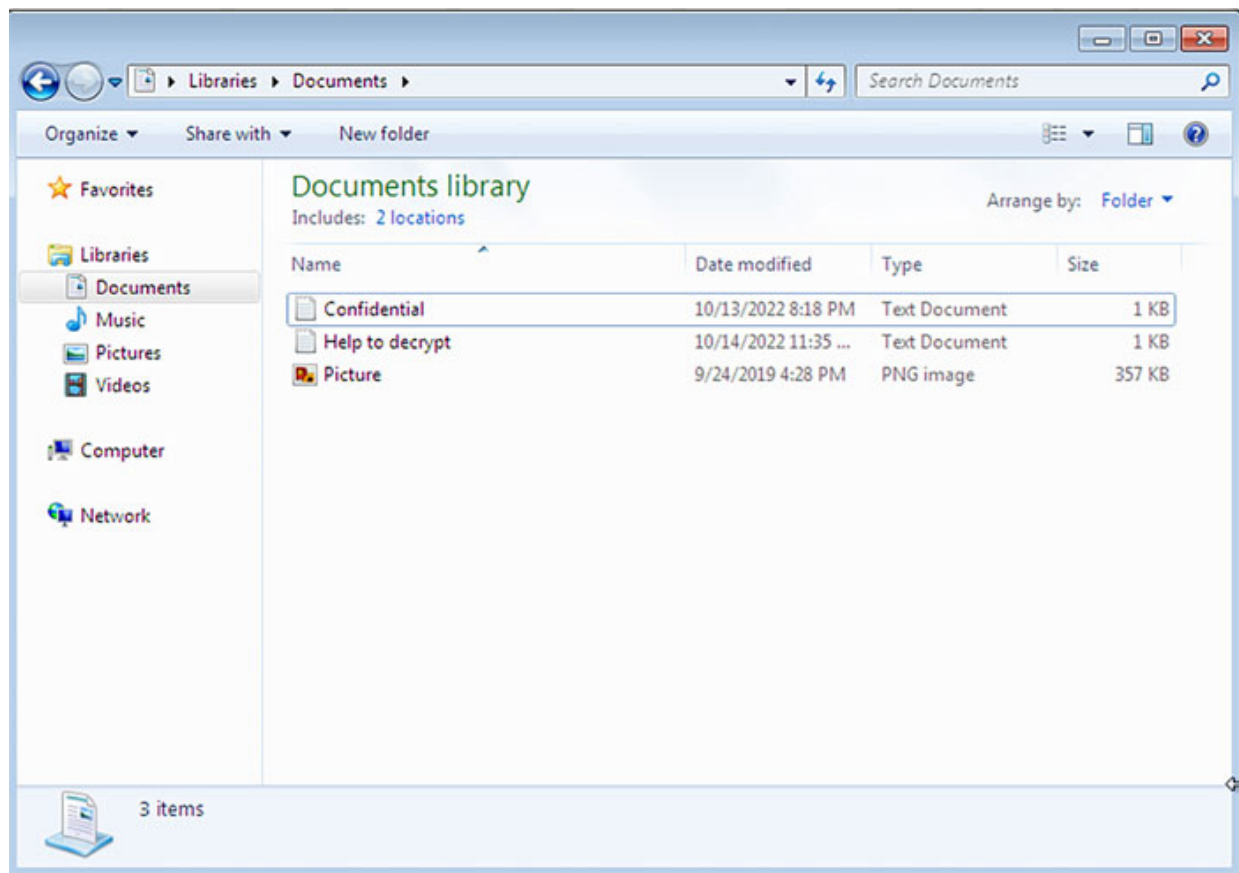


Figure 11.19: Important Files

Now, open the ransomware sample in **x32dbg**, and it will break at the entry point as configured on **x32dbg**, as shown in [Figure 11.20](#):

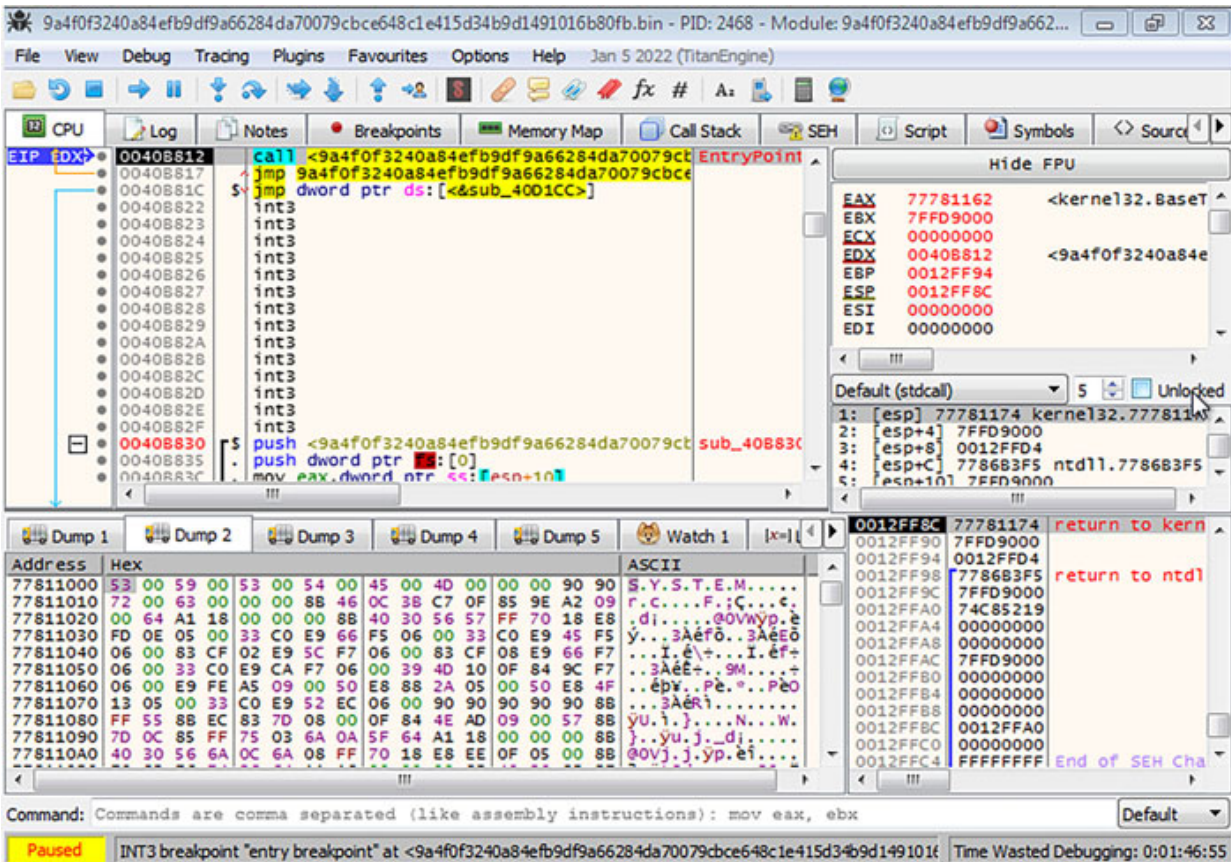


Figure 11.20: Entry Point

Now during static analysis, we figured that the **CryptGenRandom** function is called in a loop to generate a 16-byte key. In order to perform dynamic analysis, we will set breakpoints at a few memory locations, as shown in [Figure 11.21](#). We will set the breakpoints at the following locations (all these breakpoints are set after analyzing code logic):

0x004082B5 - `push 34`: At the start of function call we identified earlier

0x004082E8 - `push 10`: Put 16 in EBX as a counter to generate random bytes for chars

0x0040830E - `cmpdword ptr ds:[ESI+14], 10`: At this point, some ransom characters are generated

0x00404C63 - `mov ebx, dword ptr ds:[<&CryptEncrypt>]`: Breakpoint at `CryptEncrypt`

0x00404CC3 - `call EBX`: Random characters get encrypted with RSA

Refer to [Figure 11.21](#):

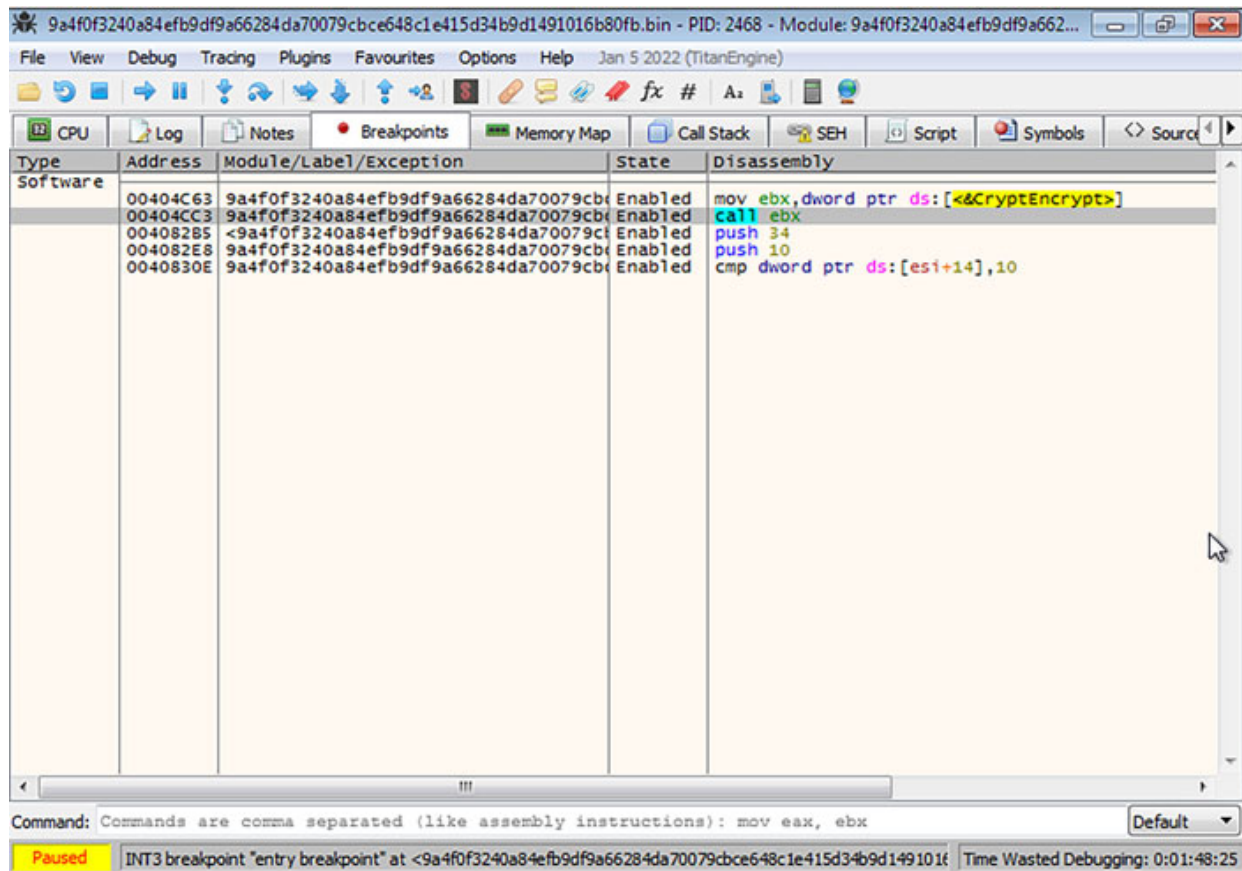


Figure 11.21: Set Breakpoints

After setting breakpoints, click on *run* to hit the first breakpoint at 0x004082B5, as shown in [Figure 11.22](#):

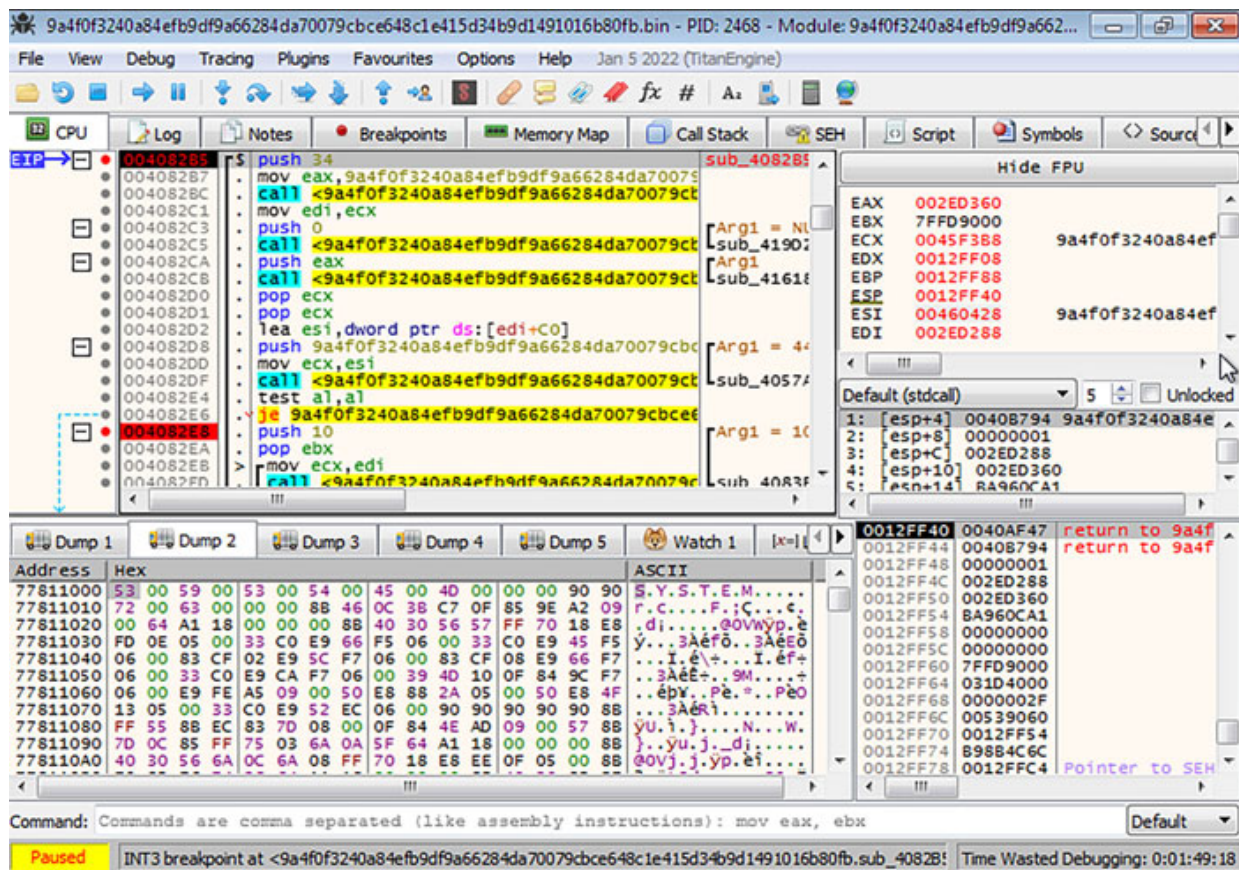


Figure 11.22: Breakpoint at 0x004082B5

In this function, ransomware is running the key generation logic 16 times. This can be seen at location 0x4082E8, where we have set our second breakpoint, as shown in [Figure 11.23](#). At 0x4082E8, we are pushing 0x10 on to stack and then pushing it on to EBX to set the counter set to 16 in decimal in EBX.

Figure 11.23: Breakpoint at 0x004082E8

As can be seen in [Figure 11.23](#), a block shown in red is executed 16 times to generate a key of length 16 char. In order to fetch these random characters (or we can say, key) from memory after running the loop 16 times, we will set a breakpoint at 0x40830E. After pressing *run* in x32dbg, execution will stop at breakpoint. At this point, as per the code, we will observe the ESI value, as shown in [Figure 11.24](#):

Figure 11.24: Breakpoint at 0x0040830E

As we can see in [Figure 11.24](#), ESI is holding the memory location of depicted by & sign. We will dump the memory location 0x0045F478 in Dump1 of x32dbg. At the location, what we get is the location at which the key is stored in memory, which is 0x002EF130 (Little endian format). On dumping this location in Dump2, we get the ransomware key in memory at 0x002EF130, as shown in [Figure 11.25](#):

Figure 11.25: Breakpoint at 0x0040830E and Key at 0x002EF130

We will copy this 16-byte key in the hex editor for analysis later. Now, let us hit our next breakpoint at call of **CryptEncrypt**, as shown in [Figure 11.26](#):

Figure 11.26: Breakpoint at 0x00404C63

On pressing *run* again in x32dbg, we hit on the code just before the call to the **CryptEncrypt** function. As we can see in [Figure 11.27](#), all the parameters to **CryptEncrypt** are pushed on to stack before the call to the **CryptEncrypt** function. At this point, we will dump the ESI on Dump2. We can see that the ransomware key is stored at the location in an unencrypted form.

Figure 11.27: Breakpoint at 0x00404CC3

As soon as we stop over the instruction at **0x00404CC3** which is **CALL EBX**, the key stored in unencrypted form is encrypted with the execution of the **CryptEncrypt** function, as shown in [Figure 11.28](#):

Figure 11.28: Breakpoint at 0x00404CC3 and key encrypted

This is the same encrypted key that the ransomware, upon execution, stores in each directory to demand ransom. This can be seen in [Figure 11.29](#), which is achieved after running the remaining ransomware by pressing **Run** in the **x32dbg**. We can also see that ransomware encrypted our confidential file and renamed it to **Confidential.txt.tuki17@qq.com!!**

Figure 11.29: Encrypted Files

Our sample file **Confidential.txt** is encrypted by the ransomware using the TEA encryption algorithm and has been renamed to **Confidential.txt.tuki17@qq.com!!**

The algorithm by which our data is encrypted, we found this using PEiD in [Figure 11.18](#). Now we have everything to recover our data:

- We know the type of encryption algorithm used to encrypt data, that is, TEA.
- We have recovered the 128-bit key from the memory while the ransomware was running, as shown in [Figure 11.25](#).
- Also, we have the encrypted data for recovery.

In order to check our key and break the ransomware from the key we recovered from memory, we will take the first 64 bits from the encrypted file for decryption using HxD hex editor, as shown in [Figure 11.30](#):

Figure 11.30: Hexview of Plaintext and Encrypted File

We will feed these 64 bits in the following Python code we created to decrypt the TEA encrypted data using the key recovered from memory, as shown in [Figure 11.31](#).

Key recovered from memory:

78 77 62 70 78 69 72 77 69 61 75 79 76 69 6F 71

First 64 bits of encrypted text from **Confidential.txt.tuki17@qq.com!!**:

97 7D A8 9E 6F 30 D8 E7

Once we decrypt the first 64 bits, we just need to fine-tune our Python code to decrypt the whole encrypted file.

Figure 11.31: Python Code for TEA Decryption

In the code in [Figure 11.31](#), at line 37, we are feeding the first 64 bits we received in the encrypted file (**Confidential.txt.tuki17@qq.com!!**). We are also feeding the key we recovered from memory in the Python code at

line 39. After execution of the code shown in [Figure 11.31](#), we will be able to retrieve the initial 64 bits of the unencrypted file (**Confidential.txt**). However, before this, we will have to take care of little endianness. Decrypting data with little endianness will result in the first 64 bits of encrypted data. Refer to [Figure 11.32](#):

Figure 11.32: Unencrypted First 64 bits

A little modification to the Python code of little endianness capability and the ability to read the whole file will result in us being able to retrieve the unencrypted file from the ransomware attack. Thus, we saw how to use dynamic analysis to extract ransomware key from the memory.

This same logic of taking a memory snapshot while the ransomware is running is being used by various modern software to recover data from the ransomware attacks.

Conclusion

In this chapter, we revised what we learned in the earlier chapters. We took a ransomware sample to perform static and dynamic analysis using disassembler and debuggers. During static analysis, we used tools we discussed in [Chapter 9, Performing Static Analysis](#), and identified points that helped us break the ransomware. After the static analysis, we disassembled the ransomware to understand its code flow and logic. After identifying relevant functions in ransomware disassembled logic, we used a debugger to set the breakpoint and understand its dynamic behavior. We went through the code in memory to understand the different phases of ransomware infection, from infecting the host to generating key(s), encrypting user data and demanding ransom. It is during the key management phase that we extracted the key from the ransomware logic and used it to decrypt our locked data from the ransomware.

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

<https://discord.bpbonline.com>

CHAPTER 12

LockCrypt 2.0 Ransomware Analysis

Introduction

In earlier chapters we covered some important Crypto API functions that are used to meet the need for different types of encryption in different applications. However, you will be surprised to know that this encryption functionality is abused by malware writers to write ransomware. To understand this, we will take up a ransomware sample and reverse engineer it to check how a vulnerability in ransomware can be used to break it.

Structure

In this chapter, we will discuss the following topics:

- Static analysis
 - Take sample ransomware for analysis
 - Perform static analysis on the ransomware sample
- Dynamic analysis
 - Perform dynamic analysis using disassemblers and debuggers
 - Break the ransomware by extracting key from memory

Objectives

This chapter's goal is to make it easier for the reader to comprehend what we studied in the previous chapters. Using debuggers and disassemblers, we will perform static and dynamic analysis on a ransomware sample in this chapter. We will use the tools we discussed in [Chapter 9, Performing Static Analysis](#), to perform static analysis and find spots that could aid in decrypting the ransomware. Following the static analysis, the malware will be disassembled so that its logic and code flow can be examined. We will utilize the debugger to create the breakpoint and comprehend its dynamic behavior after determining the pertinent functions in the disassembled logic of the ransomware. To comprehend the various stages of ransomware infection, from infecting the host to creating

key(s), encrypting user data, and demanding ransom, we will walk through the code in memory. We will attempt to extract the key from the ransomware logic during the key management phase, and we will use that key to unlock our locked data that has been encrypted by the ransomware.

Static Analysis

The first step involved in malware analysis is to run static analysis of the malware. In our case, it will be LockCrypt ransomware 2.0. To run the static analysis, we will download the famous LockCrypt ransomware from the given website: <https://www.malshare.com/sample.php?action=detail&hash=307bca9a514b1e5038926a0bafc7bc08d131dd6fe3998f31cb1e614e16effe32>

Before downloading the ransomware, you will have to be cautious about the machine on which you will be downloading it. Never forget to do the test in a controlled environment (sandbox). The details of the ransomware sample are as follows:

Details

File Type: PE32

Hashes

MD5: 3cf87e475a67977ab96dff95230f8146

SHA1: 1fb3dbd6e4ee27bddfcd1935065339e04dae435c

SHA256:

307bca9a514b1e5038926a0bafc7bc08d131dd6fe3998f31cb1e614e16effe32

Once the LockCrypt ransomware is downloaded, we will open the file in PESTudio for static analysis, as shown in [*Figure 12.1*](#):

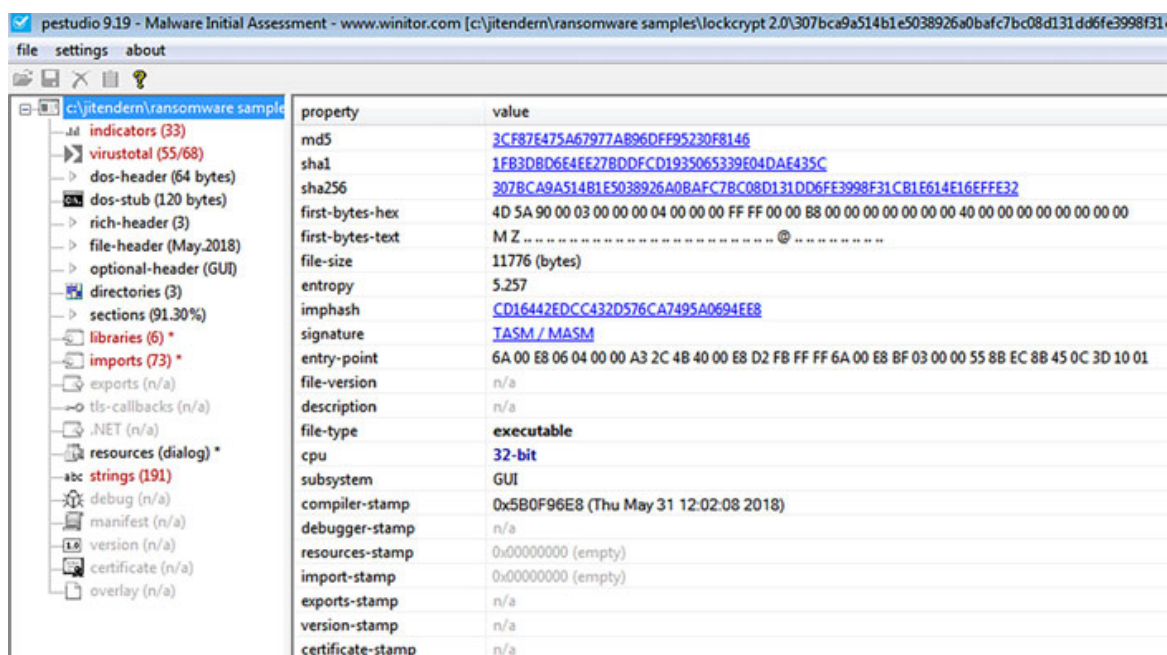


Figure 12.1: LockCryptpestudio

As we can see in [Figure 12.1](#), the hashes are the same ones we saw while downloading ransomware. The **First-bytes-hex** value shows that the file is portable executable with the initial hex value of 0x4D5A, equivalent to MZ in text form. For more details, refer to [Chapter 7, Portable Executable Insides](#), and [Chapter 8, Portable Executable Sections](#).

Before we move on to the next value, we will understand entropy analysis for a malware. Malware is written with a destructive approach in mind, but we all know that antivirus plays a great role in securing our systems. To avoid being detected by **Antivirus (AV)**, malware writers use obfuscation techniques to bypass AV protection. They use various encryption techniques, compression packages and encoding conversion to pack the original executable into something else. When the executable is obfuscated, it is difficult to reverse engineer the malware.

Entropy analysis helps malware researchers in determining whether the executable or malware sample is obfuscated or encrypted. Entropy analysis works on the well-known Shannon's Formula. It measures the entropy of the executable on a scale of 0-8. The lower the value of entropy, the lower will be the chances of that code obfuscation. On the other hand, the higher the entropy, the higher the chances of executable obfuscation. Normal executable usually has entropy of around 5, which means binary is not obfuscated, compressed or encrypted. In our

case, LockCrypt sample has an entropy value of 5.257, which means that the binary is not obfuscated, compressed or encrypted.

We can also confirm, as shown in [Figure 12.1](#) that the sample is portable executable of 32 bit. Compiler-stamp shows that the executable is compiled on May 31, 2018. LockCrypt ransomware came around the same time, but we should not heavily rely on this value as it can be modified.

If you remember, we studied the *Rich* signature in the structure of the PE file. It is an undocumented feature of Microsoft linker wherein *Rich* signature is inserted in Microsoft linked executable to identify the machine and compiler on which the binary is built. In [Figure 12.2](#), the rich-header shows that the executable is compiled on Visual Studio:

Figure 12.2: *LockCrypt rich header*

As shown in [Figure 12.3](#), the main directories of our interest are import and resource:

Figure 12.3: *LockCrypt directories*

[Figure 12.4](#) lists the imported libraries in our sample:

Figure 12.4: *LockCrypt imported libraries*

The main imported library to notice is the **advapi32.dll**. This is the same library used to add encryption capabilities into an application. [Figure 12.5](#) features a list of imported functions that use cryptography functionalities:

Figure 12.5: *LockCrypt imported functions*

We will analyze these functions in our dynamic analysis to break the ransomware.

Next, we will use resource hacker to see the resource dialog used in our sample. As shown in [Figure 12.6](#), this is the same dialog that pops up when the ransomware is executed:

Figure 12.6: *LockCrypt resource hacker*

Lastly, in our static analysis, we see some interesting clear text strings in our executable, as shown in [Figure 12.7](#):

Figure 12.7: *LockCrypt strings*

In string analysis, we see some windows binaries being called and a few commands being executed. There is one command with string starting **Vsadmin**, which is used to delete all shadow backups on the system. This is what a ransomware does upon execution, so one should be cautious while working on a ransomware sample. **DECODE.key** and **notepad++ .exe** file are some interesting binaries for us, which we will consider in our dynamic analysis.

Dynamic analysis

After the static analysis, we will analyze the LockCrypt sample using Ghidra. We will open the Ghidra and create a new project by navigating to **File | New Project**; we will name it *LockCrypt 2.0*. Select **Non-Shared Project** and then click on **Next**, as shown in [Figure 12.8](#):

Figure 12.8: *Ghidra new project*

Enter **Project Directory** and **Project Name**, as shown in [Figure 12.9](#). We named the project Lockcrypt 2.0.

Figure 12.9: *Ghidra new project name*

Now, drag and drop the LockCrypt sample in the Active Project Window. This will import the LockCrypt PE sample. Once the import is done, Import Result Summary will pop up with binary format, Compiler ID, Endianness, Processor and many more, as shown in [Figure 12.10](#):

Figure 12.10: *Lockcrypt import summary*

Click on **OK** to return to the *Active Window* screen, and double-click on the imported LockCrypt sample file to open CodeBrowser with a prompt asking whether or not to analyze the sample. Click on **Yes** to analyze the sample. You will be prompted to select the *Analysis Options*. Keeping the default options checked, enable a few more analysis options, such as *WindowsPE x86 Propagate External Parameter* and *Decompiler Parameter ID*. Click on Analyze and ignore any warning message. Refer to [Figure 12.11](#):

Figure 12.11: *Enable Analysis options*

This will load the LockCrypt ransomware for further analyses. On the top right, PE sections are displayed in the *Program Trees* window, symbols are displayed in *Symbol Tree* window, and the disassembled view of the LockCrypt ransomware is shown in the middle window. To check the entry point of this ransomware, type **entry** in the *functions* window **Filter** option. Double-click on the entry function to move to the entry point of ransomware disassembled code; this is like the **main()** or **WinMain()** function. Refer to [Figure 12.12](#) to see the LockCrypt entry point:

Figure 12.12: *Lockcrypt entry point*

Now we will drill down the ransomware code flow from the entry point and find the list of functions called from the entry point. To get the list of functions called from the entry point, go to *Windows* menu | *Function Call Tree*. As we can see in [Figure 12.13](#), **FUN_004018a0** is the main function called from the entry point. On further expanding the list of functions, we see that the Windows cryptographic function calls **CryptAcquireContextA**, **CryptImportKey**, **CryptGenKey**, **CryptExportKey**, and **CryptEncrypt** to **CryptDestroyKey**. Refer to [Figure 12.13](#):

Figure 12.13: Lockcrypt calls cryptographic functions

As we read in the earlier chapters, the first step to implement encryption is to use Crypto API in the application, which is implemented by opening a handle to appropriate CSP. This handle is opened using the **CryptAcquireContext** function. As we can see in the ransomware assembly listing, the **CryptAcquireContext** function is called in the main **FUN_004018a0** function, which is followed by the **CryptImportKey** function, as shown in [Figure 12.13](#) of Ghidra assembly listing.

If we note the parameters to the **CryptImportKey** function shown in [Figure 12.14](#), they are pushed from right to left as per the function definition:

```
BOOL CryptImportKey(HCRYPTPROV hProv, const BYTE *pbData, DWORD
dwDataLen, HCRYPTKEYhPubKey, DWORDdwFlags, HCRYPTKEY *phKey);
```

Figure 12.14: Lockcrypt ransomware assembly listing-1

Let us look at the implementation of the **CryptImportKey** function in LockCrypt ransomware sample:

- **phKey: Load Effective Address (LEA)** loads the EAX with the memory address of imported key. EAX is then pushed on to stack as the first parameter to **CryptImportKey** function.
- **dwFlags:** This is the second parameter pushed on to stack with the value of zero. This specifies the options used in importing keys.
- **hPubKey:** This is the third parameter to be pushed on to stack, and it signifies the handle to the key used to decrypt the key BLOB stored in *pbData*. So, the value of this parameter depends on the CSP type and the key BLOB being imported. This parameter is zero, as the key BLOB being imported is PAINTEXTBLOB, PUBLICKEYBLOB or OPAQUEKEYBLOB, which is not encrypted.
- **dwDataLen:** The value pushed on to stack is 0x114 bytes. It is the size, in bytes, of the key string or key BLOB.
- **pbData:** This buffer holds the key BLOB data.
- **hProv:** This is the last parameter pushed before the call to the **CryptImportKey** function. This is the handle to CSP created with the

CryptAcquireContext function before the **CryptImportKey** function.

The next function of our interest is **CryptGenKey**, which is used to generate encryption keys for symmetric encryption and public/private keys for asymmetric encryption. Let us look at the implementation of the **CryptGenKey** function in LockCrypt ransomware sample.

```
BOOL CryptGenKey(HCRYPTPROV hProv, ALG_ID Algid, DWORD dwFlags  
HCRYPTKEY *phKey);
```

- **phKey**: This is the first parameter to be pushed, and it is the handle to generated keys. This handle should not be deleted until all the encryption and decryption is done.
- **dwFlags**: This is the second parameter to be pushed on to stack. It specifies the options used for generating keys. The value that is pushed on to stack is 0x01, which is **CRYPT_EXPORTABLE**. As this flag is set, the key can be transferred out of CSP into key BLOB using the **CryptExportKey** function.
- **Algid**: The next parameter that is pushed is the Algid, which is 0x6610. This is the integer identifier of the encryption algorithm AES 256.
- **hProv**: The last parameter pushed is the handle to CSP created with the **CryptAcquireContext** function.

After the **CryptGenKey** function, we see parameters to **CryptExportKey** being pushed on to stack. As we discussed earlier, when keys are exported, they are converted into series ASCII characters, which are then passed on to other users; and when keys are imported, they are converted back into binary format to change them into the original keys. Usually, exported keys are encrypted with the recipient public key to ensure transport of keys in secure manner. Now we will see how this function is implemented in LockCrypt ransomware.

The **CryptExportKey** function is defined as follows:

```
BOOL CryptExportKey(HCRYPTKEY hKey, HCRYPTKEY hExpKey, DWORD  
dwBlobType, DWORD dwFlags, BYTE *pbData, DWORD  
*pdwDataLen);
```

- **pdwDataLen**: The first parameter that is pushed on to stack is the value that specifies the size of key blob data in bytes. We will talk about this parameter while debugging ransomware sample in the next section.
- **pbData**: This parameter holds the pointer to the buffer that holds the key blob data.

- **dwFlags:** This parameter specifies additional options. In our sample, this is set to zero.
- **dwBlobType:** This is the main parameter that is pushed on to stack having value set to 0x08. When *dwBlobType* is set to 0x08, it means that it is exporting blob type asPLAINTEXTKEYBLOB.

Note: Value of any parameter can be referred back from WinCrypt.h. Exported key blob definitions in WinCrypt.h are as follows:

```
#define SIMPLEBLOB                0x1

#define PUBLICKEYBLOB              0x6

#define PRIVATEKEYBLOB            0x7

#define PLAINTEXTKEYBLOB          0x8

#define OPAQUEKEYBLOB             0x9

#define PUBLICKEYBLOBEX           0xA

#define SYMMETRICWRAPKEYBLOB      0xB
```

Absolute path of winCrypt.h in our case is C:\Program Files\Microsoft SDKs\Windows\v7.0A\Include\WinCrypt.h.

- **hExpKey:** The second last parameter pushed is the handle to the key that is used to encrypt the exported key. As this, the parameter is zero (0) if we are exporting Plain text key.
- **hKey:** The last parameter pushed before call to **CryptExportKey** function is the handle to the key to be exported.

In our ransomware sample, we caught an interesting part where keys are exported in plain text blob. In order to further analyze the LockCrypt ransomware behavior, we run the sample in debugger.

To get inside of the ransomware key generation mechanism, we will analyze the memory by running this ransomware sample in a debugger using x32dbg. As we have to analyze the key generation mechanism in memory, we will start by putting the breakpoint on the **CryptImportKey** function call. The address to insert breakpoint can be captured from Ghidra (*Windows menu | Function Call Tree*), by

double-clicking on the **CryptImportKey** function and on the lower status bar we can see the function address 0x004019bd.

Refer to [Figure 12.15](#):

Figure 12.15: *CryptImportKey function address*

Before we start debugging ransomware sample in x32dbg, we must ensure that we are using a closed or sandboxed environment. Open the ransomware sample in the x32dbg. As we have enabled *Entry Breakpoint* setting (from **Options | Preferences | Events | Entry Breakpoint**) in x32dbg, debugger will first break at the entry point of ransomware sample, as shown in [Figure 12.16](#):

Figure 12.16: *Opening lockcrypt ransomware in x32dbg*

Now, we will set the breakpoint at the **CryptImportKey** function located at the 0x004019bd address by using the **bp 0x004019bd** command in x32dbg. By setting breakpoint on the call of the **CryptImportKey** function, we will analyze the parameters passed to the function in memory. Refer to [Figure 12.17](#):

Figure 12.17: *Setting breakpoint at CryptImportkey*

Once we click on **Run** in our debugger, we will again break at the **CryptImportKey** function. We can see in [Figure 12.18](#) the list of parameters pushed on to stack just before the call instruction to the **CryptImportKey** function. As discussed in the earlier chapters, the **CryptImportKey** function is used to import keys from key BLOB to CSP. If we look carefully at the **dwdataLen** parameters, we can understand that the length of imported key blob is 0x114 bytes. Just after the pushing the **dwdataLen** parameters on to stack, we can see that the pointer to the imported key (0x004049FD) is pushed onto stack. On dumping the memory location in Dump1, we can see that the RSA1 key of length 0x114 bytes was imported, as shown in the preceding screenshot.

Refer to [Figure 12.18](#) to look at the parameters of **CryptImportKey**:

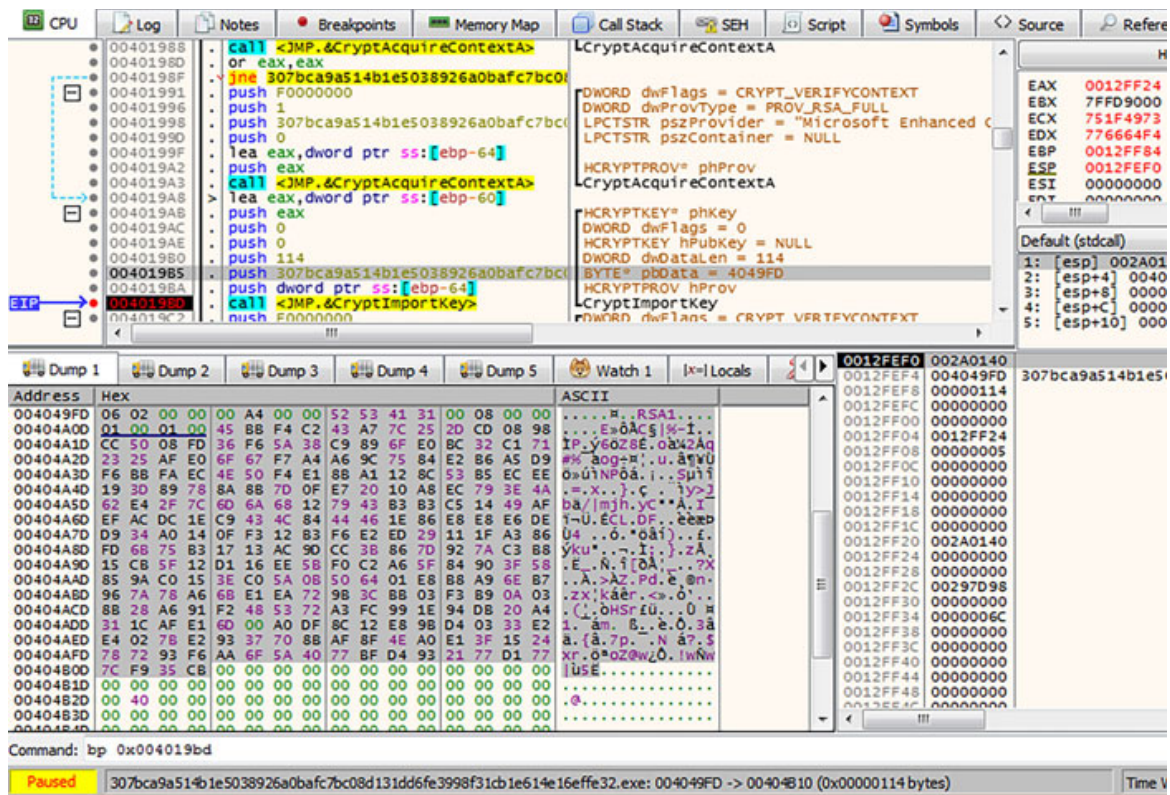


Figure 12.18: Parameters to `CryptImportKey`

Further moving down, we will set another breakpoint at the **CryptGenKey** function. Then, press **Run** again to hit the breakpoint just before the call to the **CryptGenKey** function. In [Figure 12.18](#), we can see the list of parameters pushed on to stack just before the call instruction to the **CryptGenKey** function.

As discussed in the earlier chapters, the **CryptGenKey** function is used to generate encryption keys for symmetric encryption and public/private keys for asymmetric encryption. If we look a closer look at the first parameter pushed on to stack **phKey**, we see that it is the pointer to the generated key (0x00404B20). As we have not executed the **CryptGenKey** function yet, memory *Dump2* to this memory location shows all zeros, as shown in [Figure 12.19](#):

Figure 12.19: Breakpoint at `CryptGenKey` function

On stepping over the function call to **CryptGenKey**, we will see that the pointer to **phKey** is populated with the memory location (0x00404B20) of **phKey**; this is shown in [Figure 12.20](#):

Figure 12.20: Pointer to *phKey* after stepping over *CryptGenKey*

Then, we will set the next breakpoint at the **CryptExportKey** function and click on *run* to hit the breakpoint. When the keys are generated using the **CryptGenKey** function, they are automatically stored by the CSP in key store. To export cryptographic keys in a secure manner, we use the **CryptExportKey** function. The first parameter to **CryptExportKey** is the length of exported key, which is 0x2C, in decimal 44 bytes, as shown in [Figure 12.21](#):

Figure 12.21: Breakpoint at *CryptExportKey* function

If we closely observe the **dwBlobType** parameter in the **CryptExportKey** function, we will observe strange behavior where the key is exported in **plaintextblob** and pointer to the key blob is the second parameter that is pushed on to stack. As we have not yet executed the **CryptExportKey** function, if we dump the pointer to the key blob in *Dump 4*, we see some garbage value with a path to the ransomware sample, as shown in [Figure 12.22](#):

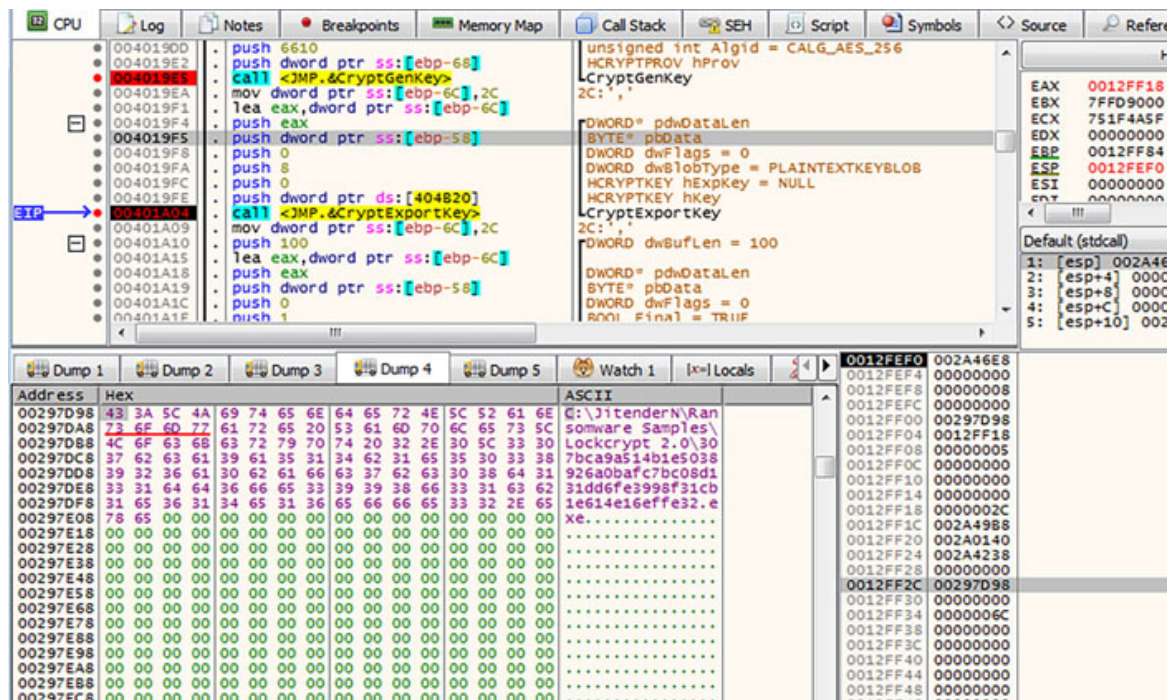


Figure 12.22: Pointer to the key blob

On stepping over the call to **CryptExportKey** function, we can see that the key of size 44 bytes is written at the pointer to the key blob in *Dump 4*. This is plain text blob of 44 bytes, and we will save it in a file. After saving the key blob in a file, we will copy the file to the external drive to preserve it from ransomware havoc.

Saving can be done by selecting 44 bytes and then right-clicking on it to open context menu, binary, and save to file. Refer to [Figure 12.23](#):

Figure 12.23: *PlainTextBlob*

Next, we will set the breakpoint at **CryptEncrypt** and click on run to stop debugger at the **CryptEncrypt** function. The second parameter to be pushed before the call to **CryptEncrypt** is the length of key blob to be encrypted, which is the same as 0x2C. The third parameter is the pointer to the key blob we generated using **CryptExportKey**. The same parameters can be seen on the stack, as shown in [Figure 12.24](#):

Figure 12.24: *CryptEncrypt parameters*

On stepping over the **CryptEncrypt** function, we see that the plain text blob is encrypted with the RSA key imported earlier. Refer to [Figure 12.25](#):

Figure 12.25: *PlainTextBlob is encrypted*

If we move down in the assembling code, we see the **CreateFile** function. It seems that the ransomware is creating a file, **c:\Windows\DECODE.key**, to write the encrypted key into it. This is the same encrypted key that is asked by ransomware writer in ransomware note. Refer to [Figure 12.26](#):

Figure 12.26: *CreateFile function*

On further clicking on *run*, we will see that the ransomware starts encrypting and we can see the ransomware note. Refer to [Figure 12.27](#):

Figure 12.27: Ransomware in action

Ransomware encrypts the files in our system after execution. However, the catch is that LockCrypt ransomware was not programmed properly, as it exported the blob in plaintextkey format. Now, we will extract the key from the blob we saved in a file in the external drive and use it to decrypt the encrypted file on our system. PlainTextBlob we extracted from memory is as follows:

```
08 02 00 00 10 66 00 00 20 00 00 00 AD 3E F2 D7 26 D1 7E 1E B4 7F B7
A1 9B 91 45 94 D5 42 58 60 50 C4 AA 0D DE CB 2D A1 BC 33 B5 D1
```

PlainTextKey blob contains plaintext key in combination with a blob header. The structure of plaintextblob is as follows:

```
typedef struct _PLAINTEXTKEYBLOB {
    BLOBHEADER hdr;
    DWORD      dwKeySize;
    BYTE       rgbKeyData[];
} PLAINTEXTKEYBLOB, *PPLAINTEXTKEYBLOB;
```

Hdr: This is Publickeystruc, also known as blobheader, and the structure of Publickeystruc is as follows:

```
typedef struct _PUBLICKEYSTRUC {
    BYTE    bType;
    BYTE    bVersion;
    WORD    reserved;
    ALG_ID  aiKeyAlg;
} BLOBHEADER, PUBLICKEYSTRUC
```

This occupies 8 bytes in blob header.

dwKeySize: It is the size of the key material in bytes; this occupies 4 bytes in blob header. In our plaintextblob, it is 0x00000020, and 32 bytes in decimal. When 32 bytes is multiplied by 8, it gives 256 bits. This shows that the key is AES256.

rgbKeyData: This is the key we want. In our blob, it is as follows:

AD 3E F2 D7 26 D1 7E 1E B4 7F B7 A1 9B 91 45 94 D5 42 58 60 50 C4 AA
0D DE CB 2D A1 BC 33 B5 D1

Now that we have the AES key, let us take a sample encrypted file and decrypt the encrypted file using an online AES decryptor, as shown in [Figure 12.28](#):

Figure 12.28: *Decrypting encrypted file with AES key*

Download the decrypted text in a file. What we got is the original **desktop.ini** file from the encrypted version, as shown in [Figure 12.29](#):

Figure 12.29: *Decrypted encrypted file*

Conclusion

This chapter's goal was to make it easier for you to understand what we studied in the previous chapters. Using debuggers and disassemblers, we performed static and dynamic analysis on a ransomware sample in this chapter. We used the tools we discussed in [Chapter 9, Performing Static Analysis](#), to run a static analysis and find spots that could aid in decrypting the ransomware. Following the static analysis, the malware was disassembled so that its logic and code flow could be examined. We utilized the debugger to create the breakpoint and understand its dynamic behavior after determining the pertinent functions in the disassembled logic of the ransomware. To understand the various stages of ransomware infection, from infecting the host to creating key(s), encrypting user data, and demanding ransom, we walked through the code in memory. We extracted the ransomware key from the ransomware logic and used it to unlock our locked data that had been encrypted by the ransomware.

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

<https://discord.bpbonline.com>

CHAPTER 13

Jigsaw Ransomware Analysis

Introduction

Imagine that you have important project data into your computer and all of sudden, all the files are renamed to an unknown format and a ransom note appears into your screen; imagine how stressful the situation is. Consider how beneficial it would be to examine malware before executing it. Only static analysis makes this possible. Using tools to perform static analysis substantially aids us in avoiding several mathematical calculations. We will discuss a few open-source and paid tools for static analysis in this chapter. The most challenging part of the approach may be manually unpacking samples that were packaged with the intention of resisting static analysis.

Structure

In this chapter, we will discuss the following topics:

- Ransomware static analysis
- Decompile ransomware sample
- Static analysis tools like PE studio, ILSpy and Kali Linux
- Break the ransomware

Objectives

This goal of this chapter is to understand static analysis and how it can be done to break ransomware. In this chapter, we will perform static analysis on a ransomware sample. We will disassemble the malware, look over the disassembled code, and relate it to various stages of ransomware in general. We will also talk about how static analysis alone can be used to defeat ransomware. The ransomware will be easier to crack if the malware author has low development skill level (as a script kid) or is just carefree. Additionally, we will look at the tools needed for static analysis.

Ransomware analysis

With the aim of locating the kill button and locating the decryption key, we will be investigating ransomware using both static and dynamic methods in this chapter. As we have already studied, static and dynamic analyses are the two fields that make up the umbrella of ransomware analysis.

We discussed in [Chapter 12, LockCrypt 2.0 Ransomware Analysis](#), static analyses involve looking into the binary's source code, structure, and flow to discover its intended behaviors and various functions. On the other hand, dynamic analysis involves studying ransomware while running it for real-time evaluation of the processes, services, and system modifications as they are being used.

This ransomware differs from others that typically only encrypt your data and demand a payment as soon as possible by erasing files every hour for the first 24 hours. Jigsaw puts even more pressure on the user by threatening to erase 1000 files if the system is restarted or process is modified.

Static analysis

Static analysis of any malware is the first stage in the ransomware analysis process. In this instance, a sample of Jigsaw ransomware will be collected. Consider your testing machine cautiously before beginning a ransomware analysis: the environment should be a sandbox.

Details of the ransomware sample will be as follows:

Details

File Type: PE32

Hashes

md5: AC5480B849C4A3F21FC99CA17910B187

sha1: 338FE99B143792AA486C406BC392C7B39F2DF03D

sha256:

A161CAB9A59134509D0CABA543FF727A793B9F0D99A7B3C841835F
3CE8B16817

Now we will open the file in PEStudio for static analysis, as shown in [Figure 13.1](#):

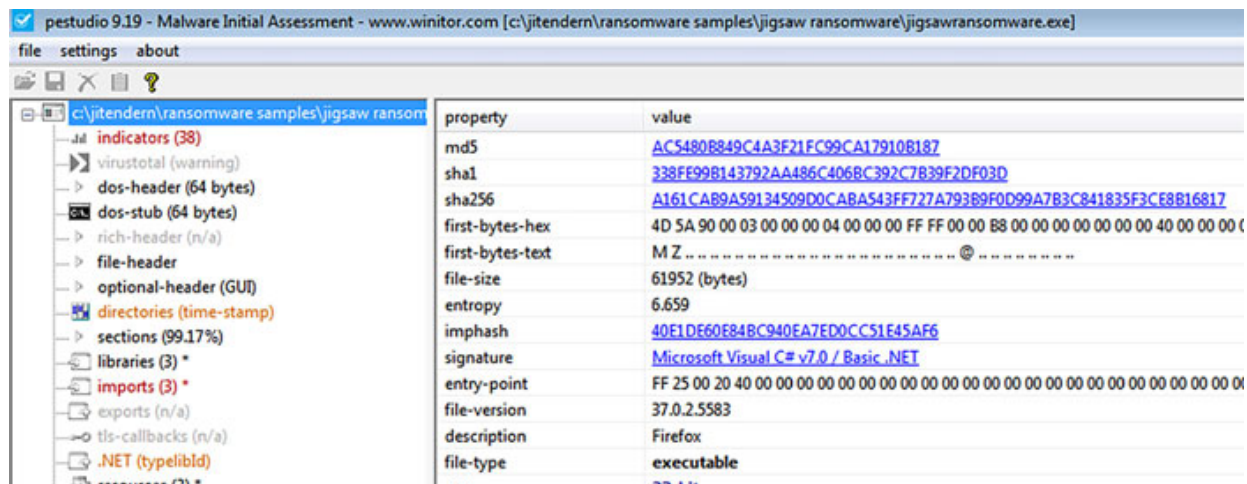


Figure 13.1: Jigsaw in PE Studio

The **First-bytes-hex** value shows that the file is portable executable with initial hex value of 0x4D5A, equivalent to MZ in text form. For more details, refer to [Chapter 7, PE \(Portable Executable\) Insides](#).

We have learned that lower entropy values reduce the likelihood of code obfuscation, while higher entropy values increase the likelihood of executable obfuscation. Entropy of a typical executable is usually around 5, indicating that the binary is not compressed, encrypted, or disguised. The ransomware sample in our case has an entropy value of 6.659, indicating that the binary is neither obscured or compressed nor encrypted.

Now, moving on to the strings sections, we see some interesting clear text strings in our executable, as shown in [Figure 13.2](#):

jigsawransomware.exe]	
	value (993)
	4.0.0.0
	14.0.0.0
	http://btc.blockr.io/api/v1/
	RegistryKey
	SOFTWARE\Microsoft\Windows\CurrentVersion\Run
ig	SetWindowPos
	_CorExeMain
	9<e.H
	j5#.Vb
	JigsawRansomware.exe
	System.Net
	kernel32.dll
	user32.dll
	c:\JitenderN\Ransomware Samples\Jigsaw Ransomware\Jigsaw
	mscoree.dll
	Address.txt
	\DeleteItself.bat
	TxtTest.txt
	EncryptedFileList.txt
	JigsawRansomware.exe
	JigsawRansomware.exe
	!This program cannot be run in DOS mode.
	OoIsAwwF32cICQoLDA00De==
	Registry
on	FromBase64String
on	CreateEncryptor
	DownloadString

Figure 13.2: Jigsaw Strings

In the strings, there are some strings marked in the boxes. The first one is the URL pointing to some **bitcoin (BTC)**. Another is a .bat file, which looks like it is a batch file that will delete the ransomware binary itself after execution. **EncryptedFileList.txt** will most probably list all the files encrypted. Some encoded text in base64 format is something which we cannot comment right now, but seems to be an interesting one to be noted. Lastly, some marked in the box named **CreateEncryptor** shows the presence of ransomware code that will encrypt victim data.

[Figure 13.3](#) shows what we will see if we scroll down in the list of strings:

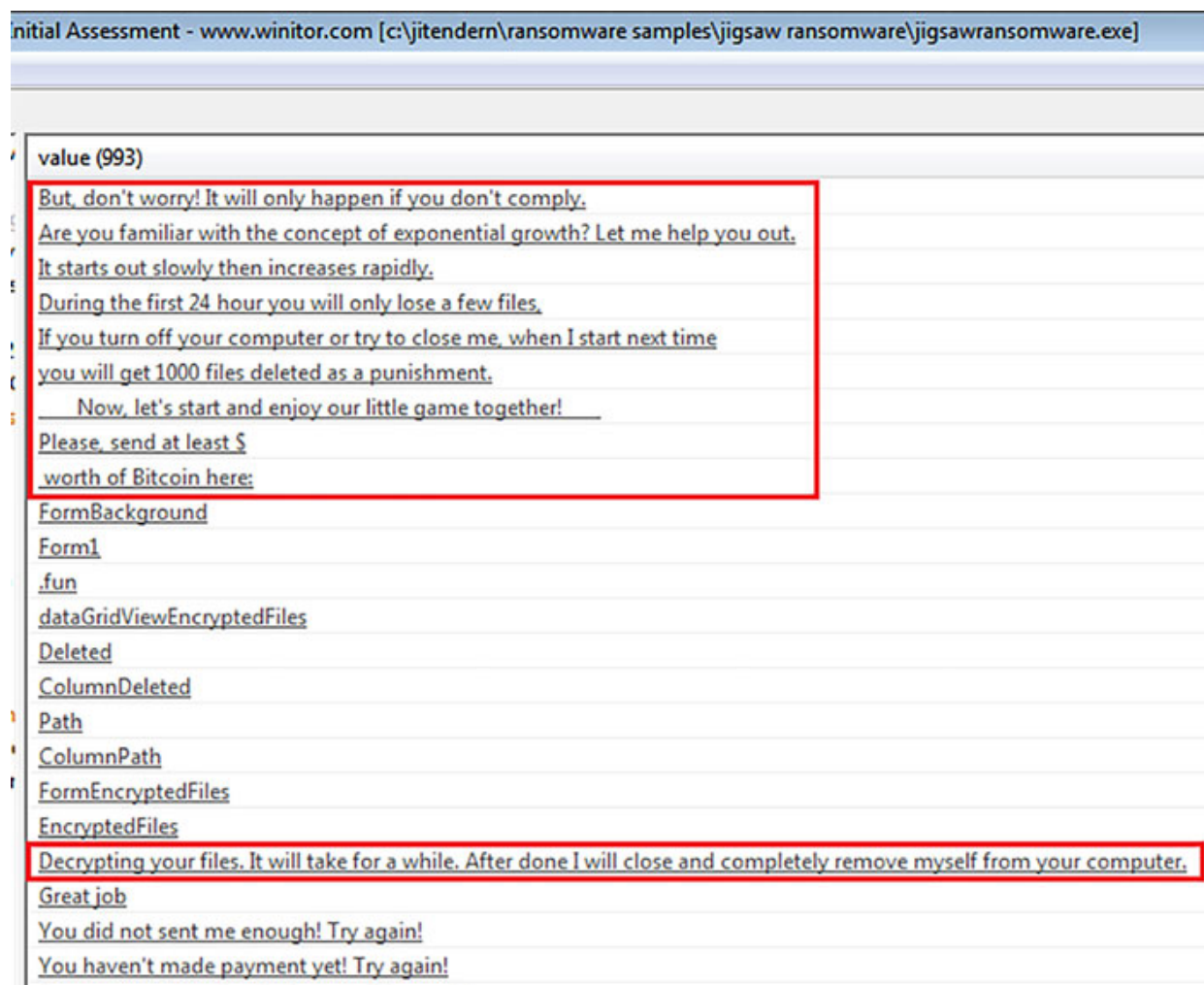


Figure 13.3: Jigsaw Strings More

The strings marked in [Figure 13.3](#) show the ransom note for the victim to be scared and pay ransom on time. Now, we will drill deeper into the internals

of ransomware using the open-source .NET assembly browser and decompile called ILSpy, as the ransomware binary is .NET executable.

ILSpy makes an effort to produce source code files in high-level languages that are similar to the original source code. [Figure 13.4](#) shows Jigsaw ransomware sample in ILSpy:

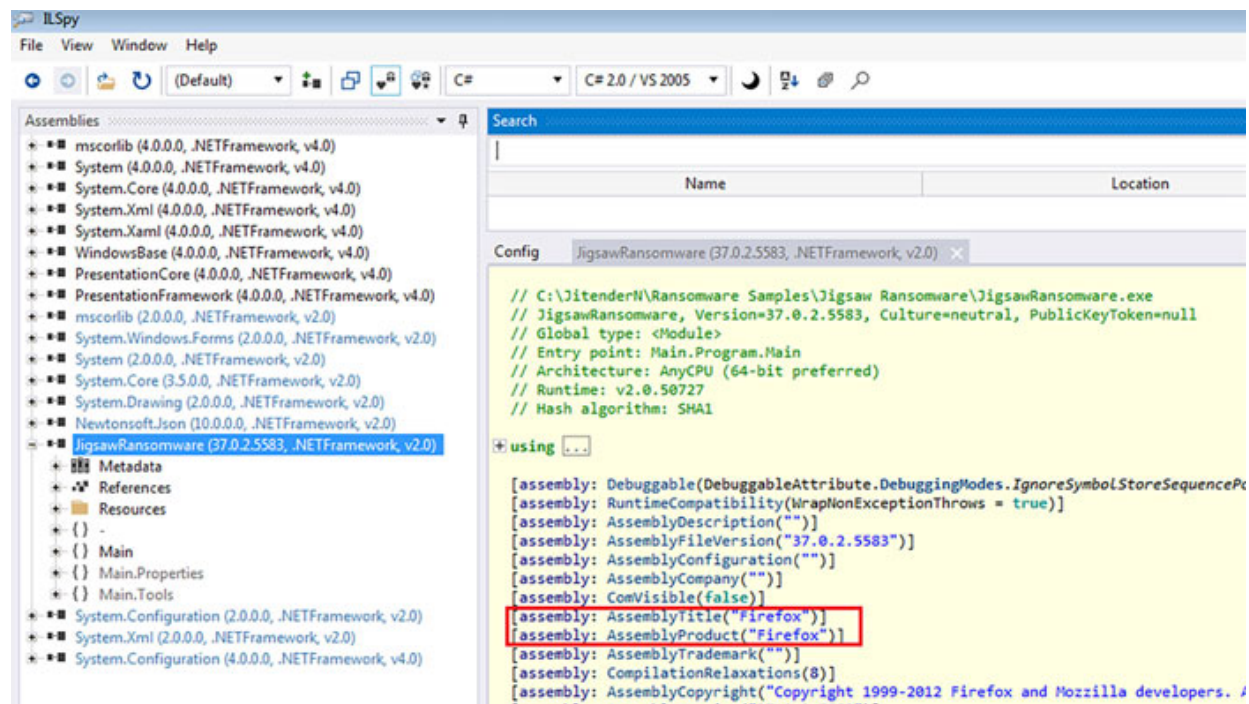


Figure 13.4: Jigsaw ransomware in ILSpy

To open the ransomware sample in ILSpy, go to **File | Open** ransomware sample. As we can see, ILSpy decompiles source code files that are similar to the original source code.

Now, to extract the key generation login, we will go through the decompiled code and see how a ransomware writer has written the key generation logic. The objective is to check the key generation logic and find ways to break it.

As we can see in [Figure 13.4](#), the Assembly title given in ransomware code is *Firefox*. In order to understand the logic behind using *Firefox* as assembly title in jigsaw ransomware, we will search Firefox in a whole disassembled code, as shown in [Figure 13.5](#):

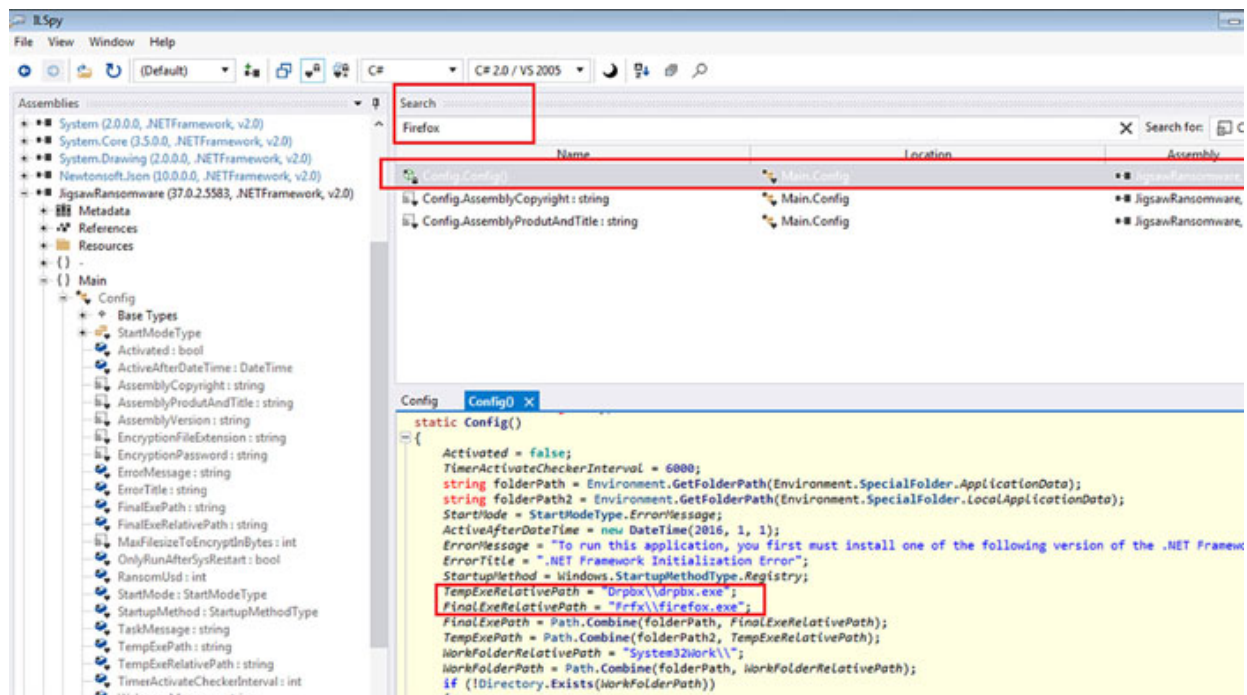


Figure 13.5: Firefox Used in Jigsaw ransomware

We can see that in the decompiled code, the ransomware writer is dropping ransomware on the victim with **drpbx.exe** and **firefox.exe**. This is just to fool the victim into believing that the executable running on computer is *Firefox* or *Dropbox* binary. Moving further up, we see something very interesting in the disassembled code, as shown in [Figure 13.6](#).

We see the Jigsaw hard coded key inside the code highlighted in the box. We will take note of this key for now:

```
internal      const      string      EncryptionPassword      =
"0oIsAwwF32cICQoLDA00De==" ;
```

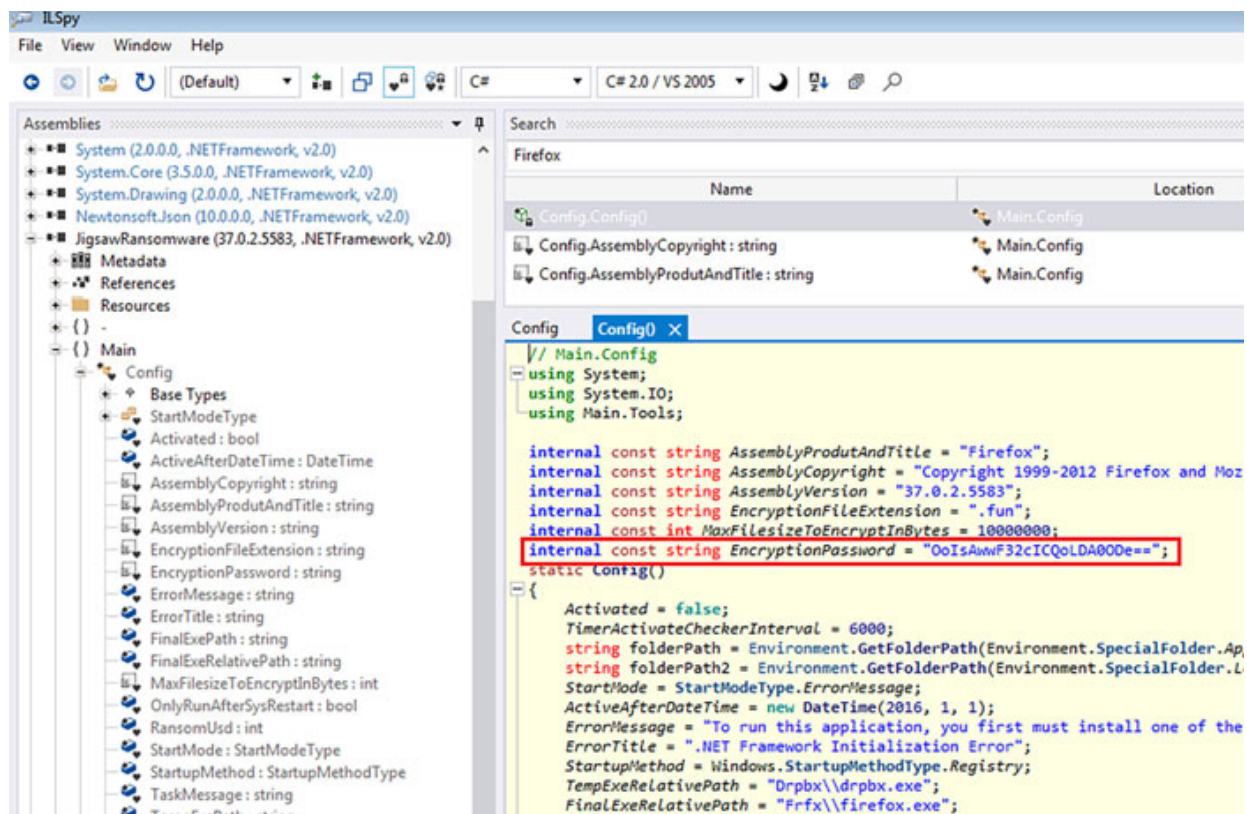



Figure 13.6: Jigsaw key Harcoded

The key in bold is base64 encoded, and the representation of same key in Hex can be achieved using a small piece of Python code, as follows:

```
C:\Users\iicybersecurity>python
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul 8 2019, 19:29:22)
[MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more
information.
>>>
>>>
>>> import base64
>>> base64.b64decode('0oIsAwwF32cICQoLDA00De==').hex()
'3a822c030c05df6708090a0b0c0d0e0d'
```

Thus, the key used by jigsaw ransomware is **3a822c030c05df6708090a0b0c0d0e0d**.

This key is in hexadecimal and its length is 128bit. Now, let us check on the encryption algorithm that Jigsaw is using by drilling more down into the

disassembled code. We see that Jigsaw ransomware used AES CBC with the Initialization vector mentioned in the code, as shown in [Figure 13.7](#):

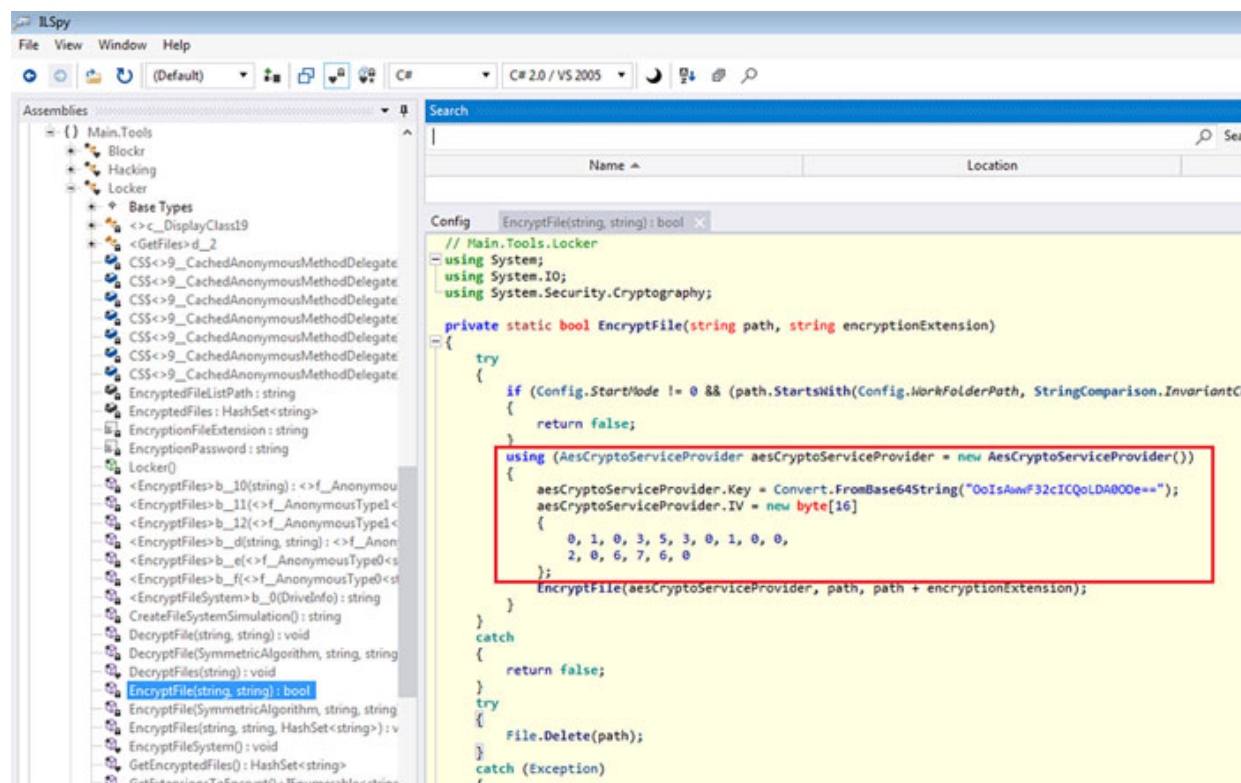


Figure 13.7: Jigsaw Encryption Algorithm and IV

We now have **Initialization Vector (IV)**, along with the key we extracted from the code, as follows:

Key: **3a822c030c05df6708090a0b0c0d0e0d** (128bit)

Initialization vector (IV): **00010003050300010000020006070600** (128 bit)

With this in hand, we will run the jigsaw ransomware sample in a controlled environment. We can see that it converts all files with the `.fun` extension, as shown in [Figure 13.8](#):

 camera images.png.fun	11/19/2022 11:15 ...	FUN File
 code-png.jpg.fun	11/19/2022 11:15 ...	FUN File
 computer-server-icon-24.png.fun	11/19/2022 11:15 ...	FUN File
 Email Hacking.png.fun	11/19/2022 11:15 ...	FUN File
 figures.pptx.fun	11/19/2022 11:15 ...	FUN File
 IAT issue that took weeks.pptx.fun	11/19/2022 11:15 ...	FUN File
 LINK fatal error LNK1123 failure during conversion to C...	11/19/2022 11:15 ...	FUN File
 main-qimg-9d603351e27d57b2f83e9367438a3c50.png.f...	11/19/2022 11:15 ...	FUN File
 MalvertisingInfoDraft.png.fun	11/19/2022 11:15 ...	FUN File
 PLAINTEXTBLOB_SAMPLE_44-BYTES.bin	10/25/2021 7:13 PM	BIN File
 PLAINTEXTBLOB_SAMPLE_44-BYTES.txt.fun	11/19/2022 11:15 ...	FUN File
 RE4Beginners.pptx - Shortcut	12/28/2020 1:24 AM	Shortcut
 Security Trends_v0.5.pptx.fun	11/19/2022 11:15 ...	FUN File
 Social Engineering_v0.1.ppt.fun	11/19/2022 11:15 ...	FUN File
 software - Shortcut	12/27/2020 4:13 AM	Shortcut
 stairs.png.fun	11/19/2022 11:15 ...	FUN File
 Thumbs.db.fun	11/19/2022 11:15 ...	FUN File
 Tor_logo1.png.fun	11/19/2022 11:15 ...	FUN File

Figure 13.8: Jigsaw Execution Output

All the files are converted to the **.fun** extension. We do not need to worry because we have the key and IV. There are multiple ways of recovering the files as we have the key and IV. Either we can code in Python, write some code in C++ or simplify our efforts by using **openssl** to recover our file. We will use Kali Linux as it has **openssl** already packed into it; the command we will use is as follows:

```
openssl enc -aes-128-cbc -nosalt -d -in <Encrypted-file-name> -K '3a822c030c05df6708090a0b0c0d0e0d' -iv '00010003050300010000020006070600' ><Decrypted-file-name>
```

enc = Encoding with Ciphers, these are the list of cipher supported

```
root@iicybersecurity:/tmp# openssl enc -help
```

Usage: enc [options]

Valid options are:

- help Display this summary


```

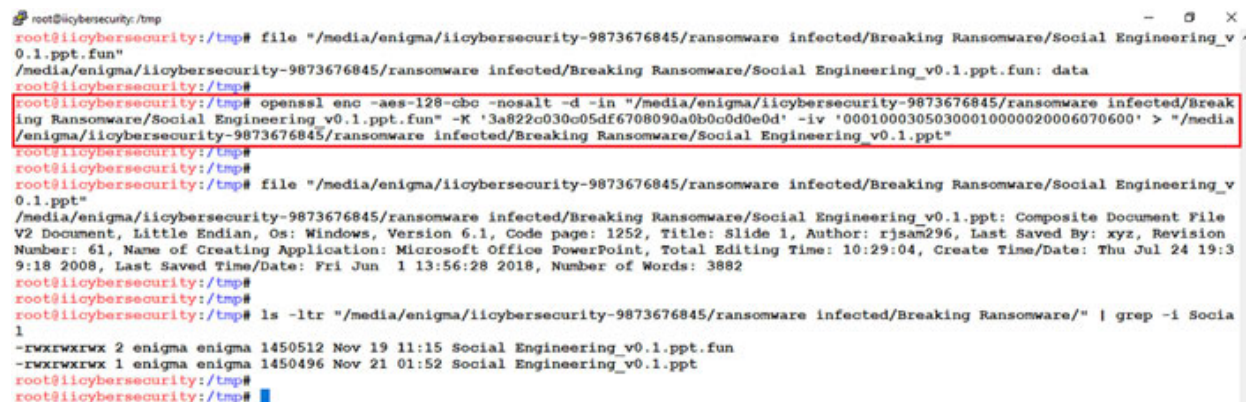
-ciphers          List ciphers
-in infile        Input file
-out outfile      Output file
-pass val         Passphrase source
-e               Encrypt
-d               Decrypt
-p               Print the iv/key
-P               Print the iv/key and exit
-v               Verbose output
-nopad           Disable standard block padding
-salt            Use salt in the KDF (default)
-nosalt          Do not use salt in the KDF
-debug           Print debug info
-a               Base64 encode/decode, depending on encryption
flag
-base64          Same as option -a
-A              Used with -[base64|a] to specify base64 buffer
as a single line
-bufsize val     Buffer size
-k val           Passphrase
-kfile infile    Read passphrase from file
-K val           Raw key, in hex
-S val           Salt, in hex
-iv val          IV in hex
-md val          Use specified digest to create a key from the
passphrase
-none            Don't encrypt
-*              Any supported cipher
-engine val      Use engine, possibly a hardware device
root@iicybersecurity:/tmp# openssl enc -ciphers
Supported ciphers:
-aes-128-cbc          -aes-128-cfb          -aes-128-
cfb1
-aes-128-cfb8         -aes-128-ctr          -aes-128-
ecb
-aes-128-ofb          -aes-192-cbc          -aes-192-
cfb

```

-aes-192-cfb1	-aes-192-cfb8	-aes-192-
ctr		
-aes-192-ecb	-aes-192-ofb	-aes-256-
cbc		
-aes-256-cfb	-aes-256-cfb1	-aes-256-
cfb8		
-aes-256-ctr	-aes-256-ecb	-aes-256-
ofb		
-aes128	-aes128-wrap	-aes192
-aes192-wrap	-aes256	-aes256-
wrap		
-bf	-bf-cbc	-bf-cfb
-bf-ecb	-bf-ofb	-blowfish
-camellia-128-cbc	-camellia-128-cfb	-
camellia-128-cfb1		
-camellia-128-cfb8	-camellia-128-ctr	-
camellia-128-ecb		
-camellia-128-ofb	-camellia-192-cbc	-
camellia-192-cfb		
-camellia-192-cfb1	-camellia-192-cfb8	-
camellia-192-ctr		
-camellia-192-ecb	-camellia-192-ofb	-
camellia-256-cbc		
-camellia-256-cfb	-camellia-256-cfb1	-
camellia-256-cfb8		
-camellia-256-ctr	-camellia-256-ecb	-
camellia-256-ofb		
-camellia128	-camellia192	-
camellia256		
-cast	-cast-cbc	-cast5-
cbc		
-cast5-cfb	-cast5-ecb	-cast5-
ofb		
-chacha20	-des	-des-cbc
-des-cfb	-des-cfb1	-des-cfb8
-des-ecb	-des-edc	-des-edc-
cbc		

-des-edc-cfb	-des-edc-ecb	-des-edc-
ofb		
-des-edc3	-des-edc3-cbc	-des-
ede3-cfb		
-des-edc3-cfb1	-des-edc3-cfb8	-des-
ede3-ecb		
-des-edc3-ofb	-des-ofb	-des3
-des3-wrap	-desx	-desx-cbc
-id-aes128-wrap	-id-aes128-wrap-pad	-id-
aes192-wrap		
-id-aes192-wrap-pad	-id-aes256-wrap	-id-
aes256-wrap-pad		
-id-smime-alg-CMS3DESwrap	-rc2	-rc2-128
-rc2-40	-rc2-40-cbc	-rc2-64
-rc2-64-cbc	-rc2-cbc	-rc2-cfb
-rc2-ecb	-rc2-ofb	-rc4
-rc4-40	-seed	-seed-cbc
-seed-cfb	-seed-ecb	-seed-ofb

We took one encrypted file as a sample and used the **openssl** command to recover our encrypted file, as shown in [Figure 13.9](#):



```

root@iicybersecurity:/tmp# file "/media/enigma/iicybersecurity-9873676845/ransomware infected/Breaking Ransomware/Social Engineering_v0.1.ppt.fun"
/media/enigma/iicybersecurity-9873676845/ransomware infected/Breaking Ransomware/Social Engineering_v0.1.ppt.fun: data
root@iicybersecurity:/tmp#
root@iicybersecurity:/tmp# openssl enc -aes-128-cbc -nosalt -d -in "/media/enigma/iicybersecurity-9873676845/ransomware infected/Breaking Ransomware/Social Engineering_v0.1.ppt.fun" -K '3a822c030c05df6708090a0b0c0d0e0d' -iv '00010003050300010000020006070600' > "/media/enigma/iicybersecurity-9873676845/ransomware infected/Breaking Ransomware/Social Engineering_v0.1.ppt"
root@iicybersecurity:/tmp#
root@iicybersecurity:/tmp# file "/media/enigma/iicybersecurity-9873676845/ransomware infected/Breaking Ransomware/Social Engineering_v0.1.ppt"
/media/enigma/iicybersecurity-9873676845/ransomware infected/Breaking Ransomware/Social Engineering_v0.1.ppt: Composite Document File V2 Document, Little Endian, Os: Windows, Version 6.1, Code page: 1252, Title: Slide 1, Author: rjsam296, Last Saved By: xyz, Revision Number: 61, Name of Creating Application: Microsoft Office PowerPoint, Total Editing Time: 10:29:04, Create Time/Date: Thu Jul 24 19:39:18 2008, Last Saved Time/Date: Fri Jun 1 13:56:28 2018, Number of Words: 3882
root@iicybersecurity:/tmp#
root@iicybersecurity:/tmp# ls -ltr "/media/enigma/iicybersecurity-9873676845/ransomware infected/Breaking Ransomware/" | grep -i Social
-rwxrwxrwx 2 enigma enigma 1450512 Nov 19 11:15 Social_Engineering_v0.1.ppt.fun
-rwxrwxrwx 1 enigma enigma 1450496 Nov 21 01:52 Social_Engineering_v0.1.ppt
root@iicybersecurity:/tmp#

```

Figure 13.9: Decrypted File using Openssl in Kali Linux

The File command is used to determine the file type of encrypted as well as decrypted files. No data can be extracted from an encrypted file, which is shown in [Figure 13.9](#). But when we run the file command against a decrypted file, we can determine that the file is of type Microsoft Office PowerPoint.

Thus, we looked at the process to break the jigsaw ransomware and a way to recover encrypted files.

Conclusion

We discussed the static analysis procedure in this chapter. Static analysis was conducted on a ransomware sample as we worked on it, and we disassembled the malware, looked through the disassembled code, and mapped it to various stages of ransomware in general. Using static analysis, we disassembled the ransomware key and IV. Using the key and IV, we also looked at how we can recover files from a ransomware attack. Further on, we looked at the tools needed for static analysis.

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

<https://discord.bpbonline.com>



Section V: Ransomware Rescue

CHAPTER 14

Experts Tips to Manage Attacks

Introduction

Modern ransomware assaults frequently incorporate numerous cutting-edge hacking methods. You require advanced defense that keeps track of and safeguard the entire assault chain to reduce your risk of becoming a victim. Throughout this book, we studied the internal details about ransomware and how ransomware is evolving from time to time. We started with the building blocks of ransomware and moved on to the ransomware internals and other things. It is very difficult to eradicate ransomware attacks where victims are not machines but humans who are forced to be the target. There are many ways in which we can prevent ransomware attacks in the current environment. We will talk about them in this chapter.

Structure

In this chapter, we will discuss the following topics:

- Ransomware incident response plan
- Ransomware mitigation strategies
- Endpoint security solutions
- Network security solutions
- End user level security

Objectives

The objective of this chapter is to help you understand ransomware incident response plan. Organizations are more likely to experience repeated ransomware attacks if they do not have a response strategy in place, which includes measures to stop more data breaches. In this chapter, we will talk about the different steps to take toward incident response plan, from identification to containment, eradication, communication, and recovery. In

the end, we will look at the tasks that need to be done post incident. In the next section, we will talk about ransomware mitigation strategies that involve solutions from end security solutions, network security solutions and end user level security. End security solutions will cover security solutions designed to help detect, prevent and eradicate malware on end devices. Network security solutions are those that keep data secure, ensure reliable access with good network performance and provide protection from cyber threats. The end user level security section will walk you through the solutions that are specifically focused on the end user.

Ransomware incident response plan

There are many reasons to create a ransomware incident response plan instead of having no plan and managing ransomware incidents like regular cyber attacks. A ransomware attack can cost a business millions of dollars of losses, and its data may be rendered unavailable by the attack. A ransomware incident response plan can help an organization save money and time and recover from an attack without paying the ransom. If recovery is expected to take a lot of time, then the company may choose to pay the ransom for restoring its operations, but this can also harm the company's reputation.

An incident response plan also makes sure that organizations learn from their mistakes and are in a stronger position to prevent future ransomware attacks. If an organization does not have a response plan in place, which includes steps to prevent future data breaches, it is more likely to become a victim of ransomware attacks over and over.

Another reason to have a ransomware incident response plan is to protect the organization's reputation, as ransomware attacks can bring its operations to a halt. This downtime can damage the reputation and customers can lose trust in the organization. With a response plan, an organization can ensure speedy recovery from ransomware attacks.

An incident response plan can vary from one organization to another, depending upon the risks, the backup tools, cyber security tools and processes, and the resources in place for responding to ransomware attacks.

Following are the important phases in a ransomware incident response plan.

Identification

This phase involves analyzing and identifying what type of ransomware has infected the organization, determining the scope of the attack, assessing the impact and getting the right people working on it quickly. Ransomware can spread very fast through networks, so quick action as per the response plan can make a big impact.

Analysis

Organizations must analyze and validate each incident to confirm whether it is a ransomware attack or other malware. This analysis should provide enough information to prioritize containment. What kind of ransomware targeted the system is the first important question to be answered during the analysis process. Although the name “*ransomware*” is most often associated with file encryptors, there are other ways for malware to demand a ransom. The main objective of ransomware is to block access to the entire system, a specific section of the system, or data, or to pretend to do so, and then demand money from the system’s user to undo the alterations.

Following are the different types of ransomware; determining the type of ransomware can help you a lot with incident recovery:

- **File encrypter:** The file encrypter ransomware encrypts files on the hard disk but only targets specific file extensions like doc, docx, and jpeg. After encrypting, it changes the file extension to a ransomware-specific extension, such as .locked, .xxx, and .aaa.
- **MBR encrypter/Overwriter:** This ransomware overwrites or encrypts the **Master Boot Record (MBR)** and demands a ransom to retrieve a password and restore the original MBR. In case MBR is just overwritten by a corrupted MBR, there is a possibility of recovering old MBR records using forensics tools and not paying the ransom. If MBR is encrypted, then it means the whole disk is encrypted and encryption will be difficult. In both cases, the victim will not be able to boot the operating system.
- **Wiper/disk erasure:** Some threat actors have modified traditional ransomware to corrupt the hard disk during the encryption process. The malware infects the MBR and then encrypts the internal file table of the NTFS file system. Thus, instead of encrypting data, it corrupts or wipes

it. This ransomware is used when a threat actor wants to harm a company. Although it will demand a ransom, that is of no use, because even if a victim pays the ransom and gets the keys, the payload will not be able to revert its changes. Thus, it is important to consider that recovering the keys or paying the ransom is not a solution to remediate the incident in this case.

- **Fake encryptor:** This ransomware does not actually encrypt data. It simply renames files, for example, by adding ransomware-like suffixes, to trick people into thinking that their files are encrypted. Windows selects the software it uses to open a file based on the file extension. Therefore, altering the extension will make it appear as though the files “no longer function.” The functionality of the file will also be restored when the file extension is changed again. Thus, it is important to consider that recovering the keys or paying the ransom is not a solution, since changing the file extension to the original extension can easily mitigate this attack.
- **Screen locker:** This ransomware is less harmful because the locking mechanism can be undone as it just adds a screen lock and does not encrypt the file. There is no need to pay ransomware, and the screen lock can easily be removed by killing the process. However, some ransomware families combine screen locking and file encryption, which means that they lock the screen while simultaneously encrypting system data. You typically just need to know if they are pure screen lockers to undo the harm.
- **Leakware/Double extortion:** Doxware or leakware threatens to publish confidential information about individuals or businesses online, and many people become alarmed and pay the ransom to stop this from happening. Police-themed ransomware is one variant; it poses as law enforcement and informs users that it has discovered illicit internet conduct and that jail time may be avoided by paying a fee. It may or may not encrypt files. Analysis should be done to determine whether files are encrypted. It is also called double extortion ransomware when it encrypts files and exports data to blackmail victims into paying a ransom.
- **Triple extortion:** A more advanced variation of double extortion ransomware is triple extortion ransomware. In this scenario, the threat

actor covers all double extortion ransomware techniques and threatens to expose important information unless the victim firm pays a ransom. With a triple extortion attack, the threat actor will pressurize the victim with another technique. The attacker may approach the victim's clients or vendors and demand ransom by making threats about data leaks. To add more pressure, the attackers may also conduct a **Distributed Denial of Service (DDoS)** attack or make phone calls or send text messages or take over company printers to print ransom notes. In some types, attackers can even enter in contact with the customers of that company. Here in my country, we had a case about a threat actor that got in contact with some customers and said that all their data coming from that company are on-wild, and if he wants to remove any information about him, he needs to pay a value in crypto to do that. The ransom note also said that the company's security was horrible. Even said about how the Data Protection law will affect them, so a whole another new level of extortion.

- **Smartphone Ransomware:** As ransomware has grown in popularity on PC platforms, more malware has emerged that target mobile operating systems. Typically, mobile ransomware targets the Android operating system using malicious APK files or click jacking attacks. For iPhones, threat actors take control of iCloud accounts and use the Find My iPhone system to lock access to the device.
- **Ransomware as a Service (RaaS):** It is the term used to describe ransomware that is hosted secretly by a threat actor group that manages all facets of the attack, from disseminating ransomware to obtaining payments and regaining access, in exchange for a percentage of the proceeds. In the RaaS model, cyber criminals rent a particular ransomware from the malware's creator, as a pay-per-use service. Similar to a SaaS concept, developers of RaaS put their ransomware on dark net websites and allow criminals to subscribe to it. These days, most ransomware infections are caused by RaaS groups, and each group has their support portal to receive ransom and chat with clients.

[Determining the strain of ransomware](#)

The objective of this is to determine the family or variant of the ransomware that infected the devices. Normally, the ransomware family or variant is

denoted by the name of ransomware gang. This step is crucial because by identifying the ransomware family, the team can determine whether a free decryptor is available for the specified ransomware, whether there is a ransomware vulnerability that can be exploited to recover data without paying, and whether FBI or Europol have the solution to the ransomware problem. As an alternative, it can be helpful if a ransomware gang is one of the US Treasury Department's or other government's sanctioned entities and paying any ransom to the sanctioned group could result in legal issues for the company. To do this, you will have to look out for clues in the **Graphical User Interfaces (GUIs)** or HTML or text files created by the ransomware, as some gangs mention the details of ransomware, time left and ransom amount and how to contact them for paying the ransom. Moreover, you could find these details in contact emails in encrypted file extensions or pop-ups after trying to open an encrypted file. Sometimes ransomware can also change the wallpaper of the devices, and this wallpaper can have details related to gang names and their contact details. A few ransomware create image files or voice messages containing the details. The following clues should help you find the strain of the ransomware:

- Deep web links where data will be published if ransom is not paid or support chat links.
- Contact email or user of the email; you can Google the email address.
- File renaming scheme or the type of extension of the encrypted files (for example, .xxx, .abc, .locked).
- Ransom demand, for example, digital currency (Bitcoin, Monero, Ethereum and so on) or gift cards can give a hint as some gang only accepts certain digital currency or cryptocurrency payment addresses.
- You can also get a clue via the language, structure, phrases or artwork used in the ransom note.
- Targeted file types or their locations or file owning user/group of affected files can also tell you a lot about the type of ransomware.
- Icon or file markers for encrypted files are an important clue.
- Some ransomware strains only affect particular software files, operating systems, or files related to particular hardware, as they exploit particular vulnerabilities related to them.

In addition to finding clues via the above-mentioned methods, you can upload the encrypted files and ransom note to services like Crypto Sheriff/No more ransomware (<https://www.nomoreransom.org/crypto-sheriff.php>) or ID Ransomware (<https://id-ransomware.malwarehunterteam.com/>), as referred to in [Chapter 1, Warning Signs, Am I Infected](#). These websites can guide any non-technical user in finding the type of ransomware.

Determining the scope of the ransomware infection

Identifying which devices have been affected by the attack is important. You can scan for **Indicators of Compromise (IOCs)**, such as files/hashes, processes, network connections, IP address of command-and-control server, TOR links, similar users, groups, data, tools, department, configuration, patch status and so on, via system logs, SIEM solution, EDR solutions, IAM tools Active directory logs and firewall logs.

The next step is to determine what data has been encrypted, such as file types, department or group, and affected software or servers. You can scan for changes to file metadata, such as mass changes to creation or modification times via metadata search tools, file integrity monitoring tools, and DLP logs. Information collected during phase can be used to find the initial infection vector using Initial Access tactic of MITRE ATT&CK framework as reference.

Assessing the impact of the ransomware attack

Assessing the functional impact of the ransomware attack to a business is very important to take further decisions for recovering from it. You might need the involvement of company directors for finding out the following:

- How much monetary damage or risk is involved?
- For how many days will business be affected or come to halt; what is at loss or risk?
- What is the impact on confidentiality, integrity, and availability of data?
- How critical or sensitive is the data to the business/mission?

- What is the regulatory status of the encrypted data (for examples, PII, PHI)?

Identifying the initial infection vector of the source of the infection

The team should determine the first infection vector, given how ransomware infection happened. This information can be helpful in identifying ransomware strain and preventing further spread. Match the tactics captured in the Initial Access tactic of MITRE ATT&CK. The common infection vectors and well-known ransomware families that employ them are listed as follows:

- **Email attachment infection:** Analyze email logs, email security solutions logs and SIEM logs.
- **Insecure Remote Desktop Protocol (RDP):** Check previous vulnerability assessment reports, firewall configurations changes and logs.
- **Network self-propagation (worm or virus):** Analyze EDR logs, Sys logs and Active directory logs.
- **Infection via removable drives:** Analyze EDR logs and USB logs.
- **Software downloads via malicious link:** Analyze SIEM and UEBA logs.

Containment

This phase focuses on limiting the damage, preventing further infection in the network, and retaining data for digital forensics or possible use in legal proceedings. During a ransomware attack, short-term containment typically happens at the same time as the identification of the attack during the identification phase. The process includes quarantines (logical or physical) to stop the transmission of ransomware from infected systems to other parts of the network. Comprehensive quarantines should be done if a ransomware is known to spread; the ransomware incident management team can take the following steps:

- Quarantine the infected endpoints or server.
- Quarantine affected users and groups from Active Directory and IAM.

- Quarantine infected or uninfected file share file shares via network firewall or DLP rules.
- Quarantine infected or uninfected shared databases and CI/CD pipelines.
- Quarantine backups and make sure infected machines' backup does not replace the clean backup.
- Identify and block command and control server domains or TOR links.
- Remove phishing emails from employee's inboxes.
- Disable all single sign on to cloud resources.
- Disconnect online backup from the network.
- Update and enable endpoint protection solutions such as AV, NGAV, EDR.
- Confirm patches are deployed on all machines and servers.
- Configure security solutions to use discovered IOCs and signatures.
- Do not power off the device; keep it disconnected is the ideal state. In many cases, the ransomware decryption keys remain in RAM and can be recovered using forensics tools.

In ransomware attacks, containment is a very important phase as putting devices in quarantine can lead to faster recovery and less impact. It can prevent spread from infected devices and also prevent spread to critical systems and data. During the containment phase, you should quarantine infected devices, affected users and groups, file shares, shared databases and secure backups.

Eradication

This phase involves removal of ransomware, backdoors malware and patch vulnerabilities exploited during the initial attack vector so that there is no re-infection. Restoration of devices is also a part of the initial phase. This phase involves the following steps:

- Take forensic images of the infected devices and system for later investigation use
- Restore devices from clean backups
- Confirm that enterprise security solutions are up to date and patched

- Deploy custom signatures to enterprise security solutions based on discovered IOCs
- Run an AV scan on infected machines and clean them

Communication

Communication is a very important step for an organization during a ransomware incident. Communication to authorities, clients, employees, insurance providers, security and IT vendors and press is required. All laws require organizations to notify clients in the event of a security breach that involves PII. Moreover, you should communicate with internal and external legal counsel, including compliance auditors, regulators and law enforcement agencies. When you start notifying customers, be very careful. Organizations should work with law enforcement to ensure the timing of the communication, lest it should impede the investigation. Notifying and involving local or federal law enforcement agencies is a good practice as in some cases, they can recover keys or recover the ransom and repay the company. Incident response teams within an organization must communicate with security tools (firewall and so on) and IT vendors and collaborate with them to block the IOC, secure the network and do any forensic investigation.

Designate a point of contact within your organization to provide updates and explain how individuals should protect themselves based on applicable regulations. Organizations should also contact the cyber insurance provider to check for eligibility and understand the claims process.

Recovery

Regardless of all mitigation steps described in this book, no organization or individual can be 100 percent immune to ransomware attack until they have a solid backup recovery plan. Paying the ransom is not recommended as ransomware gangs don't offer any guarantee, and you can't trust criminals. Also, paying a ransom makes your organization a target for new attacks from other ransomware gangs.

During this phase, you can consider the following:

- Have a business continuity/disaster recovery plan(s) ready for starting the recovery and ensure business continuity.

- Recover data from clean backups and make sure they are not infected with malware or backdoor. By checking for indicators of compromise or team, we can also consider partial recovery and backup integrity testing. We will cover a backup strategy to recover from ransomware later in this chapter.
- All administrator and user passwords are changed.
- Find and try known decryptors for ransomware strain that infected your organization using resources.
- If your organization decides on paying the ransom, make sure to consider ramifications with appropriate stakeholders, and finance, legal, regulatory, and insurance implications. Also, you can hire an experienced ransomware negotiator to reduce the ransom amount.
- Also, organizations can consider having a ransomware attack insurance policy considering any future re-infection scenarios.

Post-incident

It is a very important step as it can prevent any future infections or malware attacks. During this process, the focus should be on the lessons learned and on defining a remediation plan. A “*lessons learned*” meeting with all involved parties must take place after recovering from the incident. The focus of this should be to formulate a remediation plan and understand the following points:

- What document, process and tool were used?
- What document, process or tool was missing?
- How the team would react differently if this incident happens again.
- How information was shared and what could be improved.
- How this could have been prevented.
- Review and revise the ransomware policy as needed.
- Review and update IT disaster recovery plans.
- Request an “*indicators of compromise*” audit of the whole network from an external IT security vendor for locating and removing any leftover malware, related files, or remnants. A thorough post-incident

evaluation conducted independently can boost confidence that the ransomware or backdoor has been eradicated.

- Review and update data backup strategy and cyber security awareness policy.

Ransomware mitigation strategies

According to research, every organization should focus on developing strategies for lowering ransomware attacks and an incident response strategy that is prepared for any eventuality. In reality, most businesses that were the targets of cyber attacks were forced to comply with the ransom requests. The way we are prepared to respond to the threat will have a significant impact, so we must acknowledge that a ransomware assault is not a matter of “if” but “when.” Following is a list of strategies that anybody can use to mitigate the ransomware threats.

Endpoint security solutions

Organizations worldwide are at risk from external and internal threat actors like employees, nation-states hackers, hacktivists and organized crime. Endpoint security solutions are frontline soldiers as they represent one of the first elements that one should consider for protecting against ransomware attacks. Endpoint security is the methodology of securing endpoints or end-user devices like PCs, laptops, and cell phones from being exploited by threat actors and malware like ransomware. Endpoint security solutions defend endpoints in an enterprise or home network or in the cloud from ransomware threats.

Why are endpoint security solutions so important?

Day by day, the threat landscape is becoming more complicated, as threat actors are finding new ways to infiltrate, gain access, and steal information or influence people into giving out confidential information. Sum in the cost of reallocating resources from business goals to addressing digital threats: the reputational cost of a data breach, the cost of recovering encrypted data and the financial penalty of compliance violations, it is easy to understand why endpoint security solutions are becoming a prerequisite in terms of securing enterprises or personal devices. Data is the reason why endpoint

security solutions are given so much importance. It is the most valuable asset, and losing it or its access could cause bankruptcy or financial losses.

However, the endpoint security solutions market is highly crowded as there is a wide variety of vendors with distinctive solutions to protect your devices from ransomware threats. Some solutions are for large organizations, while others are for small and mid-sized organizations and home users. The fundamental objective is to lower the attack surface of cyber attacks against endpoint devices, concentrating mainly on ransomware threats.

We will cover different types of endpoint security solutions and network security solutions. Some solutions work for home users and others for enterprises. Endpoint security solutions that we will be focusing on are as follows.

Endpoint antivirus solutions

Endpoint antivirus is a security solution designed to help detect, prevent and eradicate malware on devices. Endpoint antivirus solutions are installed on endpoint devices like desktops, laptops, network servers and mobile phones. Traditional antivirus solutions have big databases of virus signatures and definitions. They detect malware by scanning the memory and finding patterns that match virus signatures and definitions from the database. Antivirus vendors must research new kinds of malware so that their signatures and definitions can be added to the databases. If the databases are not updated, then the endpoint antivirus will be unable to detect new malware, leaving the user exposed to new threats.

If an antivirus finds the malware on an endpoint, it will automatically block, quarantine or remove it from the endpoint, or it will prompt the user that malware has been found and ask them to take action to eradicate the threat. Traditional antivirus solutions also notify users for updating virus definitions and vulnerable software.

Next-generation endpoint antiviruses have artificial intelligence and machine learning that can help organizations keep pace with new sophisticated threats like ransomware. They also offer automation functions that can help security staff overwhelmed by the time and skill level needed to effectively manage threats.

Endpoint antivirus software's are available from different vendors, with versions designed for personal use, small businesses, and large organizations. Some vendors also offer free antivirus solutions; nevertheless, they might lack the necessary features to prevent advanced malware threats like ransomware. Any endpoint antivirus solutions must have the following capabilities:

- They must allow running scans at scheduled intervals and manually.
- They must automatically block threats when a user visits a malicious website.
- They should have a user-friendly console and must support automatic updates for protection against the latest threats.
- They must have the capability to identify the type of malware and must not slow endpoint performance.

Antivirus solutions alone are ineffective against modern ransomware attacks. You should keep in mind that having an antivirus program is not guaranteed protection against ransomware.

Endpoint Detection and Response (EDR)

Endpoint Detection and Response (EDR), also known as **Endpoint Threat Detection and Response (ETDR)**, is a unified endpoint security solution that incorporates real-time threat monitoring at endpoints for analysis and collection of data with rules-based automated response capabilities.

The fundamental functions of an EDR security solution are as follows:

- **Endpoint data collection:** Program agents perform endpoint monitoring and collect data related to system processes, connections, data transfer, processor and disk activity and upload them to a server or a central database.
- **Automated response:** Preconfigured rules in the solution can recognize indicators of compromise and trigger an automatic response, such as logging out the end user or quarantining the device and sending an alert to the IT team.
- **Analysis and forensics:** An EDR solution may incorporate real-time analytics for rapid diagnosis of threats that preconfigured rules might

ignore and cyber forensics tools for threat hunting or conducting a forensic analysis of a cyber attack.

- **Real-time analytics:** Engine employs algorithms to assess and correlate large volumes of data, searching for patterns.
- **Digital forensics:** Tools enable IT security teams to inspect past breaches to better understand how vulnerability was exploited to infiltrate into the endpoint. IT security teams can use this collected data to hunt for threats, such as malware or other backdoors that might slide undetected on an endpoint.

Managed Detection and Response (MDR)

Managed Detection and Response (MDR) refers to outsourced cyber security services tailored to protect your data and assets even if a threat evades standard enterprise security tools.

As the sophistication of cyber attacks advances exponentially, organizations cope to maintain security operation centers staffed with extremely skilled personnel and resources. As a result, MDR companies offer cost-effective services aimed to strengthen the defenses and reduce risk without an upfront cyber security investment.

MDR service providers offer cyber security experts and teams with advanced tools and up-to-the-minute global threat intelligence databases, which are beyond the reach and cost-effectiveness of most small organization budgets. Thus, it helps organizations to keep up with continually evolving threat actor **tactics, techniques, and procedures (TTPs)**. MDR tools provide an alternate solution for organizations aiming to get the latest cyber security tools by integrating Endpoint Detection and Response tools. The organization's staff does not need to manage threat monitoring, detection and response. The risk analysis gets better without the effort and expense required to keep the staff fully trained with the latest tools.

Extended Detection and Response (XDR)

Extended Detection and Response (XDR) is a threat detection and incident response solution that is SaaS-based. It combines different security solutions

into an integrated security operations console that unifies all licensed solutions.

With threat actors like hacking groups, nation states and even potentially malicious insiders constantly looking out for opportunities to infiltrate enterprises, security teams have to overcome challenges like too many disconnected security tools and event data for each one of them. Security teams struggle with huge data sets with too many false positives and little help for integrating this data in one place. This is where XDR solutions come into the picture by integrating all tools and offering advanced protection capabilities to an organization. An **Extended Detection and Response (XDR)** solution should offer the following capabilities:

- Threat detection and response to advanced ransomware attacks
- User behavior analysis and technology to protect against insider threats
- Local threat intelligence, along with intelligence from different threat intelligence sources
- Reduction of false positives by automatically correlating and analyzing alerts
- Integration of relevant data from different tools for faster, more accurate incident triage and forensics
- A centralized configuration console and infrastructure hardening capability
- Analysis of threat vectors detected by different tools
- Security orchestration and automation to streamline SOC processes
- Knowledge base and support of the cyber security team

Endpoint protection platform (EPP)

An **Endpoint Protection Platform (EPP)** is an integration of endpoint protection technologies like antivirus, data loss prevention, and intrusion prevention system and data encryption. Its objective is to detect and stop various threats at the endpoint level, for everything from smartphones to printers. EPP offers a data sharing framework between different endpoint protection technologies. This data sharing framework offers more effective security as this ability to communicate with different endpoint protection technologies can stop advanced ransomware attacks.

The main characteristics of any EPP are as follows:

- Ability to detect threats and remediate them
- Collecting real-time threat data from global sources and locally from the organization
- A robust integration framework capable of sharing information between different third-party security solutions
- A user-friendly centralized console for managing the information gathered from all security solutions

Endpoint operating system hardening

Endpoint system hardening or operating system hardening is a technique of securing endpoints like PCs, laptops, and servers via reducing their attack surface or surface of vulnerability and potential attack vectors. It is a kind of cyber security protection that involves closing vulnerabilities that threat actors can exploit to infiltrate and gain access to confidential information. The following are the different techniques of operating system hardening.

Display file extensions

Digital files have two elements to their names: the descriptive name of the file and its extension. The file extension is used by the device to know which applications can be used to open the file. By default, extensions are hidden in Microsoft Windows. Hidden extension makes recognizing potentially malicious files difficult. For example, threat actors normally change the icon of the executable ransomware file with PDF icon and change the extension to **filename.pdf.exe**. The victim, who is unaware, opens the executable file thinking it is a PDF file as the extension would be hidden. It is very important to display file extensions by default.

How to display file extensions in Windows 10 and 11

To display file extensions in Windows 10 and 11, follow the given steps:

1. Open a File Explorer window or any folder and select the **view** tab from the top menu.
2. On the right-hand side, you will see a checkbox called File name extensions.

3. Select the checkbox; you should now be able to see the file extensions of all the files by default.

Refer to [Figure 14.1](#):

Figure 14.1: Display File extensions

Disable AutoPlay

Microsoft Windows AutoPlay detects when a removable storage device is connected to a PC and performs preset actions like opening the removable storage file explorer. Threat actors exploit this feature to execute malware or ransomware.

This type of attack is called USB baiting or USB drop attack. It is a kind of social engineering attack and is performed by planting USB sticks containing malware at places where the victims can pick them up and later connect them to their PC just out of curiosity to find out the contents or the owner of the drive. With the Auto play enabled, these malicious USB sticks can be used to infect the device with ransomware.

You can disable AutoPlay so that no action will be taken after you connect to a USB drive. To disable AutoPlay, go to Settings or control panel or use the group policy feature for an enterprise network.

To disable via settings, search for Auto play in the search box. You can also configure it to take no actions or ask what it should do when a USB drive or Memory card is inserted.

Follow the given steps:

1. Toggle the switch to enable or disable AutoPlay.
2. You can also choose default actions for different types of removable media. This helps when you personalize it based on the type of removable drive or memory card.
3. The recommended option is to ask you every time. This way, you will get a notification if any malware tries to run from a USB media.

Refer to [Figure 14.2](#):

Figure 14.2: Disable Autoplay

Disable Remote Desktop protocol (RDP)

Microsoft Windows offers an excellent feature of Remote Desktop, which allows establishing connections with physically distant Windows computers. This remote desktop function works via **Remote Desktop Protocol (RDP)**. This feature is very useful, but it presents security risks such as ransomware and uninvited guests. It is recommended that users disable Remote Desktop or use very strong passwords.

The following are some of the security risks presented by the RDP:

- RDP can be exploited using brute-force attacks, which can later be used to implant ransomware.
- Remote desktop connections are not encrypted, so man-in-the-middle attacks can exploit this to intercept communications.
- Another big risk is credential harvesting attacks. Using these attacks, RDP logins and passwords can be harvested and sold on the dark web.
- Malicious Power Shell scripts can be installed on the system to exploit RDP.

Disabling remote access to computer in Windows 10/11

You can disable Remote Desktop manually by editing the Windows registry or via settings by following these steps:

1. Type remote settings in the Windows search box.
2. Choose Allow remote access to the machine to open the Control Panel's Remote System Properties dialog pane.
3. Check Don't Allow Remote Connections to disable Remote Desktop.

Software restriction policies (SRP) And AppLocker

Software Restriction Policies (SRP) are Microsoft Windows features that allow users to control the execution of software. SRPs are a part of the Group Policy feature that controls the execution of an application. SRPs work just like a set of firewall rules that can allow or deny the execution of specific applications. The types of SRP rules include zone, path, certificate, and hash. An SRP policy can prevent malware attacks like ransomware.

Windows AppLocker is a new security function that replaces the Software Restriction Policies feature for controlling software usage. It was first introduced in Windows 7, and it can control which applications and files users can run. These files may be Windows Installer files, executable files, scripts, **dynamic-link libraries (DLLs)**, packaged apps or installers. When a user runs an application or process, it has the same level of rights and access to data that the user has. Ransomware exploits this function and encrypts, deletes or transfers data to the command and control server if a user knowingly or unknowingly runs a malicious application or file.

This danger may be reduced by using the AppLocker feature, which limits the files or programs that individuals or groups are permitted to launch. Nowadays, many applications may be installed without administrator access. Ransomware uses the same Windows features to avoid conventional app control strategies that rely on users' incapacity to install apps. Using AppLocker's white listing feature, it is possible to stop per-user programs from being launched. Using Group Policy, you may activate and customize both these functions based on your operating system.

Disable Windows Script Host

Windows Script Host (WSH) is a scripting language that the Windows operating system supports. A WSH script is written in VBScript or other languages like JavaScript and Perl. These scripts are used for automating normal Windows tasks. Threat actors use social engineering attacks to influence end users in executing a script for downloading ransomware or malware payloads. WSH scripts can be a very big risk for an organization if exploited by threat actors. Normally, threat actors send phishing emails with .zip file attachments or containing JScript files, and then run via Windows script host if clicked. Our recommendation is to edit your Windows Registry and disable WSH.

To disable Windows Script Host, type **regedit.exe** in the Run box to open the Registry Editor, and follow the given steps:

1. Navigate to the following key:
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows Script Host\Settings.
2. In the right panel, you will see Enabled. If you see the entry 1, it means that the Windows Script Host is enabled.

3. Double-click on it and give it Value Data 0 to disable it. Click on **OK**.
4. If you do not see this registry entry, then you will need to create it.

Disable Server Message Block (SMB)

Windows employs the **Server Message Block (SMB)** protocol to share printers, files and serial ports. Servers make file systems and resources accessible to clients, clients make SMB requests for resources, and servers respond with SMB queries. Ransomware gangs exploit SMB protocol to propagate ransomware across networks. Thus, disabling SMB protocol outbound traffic and removing outdated versions of SMB is an important step to prevent network-wide ransomware infection. Network administrators should use SMBv3 or the current versions and disable SMBv1 and v2, considering they have mitigated any existing dependencies on old versions. Moreover, it is important to block SMB from being accessible externally to the network by blocking TCP port 445, UDP ports 137–138 and TCP port 139.

Secure domain controllers (DCs)

Domain Controller (DC) is a type of computer server that handles authentication requests and validates users within the domain of an enterprise network. For granting access to domain resources, the DC acts as a gatekeeper as it implements safety guidelines, maintains user data, and validates user identities for domains. Ransomware gangs normally exploit DCs to spread ransomware in the network. It is important to consider the following for securing domain controllers from ransomware attacks:

- Regularly patch the domain controller and use the latest version of Windows Server OS.
- Remove any unwanted applications from the servers, as they can increase the attack surface.
- SMB signing should be enforced throughout the entire domain, between hosts and the DCs to prevent the use of replay attacks on the network. SMB signing, also known as server message block signing, is a Windows feature that enables digital signatures at the packet level to prevent manipulation and man-in-the-middle attacks.

- DC host firewalls should be configured to prevent internet access to reduce the attack surface.
- It is recommended to consider the following in DC Group security Policy:
 - Although using the Kerberos default authentication protocol is advised, enabling NTLM auditing will make sure that only NTLMv2 replies are being transmitted over the network. If at all feasible, steps should be taken to guarantee that LM and NTLM responses are rejected.
 - To stop code injection that can steal system credentials, enable extra safeguards for local security authentication.
 - The Administrators group should be the only group with access to DCs. Users inside this group should be restricted and have separate, non-administrative accounts that are used for daily activities.

Restricting the use of PowerShell

Microsoft's PowerShell is a Windows OS tool for task automation and configuration management that combines a command-line shell with a scripting language. Ransomware gangs exploit PowerShell to launch a ransomware infection, so it is very important to restrict its usage on the network devices. You can disable PowerShell for certain users using a group policy or security policy or Disable access PowerShell ISE.

The following recommendation must be taken into consideration while configuring PowerShell:

- Only users or administrators who manage the network or machines should be allowed to use PowerShell.
- Update to the latest version of PowerShell and uninstall all earlier versions.
- Ensure that PowerShell instances have module, script block, and transcription logging enabled. The past OS and registry changes, and **Tactics, Techniques, and Procedures (TTPs)** used by ransomware gangs will be stored in PowerShell logs. So, it is very important to

enable enhanced logging with PowerShell Windows Event Log and the PowerShell Operational Log capturing all the changes.

Network security solutions

Network security solutions are very important for protecting enterprise data and ensuring its confidentiality, integrity and availability. These solutions keep data secure, ensure reliable access with good network performance and provide protection from cyber threats. A good network security solution should protect an organization from costly losses that occur from a data breach or other security incidents, by ensuring secure access to systems, applications and servers. Network security solutions include hardware and software solutions as well as processes or policies and network configurations solutions. However, the network security solutions market is extremely crowded, like the endpoint security solutions market, and there is a wide variety of vendors with different solutions to detect and stop threats from infiltrating into an organization. The main aim of network security solutions is to reduce the surface for attacks against the network.

We will be covering distinct network security solutions that can bolster enterprise network defenses against ransomware attacks.

Next Generation Firewall (NGFW)

A traditional firewall does stateful analysis of the network traffic, based on state, port, and protocol, and it allows or blocks network traffic based on previously defined rules. A **next-generation firewall (NGFW)** does all the functions of a traditional firewall, and much more. A NGFW can offer access control and block threats like advanced malware-like ransomware and application-based threats. Some other features of NGFWs are URL blocking, and **Virtual Private Networks (VPNs)** – with **Quality of Service (QoS)** functionality, packet filtering, **Network Address Translation (NAT)** and **Port Address Translation (PAT)**, intrusion prevention, SSL and SSH inspection, deep-packet inspection, application awareness and reputation-based malware detection.

NGFWs can block **Advanced Persistent Threats (APTs)** as they can be integrated with threat intelligence sources and can block malware before it enters a network. Using an NGFW, network administrators can block

unauthorized application traffic via network security policy. This can help maintain an application's blacklist and whitelist, along with blocking network traffic at protocol and port level. For example, the RDP protocol, which is exploited by many malware, can be blocked entirely across the network via NGFW.

NGFW can integrate active directory or host-based firewalls to apply security policies for users or groups. Host-based firewalls are firewall agents installed on endpoints to monitor incoming and outgoing traffic of the device, run malware scan and allow security policies' implementation from a centralized console. For example, the Microsoft Windows Defender Firewall is a host-based firewall. NGFWs can greatly strengthen organizations' defenses against ransomware attacks.

Network sandboxing

Sandboxing technology is used by malware researchers; it is an intensive effort that requires advanced skills. The malware reverse engineer manually runs the malware in the sandbox, which is a quarantine environment, and analyzes it to understand TTPs of the malware. Network sandboxing solutions automate this manual process. Network-based sandboxing allows detecting malware and zero-day attacks. Network sandboxes analyze network traffic for malicious objects and indicators of compromise to automatically submit them to the sandbox. In the network sandbox, these threats are analyzed and assigned malware probability scores and severity ratings.

The sandbox is designed such that it appears as a fully-functioning end-user environment to the potential malware and it coaxes malware into running its routines — executing, downloading other files, connecting to URLs, and so on. The sandbox monitors the suspicious file behavior and network traffic for allowing it to execute, blocking it, or quarantining it for further investigation. Many NGFW vendors offer a cloud sandboxing function on a subscription basis, where suspicious files are sent to the sandbox to analyze its static and dynamic behavior.

Remember, threat actors often try to detect and evade a sandbox — or discover and exploit vulnerabilities in that sandbox during their attack. Network sandboxes are integrated with endpoint and network security solutions to round out ransomware detection and response capabilities.

Intrusion detection system (IDS)

IDS should be regarded as a diagnostic solution because it tracks and finds malicious behavior throughout a network. If the system discovers an issue, it will notify the security team so that they may look into it.

There are five types of IDS that leverage two types of detections:

- **Signature-based detection:** IDS systems based on signatures notify administrators when a specific attack or malicious activity is detected.
- **Anomaly-based detection:** The goal of anomaly-based detection is to find activity that deviates from a predetermined baseline.

IDS types differ depending on where threats are monitored and how they are discovered. They can be used to identify ransomware spread in the network.

- **Network intrusion detection systems (NIDS):** A network intrusion detection system will track traffic using various sensors that are installed either through hardware or software on the network.
- **Host intrusion detection systems (HIDS):** In order to monitor traffic, an HIDS is installed directly on devices, providing network managers a little more freedom and control.
- **Protocol-based intrusion detection systems (PIDS):** Frequently installed in front of a server, a protocol-based IDS keeps track of traffic going to and coming from devices.
- **Application protocol-based intrusion detection systems (APIDS):** A protocol-based system is comparable to an APIDS. However, APIDS monitors particular application protocols to explicitly monitor activity.
- **Hybrid intrusion detection systems:** Hybrid intrusion detection systems combine the other methods of intrusion detection.

Intrusion prevention system (IPS)

In terms of detection, an IPS is similar to an IDS system, but it also has reaction capabilities. When a possible attack, malicious behavior, or an unauthorized user is detected, an IPS solution takes action.

The specific functions of an IPS vary depending on the type of solution, but having one in place is beneficial for automating activities and controlling ransomware threats without the need of continuous monitoring.

- **Network-based intrusion prevention system (NIPS):** NIPS monitors and protects a network from unusual or suspicious activities.
- **Wireless intrusion prevention system (WIPS):** WIPS are used to monitor wireless networks to provide better targeted detection and response.
- **Host-based intrusion prevention system (HIPS):** An HIPS monitors all traffic passing through and coming from the host to detect and respond to malicious activities.
- **Network behavioral analysis (NBA):** NBA seeks unusual behavior inside network patterns, making it critical for incident detection.

Data Loss Prevention

Data Loss Prevention (DLP) solutions defend against incidents of data loss and help in data leakage prevention. Data loss refers to an incident where confidential data is lost, such as in a ransomware attack. Data loss prevention focuses on preventing confidential data from being transferred outside the organizational bounds. DLP tools check the content as well as the context of the data sent. A robust DLP solution gives the information security team full visibility into all network data, including the following:

- **Data in use:** Using user authentication and access control to secure data utilized by an application or endpoint.
- **Data in motion:** Encryption and/or other e-mail and messaging security procedures ensure the secure transfer of sensitive, confidential, or proprietary data over the network.
- **Data at rest:** Access controls and user authentication are used to protect data stored on any network location, including the Cloud. DLP also allows businesses to classify business-critical information and verify that their data policies conform with the relevant requirements.

There are three types of DLP:

- **Network DLP:** All data in use, in motion, or at rest on the company's network, including the cloud, is monitored and protected.
- **Endpoint DLP:** It monitors all endpoints, including servers, PCs, laptops, mobile phones, and any other device on which data is utilized, transported, or saved.

- **Cloud DLP:** It is a variant of Network DLP specially designed to safeguard enterprises that store data in cloud repositories.

Honeypot

A honeypot is a realistic decoy solution designed to catch the attention of hackers and entice them to launch an attack. Honeypots are instruments for monitoring, risk mitigation, and early warning. A honeypot is essentially a bogus system that first convinces a hacker that it is authentic and then convinces them to initiate an attack against it. This enables IT pros or MSPs to gain a better understanding of attacker motives, behavior, and strategies. This contributes to the reinforcement of cyber security tactics and processes to prepare for true assaults. Companies utilize honeypots to obtain information and insights on their cyber security weaknesses and the types of attacks they encounter.

A honeynet consists of one or more honeypots, which are internet-connected computer systems. They are specifically designed to attract and snare threat actors attempting to infiltrate other people's computer systems. A honeynet typically hosts genuine apps and services to seem to be an actual production network and become a suitable target. However, because the honeynet does not really service any authorized users, every attempt to contact the network is deemed an illegal effort to break its security. As a result, any outbound activity is a probable indication that a system has been hacked.

There are several open-source honeypots available. Open-source honeypots are one sort of deception technology that may be installed quickly and for free. Dionaea, Kippo, Cowrie, Honeything, and Conpot are a few examples.

DNS (Domain Name System) Security

DNS is a protocol that converts a human readable domain into an IP address when you use the internet. It makes it easy for individuals to use the internet by easily accessing websites and business application. Unfortunately, the DNS protocol is not very secure as traditional security solutions seldom monitor DNS packets. As a result, many malware, including ransomware, exploit this feature and communicate to their server over the DNS layer. Most ransomware attacks include data exfiltration. Prior to encryption, the software uses DNS tunnels to send business-critical data from the client's

network to the threat actor. This gives the threat actor more power over their victim: instead of merely losing their data, corporations now face the potential of having their data published publicly or sold to the highest bidder on the dark web.

DNS security is the broad notion of safeguarding the DNS service as a whole. DNS security examines a query's purpose. It uses DNS data and DNS query traffic to assure security. DNS security as a strategic instrument is integrated into a network security strategies such as filters, DNSSEC, firewalls, or on-device agents. DNS security gives detailed insights into traffic.

DNS-layer security extends beyond data collection; a robust security posture should also defend networks from threats. DNS security solutions help in two ways: by preventing clients from connecting to suspicious domains, thereby preventing attacks from starting; and by detecting unusual DNS-layer activity that could indicate an ongoing attack, thereby allowing security teams to isolate infected systems and mitigate damage.

There are many DNS-based threat blocking solutions that provide malware, ransomware, and anti-phishing security. The following are the most well-known:

- OpenDNS (<https://www.opendns.com>), 208.67.222.222 to 208.67.220.220
- Comodo (<https://www.comodo.com/secure-dns>), 8.26.56.26 to 8.20.247.20
- Yandex.DNS (<https://dns.yandex.com/advanced>), 77.88.8.88 to 77.88.8.2
- Quad9 (<https://www.quad9.net>), 9.9.9.9

For optimal security, businesses should consider installing a secure in-house DNS server or subscribing to a paid threat-blocking DNS service with more extensive features than free ones.

Security information and event management (SIEM)

Security Information and Event Management (SIEM) software assists IT security professionals in protecting their organization network against

threats. An SIEM system collects log data from all infrastructure components. Once the logs have been collected within the SIEM software, they are normalized using various analytical approaches, including log correlation and machine learning algorithms.

SIEM includes two functions: SIM and SEM:

- **Security Information Management (SIM):** SIM is the compilation of all network operations. This can include log data gathered from servers, firewalls, domain controllers, routers, databases and netflows as well as unstructured data found in the network, such as emails.
- **Security Event Management (SEM):** SEM refers to data analysis. For any anomalous activity, the data is examined using multiple approaches, alarms are raised, and/or a process is started.

The goals of data analysis and correlation are as follows:

- Prevent data breaches by detecting danger indications early on.
- Discover unusual user activity patterns to detect complex assaults and respond fast.
- Generate real-time warnings for every security event discovered.
- Assist businesses in meeting IT requirements.
- Perform forensic analysis and expedite post-incident recovery.
- Monitor all network events to resolve IT operations issues and ensure network security.

The interpreted data collected from the mentioned methodologies is displayed in the form of bar graphs or pie charts, which allows security administrators to make decisions quickly and easily.

A SIEM system can be extremely beneficial in preventing ransomware attacks. In the event of a ransomware attack, SIEM can assist at several stages of infection, including detecting the following:

- Execution parameters the ransomware runs with
- Privilege escalation
- Disablement of Windows behavior scanning
- High-frequency file deletion, process cessation and service cessation
- High-frequency generation of ransomware notes

- Alteration of wallpaper
- Techniques for persistence
- Extension files for ransomware
- Volume shadow copy deletion
- Data leakage

Security Orchestration, Automation, and Response (SOAR)

SOAR is a cyber security solution that enables organizations to optimize cyber security operations. It focuses on vulnerability management, threat hunting, security operations' automation and automated incident response. Security automation, in a nutshell, is the automated management of security operations-related duties. It is the process of carrying out these duties without human intervention, such as scanning for vulnerabilities or looking for logs. Security orchestration is a technique for linking and combining various security systems. It is the linked layer that automates security and simplifies security operations.

An ideal SOAR solution should be able to do the following:

- Ingest and evaluate data and warnings from multiple security systems
- Develop, build, and automate procedures required by teams to detect, prioritize, investigate, and respond to security alarms
- Enhance operations, orchestrate and integrate a wide range of technologies
- Possess forensic capability to conduct post-incident analysis, allowing teams to improve their procedures and avoid repeat problems
- Ensure automation to minimize repetitive activities and allow teams to save time and focus on more complicated duties that require human input

Ransomware infection can also be stopped using SOAR solution in the following ways:

- SOAR helps with security orchestration and automation by identifying advanced threats like ransomware. The most important step to take

during a ransomware attack is to detect and stop it from spreading. SOAR's capabilities are well-known for enhancing threat hunting and providing a significantly quicker incident reaction time.

- SOAR can orchestrate and automate the process of examining every suspicious email without requiring human participation, and experts will only be alerted if there is any risky activity and will be involved in the process as soon as possible.
- SOAR's playbooks take care of repetitious and menial tasks, such as analyzing emails, attachments, URLs, and other potentially dangerous actions, reducing the need for manual assessment of cyber risks. While SOAR does the mundane work, analysts can concentrate on more significant inquiries.
- It ensures improved incident response time. In the struggle to thwart ransomware attacks, only seconds can be significant. Furthermore, by collecting data from alerts into a uniform platform, SOAR enables the whole SOC team to take early and appropriate actions to mitigate ransomware attacks, significantly boosting incident reaction time.

When a ransomware infects your machine, SOAR identifies the intrusion process and promptly isolates the malware in its early stages, limiting additional damage and keeping the organizational effect to a minimum. SOAR depends on pre-defined incident response playbooks for speedier execution once the ransomware attack has been contained.

Zero Trust Network Access (ZTNA)

Zero Trust is a cyber-security strategy that safeguards an enterprise by removing implicit trust and continually verifying every phase of a digital connection. Zero Trust is developed to secure modern digital ecosystems and is based on the philosophy of "never trust, always verify." It represents a transition from the "trust but verify" philosophy to "never trust, always verify." No user or device is allowed to access a resource under the Zero Trust paradigm until their identity and credentials are validated.

The Zero Trust model requires rigorous authentication and authorization for every device and user before it may access or transmit data on a network, regardless of whether it is within or beyond the network's perimeter. In addition, the method integrates analysis, filtering, and logging to validate

behavior and continuously monitor for signs of intrusion. If a user or device behaves differently than usual, it is noted and monitored as a possible threat.

Stop ransomware attacks

Adopting zero-trust architecture is one method of preventing ransomware attacks. Here are a few ways that zero trust may aid your enterprise in protecting against ransomware:

- Zero trust architecture reduces the attack surface by making applications invisible to adversaries. All apps remain private and hence, invisible to adversaries while using ZTNA. Extending this strategy to all applications and devices in your IT infrastructure makes ransomware operators' job nearly impossible.
- All traffic, including the encrypted one, is subject to detailed scrutiny under zero-trust architecture.
- Zero Trust solution prevents lateral movement of computers, servers, software and cloud through an organization's network. By preventing this lateral movement across the IT infrastructure, ransomware can be securely halted even if the perimeter has been infiltrated.
- Intruders are prevented from abusing workloads by zero trust architecture. Security policies are implemented in a zero trust architecture based on the identification and authorization of the workloads, attempting to interact with one another. These identities are regularly validated; unauthorized workloads are prevented from communicating with others. This means malware cannot communicate with remote command and control servers or with internal hosts, users, apps, and data.

Least privilege account

The IT staff must grant local administrator access to corporate users in order for them to run permitted and essential programs. Once privileges are granted, they are seldom removed, and organizations may find that many of their users now have local administrator privileges. This privilege opens a security vulnerability associated with excessive privileges and renders organizations more exposed to attacks, even if they feel they are well-protected. Organizations may control this "*privilege creep*" and guarantee

that human and non-human users only have the minimum levels of permission necessary by using least privilege access restrictions.

A user is only provided the minimal levels of accessor permissions required to carry out their job tasks under the **Principle of Least Privilege (PoLP)**, which is a notion in cyber security. It is commonly considered as a best practice in cyber security and is a crucial step to safeguarding privileged access to highly sensitive data and assets. Beyond human access, least privilege can be applied to applications, systems, and linked devices that require rights or permissions to perform a necessary operation. Least privilege enforcement makes sure a device only has the minimal access required.

A fundamental element of zero trust frameworks is the idea of least privilege. The entire cyber attack surface could be lowered, and compliance and audits could be made simpler by PoLP. By imposing least privilege on endpoints, malware threats are blocked from using elevated rights to gain more access and move laterally, therefore damaging the device or installing or executing malware.

Implementing role-based access control can help reduce these attacks. In this framework, permissions can be assigned to a role. Users' accounts can have more than one role assigned, and people can have more than one user account.

Refer to [Figure 14.3](#) for an illustration of the Least Privilege Account:

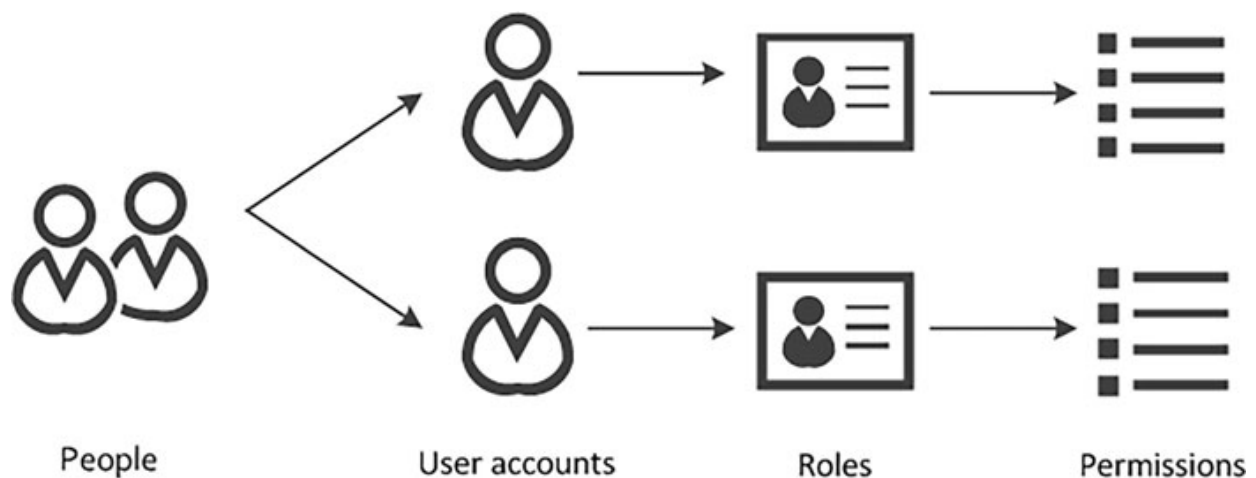


Figure 14.3: Least Privilege Account

Patch management

A piece of code created specifically to address a flaw or provide a new functionality in software is known as a software patch. The process of regularly releasing software updates, or patches, is known as patch management. By doing this, the software's vulnerabilities are fixed, new features are added, or the existing ones are shielded from threat actors' exploitation. By addressing software security flaws, security updates prevent hackers and other threat actors from exploiting a system. Ransomware attacks that target network flaws in enterprises are on the rise as cybercriminals realize that patching issues, such as incomplete patches and poor patch management, are prevalent in enterprises and provide the possibility of financial gain.

Organizations can learn from different ransomware incidents worldwide and take steps to ensure security of their IT infrastructure. In the future, we may anticipate seeing more and "better" ransomware protection. The most important lesson to be learned from the numerous ransomware incidents is that running security updates as soon as they become available will help prevent being a victim of ransomware when any vulnerability is exploited.

The main problem is that many organizations lack the resources, including personnel and technology, needed to update systems and applications with the latest patches released by original equipment manufacturers and software companies. Making sure that every server and computer in an organization has the latest patches will be difficult and time-consuming. Manually applying patches is not only time-consuming but also poses a significant danger to organizations. Although applying patches is required, doing so without testing them might have disastrous effects for IT infrastructure. It is strongly advised that organizations have a strategic plan and patch management solution that carefully balances the patch priorities and effectiveness. The following points must be considered before buying comprehensive patch management solution:

- Select centralized patch management software that can automate the process.
- Before deploying fixes, test them in a pilot environment to see if they cause potential failures and crashes.

- Consider their seriousness and systematically deploy the most crucial fixes first.

Given your business constraints and architectural needs, selecting the appropriate patch management software is crucial.

Environment hardening

The practice of examining networked systems' applications, determining their vulnerabilities, and putting defenses in place to lessen known vulnerabilities and attack surfaces is known as environment hardening. An organization can improve its connected environment inside a corporate setting by following a five-step process:

1. **Making use of virtual local area networks (VLAN):** By using broadcasting domains, VLANs divide a single physical network into several logical networks that may be utilized independently, each with its own set of hardware. This ensures that the network's resources and parts are segregated into several levels. These methods are frequently used to prioritize and manage traffic, allocating large amounts of bandwidth to the services that require it the most. By serving as a buffer between secure zones and vulnerable ones and isolating them, VLANs enhance network security.
2. **Network segmentation:** Network segmentation is a successful defense against ransomware because it reduces the consequences of network intrusion and makes it difficult for an attacker to locate and reach the segment that contains critical data. During network segmentation, a big network is divided into several smaller parts using firewalls, routers, VLANs, and other network division strategies. Network segmentation will not only isolate sensitive information into safe segments but also increase the time it takes for intruders to identify and map the target network after the breach, making them more vulnerable to being detected by firewalls and intrusion detection systems. If a ransomware successfully infects one endpoint in a network without segmentation, it can spread to all other endpoints on the same network.
3. **Hardware updates:** Updates for software and hardware are published frequently by manufacturers in the quickly growing field of information security. These updates often fix newly found and well-

known vulnerabilities that might compromise user data privacy. To guarantee that equipment performs to its full potential, vendors offer upgrades. Upgrading the network infrastructure itself might play a significant role in addition to software, hardware, and firmware, since problems with outdated technology often cannot be resolved by software upgrades.

4. Using ZTNA, NGFW, and other solutions mentioned earlier.
5. **Vulnerability management tools:** Unpatched vulnerabilities are used by ransomware gangs to infiltrate an organization. The process of finding, prioritizing, mitigation and disclosing vulnerabilities in IT infrastructure and the software that runs on it is known as vulnerability management. It helps prioritize potential risks and reduce the attack surface. Normally, the process is manual, but with vulnerability management tools, it is automated. These tools will inventory a range of devices on a network and detect vulnerabilities on them using a vulnerability scanner. Once vulnerabilities have been discovered, it is necessary to assess the risk they offer in various situations to make decisions regarding the best mitigation action. Any vulnerability management program typically has four steps:
 - a. **Discovery:** List every piece of IT equipment and software that your company has; software for IT inventory management will make this process easier.
 - b. **Reporting:** By scanning your IT devices using a vulnerability scanner tool, you can report the vulnerabilities discovered for each one.
 - c. **Prioritization:** Put the vulnerabilities found in order of severity.
 - d. **Remediate:** Start with the most important vulnerability and implement remediation to fix it.
 - e. **Verify:** Use further scans and/or IT reports to validate the remediation.
 - f. **Report:** Finally, it is important for the C-suite, executives, and IT department to be aware of the risk associated with the present vulnerabilities. Company executives need an overview of the current state of security vulnerabilities (think red/yellow/green

type tracking), and the board of directors need high-level risk analysis reports across the entire enterprise.

To simplify and automate the vulnerability management process, many commercial tools offer a comprehensive vulnerability management process with the main aim of improving the security of IT infrastructure. Organizations must choose wisely if they want a comprehensive or merely a vulnerability scanning/assessment tool.

Enterprise Mobility Management (EMM)

The technique of protecting enterprise data on mobile devices belonging to the employees or company is known as **Enterprise Mobility Management (EMM)**. While incorporated with other corporate IT technologies and software to provide a wide range of enterprise features, EMM systems usually provide a comprehensive set of services intended to protect an organization's intellectual assets and confidential information.

EMM solutions differ greatly from one another. Some are focused on safeguarding certain programs, while others aim to totally lock down the device by controlling applications to prevent any kind of data leakage and wiping the device if it is lost or stolen.

EMM offers a single platform for managing smartphones, email, apps, data, browsing, and more. It also offers a customizable method of managing smartphones or a secure profile on phones to accommodate various scenarios and organizational demands.

Components inside an EMM Solution

Enterprise Mobility Management uses a wide range of diverse and ever-evolving technology and solutions, listed as follows:

- **Mobile expense management (MEM):** MEM analyzes mobile communication costs and provides a company with information about the use of the mobile calling and the mobile data used so that employees can be reimbursed based on corporate policies. MEM is very useful for charge-backs and audits of mobile plan usage.
- **Mobile Device Management (MDM):** Using the profiles put on each device, MDM is employed to control smart phones. This makes it

possible to operate devices remotely, encrypt content, establishments, and wipe out some programs and data from a phone if it is misplaced, stolen, or used by an employee who has left the company.

- **Mobile information management (MIM):** The MIM service, which is typically a component of MDM or MAM services, handles remote database accessibility from smartphones and integration with the numerous cloud storage and productivity services like Dropbox and AWS.
- **Mobile identity management (MIM):** In order to guarantee that only registered users and trustworthy devices may access business assets. MIM is responsible for authentication, login, identity, certificates and code signatures.
- **Mobile content management (MCM):** MCM is in charge of managing content on mobile devices, including information access, security, content delivery to devices, and file-level content protection. For each user's access and data authorization, several MCM technologies integrate seamlessly with well-known cloud storage services.
- **Mobile application management (MAM):** MAM is primarily concerned with installing, administering, and upgrading the mobile apps used by a company. Sending updates, licensing control, and ensuring application security are some of the MAM features that make it possible to control and remove certain apps when they are no longer in use. MAM is gaining popularity as a technique to apply regulations and protection measures to particular programs and associated information without having to erase the data on a smart phone entirely.

Different EMM solutions are widely available in the market right now. Some offer basic BYOD security features, while others are MAM-only. The organization must search for an all-encompassing EMM security solution.

[End user level security](#)

The responsibility of data security extends beyond IT specialists. An end user's simple error can bring down secure networks. Information security at the user level is a priority for smart organizations. You can teach end users some of these tactics to successfully prevent ransomware attacks.

Virtualization technology

A user may safeguard a device and data from malware threats like ransomware by using virtualization technologies. A **virtual machine (VM)**, as the term indicates, is a virtual environment that mimics a real physical machine. Despite depending on the actual hardware, VMs contain their respective CPU, network interface, memory and storage. In one physical machine, many virtual machines can operate without crashing. Sandboxing is also supported by virtual machines that allow different operating systems to be kept apart in a sandbox so that they cannot influence one another. It serves as a type of security for situations when one operating system or program is infected with malware. You can separate the virtual machine from the host machine such as a sandbox and avoid using any services that may damage the host computer, including network files, shared folders, and drag-and-drop between the two computers.

Because the virtual machine is isolated from the host machine, malware in the virtual machine cannot infect your host PC. Virtual machines are an excellent technique to safeguard your host system from malwares: you may access suspicious websites or execute dangerous applications in the virtual machine, knowing that any harm will be limited to the virtual machine, which can always be backtracked if required. As the virtual machine runs in a sandbox mode, a user can safely execute software, open email attachments, and visit compromised internet sites without fear of damaging the host machine. One more important feature of VM is that many malware do not execute in a virtualized machine as malware developers fear VM are used by malware reverse engineers so they can configure parameters in code to detect whether the machine is a virtual machine.

Virtualization has generally developed into a potent tool in computing and IT, and there are several free and paid virtualization tools available, including Parallels Desktop, VMware Workstation Player, Microsoft Hyper-V and Citrix Hypervisor. By default, Windows supports a sandboxing feature that can help with the same; this feature can be enabled in machines.

Disable macros in Office files

Microsoft Office macros boost user performance by automating complex and repetitive operations in Microsoft Office applications, particularly in

Excel and Word, using the **Visual Basic for Applications (VBA)** programming language. On the computer of the unwary user, this function can be used to execute ransomware. The latest editions of Microsoft Office come with macros disabled by default; avoid enabling them in documents you download from the Internet or emails from unreliable sources. Phishing emails are frequently sent by malicious attackers with malicious documents. When users open the emails, depending on the attachment, they might see instructions for enabling macros.

Because of the potential for nefarious schemes involving macros, Microsoft Office default security settings block them. Enabling macros enables a macro that is hidden in the office document to run. This macro establishes a connection to a certain threat actor server to download code that subsequently downloads the ransomware. When accessing any office document with macros enabled, a user should exercise great caution. One can use Office viewer software or the Google Docs online viewer to access any Office files delivered by email to mitigate this risk.

[Cyber security awareness training](#)

Viewing the data about data breaches and ransomware incidents in the last 5 years, human error is the biggest threat to cyber security. In order to survive in the current digital era, your organization must implement a cyber security awareness program. This is the most effective way to shield your organization from ransomware threats. An effective cyber security awareness program will teach employees about information security risks and also help them understand how to respond to attacks to prevent any data exfiltration. Every employee needs to be aware of the different social engineering attacks, such as phishing, smishing and vishing that can be potential points of entry that threat actors could use to gain access to their enterprise network. A security awareness program will teach them how to guard against threats and protect access to information assets. This will help define each employee's IT security responsibilities in accordance with their work roles to safeguard the IT infrastructure and sensitive information from threat actors. Additionally, it will greatly reduce incidents where employees download email malicious attachments or click on malicious links and not pass the responsibility to email scanning and malware blocking software.

E-mail security

Email is the most popular method for distributing ransomware and is the most used attack vector used by threat actors to infiltrate any organization. Email security is very important since, if any user clicks on the links inside the email body or opens an attachment, it could cause a big havoc. Phishing emails spread ransomware when users click on the malicious link or open a malicious attachment. Thus, lowering the email attack surface should be the priority for preventing ransomware attacks. The following are some recommended techniques for doing this:

- **Stop email harvesting:** The first technique is to make sure that threat actors do not get your email so that they cannot send you phishing emails. There are several ways to prevent email harvesting. Here are a few:
 - If your organization lists emails of important or VIP executives on its website, make email addresses difficult for automated bots to read. Your organization can use a structure, such as **[Last Name]. [First Name]@companydomain.com**, instead of mentioning it completely. Only humans can understand and drive the email address by reading the executive name; automated bots will not be able to harvest the email IDs.
 - Your organization can also post email addresses of the executives in images rather than text so that automated harvesters cannot read them.
 - **Temporary email account:** Users should have three email addresses: one for personal use, the second one for the offers and subscriptions, and third one for business. When a company is breached, its customers' emails can be leaked, and threat actors can use these leaked data to get your email address. There are many websites that offer temporary email addresses for days or hours; you can use those services to register for some non-essential online services. Users should not register using their personal or professional email addresses so that it prevents threat actors from getting your email address.
- **Disable HTML and code injection in emails:** Preventing HTML code injection in emails can stop malicious scripts from launching as soon as

you receive an email. Enabling the email preview window and “*disallowing remote content*” in the opened message in your email client can protect you from malware attacks.

- **Configuring email security protocols:** Properly configuring email protocols like **Domain-based Message Authentication, Reporting and Conformance (DMARC)**, **Sender Policy Framework (SPF)** and **Domain Keys Identified Mail (DKIM)** can stop spoofing of your organization domain. Email spoofing is a technique used by threat actors to conceal the true recipient’s address in a malicious email and replace it with a legitimate one while assuming the identity of a company or user by using an authentic domain name. This technique is frequently used by ransomware gangs to evade spam filters and give emails a more trustworthy appearance. DKIM uses a digital signature, which is added to the email, to verify that the mail was not altered after it was sent. To accept the email, this signature should match the email domain’s public key. The SPF protocol confirms that an email originated from its source and is actually sent from that domain. The DMARC protocol uses DKIM and SPF. Domain administrators can publish their DKIM and SPF criteria using DMARC and describe what will happen if an email does not comply with those criteria. These technical solutions can stop spoof emails. However, use of additional email security solutions like email security gateways, monitoring solutions, antimalware, and anti-spam is always recommended to stop advanced ransomware attacks.
- **Multi Factor Authentication (MFA):** It refers to employing multiple methods to confirm a user’s identity. For instance, a username and password in addition to a one-time password or a fingerprint, face or voice biometric might be used. This additional authentication step can provide an extra line of protection against common email threats like brute-force and password cracking attacks. Organizations and employees should enable MFA options whenever they can, for different platforms to protect data.
- **Password policy:** Many users have difficulty managing passwords as there are dozens of them that each one has to remember. Using strong passwords is one of the top cyber security best practices. Organizations should promote the adoption of robust, single sign-on passwords by employees, the use of secure password managers, and frequent

changing of passwords. It is recommended to have a long password rather than a very complex one. It is advised that password strength be primarily determined by length rather than complexity. Passphrases, which are created by combining a few words, like **110vemyparents@**, are one way to create passwords that are longer, simpler to remember, and more difficult to guess. This helps prevent cybercriminals from employing dictionary attacks to target weak passwords.

- **Data classification:** By using a data classification policy, an organization can manage data that is confidential and non-confidential. This can guarantee that sensitive information is handled appropriately in the organization. Using a data classification policy, employees can tag and classify files and data according to the level of sensitivity and permission to view; and edit or forward them after considering all types of organization-owned data. These files and data can be classified, for example, into sensitive, public, private, and personal. Organizations may minimize their total risk against ransomware attacks using data classification strategies and prevent exfiltration of data by controlling who has access to a certain file and how sensitive documents are distributed through the network.
- **Advanced spam filtering solutions:** These programs may safeguard your data as well as IT infrastructure from cyber security threats that can be triggered by emails. Emails with malicious links and attachments are sent by threat actors, and malware or malicious programs can propagate across your network if you or your employees become victims of these threats. However, you can secure your data and IT infrastructure if you have implemented advanced spam filtering solutions. These solutions can identify and block advanced email threats like ransomware.

Backup strategy

The best defense against ransomware is to restore data from recent backups. In the event of a ransomware attack, authorities and security experts advise against paying the demanded sum since there is no assurance that your files will be returned. As part of a comprehensive ransomware defense strategy, a backup and recovery plan can assist you in safeguarding your data and avoiding the need to pay a ransom by deploying backup solutions. You may

swiftly and effectively restore data that is essential to your organization and go back to business as usual. In other words, keeping secure backups that are inaccessible to adversaries is essential. Regular audits should be done of all in-premise and cloud data backup solutions in place. Following are the best backup practices as part of a comprehensive ransomware defense strategy.

[Auditing backup policies](#)

Regularly auditing your backup plans and processes will ensure that backups are thorough and reliable. While auditing backup policies, enterprises should consider the following:

- Do all essential systems receive frequent, automated backups?
- Has the enterprise had experience using backups to restore important systems?
- Are we following the 3-2-1 backup policy rule, which calls for retaining 3 backup copies on 2 different media types and 1 copy outside the company?
- To stop ransomware from reaching backups, do we isolate and secure backup systems?

[Encrypting backup data](#)

A data backup strategy should ideally include encryption because it is a potent method of protecting sensitive data. Backup data can be encoded using an encryption algorithm. Using a secret key, data can be written and read or processed. If an unauthorized person or ransomware gang tries to access the data, they will not be able to read it without the encryption key. Make sure your backup strategy also ensures encryption of data in rest and in transit.

[Immutable storage](#)

Data storage that does not allow data deletion or modification is referred to as immutable storage. Also known as object locking or **Write-Once-Read-Many (WORM)**, immutable storage is supported by several cloud service providers and backup solutions. Backups can be locked by organizations for a set amount of time, prohibiting users from deleting or modifying them during this retention period. During the immutable retention term, backups

cannot be removed—even if a hostile actor or ransomware is able to gain root credentials. Following are some essential qualities that enterprises should consider when choosing this technology:

- Select a backup solution that lets you specify a reasonable retention time or offers enough capacity to satisfy compliance needs.
- Backup solutions should offer policy-based scheduling that foresees and triggers alarms when backups deviate from the retention policy.
- Ensure that point-in-time backups are accessible during the retention period and that the backup solution supports encryption to safeguard files by default.

[Air gap backup](#)

An air gap is a security measure that prevents devices in network or computer systems from connecting to one another. In critical infrastructure environments like nuclear power plants, air gapped network infrastructure are used, as the risk of connected devices to the internet is very high. This guarantees complete isolation of a critical system from other networks, especially unprotected ones, physically, electromagnetically, and electrically. For air gapped networks, physical devices with an air gap technique, such as an external hard disk, are used for data backup. The same technique can be used by enterprises for backup solutions by not depending solely on cloud backup solutions. As cloud backup solutions may not be enough to secure data from ransomware, it is advised to keep an offsite copy of the data in a storage medium that is disconnected from all networks.

[Use the 3-2-1 backup rule](#)

The 3-2-1 backup approach requires you to maintain a minimum of three unique copies of your data at all times. The production data or the original data is the first copy, and backup copies are the other two. These three should all be stored and configured in a way that ensures their integrity and availability at all times. The data should be identical in each of these copies, that is, the same data from the same point in time.

Secondly, you should have at least two copies of your data on storage systems that are physically separate from one another. One may be on a

network-attached storage device or file server, while the other might be on cloud storage or flash drive that is not at all connected to the server.

The third requirement of the 3-2-1 backup rule is that at least one copy of your data should be stored offsite, that is, physically separated from other copies; this can be on a different company premises.

Utilizing hybrid backup is the recommendable best practice of the 3-2-1 backup method. In a hybrid backup, data is simultaneously backed up to local storage and the cloud using only one program and one procedure.

Ascertain backup coverage

Make sure your backup solution includes the entire enterprise IT infrastructure so that following an enterprise-wide ransomware attack, the IT team can restore data from backup. Backup solutions should support endpoints, NAS, servers, cloud infrastructure, different operating systems and legacy systems.

Test the backup plan

Test each backup and recovery strategy. Calculating recovery durations and whether or not you can recover easily using this strategy, you should think about the following issues:

- Which systems would you give top priority for restoration?
- Will your organization's rehabilitation need new and distinct networks?
- Will you need to quarantine devices for digital forensic purposes?

Disaster recovery plans must be regularly audited and tested to make sure backups are production-ready and can support the organization's **Recovery Point Objective (RPO)** and **Recovery Time Objective (RTO)**.

Conclusion

In this chapter, we revised what we learned in the earlier chapters. We learned about Ransomware Incident response plan. Organizations are more likely to experience repeated ransomware attacks if they do not have a response strategy in place that includes measures to stop future data breaches. In this chapter, we talked about the steps involved in an Incident response plan, from identification to containment, eradication,

communication, and recovery and we looked at what needs to be done post-incident.

Then, we also covered ransomware mitigation strategies, which involve solutions from end security solutions, network security solutions and end user level security. In end security solutions, we discussed security solutions designed to help detect, prevent and eradicate malware on end devices. In network security solutions, we talked about solutions that keep data secure, ensure reliable access with good network performance and provide protection from cyber threats. Lastly, in the end user level security section, we looked at solutions that specifically focus on the end user.

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

<https://discord.bpbonline.com>



[Index](#)

Symbols

3-2-1 backup approach [325](#)

A

Address Space Layout Randomization (ASLR) [86](#)

Advanced Encryption Standard (AES) [58](#)

Advanced Persistent Threats (APTs) [306](#)

air gap [325](#)

algorithms [57](#)

Antivirus (AV) [245](#)

asymmetric algorithms [61](#), [62](#)

attack vector [33](#)

Autoruns [209](#)

Avast ransomware decryptor [17](#)

AVG ransomware decryptors [14](#)

B

backup solutions [41](#)

 offline backup solutions, versus online backup solutions [41](#), [42](#)

backup strategy [323](#)

 3-2-1 backup approach [325](#), [326](#)

 air gap [325](#)

 ascertain backup coverage [326](#)

 backup data, encrypting [324](#)

 backup plan, testing [326](#)

 backup policies, auditing [324](#)

 immutable storage [324](#), [325](#)

Bitcoin (BTC) [25](#)

BitDefender ransomware decryptor [18](#)

block cipher [60](#)

Burp Suite [211](#), [212](#)

C

CIA [55](#)

cipher [57](#)

Command & Control (C&C) server [73](#), [74](#)

Comodo [310](#)

Critical Zone Table (CZT) [51](#)

CryptAcquireContext function [82](#)

- dwFlags [85](#)
- dwProvType [84](#)
- example [86-89](#)
- phProv [83](#)
- szContainer [83](#)
- szProvider [83](#)
- CryptExportKey function [100](#), [101](#)
 - dwBlobType [102](#)
 - dwFlags [106](#)
 - hExpKey [102](#)
 - hKey [102](#)
 - pbData [106](#)
 - pdwDataLen [106](#)
 - PlainTextBlob example [107-112](#)
 - SimpleBlob example [112-120](#)
- CryptGenKey function [89](#), [90](#)
 - AlgId [90](#)
 - dwFlags [90](#)
 - example [91-95](#)
 - hProv [90](#)
 - phKey [91](#)
- CryptGetUserKey function [95](#), [96](#)
 - dwKeySpec [96](#)
 - example [96-100](#)
 - hProv [96](#)
 - phUserKey [96](#)
- CryptImportKey function [121](#)
 - dwDataLen [121](#)
 - dwFlags [122](#)
 - example [122-125](#)
 - hProv [121](#)
 - hPubKey [121](#), [122](#)
 - pbData [121](#)
 - phKey [122](#)
- Crypto API [82](#)
- cryptocurrency [24](#), [25](#)
- cryptographic interceptors [50](#)
- Cryptographic Service Provider (CSP) [82-87](#)
- cryptography [55-58](#)
- cryptography algorithms [59](#)
 - asymmetric algorithms [61](#), [62](#)
 - hybrid encryption method [62-64](#)
 - symmetric algorithms [59](#), [60](#)
 - symmetric encryption algorithm [60](#)
- CryptoLocker ransomware [73](#), [74](#)
- cryptomining [26](#), [27](#)
- current solution analysis
 - Microsoft control folder access [52](#), [53](#)
 - PayBreak [50](#), [51](#)
 - redemption-anti-ransomware [51](#), [52](#)

- R-Locker [50](#)
- custom binary
 - creating [128](#), [129](#)
- Cutter [204](#)

D

- data at rest [55](#)
- data in transit [55](#)
- Data Loss Prevention (DLP) [308](#)
 - Cloud DLP [308](#)
 - data at rest [308](#)
 - data in motion [308](#)
 - data in use [308](#)
 - Endpoint DLP [308](#)
 - Network DLP [308](#)
- debuggers [205](#)
 - x32dbg/x64dbg [205](#)
- decryptor [24](#)
- Defense Advanced Research Projects Agency (DARPA) [27](#)
- Dependency Walker [209](#), [210](#)
- disassemblers
 - Cutter [204](#)
 - Ghidra [203](#), [204](#)
 - Interactive Disassembler (IDA) [201-203](#)
- Distributed Denial of Service (DDoS) attack [289](#)
- DMARC protocol [322](#)
- DNS (Domain Name System) Security [309](#), [310](#)
 - Comodo [310](#)
 - OpenDNS [310](#)
 - Quad9 [310](#)
 - Yandex.DNS [310](#)
- Domain Controller (DC) [304](#)
- Domain Keys Identified Mail (DKIM) [322](#)
- DOS header [132](#)
- DOS stub [133](#), [134](#)
- double extortion [288](#)
- dynamic analysis [200](#), [201](#)
 - memory checking [221-241](#)
- dynamic analysis tools
 - debuggers [205](#)
 - disassemblers [201](#)
 - monitors [206](#)
- dynamic behavior-based solutions [44](#)
 - entropy-based products [45](#)
 - honeyfile products [46](#), [47](#)
 - machine learning products [45](#), [46](#)
- Dynamic Link Libraries (DLL) [129](#), [161](#), [302](#)

E

- EDR security solution
 - fundamental functions [297](#)
- e-mail security
 - advanced spam filtering solutions [323](#)
 - data classification [323](#)
 - email harvesting, preventing [321](#)
 - HTML and code injection, disabling [322](#)
 - Multi Factor Authentication (MFA) [322](#)
 - password policy [323](#)
 - security protocols, configuring [322](#)
 - temporary email account [322](#)
- email spoofing [33](#)
- Emsisoft ransomware decryptors [15](#)
- encryption [56](#), [57](#)
- encryption algorithm [58](#)
- endpoint antivirus solutions [296](#), [297](#)
 - Endpoint Detection and Response (EDR) [297](#)
 - Endpoint Protection Platform (EPP) [299](#)
 - Extended Detection and Response (XDR) [298](#), [299](#)
 - Managed Detection and Response (MDR) [298](#)
- endpoint operating system hardening [299](#)
 - AppLocker [302](#)
 - AutoPlay, disabling [300](#), [301](#)
 - domain controllers (DCs), securing [304](#)
 - file extensions, displaying [300](#)
 - PowerShell usage, restricting [304](#), [305](#)
 - Remote Desktop protocol (RDP), disabling [301](#), [302](#)
 - Server Message Block (SMB), disabling [303](#)
 - Software Restriction Policies (SRP) [302](#)
 - Windows Script Host (WSH), disabling [303](#)
- Endpoint Protection Platform (EPP) [299](#)
 - characteristics [299](#)
- endpoint security solutions [295](#), [296](#)
- Endpoint Threat Detection and Response (ETDR) [297](#)
- end user level security [319](#)
 - backup strategy [323](#), [324](#)
 - cyber security awareness training [321](#)
 - e-mail security [321](#)
 - macros, disabling in Office files [320](#)
 - virtualization technology [319](#), [320](#)
- Enterprise Mobility Management (EMM) [318](#)
 - mobile application management (MAM) [319](#)
 - mobile content management (MCM) [319](#)
 - mobile device management (MDM) [318](#)
 - mobile expense management (MEM) [318](#)
 - mobile identity management (MIM) [318](#)
 - mobile information management (MIM) [318](#)
- entropy-based products [45](#)

- environment hardening [316](#)
 - process [316](#), [317](#)
- ESET ransomware decryptor [14](#)
- Ethereum (ETH) [26](#)
- exploit [31](#), [32](#)
- exploit kits [34](#), [35](#)
- Export Address Table (EAT) [168](#), [172](#)
- Export Name table (ENT) [168](#)
- Export Ordinal table (EOT) [168](#)
- Extended Detection and Response (XDR) [298](#), [299](#)

F

- fake encryptor [288](#)
- file alignment [131](#)
- file encrypter ransomware [287](#)

G

- Ghidra [203](#), [204](#)
- Graphical User Interfaces (GUIs) [289](#)

H

- honeyfile products [46](#), [47](#)
 - limitations [47](#)
- honeypot [309](#)
- Human Interface Device (HID) [35](#)
- hybrid approach
 - used, in ransomware [64-66](#)
- hybrid encryption method [62-64](#)

I

- IDA Pro [202](#)
- iFrames [37](#)
- IMAGE_SECTION_HEADER structure [145](#)
 - Characteristics [148](#), [149](#)
 - PointerToRawData [147](#)
 - SizeOfRawData [146](#)
 - VirtualAddress [146](#)
 - VirtualSize [146](#)
- immediate remedial actions, ransomware infection
 - action plan [9-19](#)
 - infected computer, disconnecting [7](#)
 - scope of infection, checking [7](#)
 - type of ransomware, checking [8](#), [9](#)
- Import Directory [170](#)
- Indicators of Compromise (IoCs) [42](#), [290](#)
- Initialization Vector (IV) [277](#)

- Interactive Disassembler (IDA) [201-203](#)
- intrusion detection system (IDS) [307](#)
 - anomaly-based detection [307](#)
 - application protocol-based intrusion detection systems (APIDS) [307](#)
 - host intrusion detection systems (HIDS) [307](#)
 - hybrid intrusion detection systems [307](#)
 - network intrusion detection systems (NIDS) [307](#)
 - protocol-based intrusion detection systems (PIDS) [307](#)
 - signature-based detection [307](#)
- intrusion prevention system (IPS) [307](#)
 - host-based intrusion prevention system (HIPS) [308](#)
 - network-based intrusion prevention system (NIPS) [307](#)
 - network behavioral analysis (NBA) [308](#)
 - wireless intrusion prevention system (WIPS) [308](#)

J

- Jigsaw ransomware analysis [270](#)
 - static analysis [270-281](#)

K

- Kaspersky ransomware decryptor [12](#), [13](#)
- key management [67-69](#)
- key on attacker network [72-77](#)
- key on hacker machine [72-77](#)
- key on victim machine [70-72](#)
- keyspace [58](#)

L

- law enforcement agency [25](#)
- leakware [288](#)
- Load Effective Address (LEA) [253](#)
- LockCrypt [9](#)
- LockCrypt 2.0 ransomware analysis
 - CryptEncrypt function [263](#)
 - CryptExportKey function [255](#)
 - CryptGenKey function [254](#)
 - CryptImportKey function [253](#)
 - dynamic analysis [249-266](#)
 - static analysis [244-249](#)

M

- machine learning products [45](#), [46](#)
- malvertising [36](#), [37](#)
- malware [23](#)
 - types [23](#), [24](#)
- Managed Detection and Response (MDR) [298](#)

- Master Boot Record (MBR) [287](#)
- MBR encrypter [287](#)
- McAfee Ransomware Decryptor [11](#)
- McAfee Ransomware Recover (Mr2) [11](#)
- Microsoft control folder access [52](#), [53](#)
- monitors
 - Autoruns [209](#)
 - Burp Suite [211](#), [212](#)
 - Dependency Walker [209](#), [210](#)
 - Process Explorer [207](#), [208](#)
 - process monitors [206](#), [207](#)
 - Wireshark [210](#), [211](#)

N

- National Security Agency (NSA) [203](#)
- Network Address Translation (NAT) [305](#)
- network security solutions [305](#)
 - Data Loss Prevention (DLP) [308](#)
 - DNS (Domain Name System) Security [309](#), [310](#)
 - Enterprise Mobility Management (EMM) [318](#)
 - environment hardening [316](#), [317](#)
 - honeypot [309](#)
 - intrusion detection system (IDS) [307](#)
 - intrusion prevention system (IPS) [307](#)
 - least privilege account [314](#)
 - network sandboxing [306](#)
 - next-generation firewall (NGFW) [305](#), [306](#)
 - patch management [315](#), [316](#)
 - SIEM [310](#)
 - SOAR [311](#), [312](#)
 - Zero Trust Network Access (ZTNA) [313](#)
- next-generation firewall (NGFW) [305](#), [306](#)
- no key management technique [69](#), [70](#)

O

- OpenDNS [310](#)

P

- patch management [315](#)
- PayBreak [50](#), [51](#)
- payload [31](#), [32](#)
- PE header structure [134](#), [135](#)
 - data directories [141-144](#)
 - FileAlignment [139](#)
 - File header [135](#)
 - NumberOfRvaAndSizes [141](#)
 - Optional header [136](#), [137](#)

- SectionAlignment [138](#)
- Section table [145](#)
- SizeOfHeaders [139](#), [140](#)
- SizeOfImage [139](#)
- subsystem [140](#)
- PE sections [152](#), [153](#)
 - .data [160](#)
 - export section [161-169](#)
 - import section [169-175](#)
 - PE file, loaded in memory [176-179](#)
 - .rdata [158](#), [159](#)
 - .rsrc section [155-158](#)
 - .textbss section [153](#)
 - .text section [154](#)
- phishing scam [33](#)
- PLAINTEXTKEYBLOB [106](#)
- Portable Executable (PE) [128](#)
 - custom binary [128](#), [129](#)
 - DOS header [132](#)
 - DOS stub [133](#), [134](#)
 - structure [129-131](#)
- Port Address Translation (PAT) [305](#)
- Principle of Least Privilege (PoLP) [314](#)
- PRIVATEKEYBLOB [105](#)
- Process Explorer [207](#), [208](#)
- process monitor [206](#), [207](#)
- proprietary algorithms [57](#)
- PUBLICKEYBLOB [104](#)

Q

- Quad9 [310](#)
- Quality of Service (QoS) [305](#)

R

- RaaS business models
 - affiliate business model [30](#)
 - licensing, for lifetime [30](#)
 - subscription-based [30](#)
- randomness [58](#)
- ransomware [3](#)
 - building blocks [21](#)
 - defining [22](#), [23](#)
 - hybrid approach, using [64-66](#)
 - key management [67](#)
- Ransomware as a Service (RaaS) [28](#), [289](#)
 - working [28-30](#)
- ransomware decryption key [73](#)
- ransomware decryptor [9](#)

- ransomware distribution methods
 - email [33](#)
 - email spoofing [33](#)
 - exploit kits [34](#), [35](#)
 - malvertising [36](#), [37](#)
 - phishing scam [33](#)
 - spear phishing [33](#)
 - unsolicited advertisements [33](#)
 - USB and removable media [35](#), [36](#)
- ransomware incident response plan [286](#), [287](#)
 - analysis [287-289](#)
 - communication [293](#)
 - containment [292](#)
 - eradication [293](#)
 - identification [287](#)
 - impact of ransomware attack, assessing [291](#)
 - initial infection vector, identifying [291](#)
 - post-incident [294](#), [295](#)
 - recovery [294](#)
 - scope of ransomware infection, determining [290](#), [291](#)
 - strain of ransomware, determining [289](#), [290](#)
- ransomware infection [3](#)
 - immediate actions [7](#)
 - symptoms [4-6](#)
- ransomware mitigation strategies [295](#)
- ransomware, stages [37](#)
 - data encryption [38](#)
 - destruction of backup [38](#)
 - infection and exploitation [37](#)
 - notification and extortion [38](#)
 - ransomware delivery [38](#)
- RansomWarrior [72](#)
- Recovery Point Objective (RPO) [326](#)
- Recovery Time Objective (RTO) [326](#)
- redemption-anti-ransomware [51](#), [52](#)
- Relative Virtual Address (RVA) [137](#)
- Remote Desktop Protocol (RDP) [301](#)
- REvil ransomware decryptor [18](#)
- Rivest Shamir Adleman (RSA) key pair [77](#)
- R-Locker [46](#), [50](#)

S

- scareware [69](#), [70](#)
- screen locker [288](#)
- secret key [59](#)
- section alignment [131](#)
- Security Information and Event Management (SIEM) [310](#), [311](#)
 - Security Event Management (SEM) [310](#)
 - Security Information Management (SIM) [310](#)

- Security Orchestration, Automation, and Response (SOAR) [311](#), [312](#)
- Sender Policy Framework (SPF) [322](#)
- Server Message Block (SMB) [49](#)
- SHA256 value [42](#)
- SIMPLEBLOB [102](#), [103](#)
- Small and Medium Enterprises (SME) [215](#)
- smartphone ransomware [289](#)
- Software Restriction Policies (SRP) [302](#)
- solutions, ransomware attacks [40](#)
 - backup solutions [41](#)
 - dynamic behavior-based solutions [44](#)
 - static or signature-based solutions [42-44](#)
- spear phishing [33](#)
- static analysis [184-220](#)
 - host infection [185-189](#)
 - key generation [190-193](#)
 - ransom demand [194](#), [195](#)
 - user data encryption [193](#)
- static analysis tools [195](#)
 - CFF explorer [196](#)
 - PE Studio [196](#)
- static or signature-based solutions [42-44](#)
- stream cipher [60](#)
- symmetric algorithms [59](#), [60](#)
- symmetric encryption algorithm [60](#)
- symmetric key [59](#), [69](#)

T

- tactics, techniques, and procedures (TTPs) [298](#)
- The Onion Router (TOR) [27](#), [28](#)
 - URL [27](#)
- threat actors [31](#)
- Transport Layer Security (TLS) encryption [74](#)
- Trend Micro Ransomware Decryptor [10](#)
- triple extortion [288](#), [289](#)

U

- unsolicited advertisements [33](#)
- USB and removable media [35](#), [36](#)
- user awareness trainings [48](#)

V

- virtual machine (VM) [319](#)
- Virtual Private Networks (VPNs) [305](#)
- Visual Basic for Applications (VBA) [320](#)
- vulnerability [31](#), [32](#)
- vulnerability management cycles [49](#)

vulnerability management tools [49](#)

W

WannaCry ransomware [76](#), [77](#)

Wildfire decryptor [13](#)

wiper/disk erasure [288](#)

Wireshark [210](#), [211](#)

Write-Once-Read-Many (WORM) [324](#)

X

x32dbg/x64dbg [205](#)

Y

Yandex.DNS [310](#)

Z

Zero Trust Network Access (ZTNA) [313](#)

ransomware attacks, preventing [313](#), [314](#)