v2

Threat Hunting Professional

Introduction to Malware Hunting

Section 03 | Module 02

https://t.me/learningnets

© Caendra Inc. 2020
All Rights Reserved

Table of Contents

MODULE 02 | INTRODUCTION TO MALWARE HUNTING

https://t.me/learningnets

2.1

Introduction

https://t.me/learningnets

2.1 Introduction

During our threat hunting missions on endpoint systems, we will be hunting malware in various forms: exe's, dll's, ps1's, etc.

In order for us to successfully hunt malware, we need to fully understand what it is, how it infects an endpoint, techniques it uses to hide within the endpoint, etc.

https://t.me/learningnets

2.1 Introduction

What is malware?

Malware is the short form of **mal**icious soft**ware**.

It is software written to infiltrate or damage a computer system without the owner's consent. It can be considered one of the following: intrusive, hostile, and/or annoying.

https://t.me/learningnets

2.1 Introduction

We will not focus on annoying malware, such as adware and/or PUPs (Potentially Unwanted Programs).

We will be looking at intrusive and hostile forms of malware.

https://t.me/learningnets

2.2

Malware Classifications

https://t.me/learningnets

2.2.1 Viruses

A computer **virus** is a program that copies itself and spreads without the permission or knowledge of the owner. Viruses do not spread via exploiting vulnerabilities (the ones that do that are called **worms**).

The only way viruses are supposed to spread is with the host, at least in their rigorous classification. Let's say that a virus has infected a file; now if the owner moves the file to a different system, the virus has a chance to spread and survive.

https://t.me/learningnets

2.2.1 Viruses

Viruses can be classified into the following sub-types:

- Resident
- Non-resident
- Boot Sector
- Multi-Partite

https://t.me/learningnets

2.2.1 Viruses

Resident

When the virus is executed and becomes memory resident. It waits for some trigger, such as loading another program. It then infects other programs and so on.

Non-Resident

When the virus is executed, it will search for files it can infect. After infecting them, it will quit. When the infected program is run again, it will again find new targets to infect and so on.

https://t.me/learningnets

2.2.1 Viruses

Boot Sector

Spreads via boot sectors. For example, if a user leaves an infected CD-ROM in the drive while turning off a system. The next time the system boots up, the boot sector virus will activate and will spread to the hard disk, which will then spread to USB flash drives. When the flash drives are moved, the cycle repeats.

Multi-Partite

These viruses have several types of infection mechanisms. They can have both **Boot-Sector** and **Resident** type viruses or even more.

https://t.me/learningnets

2.2.2 Worm

Worms are basically software which uses network/system vulnerabilities to spread themselves from system to system. They are typically part of other software, such as rootkits, and are normally the entry point into the system. They basically compromise the system (locally or remotely) and provide access to other malware.

https://t.me/learningnets

2.2.3 Rootkits

A **rootkit** is malware which is designed to hide the fact that a compromise has occurred, or to compromise the system at a deeper level. A rootkit is basically used as a supplement to other malware.

Rootkits can be used to:

- Hide processes
- Add files to the file system
- Implement backdoors
- Create loopholes

https://t.me/learningnets

2.2.3 Rootkits

When a rootkit is installed, the entire operating system is compromised. Rootkits exists for all major operating systems. They are installed as drivers (or kernel modules).

They are known to exist at the following levels (even lower levels are possible):

- Application Level
- Library Level
- Kernel Level
- Hypervisor Level
- Firmware Level

https://t.me/learningnets

2.2.3 Rootkits

Application Level

They actually replace programs with copies of other programs.

Library Level

Let’s say that 10 applications are sharing a library. Taking control of the library means taking control of all 10 applications.

https://t.me/learningnets

2.2.3 Rootkits

Kernel Level

This is the most common type. They are known for their resistance to removal, since they run at the same privilege level as antivirus software.

Hypervisor Level

Modern processors have support for virtualization. Rootkits which use such processor specific technologies are called hypervisor rootkits, such as **blue pill** and **SubVirt**.

https://t.me/learningnets

http://theinvisiblethings.blogspot.com/2006/06/introducing-blue-pill.html
https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/subvirt.pdf

2.2.3 Rootkits

Firmware Level

Rootkits for firmware such as BIOS, ACPI tables, and device ROMS are known to exist. They have the highest chance of survival because currently no tools exist to verify/scan for firmware level rootkits.

https://t.me/learningnets

2.2.4 Bootkits

Bootkits differ from rootkits in their installation process and how/when they take control of the operating system.

Bootkits start attacking the operating system before the operating system has started. They are able to completely violate the security of the target operating system.

https://t.me/learningnets

2.2.5 Trojans

A **trojan** (or **trojan horse**) is a kind of malware that appears to the user to perform a legitimate function, but actually facilitates unauthorized access to the owner's machine.

An example of a trojan would be when you install a game you downloaded from the Internet onto your machine, but it might contain additional malicious code that is not part of the game. While you're playing the game, the secondary code would execute to perform its malicious activity.

https://t.me/learningnets

2.2.6 Backdoors

A **backdoor** is software (or modification of software) which helps in bypassing authentication mechanisms. They can keep remote access open for later unauthorized purposes while trying to remain hidden.

For example, a backdoor in a login system might give you access when a specified username/password is entered, even though the credentials might not be a valid combination. **RATs** (**Remote Access Trojans**) are similar to backdoors.

https://t.me/learningnets

2.2.7 Remote Access Trojans

A **Remote Access Trojan (RAT)** is a malicious remote administration tool, which an attacker uses to issue commands to the compromised host. A RAT uses a client-server model and has a user interface for easy administration.

https://t.me/learningnets

2.2.8 Spyware

Spyware is software which spies on user activities to collect user information, such as what websites the user frequently visits, without the consent of the computer owner.

The information is sent to the author or owner of the spyware program after a certain amount of data has been collected. Normally, a system which has spyware also has other kinds of malware, such as rootkits or trojans, to hide their tracks and to keep control of the machine.

https://t.me/learningnets

2.2.9 Botnets

Botnets are a collection of compromised computers which run commands automatically and autonomously, with the help of a command and control server. Botnets are typically created when a number of clients install the same malware. The hosts are usually infected via drive-by-downloads.

The controller or owner of the botnet is called the bot master and is usually the one who gives commands to the bots. Botnets are used by the bot master for reasons such as DDoS, sending spam, etc.

https://t.me/learningnets

2.2.10 Ransomware

This type of malware encrypts files and demands the victim send money via Bitcoin, after which the user will be sent the key to unlock the files. The files are being held hostage until the victim pays the ransom, hence the term **ransomware**.

They are also called extortive malware, since they demand money in exchange for restoration of the victim's data.

https://t.me/learningnets

2.2.11 Information Stealers

This type of malware basically steals data such as private encryption keys, login credentials, credit card data, competitor data (such as proprietary data, intellectual property, etc.), and other important data which could be used for many malicious reasons.

https://t.me/learningnets

2.2.11 Information Stealers

Keyloggers

Keyloggers capture keystrokes as the victim is typing. This information is saved locally and later sent to the attacker.

Screen recorders

Screen recorders take screenshots of the active window on the victim's machine when a condition is met, such as a time interval. These images are saved locally and sent to the attacker as well.

https://t.me/learningnets

2.2.11 Information Stealers

RAM scrapers

RAM scrapers attempt to steal information in memory while it's being processed. The reason for this is because in memory, everything is decrypted. This technique is well known for stealing credit card numbers in some big-name breaches within the last few years.

https://t.me/learningnets

2.2 Malware Classifications

Basically, the point to note is that there is no clear line which distinguishes one form of malware from another. Normally, malware is found in pairs with multiple variants simultaneously active on the target system.

There are various other types of malware classifications, such as **Adware**, **Greyware**, **Scareware**, **Fakeware**, **PUP**s (Potentially Unwanted Programs), etc., but we will not be covering those types of malware as they're not what we'll be primarily hunting within our corporate environments.

https://t.me/learningnets

2.2 Malware Classifications

Being familiar with malware and which classification it would fall under will help you understand the purpose of the malware and potentially what actions it took, or is going to take.

Next, we'll look at how the malware will reach the target system.

https://t.me/learningnets

Malware Delivery

https://t.me/learningnets

2.3 Malware Delivery

There are various ways malware can reach its target. Below are a few:

- Physical media
- Email (attachments)
- URL links
- Drive-by downloads
- Web advertising
- Social media
- File shares
- Software vulnerabilities

https://t.me/learningnets

2.3.1 Physical Media

Malware that uses physical medium to spread generally doesn't have any other means to spread itself. Malware that uses a USB stick, for example, could infect the boot sector or be configured to autorun once inserted into the victim machine.

Perhaps the malware is not configured that way. Another method would be simply to put the malware into the USB stick and hope that your intended target(s) will run the malware.

https://t.me/learningnets

2.3.1 Physical Media

Another method worth mentioning is **HIDs (Human Interface Device)**.

USB Rubber Ducky, USBdriveby, Teensy, and BadUSB are examples of attacks where the attacker can create scripts to execute a set of commands, such as running malware, on a target system.

http://usbrubberducky.com/#!index.md
http://samy.pl/usbdriveby/
https://www.pjrc.com/teensy/
https://nakedsecurity.sophos.com/2014/10/06/badusb-now-with-do-it-yourself-instructions/

https://t.me/learningnets

2.3.2 Email

This is one of the most common methods to attempt to infiltrate an organization. Due to users' poor security awareness, they are susceptible to falling victim to social engineering via phishing, spear phishing, and whaling attacks.

Why would an attacker spend hours, days, or even weeks surveying the network perimeter of the target in hopes of punching a hole through the firewall when a nicely crafted email, with a malicious attachment, will do the trick and get them inside? This attack vector is fruitful compared to finding a vulnerability through the firewall.

https://t.me/learningnets

2.3.2 Email

Also, the email doesn't necessarily need to have an attachment.

The email can still be carefully crafted to lure the victim to click a link or even visit a website where the attacker hosts the malware.

https://t.me/learningnets

2.3.3 URL Links

We have been reading about links thus far, and it should have its own section as this is an attack vector to deliver malware to the target.

With URL links, the link can direct to victim to a website where the malware is hosted, or the link could simply download the malware itself.

https://t.me/learningnets

2.3.3 URL Links

As we have read already, links can be placed anywhere:

- Emails
- Documents (Word, Excel, etc.)
- Instant Messaging
- Social Media Feeds
- Etc.

URL shortening has also aided in making this a successful attack vector. The victim can't tell what they're clicking on by simply looking at the link.

https://t.me/learningnets

2.3.4 Drive-by Downloads

A **drive-by download** site installs malware once a victim visits that site.

This means that the victim doesn't need to click on a link. Code in the web page, whether client-side or server-side, will attempt to install malware based on the victim's machine configuration (Windows 7 running an unpatched version of Flash for example). Typically, this is achieved by exploit kits.

https://t.me/learningnets

https://blog.malwarebytes.com/threat-analysis/2017/03/exploit-kits-winter-2017-review/

2.3.4 Drive-by Downloads

The attackers can purchase and build specific domains to spread malware via drive-by. Attackers can use popular news/events and use search engine optimization (SEO) techniques to push their malware-hosting website to the top of the search ranks to increase the chances of victims visiting the site.

Another technique can be used which is called a **watering hole attack**.

https://t.me/learningnets

https://www.trendmicro.com/vinfo/us/threat-encyclopedia/web-attack/137/watering-hole-101

2.3.5 Web Advertising

This technique is known as **malvertising**, delivering malware through online ads. This is achieved by buying ad space in popular, legitimate websites, but the ads have a malicious purpose.

The ads will either redirect the victim to a website hosting the malware or it will insert malware directly into the victim's machine. This can be done by clicking on the ad and even **without** clicking on the ad. You can read more here.

https://t.me/learningnets

https://arstechnica.com/information-technology/2016/12/millions-exposed-to-malvertising-that-hid-attack-code-in-banner-pixels/

2.3.5 Web Advertising

Another way to use ad-serving code is through browser extensions. In more recent news, a couple of Chrome extensions were targeted and displayed malicious ads.

You can read about this here.

https://t.me/learningnets

https://www.theregister.co.uk/2017/08/03/over-a-million-coders-targeted-by-chrome-extension-hack/#670c91059c9d

2.3.6 Social Media

Social media continues to grow in popularity. The amount of users registered in various platforms are in the hundreds of millions. Even companies have social media accounts to promote their company and products.

Many of these social media platforms offer tools and features that will aid an attacker in their mission. They have the ability to post links in feeds and/or send direct messages to users.

https://t.me/learningnets

2.3.6 Social Media

Within their attack campaign, the attackers will use some social engineering techniques, again, to convince the unsuspecting victims to click on a link or to download the malware.

https://t.me/learningnets

2.3.7 Software Vulnerabilities

Software vulnerabilities refer to exploiting vulnerabilities within whitelisted applications running in the environment to inject executable code into the running application and take control of the process.

Attackers exploit the vulnerabilities (application bugs and/or flaws) with buffer overflows:

- Stack overflows
- Heap overflows

https://t.me/learningnets

2.3.7 Software Vulnerabilities

Stack overflows

This is exploited by overflowing the buffers on the stack to get control of the flow of execution for the application to execute the malicious code.

Heap overflows

This is exploited by overwriting pointers in the heap (dynamically allocated memory space) to point to malicious code instead of its original location.

https://t.me/learningnets

2.3 Malware Delivery

There are other vectors that could allow a victim to be infected that weren't discussed in these slides because the chances of these vectors being present in corporate environments are slim.

These other vectors are:

- Peer to peer (P2P) file sharing
- Instant messaging

https://t.me/learningnets

2.3 Malware Delivery

So far, we discussed how malware is classified and how it can reach a victim machines.

Next, we need to discuss how the malware attempts to remain undetected and dormant once it lands on the victim machine, as well as techniques to avoid analysis if found.

https://t.me/learningnets

2.4

Malware Evasion Techniques

https://t.me/learningnets

2.4 Malware Evasion Techniques

We'll begin this section with techniques on how malware attempts to run and **evade defenses**.

Not only is the malware looking for a successful initial **execution**, but it's also using techniques to **escalate privileges**, **steal credentials**, **exfiltrate data**, and **maintain persistence**, to name a few. Refer to the MITRE ATT&CK wiki here for more information.

https://attack.mitre.org/wiki/Defense_Evasion
https://attack.mitre.org/wiki/Execution
https://attack.mitre.org/wiki/Privilege_Escalation
https://attack.mitre.org/wiki/Credential_Access
https://attack.mitre.org/wiki/Exfiltration
https://attack.mitre.org/wiki/Persistence
https://attack.mitre.org/wiki/Technique_Matrix

https://t.me/learningnets

2.4 Malware Evasion Techniques

Note that new methods and techniques are always being discovered by researchers and, most importantly, the adversary to remain covert and prevent detection.

As a hunter, you must always be looking into the latest techniques being shared and discovered.

https://t.me/learningnets

2.4.1 Alternate Data Streams

Alternate Data Streams, or streams, are a feature of the NTFS file system. They are not available on FAT file systems or any other file system.

As per the Microsoft's article, the original data stream is the file data itself, which is the data stream with no name. All other streams have a name. Alternate Data Streams can be used to store file meta data and any other type of data.

https://t.me/learningnets

https://docs.microsoft.com/en-us/archive/blogs/askcore/alternate-data-streams-in-ntfs

2.4.1 Alternate Data Streams

To explain this concept, we can type the following into the command prompt:

echo “This is not ADS” > file.txt

echo “This is in ADS” > file.txt:stream1

We can also create streams programmatically. In the **CreateFile** Windows API, just append “:stream_name” to the file name, where “stream_name” is the name of the data stream. We could also use the **WriteFile** Windows API function to write the data.

https://t.me/learningnets

https://docs.microsoft.com/en-us/windows/win32/api/fileapi/nf-fileapi-createfilea
https://docs.microsoft.com/en-us/windows/win32/api/fileapi/nf-fileapi-writefile

2.4.1 Alternate Data Streams

Example program that writes to Alternate Data Streams is presented on the right.

```
#include <windows.h>
#include <stdio.h>

void main() {
                ...
                hStream = CreateFile('file.txt:stream2',
                                GENERIC_WRITE,
                                FILE_SHARE_WRITE,
                                NULL,
                                OPEN_ALWAYS,
                                0,
                                NULL);
                if(hStream == INVALID_HANDLE_VALUE)
                                printf("Cannot open file.txt:stream2\n");
                else
                                WriteFile(hStream, "This data is hidden in the
stream. Can you read it???", 53, &dwRet, NULL);
}
```

https://t.me/learningnets

2.4.1 Alternate Data Streams

Let's use the DIR command to attempt to see the regular txt file and the ADS.

```
C:\Users\elshunter\Desktop>dir
 Volume in drive C has no label.
 Volume Serial Number is 4247-60E2

 Directory of C:\Users\elshunter\Desktop

08/04/2017  01:22 PM    <DIR>          .
08/04/2017  01:22 PM    <DIR>          ..
08/04/2017  01:22 PM                20 file.txt
               1 File(s)             20 bytes
               2 Dir(s)   7,552,462,848 bytes free

C:\Users\elshunter\Desktop>dir /r
 Volume in drive C has no label.
 Volume Serial Number is 4247-60E2

 Directory of C:\Users\elshunter\Desktop

08/04/2017  01:22 PM    <DIR>          .
08/04/2017  01:22 PM    <DIR>          ..
08/04/2017  01:22 PM                20 file.txt
                                    19 file.txt:stream1:$DATA
               1 File(s)             20 bytes
               2 Dir(s)   7,552,462,848 bytes free
```

https://t.me/learningnets

2.4.1 Alternate Data Streams

Here, we attempt to view the contents of the txt file and ADS using **TYPE** and **MORE**.

```
C:\Users\elshunter\Desktop>type file.txt
"This is not ADS"

C:\Users\elshunter\Desktop>type file.txt:stream1
The filename, directory name, or volume label syntax is incorrect.

C:\Users\elshunter\Desktop>more < file.txt
"This is not ADS"

C:\Users\elshunter\Desktop>more < file.txt:stream1
"This is in ADS"
```

https://t.me/learningnets

2.4.1 Alternate Data Streams

Another tool to view alternate data streams is the **Streams** tool bundled within **Sysinternals**.

streams.exe c:\users\elshunter\desktop\file.txt

```
c:\Users\elshunter\Documents\Tools\SysinternalsSuite>streams.exe c:\Users\elshunter\Desktop\file.txt

streams v1.60 - Reveal NTFS alternate streams.
Copyright (C) 2005-2016 Mark Russinovich
Sysinternals - www.sysinternals.com

c:\Users\elshunter\Desktop\file.txt:
        :stream1:$DATA 19
```

https://t.me/learningnets

https://docs.microsoft.com/en-us/sysinternals/downloads/streams
https://docs.microsoft.com/en-us/sysinternals/downloads/

2.4.1 Alternate Data Streams

This tool was useful prior to the Windows PowerShell days. PowerShell's cmdlet **Get-Item** has the capability to retrieve alternate data stream information.

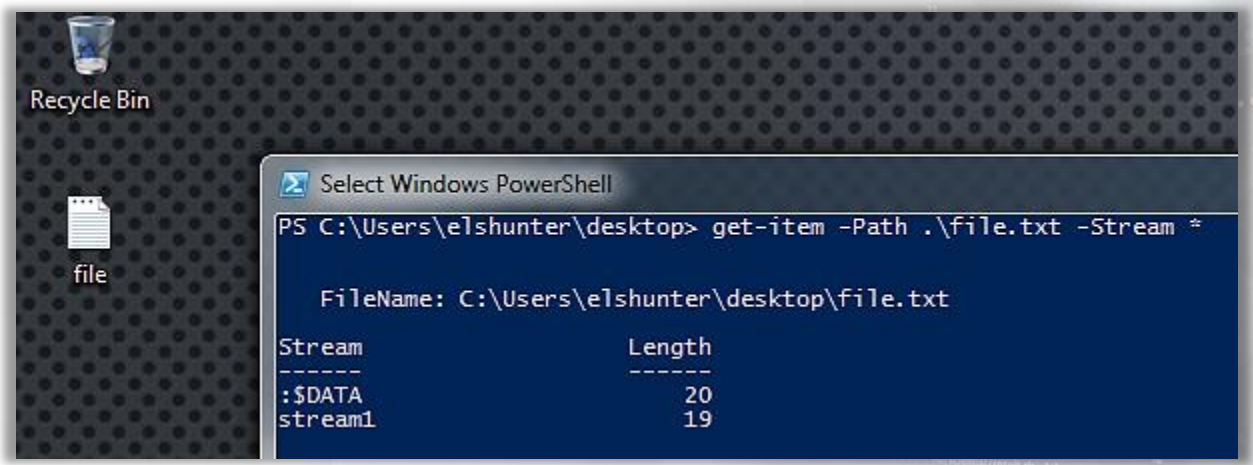
We'll need to use the "**–Stream**" parameter in order to do so.

https://t.me/learningnets

https://docs.microsoft.com/en-us/powershell/module/Microsoft.PowerShell.Management/Get-Item?view=powershell-5.1

2.4.1 Alternate Data Streams

get-item –path .\file.txt –stream *



https://t.me/learningnets

2.4.1 Alternate Data Streams

Note that Microsoft uses ADS for non-nefarious reasons. For instance, via the **Zone.Identifier (Zone 3) ADS**, we can tell if a file or binary was downloaded from the Internet (**Internet Zone**).

Value	Zone
0	My Computer
1	Local Intranet Zone
2	Trusted Sites Zone
3	Internet Zone
4	Restricted Sites Zone

https://t.me/learningnets

2.4.1 Alternate Data Streams

To retrieve this information, we need to use the **Get-Content** cmdlet within PowerShell.

get-item –path .\putty-64bit-0.70-installer.msi –stream *

```
PS C:\Users\elshunter\Downloads> get-item -path .\putty-64bit-0.70-installer.msi -stream *


   FileName: C:\Users\elshunter\Downloads\putty-64bit-0.70-installer.msi

Stream                   Length
------                   ------
:$DATA                  3048960
Zone.Identifier              26


PS C:\Users\elshunter\Downloads> get-content -path .\putty-64bit-0.70-installer.msi -stream zone.identifier
[ZoneTransfer]
ZoneId=3
```

https://docs.microsoft.com/en-us/powershell/module/Microsoft.PowerShell.Management/Get-Content?view=powershell-5.1

https://t.me/learningnets

2.4.1 Alternate Data Streams

Within PowerShell version 5 and newer versions, there is a lot more information that is displayed when using the **Get-Item** cmdlet and the "**–Stream**" parameter.

https://t.me/learningnets

https://docs.microsoft.com/en-us/powershell/scripting/whats-new/what-s-new-in-windows-powershell-50?view=powershell-7

2.4.1 Alternate Data Streams

Here you can see the Get-Item command being used with the -Stream option.

```
PS C:\Users\[redacted]\Downloads> Get-Item -Path .\NwInvestigatorSetup-10.6.1.1.696.x64.msi -Stream *


PSPath        : Microsoft.PowerShell.Core\FileSystem::C:\Users\[redacted]\Downloads\NwInvestigatorSetup-10.6.1.1.696.x64.msi
                ::$DATA
PSParentPath  : Microsoft.PowerShell.Core\FileSystem::C:\Users\[redacted]\Downloads
PSChildName   : NwInvestigatorSetup-10.6.1.1.696.x64.msi::$DATA
PSDrive       : C
PSProvider    : Microsoft.PowerShell.Core\FileSystem
PSIsContainer : False
FileName      : C:\Users\[redacted]\Downloads\NwInvestigatorSetup-10.6.1.1.696.x64.msi
Stream        : :$DATA
Length        : 74014720

PSPath        : Microsoft.PowerShell.Core\FileSystem::C:\Users\[redacted]\Downloads\NwInvestigatorSetup-10.6.1.1.696.x64.msi
                :Zone.Identifier
PSParentPath  : Microsoft.PowerShell.Core\FileSystem::C:\Users\[redacted]\Downloads
PSChildName   : NwInvestigatorSetup-10.6.1.1.696.x64.msi:Zone.Identifier
PSDrive       : C
PSProvider    : Microsoft.PowerShell.Core\FileSystem
PSIsContainer : False
FileName      : C:\Users\[redacted]\Downloads\NwInvestigatorSetup-10.6.1.1.696.x64.msi
Stream        : Zone.Identifier
Length        : 26

PS C:\Users\[redacted]\Downloads> get-content -path .\NwInvestigatorSetup-10.6.1.1.696.x64.msi -stream zone.identifier
[ZoneTransfer]
ZoneId=3
```

https://t.me/learningnets

2.4.1 Alternate Data Streams

An example of malware that attempts to remove clues that it was downloaded from the Internet by deleting the Zone.Identifier from the file is **CoinMiner**.

Deletes `Zone.Identifier` of the file:

```
004025D6 lea     edx, [esp+25A0h+FileName]
004025DD lea     ecx, [esp+25A0h+Filename] ; lpFileName
004025E4 call    copy_via_mapping
004025E9 mov     edx, offset aZone_identifie ; ":Zone.Identifier"
004025EE lea     ecx, [esp+25A0h+FileName] ; lpString2
004025F5 call    sub_401040      ; ???
004025FA mov     esi, eax
004025FC push    esi             ; C:\Users\John Doe\AppData\Local\hjfdWLFJwt\taskman.exe:Zone.Identifier
004025FD call    ds:DeleteFileW
00402603 push    esi             ; void *
00402604 call    j__free
00402609 add     esp, 4
```

Credit: https://secrary.com/ReversingMalware/CoinMiner/

https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Trojan:Win32/CoinMiner

https://t.me/learningnets

2.4.2 Injections

In the upcoming slides, we'll look at various injection techniques used by malware to inject itself into other processes, threads, etc.

https://t.me/learningnets

2.4.2.1 DLL Injections

DLL Injection is the most common technique used to inject malware into another process, or processes. Let's illustrate how this is accomplished by malware.

1. Locate Process

Malware needs to find a target process to inject the malicious DLL into.

- Windows API: CreateToolhelp32Snapshot(), Process32First(), Process32Next()

https://attack.mitre.org/wiki/Technique/T1055
https://msdn.microsoft.com/en-us/library/windows/desktop/ms682489(v=vs.85).aspx
https://msdn.microsoft.com/en-us/library/windows/desktop/ms684834(v=vs.85).aspx
https://msdn.microsoft.com/en-us/library/windows/desktop/ms684836(v=vs.85).aspx

https://t.me/learningnets

2.4.2.1 DLL Injections

2. Open Process

Once the malware finds the process, it opens the process.

- Windows API: GetModuleHandle(), GetProcAddress(), OpenProcess()

3. Allocate Memory

The malware then needs to find a location in order to write the path to the malicious DLL.

- Windows API: VirtualAllocEx()

https://msdn.microsoft.com/en-us/library/windows/desktop/ms683199(v=vs.85).aspx
https://msdn.microsoft.com/en-us/library/windows/desktop/ms683212(v=vs.85).aspx
https://msdn.microsoft.com/en-us/library/windows/desktop/ms684320(v=vs.85).aspx
https://msdn.microsoft.com/en-us/library/windows/desktop/aa366890(v=vs.85).aspx

https://t.me/learningnets

2.4.2.1 DLL Injections

4. Copy

The malware will write the path to the malicious DLL into the allocated memory location.

- Window API: WriteProcessMemory()

5. Execute

The malware will execute the malicious DLL in another process by starting a new thread.

- Window API: CreateRemoteThread(), LoadLibrary()

https://msdn.microsoft.com/en-us/library/windows/desktop/ms681674(v=vs.85).aspx
https://msdn.microsoft.com/en-us/library/windows/desktop/ms682437(v=vs.85).aspx
https://msdn.microsoft.com/en-us/library/windows/desktop/ms684175(v=vs.85).aspx

https://t.me/learningnets

2.4.2.1 DLL Injections

It’s important to note that the **documented** Windows API **CreateRemoteThread()** function is not the only function that can be used. There are **undocumented** functions that can be used as well.

https://t.me/learningnets

2.4.2.1 DLL Injections

NTCreateThreadEx()

The steps to utilize this undocumented function are a bit more involved. This function needs to be loaded from Ntdll.dll.

RtlCreateUserThread()

Both **Mimikatz** and **Metasploit** use this undocumented function. This function also needs to be loaded from Ntdll.dll.

https://undocumented.ntinternals.net/index.html?page=UserMode/Undocumented%20Functions/NT%20Objects/Thread/NtCreateThread.html
https://undocumented.ntinternals.net/index.html?page=UserMode/Undocumented%20Functions/Executable%20Images/RtlCreateUserThread.html

https://t.me/learningnets

2.4.2.1 DLL Injections

In order for this technique to work, the path of the malicious DLL needs to reside on disk.

Another stealthier technique is **Reflective DLL Injection** which loads the DLL in memory.

https://t.me/learningnets

2.4.2.2 Reflective DLL Injection

This technique was discovered and shared by Steven Fewer. As stated on the GitHub page:

"**Reflective DLL injection** is a library injection technique in which the concept of reflective programming is employed to perform the loading of a library from **memory** into a host process."

https://t.me/learningnets
https://github.com/stephenfewer/ReflectiveDLLInjection

2.4.2.2 Reflective DLL Injection

In a nutshell, this technique will load a malicious DLL without calling the normal Windows API calls as it does with DLL Injection.

The malicious DLL doesn't need a loader and will map itself into memory when run. It will resolve import addresses, fix relocations, and call the **DllMain** function (without using **LoadLibrary**).

https://t.me/learningnets

2.4.2.2 Reflective DLL Injection

Frameworks, such as **Metasploit** and **PowerShell Empire**, have this capability incorporated into them.

APTs and malware authors, especially advanced ones, will most likely not use these frameworks, but instead will create their own tools using C/C++.

https://t.me/learningnets

2.4.2.3 Thread Hijacking

With this technique, the malware will not need to create a new process or thread. Instead, it will loop through threads of a process on the target system and perform similar steps of **DLL Injection** but with slight differences.

1. Locate Thread

Malware needs to find a target thread to inject into.

- Windows API: CreateToolhelp32Snapshot(), Thread32First(), Thread32Next()

https://msdn.microsoft.com/en-us/library/windows/desktop/ms682489(v=vs.85).aspx
https://msdn.microsoft.com/en-us/library/windows/desktop/ms686728(v=vs.85).aspx
https://msdn.microsoft.com/en-us/library/windows/desktop/ms686731(v=vs.85).aspx

https://t.me/learningnets

2.4.2.3 Thread Hijacking

2. Open Thread

The malware opens the thread.

- Windows API: OpenThread()

3. Suspend Thread

The thread needs to be suspended in order to inject.

- Windows API: SuspendThread()

https://t.me/learningnets

https://msdn.microsoft.com/en-us/library/windows/desktop/ms684335(v=vs.85).aspx
https://msdn.microsoft.com/en-us/library/windows/desktop/ms686345(v=vs.85).aspx

2.4.2.3 Thread Hijacking

4. Allocate Memory

The malware then needs to find a location in order to write the path to the malicious DLL, or shellcode.

- Windows API: VirtualAllocEx()

5. Copy

The malware will write the path to the malicious DLL, shellcode, or address to LoadLibrary into the allocated memory location.

- Window API: WriteProcessMemory()

https://t.me/learningnets

https://msdn.microsoft.com/en-us/library/windows/desktop/aa366890(v=vs.85).aspx
https://msdn.microsoft.com/en-us/library/windows/desktop/ms681674(v=vs.85).aspx

2.4.2.3 Thread Hijacking

6. Resume Thread

Once the malware has performed its injection, the thread resumes.

- Windows API: ResumeThread()

https://docs.microsoft.com/en-us/library/windows/desktop/ms685086(v=vs.85).aspx

https://t.me/learningnets

2.4.2.4 PE Injection

PE (Portable Executable) Injection is similar to **DLL Injection**, but one notable difference is that, just like **Reflective DLL Injection**, the malicious DLL doesn't have to reside on disk.

WriteProcessMemory() will be used by this technique, but not to write the path to the malicious DLL. Instead, it will be used to write the malicious code into that location. There is no need to use **LoadLibrary()** either with this technique.

https://t.me/learningnets

2.4.2.4 PE Injection

Since the PE is injected into another process, it will have a new base address. The base address is the starting address of a memory-mapped EXE or DLL. The malware will need to loop through its relocation descriptors to find its addresses. You can read more about PE Headers in the document titled **Peering Inside the PE: A Tour of the Win32 Portable Executable File Format**.

https://t.me/learningnets
https://msdn.microsoft.com/en-us/library/ms809762.aspx

2.4.2.5 Process Hollowing

Process Hollowing is the technique where malware will un-map the legitimate code from memory of the process and overwrite memory of the process with a malicious binary.

1. Create Process

Malware will create a new process in a suspended state to host the malicious code. The primary thread will also be in a suspended state.

- Window API: CreateProcess()

https://t.me/learningnets

https://attack.mitre.org/techniques/T1093
https://msdn.microsoft.com/en-us/library/windows/desktop/ms682425(v=vs.85).aspx

2.4.2.5 Process Hollowing

2. Un-map Memory

The malware at this point will un-map the legitimate code from memory.

- Windows API: ZwUnmapViewOfSection(), NtUnmapViewOfSection()

3. Allocate & Write

Same as DLL Injection technique, the malware will allocate memory locations for malicious code and write each section of the malware into this space.

- Windows API: VirtualAllocEx(), WriteProcessMemory()

https://msdn.microsoft.com/ru-ru/library/ff567119(v=vs.85).aspx
https://msdn.microsoft.com/en-us/library/ff557711(v=vs.85).aspx
https://msdn.microsoft.com/en-us/library/windows/desktop/aa366890(v=vs.85).aspx
https://msdn.microsoft.com/en-us/library/windows/desktop/ms681674(v=vs.85).aspx

https://t.me/learningnets

2.4.2.5 Process Hollowing

3. Set Entry Point

The malware will set the entry point to new code section.

- Windows API: SetThreadContext()

4. Resume

The malware will take the process out of suspended state.

- Windows API: ResumeThread()

https://t.me/learningnets

https://msdn.microsoft.com/en-us/library/windows/desktop/ms680632(v=vs.85).aspx
https://msdn.microsoft.com/en-us/library/windows/desktop/ms685086(v=vs.85).aspx

2.4.2.6 Hook Injection

Malware uses **hooking** to intercept events, and based on a particular triggered event, the malware will respond as instructed.

This is accomplished using the Windows API [SetWindowsHookEx()](https://msdn.microsoft.com/en-us/library/windows/desktop/ms644990(v=vs.85).aspx).

https://t.me/learningnets

2.4.2.6 Hook Injection

This can be used to monitor the keyboard (**WH_KEYBOARD**) and mouse (**WH_MOUSE**) input. WH_KEYBOARD and WH_MOUSE are hook types. There are different hook types that can be passed as an argument to SetWindowsHookEx(). You can read more about hook types here. Malware can also use this technique to load a malicious DLL based on a specific event.

https://t.me/learningnets

https://msdn.microsoft.com/en-us/library/windows/desktop/ms644959(v=vs.85).aspx

2.4.2.7 KernelMode Rootkits: SSDT Hooks

SSDT stands for System Service Descriptor Table. The Windows Kernel uses this table to lookup system functions and each table entry points to function code. Every SSDT entry will point to the system kernel (**ntoskrnl.exe**) or the GUI driver (**win32k.sys**).

https://t.me/learningnets

2.4.2.7 KernelMode Rootkits: SSDT Hooks

For each entry in the SSDT table, there is a suitable function in kernel mode which completes the task specified by Native API. The SSDT resides in the kernel and is exported as **KeServiceDescriptorTable()**.

The representation can be pictured as:

https://t.me/learningnets

2.4.2.7 KernelMode Rootkits: SSDT Hooks

With SSDT Hooking, it will modify the pointers within this table to point to a location that the rootkit controls, such as a malicious function.

The effects of this technique are global to the system.

https://t.me/learningnets

2.4.2.7 KernelMode Rootkits: SSDT Hooks

1. Hook SSDT

Hook SSDT entry corresponding to NTQueryDirectoryFile.

2. Call Function

Now, whenever the NTQueryDirectoryFile function is called, the malicious function will be called instead.

https://t.me/learningnets

https://msdn.microsoft.com/en-us/library/windows/hardware/ff556633(v=vs.85).aspx

2.4.2.7 KernelMode Rootkits: SSDT Hooks

3. Pass Control

Right after the malicious function gets called, call the original function (NTQueryDirectoryFile) and get its results, a directory listing in this case.

4. Alter & Return Results

- Modify the results (hide a file or directory, for example)
- Pass the results back to the caller.

https://t.me/learningnets

2.4.2.8 KernelMode Rootkits: IRP Hooks

Windows architecture in kernel mode introduced the concepts of **IRP**s (I/O Request Packets) to transmit pieces of data from one component to another.

The concept of IRPs is well explained in the [Windows Driver Development Kit](https://docs.microsoft.com/en-us/windows-hardware/drivers/download-the-wdk).

https://t.me/learningnets

2.4.2.8 KernelMode Rootkits: IRP Hooks

Almost everything in the Windows kernel uses IRPs, for example, the network interface (TCP, UDP, etc.), file system, keyboard, mouse, and almost all existing drivers.

Here is a little snipped from the **Microsoft WinDDK** showing how critical IRPs are:

```
DriverObject->MajorFunction[IRP_MJ_CREATE]=DiskPerfCreate;
DriverObject->MajorFunction[IRP_MJ_READ]=DiskPerfReadWrite;
DriverObject->MajorFunction[IRP_MJ_WRITE]=DiskPerfReadWrite;
DriverObject->MajorFunction[IRP_MJ_SYSTEM_CONTROL]=DiskPerfWmi;
DriverObject->MajorFunction[IRP_MJ_SHUTDOWN]=DiskPerfShutdownFlush;
DriverObject->MajorFunction[IRP_MJ_FLUSH_BUFFERS]=DiskPerfShutdownFlush;
DriverObject->MajorFunction[IRP_MJ_PNP]=DiskPerfDispatchPnp;
DriverObject->MajorFunction[IRP_MJ_POWER]=DiskPerfDispatchPower;
```

https://t.me/learningnets

2.4.2.8 KernelMode Rootkits: IRP Hooks

Each device object has its own function table.

Hooking the function pointers of such objects is called **DKOM** (**D**irect **K**ernel **O**bject **M**anipulation). The effects of this technique are global to the system.

https://t.me/learningnets

2.4.2.8 KernelMode Rootkits: IRP Hooks

A function pointer can be hooked as follows:

```
old_power_irp=DriverObject->MajorFunction[IRP_MJ_POWER];
DriverObject->MajorFunction[IRP_MJ_POWER]=my_new_irp;
```

https://t.me/learningnets

2.4.2.8 KernelMode Rootkits: IRP Hooks

The basic IRP design is so that after an IRP has been created, it is passed to all the devices registered at lower levels. The design has a pre-processing mode and a post-processing mode.

Pre-processing is done when an IRP arrives, and post-processing is done when the IRP has been processed by all the levels below the current level.

https://t.me/learningnets

2.4.2.9 Userland Rootkits: IAT Hooks

IAT stands for Import Address Table, and it is used to resolve runtime dependencies.

It accomplishes this by listing what Windows API functions it needs and where they are located, within each DLL (including its address).

https://t.me/learningnets

2.4.2.9 Userland Rootkits: IAT Hooks

IAT Hooking involves modifying the IAT table of the executable and replacing the addresses to redirect certain functions to use the malicious function instead.

https://t.me/learningnets

2.4.2.10 Userland Rootkits: EAT Hooks

EAT stands for Export Address Table, this table is maintained in DLLs. These files contain support functions for other executables.

The difference between IAT and EAT Hooking is that since EATs exist only in DLLs (under normal settings), most of the times EAT Hooking is utilized only in DLLs, while IAT Hooking can be done in both DLLs and EXEs.

https://t.me/learningnets

2.4.2.11 Userland Rootkits: Inline Hooks

Inline Hooking is the most difficult. With this technique, the malware will modify the API function itself.

The malware can accomplish this by modifying the first few bytes of the target function code and replacing them with malicious code which tells EIP (instruction pointer) to execute code somewhere else in memory.

https://t.me/learningnets

2.4.2.12 Rootkits: Process Hiding

One characteristic all rootkits have in common is hiding their existence in the target system.

When a hunter/analyst is attempting to look for malware running on a system, rootkits will typically hide running processes from view making them difficult to detect using conventional methods.

https://t.me/learningnets

2.4.2.12 Rootkits: Process Hiding

Malware will hook **NtOpenProcess** (Native API) probably using SSDT hooking techniques.

It would also need to hide the process from the EPROCESS list. This list is maintained by the operating system for all active processes.

https://t.me/learningnets

https://msdn.microsoft.com/en-us/library/windows/hardware/ff556571(v=vs.85).aspx
https://msdn.microsoft.com/en-us/library/windows/hardware/ff544273(v=vs.85).aspx

2.4.2.12 Rootkits: Process Hiding

Within the EPROCESS list structure, one of the important members is **ActiveProcessLinks**, which is the doubly linked list with *FLINK (Forward Link) and *BLINK (Back Link) as pointers to the other structures.

In order to hide the process, the malware will need to unlink the structure relative to the malicious and compromised process from the list. If the driver is loaded, the malware will also have to unlink it from **PsLoadedModuleList**.

https://t.me/learningnets

https://msdn.microsoft.com/en-us/library/windows/hardware/ff554296(v=vs.85).aspx

2.4.3 Masquerading

Masquerading is defined as pretending to be someone one is not. This definition can apply to techniques malware uses to avoid detection.

At times, instead of going to great lengths to use an advanced technique, like one of ones previously mentioned, they can often opt for a very simple one, such as naming the malware similar to something that might go undetected.

https://t.me/learningnets

2.4.3 Masquerading

They might call the malware svch0st, scvhost, svchost32, etc. You can find a list of different variations used to masquerade malware as svchost.exe here.

Also, placing the malware in C:\Windows or C:\Windows\System32 directories is another way of masquerading as a legitimate executable or dll.

https://t.me/learningnets

http://www.hexacorn.com/blog/2014/07/04/the-typographical-and-homomorphic-abuse-of-svchost-exe/

2.4.3 Masquerading

Malware will also hide in other locations, such as in:

- Temporary folders
- Temporary Internet files
- Program Files

This doesn’t exclude other locations, such as the Recycle Bin.

https://t.me/learningnets

2.4.4 Packing / Compression

A packer is software which compresses an executable. They were initially designed to decrease the size of executable files.

However, malware authors recognized very quickly that decreasing the file size will decrease the number of patterns in the file, so it will decrease the chance of detection by AV products.

https://t.me/learningnets

2.4.4 Packing / Compression

Some malware authors have gone to the trouble of creating their own packers (such as Yoda Packer), while others use readily available packers (such as UPX).

https://t.me/learningnets

https://upx.github.io/

2.4.5 Recompiling

As we should know, an executable will have a signature, such as a MD5 hash, that will identify it.

With recompiling, especially with different compilers, the malware authors are looking to avoid detection because a different hash will be produced.

https://t.me/learningnets

2.4.6 Obfuscation

The next two techniques (**obfuscation** and **anti-reversing**) address the point that once the malware has been found, it attempts to remain irreversible.

https://t.me/learningnets

2.4.6 Obfuscation

Code obfuscation techniques transform/change a program in order to make it more difficult to analyze while preserving the original functionality. Code obfuscation is used both by malware and legitimate software to protect itself.

The difference is that malware uses it to either prevent detection or to make reverse engineering more difficult. Malware obfuscates itself every time it infects a new host, which makes it harder for a detector to recognize it.

https://t.me/learningnets

2.4.7 Anti-Reversing Techniques

There are several methods that are used by malware to detect that analysis is being conducted on the code, or simply to misdirect the malware analyst and take him for a loop, thus increasing the time required to analyze the code.

Some of these the anti-reversing techniques are:

- Detecting that malware is being run in a virtual machine
- Detecting that a debugger is attached to the malware
- Junk code can be inserted into the malware as misdirection

https://t.me/learningnets

2.5

Malware Persistence

https://t.me/learningnets

2.5 Malware Persistence

As mentioned previously, there are many goals of malware once on a host machine, as noted in MITRE's ATT&CK wiki.

In this section, we'll look at mechanisms used by malware to achieve **persistence**.

https://t.me/learningnets

https://attack.mitre.org/wiki/Persistence

2.5.1 Autostart Locations

Malware uses these autostart locations to survive reboots. By adding an entry into one of many autostart locations, either within the registry or specific folders, the program will execute when the user logs in.

The most common, and the most obvious, registry location is:

HKLM\Software\Microsoft\Windows\CurrentVersion\Run.

https://t.me/learningnets

2.5.1 Autostart Locations

Here are a few other **common** locations:

- HKCU\Software\Microsoft\Windows\CurrentVersion\Run
- HKLM\Software\Microsoft\Windows\CurrentVersion\Policies\Explorer\Run
- HKCU\Software\Microsoft\Windows\CurrentVersion\Policies\Explorer\Run

https://t.me/learningnets

2.5.1 Autostart Locations

Some **not so common** locations include:

- HKLM\Software\Microsoft\Windows NT\CurrentVersion\Winlogon\Userinit
- HKLM\Software\Microsoft\Windows NT\CurrentVersion\Image File Execution Options
- HKLM\Software\Wow6432Node\Windows NT\CurrentVersion\Image File Execution Options

https://t.me/learningnets

2.5.1 Autostart Locations

To get an idea of all the locations that malware can use within the registry or folders, look at the AutoRuns tool from Sysinternals.

You may also want to refer to ATT&CK's wiki page here.

https://docs.microsoft.com/en-us/sysinternals/downloads/autoruns
https://docs.microsoft.com/en-us/sysinternals/
https://attack.mitre.org/wiki/Technique/T1060

https://t.me/learningnets

2.5.2 Scheduled Task

Utilities such as **at.exe** and **schtasks.exe** can be used to schedule execution of malware or scripts based on a specific date/time or even an event.

An adversary can use task scheduling to execute malware or scripts on startup or on a schedule, not only for persistence, but for other tasks as well, such as privilege escalation.

https://t.me/learningnets

2.5.2 Scheduled Task

An example from ATT&CK for scheduled tasks is from APT3:

- **schtasks /create /tn “mysc” /tr C:\Users\Public\test.exe /sc ONLOGON /ru “System”**

https://t.me/learningnets
https://attack.mitre.org/wiki/Group/G0022

2.5.3 COM Hijacking

“The Microsoft Component Object Model (COM) is a platform-independent, distributed, object-oriented system for creating binary software components that can interact. COM is the foundation technology for Microsoft's OLE (compound documents), ActiveX (Internet-enabled components), as well as others.” – as referenced by Microsoft here.

https://t.me/learningnets

https://docs.microsoft.com/en-us/windows/win32/com/the-component-object-model

2.5.3 COM Hijacking

In a nutshell, it enables interaction between software components through the Windows operating system.

Adversaries can use this system (COM) to insert malicious code that can be executed instead of legitimate software by hijacking the COM references. You can read the ATT&CK document here.

https://t.me/learningnets
https://attack.mitre.org/wiki/Technique/T1122

2.5.4 DLL Hijacking: Search Order

In this technique, the malware is attacking a feature of the Windows operating system. When an executable runs, there is a search order to find the required DLL for the executable.

The first location that it will look in is within the local directory and eventually, it will look in the C:\Windows\System32 folder as well, if the executable is still searching for the required DLL.

https://t.me/learningnets

2.5.4 DLL Hijacking: Search Order

You can read more about:

- The technique on ATT&CK here.
- DLL Library Search Order here.

https://t.me/learningnets

https://attack.mitre.org/techniques/T1038
https://msdn.microsoft.com/en-us/library/windows/desktop/ms682586(v=vs.85).aspx

2.5.5 DLL Hijacking: Phantom DLL

This technique also attacks a feature of the Windows operating system. Executables may try to load old DLLs that no longer exist on modern Windows operating systems.

Attackers can place a malicious DLL and name it to what the executable is looking for, then the malicious DLL will be loaded.

https://t.me/learningnets

2.5.5 DLL Hijacking: Phantom DLL

Please reference a very resourceful blog on Hexacorn.com. He has a series titled, **Beyond Good Ol' Run Key**, where some parts of the series are dedicated to Phantom DLLs.

You can read more about this technique on his blog in one of the postings here.

https://t.me/learningnets

http://www.hexacorn.com/blog/2016/06/02/beyond-good-ol-run-key-part-40/

2.5.6 DLL Hijacking: Side Loading

This technique uses the **WinSxS** (Windows side-by-side) folder. This Windows folder (or feature) is used by applications to prevent problems which may arise due to updated and duplicated versions of DLLs. As you can imagine, a malicious DLL can be placed into this folder.

Please reference the information on ATT&CK about this technique here. You can read more about this technique in the following FireEye PDF called DLL Side-Loading: A Thorn in the Side of the Anti Virus Industry.

https://t.me/learningnets

https://www.fireeye.com/content/dam/fireeye-www/global/en/current-threats/pdfs/rpt-dll-sideloading.pdf

2.5.7 Windows Services: Service Creation

Service creation has been a popular technique to achieve persistence. A malicious service can hide in plain sight among many other services. A service can be configured to run at boot, which will often load before AV software. Services are created using the built-in Windows "**sc**" command.

The following link, here, is an old post, but still holds true to the fundamentals to use service creation for malicious purposes.

https://t.me/learningnets

https://www.bleepingcomputer.com/tutorials/how-malware-hides-as-a-service/

2.5.8 Windows Services: Service Replacement

With service replacement, attackers will look for an existing service to replace with their malware. If it's not configured to auto start, then that setting can be tweaked. With a low privilege account, the attacker can look for services with poor ACL configurations to replace the executable with the malware.

https://t.me/learningnets

2.5.9 Windows Services: Service Recovery

Another technique an attacker can accomplish using Services is to set up recovery actions when a service fails. With this technique, an attacker can configure the service to run an executable, such as malware, upon failure.

You can read more about this feature here.

https://t.me/learningnets

https://technet.microsoft.com/en-us/library/cc753662(v=ws.11).aspx

Module Conclusion

This concludes the module on malware. We have covered:

- Different classifications of malware.
- How malware infects the host system.
- Different techniques malware uses to evade detection and maintain persistence.

https://t.me/learningnets

References

https://t.me/learningnets

References

Blue Pill

http://theinvisiblethings.blogspot.com/2006/06/introducing-blue-pill.html

SubVirt

https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/subvirt.pdf

USB Rubber Ducky

http://usbrubberducky.com/#!index.md

USBdriveby

http://samy.pl/usbdriveby/

https://t.me/learningnets

References

Teensy

https://www.pjrc.com/teensy/

BadUSB

https://nakedsecurity.sophos.com/2014/10/06/badusb-now-with-do-it-yourself-instructions/

URL Shortener

https://goo.gl/

Exploit Kits

https://blog.malwarebytes.com/threat-analysis/2017/03/exploit-kits-winter-2017-review/

https://t.me/learningnets

References

Watering Hole Attack

https://www.trendmicro.com/vinfo/us/threat-encyclopedia/web-attack/137/watering-hole-101

Malvertising

https://arstechnica.com/information-technology/2016/12/millions-exposed-to-malvertising-that-hid-attack-code-in-banner-pixels/

Malvertising Example

https://www.forbes.com/sites/leemathews/2017/08/03/over-a-million-coders-targeted-by-chrome-extension-hack/#670c91059c9d

ATT&CK Technique Matrix

https://attack.mitre.org/wiki/Technique_Matrix

https://t.me/learningnets

References

ATT&CK (Evading Defenses)

https://attack.mitre.org/wiki/Defense_Evasion

ATT&CK (Execution)

https://attack.mitre.org/wiki/Execution

ATT&CK (Escalate Privileges)

https://attack.mitre.org/wiki/Privilege_Escalation

ATT&CK (Steal Credentials)

https://attack.mitre.org/wiki/Credential_Access

https://t.me/learningnets

References

ATT&CK (Exfiltration)

https://attack.mitre.org/wiki/Exfiltration

ATT&CK (Persistence)

https://attack.mitre.org/wiki/Persistence

Alternate Data Streams

https://docs.microsoft.com/en-us/archive/blogs/askcore/alternate-data-streams-in-ntfs

CreateFile

https://docs.microsoft.com/en-us/windows/win32/api/fileapi/nf-fileapi-createfilea

https://t.me/learningnets

References

WriteFile

https://docs.microsoft.com/en-us/windows/win32/api/fileapi/nf-fileapi-writefile

Streams (tool)

https://docs.microsoft.com/en-us/sysinternals/downloads/streams

Sysinternals Suite

https://docs.microsoft.com/en-us/sysinternals/downloads/

Get-Item

https://docs.microsoft.com/en-us/powershell/module/Microsoft.PowerShell.Management/Get-Item?view=powershell-5.1

https://t.me/learningnets

References

Get-Content

https://docs.microsoft.com/en-us/powershell/module/Microsoft.PowerShell.Management/Get-Content?view=powershell-5.1

PowerShell v5

https://docs.microsoft.com/en-us/powershell/scripting/whats-new/what-s-new-in-windows-powershell-50?view=powershell-7

CoinMiner

https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Trojan:Win32/CoinMiner

CoinMiner Analysis

https://secrary.com/ReversingMalware/CoinMiner/

https://t.me/learningnets

References

[Process Injection]

https://attack.mitre.org/wiki/Technique/T1055

[CreateToolhelp32Snapshot]

https://msdn.microsoft.com/en-us/library/windows/desktop/ms682489(v=vs.85).aspx

[Process32First]

https://msdn.microsoft.com/en-us/library/windows/desktop/ms684834(v=vs.85).aspx

[Process32Next]

https://msdn.microsoft.com/en-us/library/windows/desktop/ms684836(v=vs.85).aspx

https://t.me/learningnets

References

GetModuleHandle

https://msdn.microsoft.com/en-us/library/windows/desktop/ms683199(v=vs.85).aspx

GetProcAddress

https://msdn.microsoft.com/en-us/library/windows/desktop/ms683212(v=vs.85).aspx

OpenProcess

https://msdn.microsoft.com/en-us/library/windows/desktop/ms684320(v=vs.85).aspx

VirtualAllocEx

https://msdn.microsoft.com/en-us/library/windows/desktop/aa366890(v=vs.85).aspx

https://t.me/learningnets

References




[WriteProcessMemory]

https://msdn.microsoft.com/en-us/library/windows/desktop/ms681674(v=vs.85).aspx

[CreateRemoteThread]

https://msdn.microsoft.com/en-us/library/windows/desktop/ms682437(v=vs.85).aspx

[LoadLibrary]

https://msdn.microsoft.com/en-us/library/windows/desktop/ms684175(v=vs.85).aspx

[NTCreateThreadEx]

https://undocumented.ntinternals.net/index.html?page=UserMode/Undocumented%20Functions/NT%20Objects/Thread/NtCreateThread.html

https://t.me/learningnets

References

RtlCreateUserThread

https://undocumented.ntinternals.net/index.html?page=UserMode/Undocumented%20Functions/Executable%20Images/RtlCreateUserThread.html

Reflective DLL Injection

https://github.com/stephenfewer/ReflectiveDLLInjection

Thread32First

https://msdn.microsoft.com/en-us/library/windows/desktop/ms686728(v=vs.85).aspx

Thread32Next

https://msdn.microsoft.com/en-us/library/windows/desktop/ms686731(v=vs.85).aspx

https://t.me/learningnets

References

OpenThread

https://msdn.microsoft.com/en-us/library/windows/desktop/ms684335(v=vs.85).aspx

SuspendThread

https://msdn.microsoft.com/en-us/library/windows/desktop/ms686345(v=vs.85).aspx

ResumeThread

https://msdn.microsoft.com/en-us/library/windows/desktop/ms685086(v=vs.85).aspx

Peering Inside the PE

https://msdn.microsoft.com/en-us/library/ms809762.aspx

https://t.me/learningnets

References

Process Hollowing

https://attack.mitre.org/wiki/Technique/T1093

CreateProcess

https://msdn.microsoft.com/en-us/library/windows/desktop/ms682425(v=vs.85).aspx

ZwUnmapViewOfSection

https://msdn.microsoft.com/ru-ru/library/ff567119(v=vs.85).aspx

NtUnmapViewOfSection

https://msdn.microsoft.com/en-us/library/ff557711(v=vs.85).aspx

https://t.me/learningnets

References

SetThreadContext

https://msdn.microsoft.com/en-us/library/windows/desktop/ms680632(v=vs.85).aspx

SetWindowsHookEx

https://msdn.microsoft.com/en-us/library/windows/desktop/ms644990(v=vs.85).aspx

Hook Types

https://msdn.microsoft.com/en-us/library/windows/desktop/ms644959(v=vs.85).aspx

NTQueryDirectoryFile (ZwQueryDirectoryFile)

https://msdn.microsoft.com/en-us/library/windows/hardware/ff556633(v=vs.85).aspx

https://t.me/learningnets

References

[Windows Driver Development Kit]

https://docs.microsoft.com/en-us/windows-hardware/drivers/download-the-wdk

[Direct Kernel Object Manipulation]

https://www.blackhat.com/presentations/win-usa-04/bh-win-04-butler.pdf

[NTOpenProcess]

https://msdn.microsoft.com/en-us/library/windows/hardware/ff556571(v=vs.85).aspx

[EPROCESS List]

https://msdn.microsoft.com/en-us/library/windows/hardware/ff544273(v=vs.85).aspx

https://t.me/learningnets

References

ActiveProcessLinks (Linked List)

https://msdn.microsoft.com/en-us/library/windows/hardware/ff554296(v=vs.85).aspx

Svchost Abuse

http://www.hexacorn.com/blog/2013/07/04/the-typographical-and-homomorphic-abuse-of-svchost-exe/

UPX Packer

https://upx.github.io/

AutoRuns (tool)

https://docs.microsoft.com/en-us/sysinternals/downloads/autoruns

https://t.me/learningnets

References

APT3 (group)

https://attack.mitre.org/wiki/Group/G0022

COM Hijacking

https://attack.mitre.org/wiki/Technique/T1122

DLL Search Order Hijacking

https://attack.mitre.org/wiki/Technique/T1038

DLL Search Order (MSDN)

https://msdn.microsoft.com/en-us/library/windows/desktop/ms682586(v=vs.85).aspx

https://t.me/learningnets

References




Phantom DLLs

http://www.hexacorn.com/blog/2016/06/02/beyond-good-ol-run-key-part-40/

DLL Side Loading

https://attack.mitre.org/wiki/Technique/T1073

DLL Side-Loading (FireEye Report)

https://www.fireeye.com/content/dam/fireeye-www/global/en/current-threats/pdfs/rpt-dll-sideloading.pdf

Service Creation

https://www.bleepingcomputer.com/tutorials/how-malware-hides-as-a-service/

https://t.me/learningnets

References

Service Recovery Actions

https://technet.microsoft.com/en-us/library/cc753662(v=ws.11).aspx

COM Object Model

https://docs.microsoft.com/en-us/windows/win32/com/the-component-object-model

CreateToolhelp32Snapshot

https://msdn.microsoft.com/en-us/library/windows/desktop/ms682489(v=vs.85).aspx

WriteProcessMemory

https://msdn.microsoft.com/en-us/library/windows/desktop/ms681674(v=vs.85).aspx

https://t.me/learningnets

References

Windows Sysinternals

https://docs.microsoft.com/en-us/sysinternals/

Registry Run Keys / Startup Folder

https://attack.mitre.org/wiki/Technique/T1060

https://t.me/learningnets