CryptBot Infostealer: Malware Analysis

ANY.RUN

January 26, 2023

HomeMalware Analysis
CryptBot Infostealer: Malware Analysis
We recently analyzed CryptBot, an infostealer detected by the ANY.RUN online malware sandbox.

Through our research, we collected information about MITRE ATT&CK techniques used by this malware. We also learned about how this infostealer stores and encrypts its configuration information, and we wrote a Python script to extract the configuration.

Let’s go over the whole process step-by-step.

Brief description of CryptBot malware

CryptBot is an infostealer targeting Windows operation systems that was first discovered in the wild in 2019. It is designed to steal sensitive information from infected computers, such as credentials for browsers, cryptocurrency wallets, browser cookies, credit card information, and screenshots of the infected system. It is distributed through phishing emails and cracked software.

CryptBot malware

CryptBot dynamic analysis in a malware sandbox

During the analysis we’ll take a look at the sample:

MD5: 12d20a973f8cd9c6373929ae14efe123
URL: https://app.any.run/tasks/5c6e7021-f223-495c-a332-21ef1276e4cf

A single process (Fig. 1) is created when the malware starts, which actively uses the file system (15k+ events) and the registry (2k+ events).

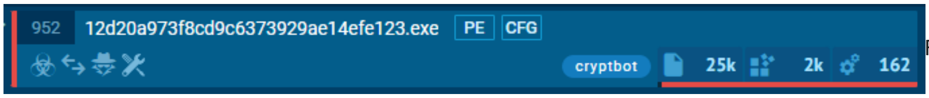

Fig. 1 — CryptBot's process

Ok, now that we got the basics out of the way, let's break down this malware and list all of the techniques it uses. We'll break sort the information by technique as we go from here.

Credentials from password stores: credentials from web browsers (T1555.003)

CryptBot steals information from popular browsers — Chrome, Firefox, and Edge, as the "Actions looks like stealing of personal data" indicator (Fig. 2) and "Reads browser cookies" indicators tell us:

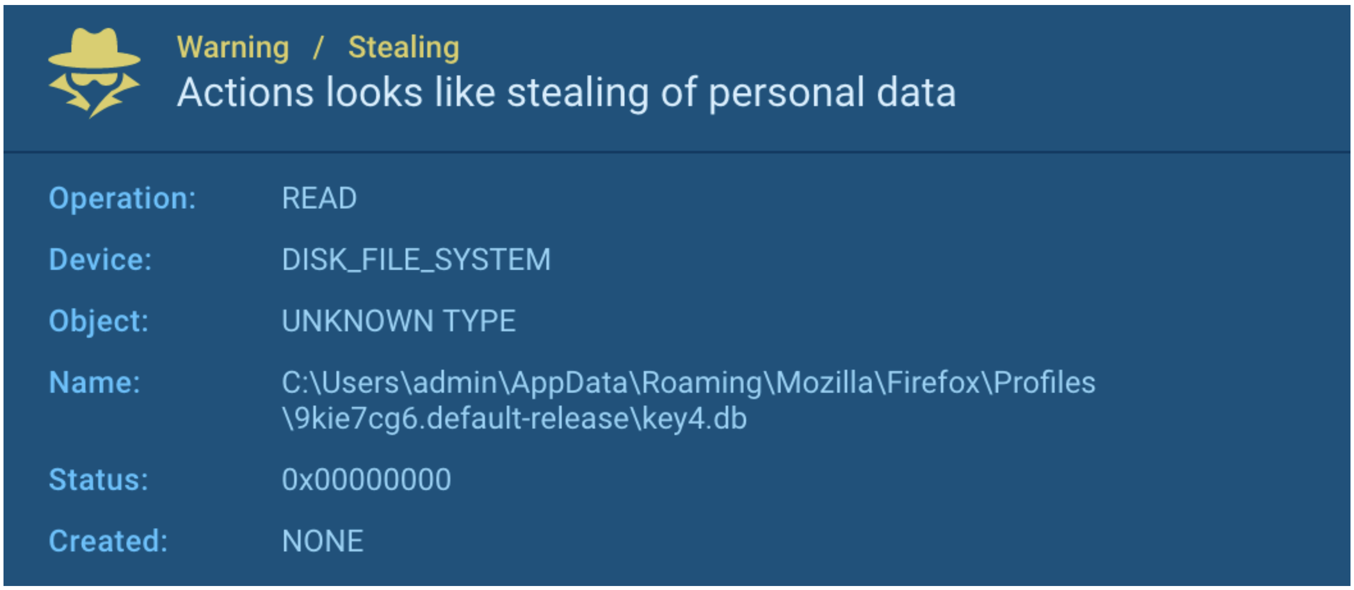

Fig. 2 — CryptBot steals Firefox data

To detect access to personal data stored in the browser, we can use the pseudo-signature:

```
process_name NOT (“chrome.exe”, ”firefox.exe”, “msedge.exe”, “opera.exe”)
AND
file_access (
%LOCALAPPDATA%\\MICROSOFT\\EDGE\\USER DATA\\*,
%APPDATA%\\Roaming\\Mozilla\\Firefox\\*,
%LOCALAPPDATA%\\Local\\Google\\Chrome\\User Data\\*
%LOCALAPPDATA%\\AppData\\Local\\Opera Software\\Opera Stable\*
)
```

Software discovery (T1518)

CryptBot checks the presence of installed software in the system by going through the "Uninstall" registry tree (Fig. 3):

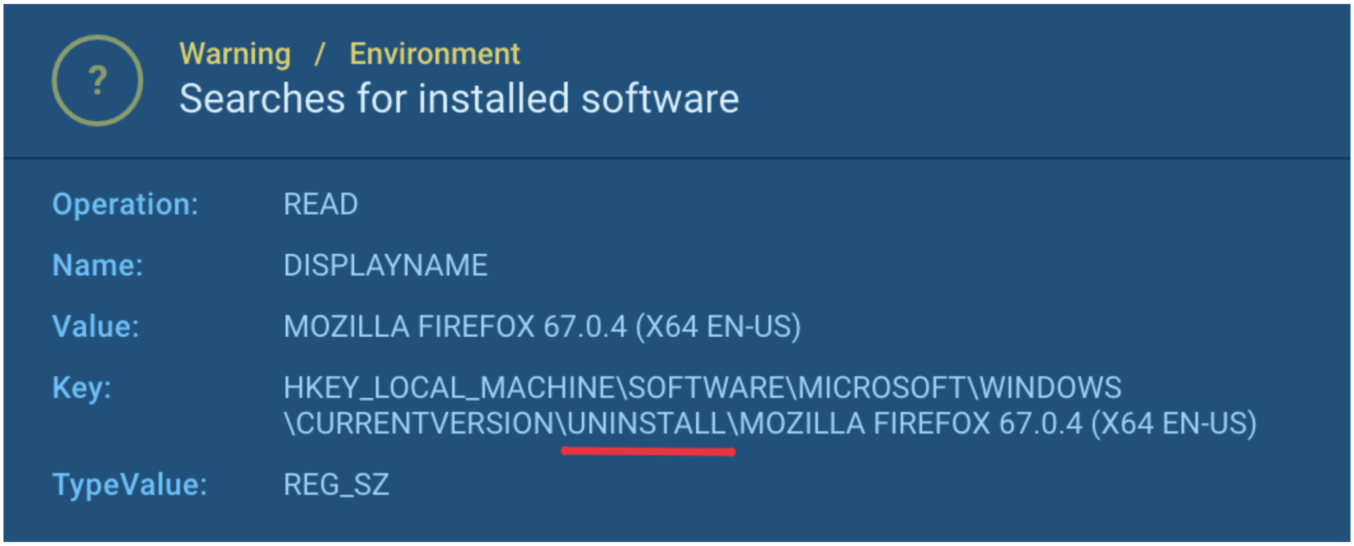

Fig. 3 — CryptBot searches for installed software

To detect an attempt to access the list of installed software, we can use a pseudo-signature:

```
reg_key is (“HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall”)
AND
operation read
```

System information discovery (T1082)

The malware collects system information, including operating system installation date, computer name, key, CPU information, and this behavior triggers the corresponding indicators (Fig. 4):

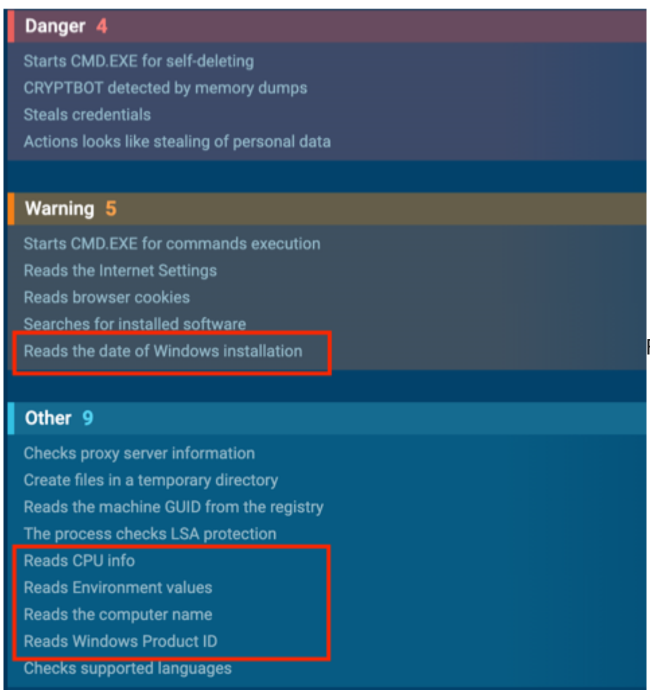

Fig. 4 — CryptBot collects system information

It is possible to detect the collection of system configuration information by accessing certain registry keys. For example, reading the system installation date can be detected by the following pseudo-signature:

```
reg_key is (“HKLM\SOFTWARE\MICROSOFT\WINDOWS NT\CURRENTVERSION”)
AND
reg_name is (“INSTALLDATE”)
AND
operation read
```

Application layer protocol: web protocols (T1071.001)

CryptBot sends the collected OS information and personal data to the control server, which we can see in multiple connection attempts (see Figure 5):

Fig. 5 — CryptBot attempts to send data to the control server

We can detect attempts to connect to the C2 server with the following pseudo-signature:

```
network connect
AND
(
domains are (“sginiv12[.]top” or “bytcox01[.]top”)
OR (ip == “23[.]217.138.108” and port==80)
)
```

Additionally, we investigated the content of the network stream and detected that the data is sent through the HTTP protocol, using a POST request with an attached file (see Fig. 6). Having restarted the malware several times we found that the file name is most likely randomly generated. However, the request is always sent to the “gate.php” page.

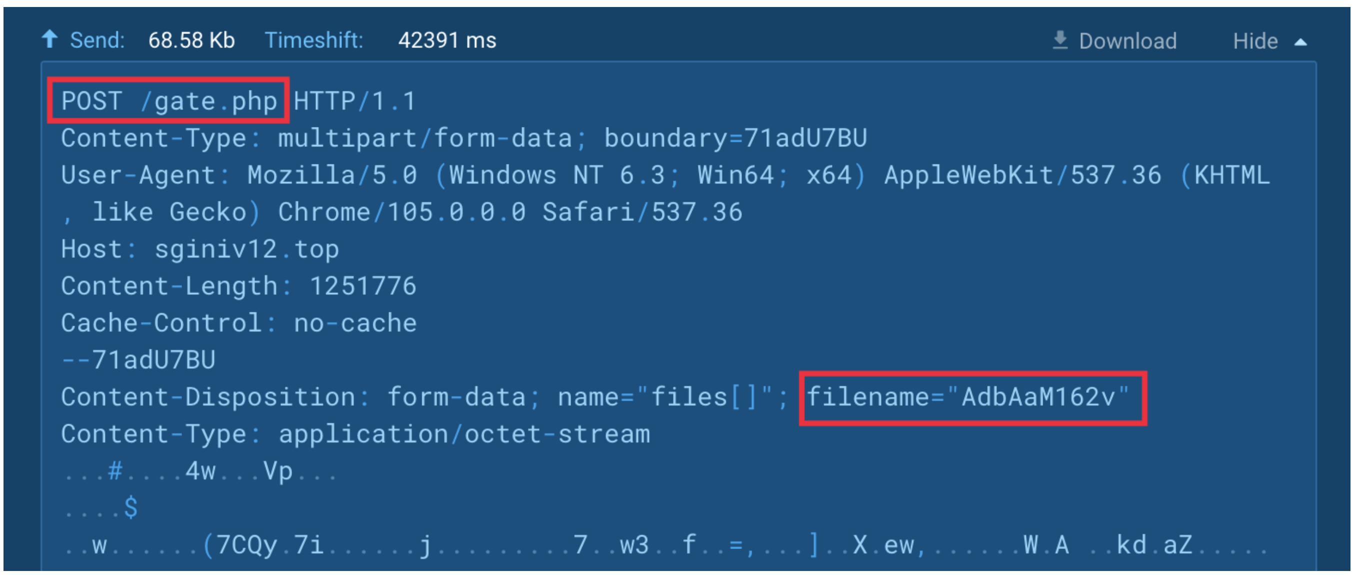

Fig. 6 — Malware sends information to the control server

Potentially malicious traffic is also detected in the results of the Suricata (see Fig. 7):

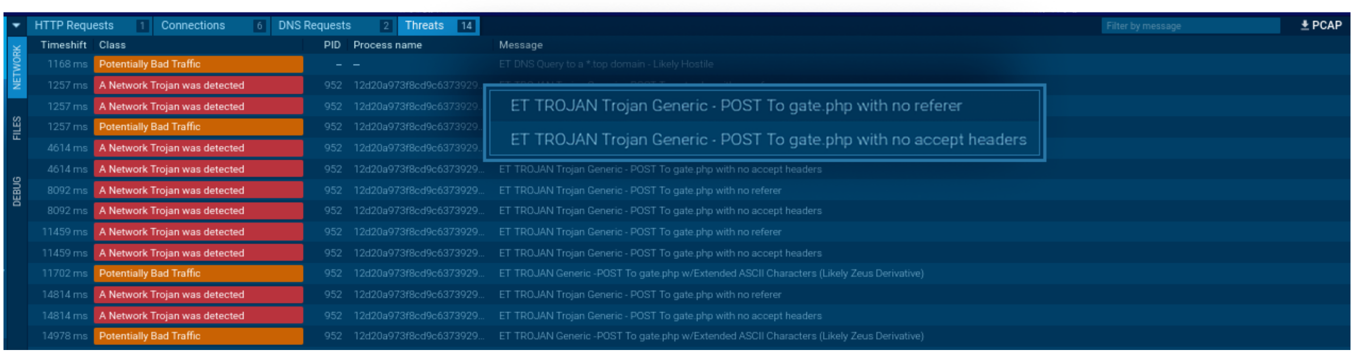

Fig. 7 — Potentially malicious traffic detected by the Suricata rules

Let’s create a pseudo-signature to detect CryptBot in the traffic:

```
network send
AND
http_verb is “POST” AND location is “gate.php”
AND
http_content includes (“form-data”, “name=\”files[]\””, “filename”)
```

Analyzing the contents of the transmitted file gives nothing of interest, since it is probably encrypted.

Data staged: local data staging (T1074.001)

1. Preventing re-runs

When we launch the malware for the first time in the “%APPDATA%” directory an empty directory-marker “0D445946B53E9551” is created (Figure 8). This directory allows the Malicious software to determine whether it has been launched before. If the CryptBot is restarted, it will stop working immediately.

Marker-directory 0D445946B53E9551

Fig. 8 — Marker-directory 0D445946B53E9551

Let’s make a pseudo-signature to detect the creation of the marker directory:

```
action create_directory
AND
directory_name is (“^%APPDATA%\\[A-F0-9]{16}$”)
```

2. Storing collected data

Collected information is stored in temporary files in various formats (sqlite, binary, text) in the %TEMP% directory (Fig. 9):

Files modification 68 | Only important | Filter by filename

Timeshift	PID	Process name	Filename	Content
265 ms	856	12d20a973f8cd9c637...	C:\Users\admin\AppData\Local\Temp\F5BF.tmp	87.0 Kb text
265 ms	856	12d20a973f8cd9c637...	C:\Users\admin\AppData\Local\Temp\F5E0.tmp	87.0 Kb text
265 ms	856	12d20a973f8cd9c637...	C:\Users\admin\AppData\Local\Temp\F5E1.tmp	32 b binary
265 ms	856	12d20a973f8cd9c637...	C:\Users\admin\AppData\Local\Temp\F5BF.tmp	87.0 Kb text
265 ms	856	12d20a973f8cd9c637...	C:\Users\admin\AppData\Local\Temp\F5E0.tmp	87.0 Kb text
265 ms	856	12d20a973f8cd9c637...	C:\Users\admin\AppData\Local\Temp\F5E1.tmp	32 b binary
281 ms	856	12d20a973f8cd9c637...	C:\Users\admin\AppData\Local\Temp\F5F2.tmp	120 Kb sqlite
281 ms	856	12d20a973f8cd9c637...	C:\Users\admin\AppData\Local\Temp\F5F2.tmp	120 Kb sqlite

Fig. 9 — Temporary files in the %TEMP% directory

For example, in Fig. 10 we see the content of one of the created temporary files, where information about the stolen logins and passwords is stored in Base64 format. Note that the data also includes a website to which each login-password pair corresponds:

PREVIEW EXIF HEX


```
{"nextId":3,"logins":
[{"id":2,"hostname":"https://www.facebook.com","httpRealm":null,"formSubmitURL":"","usernameField":"","passwordField"
:"","encryptedUsername":"MDoEEPgAAAAAAAAAAAAAAAAAAAEwFAYIKoZIhvcNAwcECDwFMZ75BKpsBBDNd+SLlHlsCHKyF
/RoXEs+","encryptedPassword":"MDoEEPgAAAAAAAAAAAAAAAAAAAEwFAYIKoZIhvcNAwcECL
/58rtWXdKbBBAQnB78n1fiZDTMi52lUBo9","guid":"{291ea46a-56ab-4330-bb38-
fe76fd9620a1}","encType":1,"timeCreated":1532091682000,"timeLastUsed":1532091682000,"timePasswordChanged":15320916820
00,"timesUsed":1}],"disabledHosts":[],"version":2}
```


Fig. 10 — The contents of the files with the collected information

To detect the creation of temporary files with personal data, we can, for example, apply the following pseudo-signature:

```
process_name NOT (“chrome.exe”)
AND
file_create (“%TEMP%\\*.tmp”)
AND
file_content includes (
*username*,
*password*
)
```

Indicator removal: file deletion (T1070.004)

When the malware is done running, it removes itself using CMD.EXE with a short delay to give the process time to finish and unblock the executable file (Fig. 11):

Danger / General

Starts CMD.EXE for self-deleting

Image: C:\Windows\SysWOW64\cmd.exe

Cmdline: "C:\Windows\System32\cmd.exe" /c timeout -t 5 && del "C:\Users\admin\Desktop\12d20a973f8cd9c6373929ae14efe123.exe"

Fig. 11 — The malware self-deletes

We can use the following pseudo-signature in the command line for detection:

```
process_name is (“cmd.exe”)
AND
command_line includes (“timeout”, “del”)
```

CryptBot dynamic analysis using a debugger

Static packer check

In general, it’s a best practice to check the file statically to figure out its type and if there’s a packer present, before conducting the dynamic analysis. Once we do that with the DiE tool shows that the file is not packed (see fig.12):

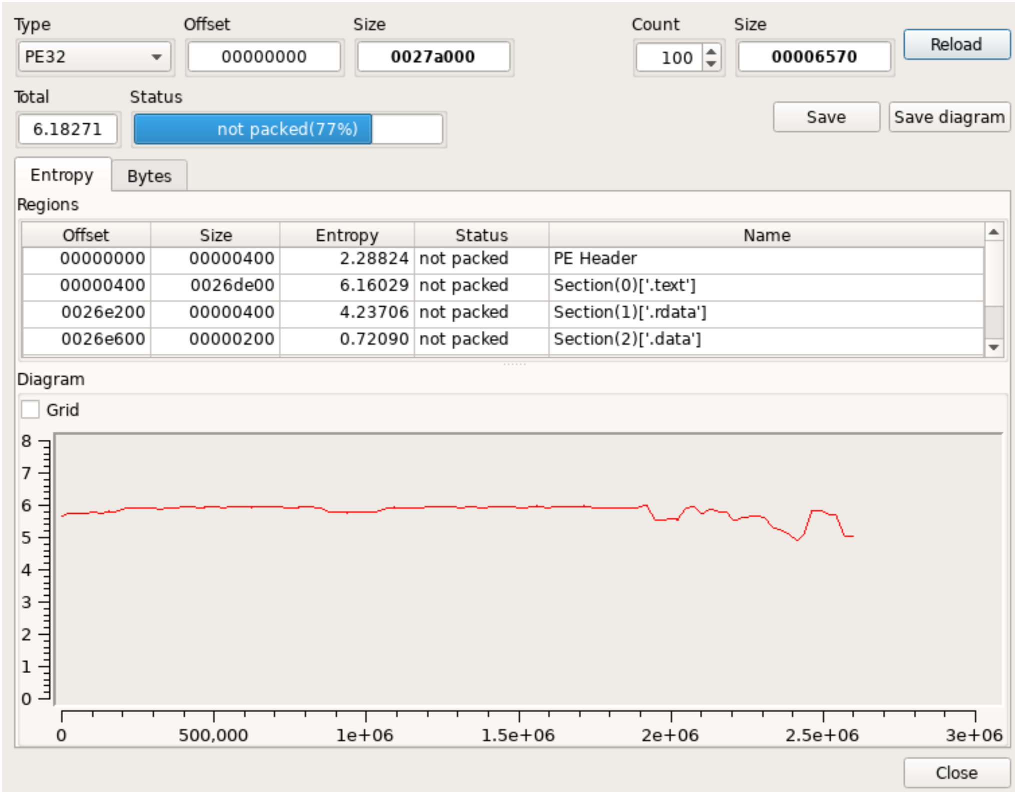

Fig. 12 — Checking the malware file statically to detect a packer

In this case, even though we didn't find a packer during our static analysis, the dynamic analysis revealed that the malware uses a T1027.002 – software packing technique.

Obfuscated files or information: software packing (T1027.002)

By analyzing the memory of a running process using Process Hacker, we stumble upon an RWX region that is not normally found in legitimate programs. The beginning of the dump of this region allows you to see the header of the PE file (see Fig. 13):

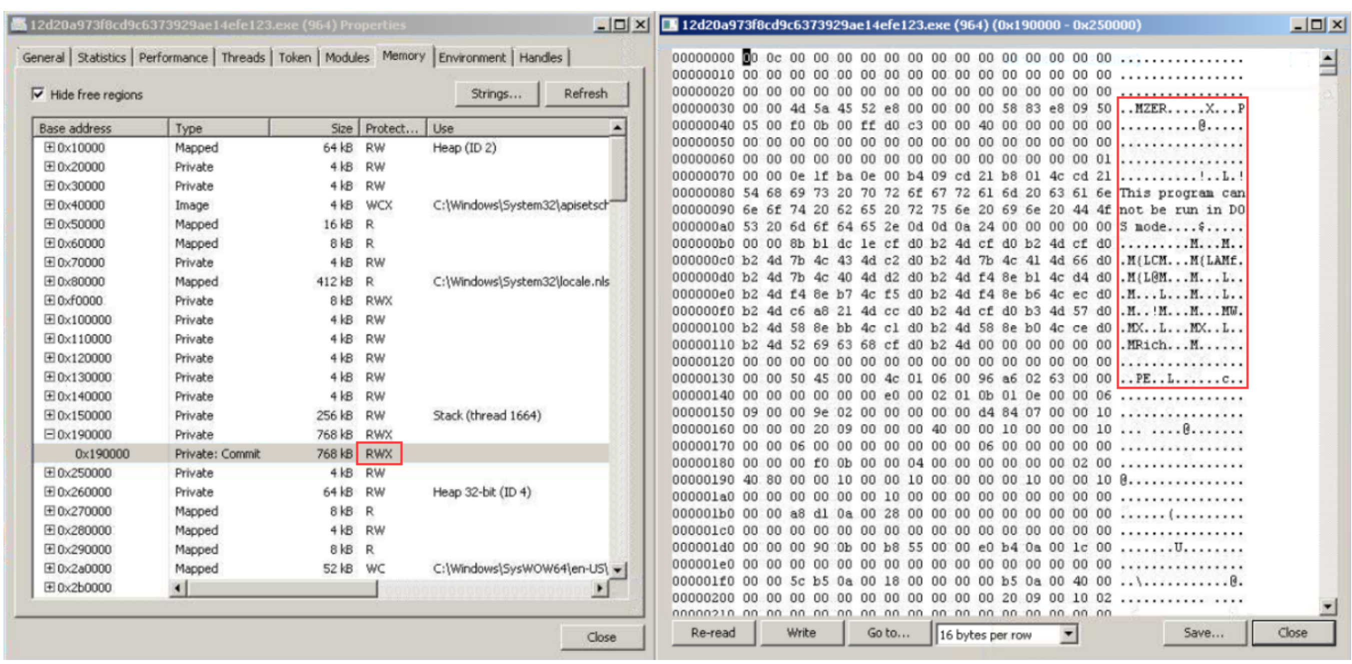

Fig. 13 — CryptBot's memory dump of a running process

On further analysis we discovered that the header of the PE file is also the beginning of the **shellcode** (see Fig. 14), which recovers the register value, gets the ImageBase and passes control to the EntryPoint:

```
user@b552020af7b7:~$ disasm 4D 5A 45 52 E8 00 00 00 00 58 83 E8
09 50 05 00 F0 0B 00 FF D0 C3
   0:    4d                      dec    ebp
   1:    5a                      pop    edx
   2:    45                      inc    ebp
   3:    52                      push   edx
   4:    e8 00 00 00 00          call   0x9
   9:    58                      pop    eax
   a:    83 e8 09                sub    eax,  0x9
   d:    50                      push   eax
   e:    05 00 f0 0b 00          add    eax,  0xbf000
  13:    ff d0                   call   eax
  15:    c3                      ret
```

Figure 14 — Disassembling the PE header

Using the x64dbg debugger we have determined that the executable memory region is allocated by the unpacker using the WinAPI's **VirtualAlloc** function. Next, the unpacker writes payload to it and decrypts it with an **XOR** operation (see Figure 15):

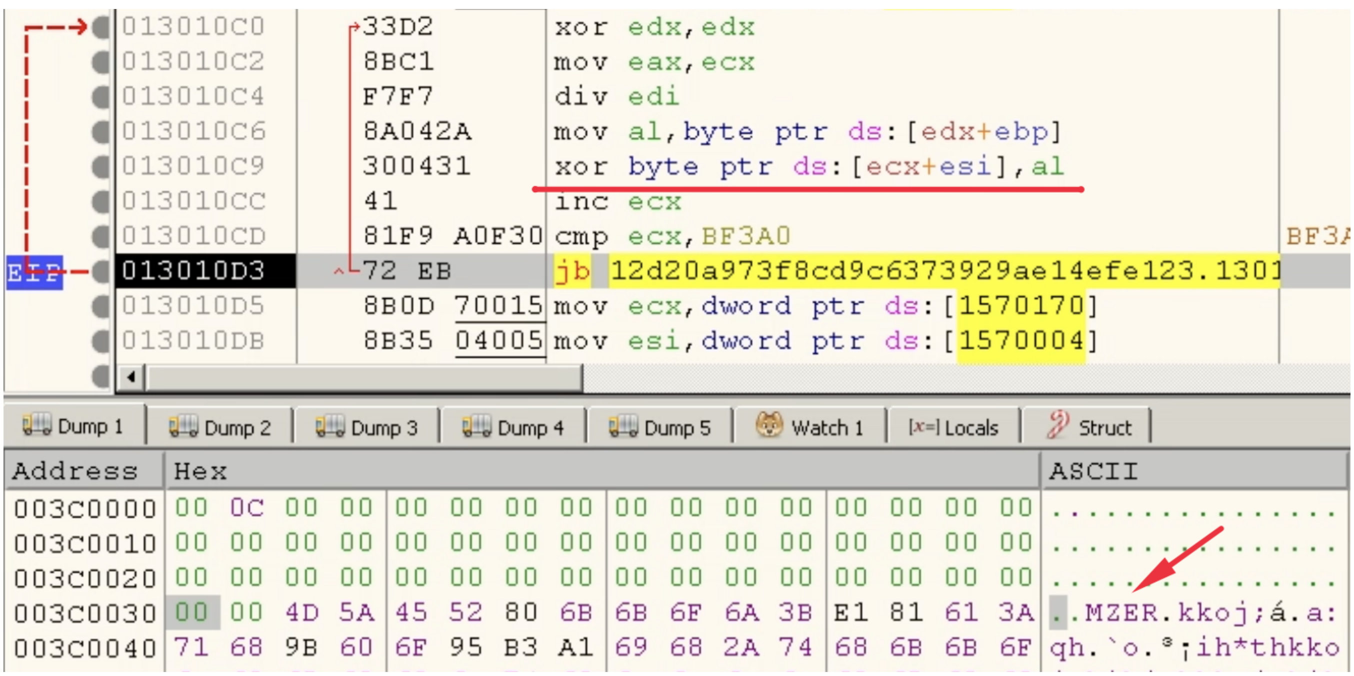

Fig. 15 — Decrypting payload using XOR

The **key** to decrypt the payload is in the ".rdata" section of the running executable:

```
Address   Hex                                              ASCII
0156F01F  00 32 00 00 00 FA 16 00 00 A0 F3 0B 00 69 68 6A  .2...ú... ó..ihj
0156F02F  74 68 6B 6B 6F 6A 63 62 00 0E 2A 68 05 74 05 6B  thkkojcb..*h.t.k
0156F03F  38 6F 1A 63 07 69 0B 6A 00 00 00 00 00 14 08 68  8o.c.i.j.......h
```

Fig. 16 — Key to decrypt the payload

Thus, we can see that despite the absence of features of the payload in the static analysis, using the dynamic one **we have identified the presence of a packer and determined the key and the encryption algorithm.**

Writing YARA rules to detect CryptBot shellcode in memory

A YARA rule for detecting a CryptBot shellcode in OS memory could look like this:

```
rule CryptBot_ShellCode
{
meta:
        author = "Any.Run"
        SHA256 = "183f842ce161e8f0cce88d6451b59fb681ac86bd3221ab35bfd675cb42f056ac"
        date = "2023-01-19"
        description = "Detect CryptBot shellcode in memory"

strings:
        $shellcode = { 4D 5A 45 52 E8 00 00 00 00 58 83 E8 09 50 05 [4] FF D0 C3 }

condition:
        uint16(0) != 0x5A4D and
        uint16(0) > 0 and
        $shellcode in (0x20..0x50)
}
```

Static analysis and configuration decoding

Finding and deciphering the configuration

The static analysis of the payload code led us to the conclusion that the malware configuration is located in the ".data" section and encrypted with an XOR operation. Moreover, the decryption key lies in plaintext just before the encrypted data (see Figure 17):

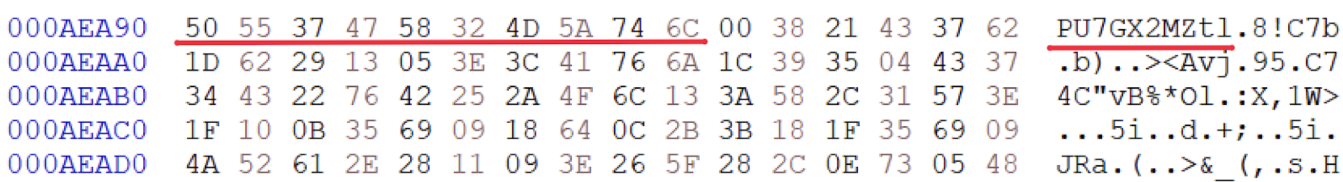

Fig. 17 — Key and encrypted configuration

The configuration is easily decrypted using CyberChef and the key "PU7GX2MZtl" (see Fig. 18):

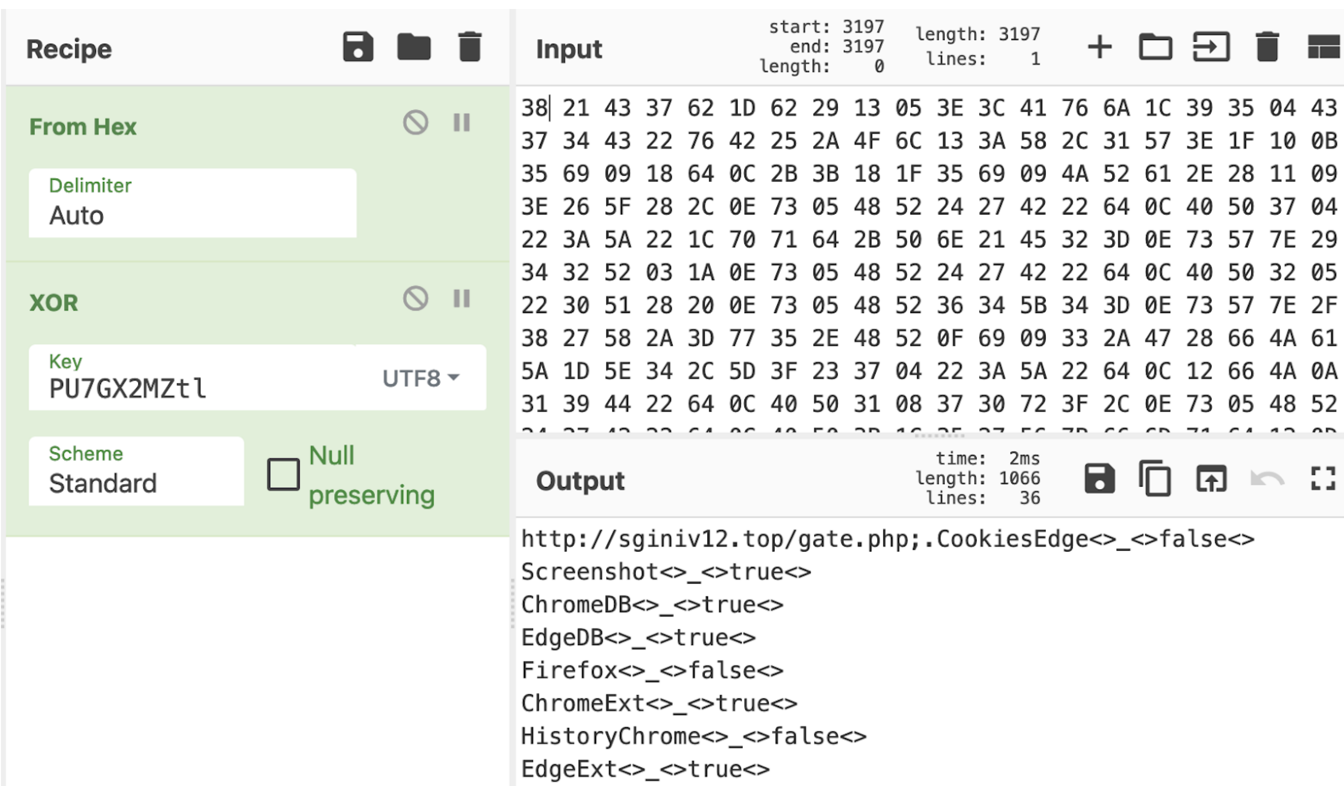

Figure 18 — CryptBot decrypted configuration

From the decrypted configuration it becomes clear what information should be stolen by CryptBot. For example, the screenshot variable tells the malware to take a screenshot, and ChromeExt — to steal data from Chrome extensions.

Automating configuration decryption

We have automated the CryptBot configuration extraction in Python and made the script public. You can always find it in our Git repo. The result of the unpacked payload script is shown in Fig. 19:

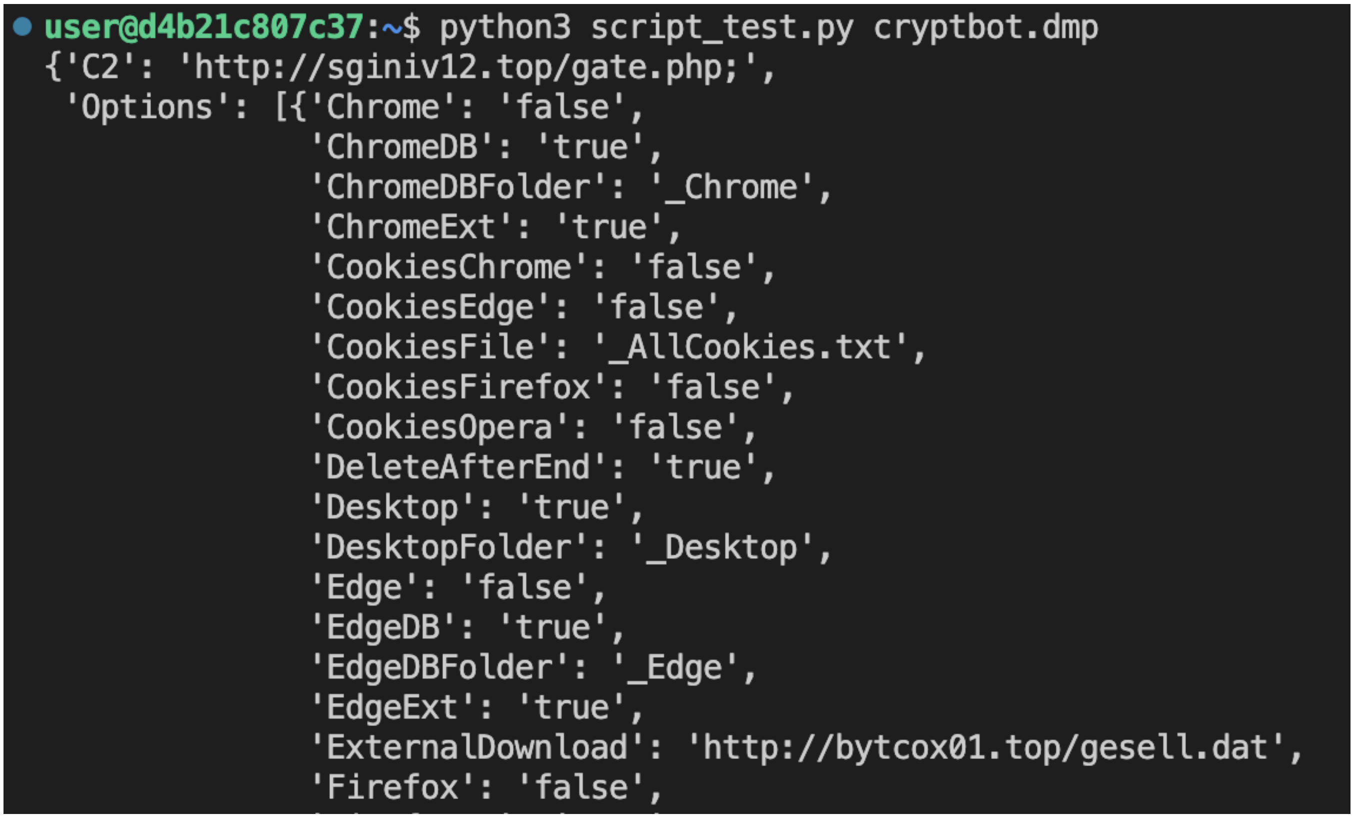

Fig. 19 — The result of the configuration extraction script

Developing YARA Rules for detecting CryptBot configuration in memory

Some strings of the decrypted CryptBot configuration can be used as part of a YARA rule to detect it in memory:

```
rule CryptBot_Config {
meta:
        author = "Any.Run"
        SHA256 = "183f842ce161e8f0cce88d6451b59fb681ac86bd3221ab35bfd675cb42f056ac"
        date = "2022-01-19"
        description = "Detect CryptBot configuration in memory"
strings:
        $s1 = "CookiesEdge"
        $s2 = "ChromeDB<>_<>"
        $s3 = "EdgeDB<>_<>"
        $s4 = "ChromeExt<>_<>"
        $s5 = "HistoryChrome<>_<>"
        $s6 = "EdgeExt<>_<>"
        $s7 = "CookiesFirefox<>_<>"
        $s8 = "HistoryOpera<>_<>"
        $s9 = "CookiesOpera<>_<>"
        $s10 = "FirefoxDB<>_<>"
        $s11 = "CookiesChrome<>_<>"
        $s12 = "HistoryFirefox<>_<>"
        $s13 = "HistoryEdge<>_<>"
        $s14 = "DesktopFolder<>_<>"
        $s15 = "ChromeDBFolder<>_<>"
        $s16 = "ExternalDownload<>_<>"
        $s17 = "ScreenFile<>_<>"
        $s18 = "MessageAfterEnd<>_<>"
        $s19 = "HistoryFile<>_<>"
        $s20 = "FirefoxDBFolder<>_<>"
        $s21 = "PasswordFile<>_<>"
        $s22 = "WalletFolder<>_<>"
        $s23 = "DeleteAfterEnd<>_<>"
        $s24 = "EdgeDBFolder<>_<>"
        $s25 = "InfoFile<>_<>"
        $s26 = "CookiesFile<>"

condition:
        7 of them
}
```

Using ANY.RUN to efficiently analyze CryptBot

For your convenience, we have integrated automatic extraction of the CryptBot configuration into ANY.RUN interactive sandbox — just run the sample and get all the IOCs in seconds (Fig. 20):

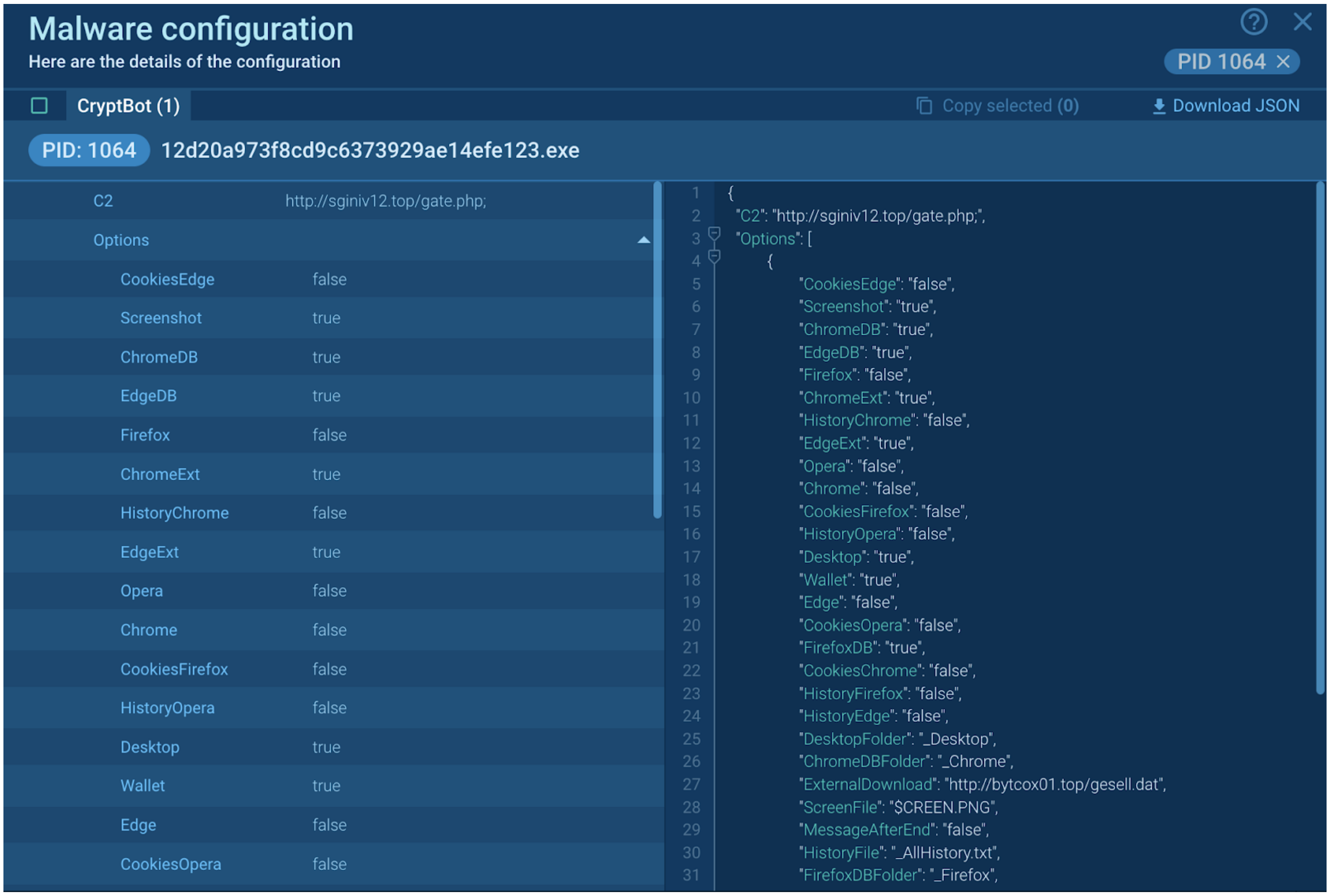

Fig. 20 – Automatic CryptBot configuration extraction in ANY.RUN sandbox

Conclusion

In this article, we looked into CryptBoT, its techniques and behavior when contained in the ANY.RUN sandbox. We also wrote a configuration extractor that you can use to gather and interpret the data.

Fortunately, ANY.RUN is already set up to detect this malware automatically, making the relevant configuration details just a click away.

If you want to read more content like this, check out our analysis of the Raccoon Stealer, or Orcus RAT.

Appendix

Analyzed files

Title	Description
Name	12d20a973f8cd9c6373929ae14efe123.exe
MD5	12d20a973f8cd9c6373929ae14efe123
SHA1	7f277f5f8f9c2831d40a2dc415566a089a820151
SHA256	183f842ce161e8f0cce88d6451b59fb681ac86bd3221ab35bfd675cb42f056ac

Extracted URLs

- http://sginiv12[.]top/gate.php
- http://bytcox01[.]top/gesell.dat

MITRE (ARMATTACK)

Tactics	Techniques	Description
TA0005: defence evasion	T1070.004: Indicator Removal: File Deletion	Self-deleting after completion
	T1027.002: Obfuscated Files or Information: Software Packing	Malware is decrypted into memory before it starts working
TA0006: Credential access	T1555.003: Credentials from Web Browsers	Steals data from installed browsers
TA0007: Software discovery	T1518: Software Discovery	Searches for installed software in the system in the "Uninstall" key
	T1082: System Information Discovery	Collects system data
TA0009: Collection	T1113: Screen capture	Has an option to take a configuration screenshot
	T1074: Data Staged	Saving of gathered data in a temporary directory before sending; prevention of relaunch
TA0011: Command and Control	T1071: Application Layer Protocol	Sending collected data to the control server

malware analysis

What do you think about this post?

16 answers

- Awful
- Average
- Great

No votes so far! Be the first to rate this post.

1 comments