2022

Practical malware analysis

MALWARE.UNKNOWN.EXE.MALZ

DIRK F.

Table of contents

LIST OF FIGURES **2**
EXECUTIVE SUMMARY **3**
STATIC ANALYSIS **4**
DYNAMIC ANALYSIS **7**
YARA RULES **12**
FINDINGS SUMMARY **12**
OBSERVED INDICATORS OF COMPROMISE **13**

List of figures

Figure 1 - File-type and -architecture identification 4
Figure 2 - PE-header analysis result from pedump 4
Figure 3 - Abbreviated floss output for the specimen 4
Figure 4 - Imports by the specimen as identified by peframe 5
Figure 5 - DetectItEasy analysis result 5
Figure 6 - De-obfuscation of strings contained in specimen 6
Figure 7 - Process Hacker output when running the specimen 7
Figure 8 - Process Monitor tree-view output 7
Figure 9 - Process Monitor tree for specimen with "cosmo.jpeg" present 8
Figure 10 - Specimen beaconing to hey[.]youup[.]local (Wireshark) 8
Figure 11 - GET request details for the "beacon" traffic (Wireshark) 9
Figure 12 - ProcessMonitor with DNS traffic and access to cosmo.jpeg 9
Figure 13 - fakends output for specimen running without access to the beacon-URL 10
Figure 14 - Wireshark capture of subdomain DNS resolution 10
Figure 15 – base64 decoded subdomain 10
Figure 16 - Modified "cosmo.jpeg" for verification 10
Figure 17 - Wireshark traffic running the specimen on updated "cosmo.jpeg" 11
Figure 18 - base64 decoded subdomain request strings 11
Figure 19 - Indicators of compromise found during analysis 13

Executive summary

Sample name	Malware.unknown.exe.malz
SHA1 Hash	ec0d565afe635c2c7863b2a05df8a49c58b703a3
MD5 Hash	812a7c7eb9d7a4332b9e166aa09284d7
VirusTotal detection rate	4 detections (generic / CobaltStrike)
Source	GitHub Link
Operating system and architecture	Windows, 32Bit
Language Compiler	Nim Language (not packed)
Analysis date	13.02.2022
Analysis author	Dirk F.

The specimen examined for this report, was obtained from a GitHub address in .7z format. At the time of report creation, it has a very low detection rate on VirusTotal with only four generic detections (MalwareBytes flags it as CobaltStrike).

Static- and dynamic analysis, lead to the conclusion that the specimen's purpose is to exfiltrate data via DNS traffic. The malware tries to connect to a hard-coded domain and only proceeds with the exfiltration if the domain **can't be reached** ("kill-switch").

The only network traffic observed is to the kill-switch domain and to DNS for data exfiltration. No additional activities (downloading additional stages of malware, persistence mechanisms, project injection/spawning) has been observed during analysis.

Three Indicators of Compromise (one host based, two network based) have been observed that can help identify the specimen.

The next chapters will substantiate the findings above via static- and dynamic analysis.

Static analysis

File type and target operating system analysis confirm, the specimen targets Microsoft Windows operating systems and has been compiled for 32bit versions of Windows:

file

```
remnux@remnux:~/malware/PMAT$ file Malware.unknown.exe.malz
Malware.unknown.exe.malz: PE32 executable (GUI) Intel 80386 (stripped to external PDB), for MS
Windows
```

Figure 1 - File-type and -architecture identification

Looking at the PE header, no indications of packer usage are apparent

pedump

=== SECTIONS ===

NAME	RVA	VSZ	RAW_SZ	RAW_PTR	nREL	REL_PTR	nLINE	LINE_PTR	FLAGS	
.text	1000	19b14	19c00	400	0	0	0	0	60500060	R-X
.data	1b000	a0	200	1a000	0	0	0	0	c0600040	RW-
.rdata	1c000	33f0	3400	1a200	0	0	0	0	40600040	R--
.bss	20000	bff4	0	0	0	0	0	0	c0600080	RW-
.idata	2c000	7f4	800	1d600	0	0	0	0	c0300040	RW-
.CRT	2d000	34	200	1de00	0	0	0	0	c0300040	RW-
.tls	2e000	8	200	1e000	0	0	0	0	c0300040	RW-

Figure 2 - PE-header analysis result from pedump

String output (abbreviated for clarity) of *floss* shows several indicators of HTTP protocol usage from the nim language and a potential host based indicator “Desktop\cosmo.jpeg”. There are no strings that indicate capabilities to launch new processes or to inject into running processes.

The last three strings look like they’re obfuscated with repeating characters (w, x and B):

floss

```
[…]
@Can't load inet_ntop proc from Ws2_32.dll
@kernel32
@kernel32
@Ws2_32.dll
@Ws2_32.dll
[…]
@Bcrypt.dll
[…]
@Desktop\cosmo.jpeg
@200 OK
@Authorization
@Host
@httpclient.nim(1144, 15) `false`
@Transfer-Encoding
@Content-Type
@Content-Length
@httpclient.nim(1082, 13) `not url.contains({'\r', '\n'})` url shouldn't contain any newline
characters
@Nim httpclient/1.6.2
@hwtwtwpw:w/w/whwewyw.wywowuwuwpw.wlwowcwawlw
@axuxtxhx.xnxsx.xlxoxcxaxlx
@.BcBoBsBmBoBsBfBuBrBbBoBoBtBsBeBmBpBoBrBiBuBmB.BlBoBcBaBlB
[…]
```

Figure 3 - Abbreviated floss output for the specimen

The specimen only imports three DLLs – none of which hints at any network connectivity. Having identified this to be nim executable, nim’s dynamic DLL loading capabilities need to be considered. The *floss* output also hints a string “@Ws2_32.dll” as a possible candidate.

peframe

```
--------------------------------------------------------------------------------
Import function
--------------------------------------------------------------------------------
KERNEL32.dll      23
msvcrt.dll        49
USER32.dll        1
```

Figure 4 - Imports by the specimen as identified by peframe

The string *“@Bcrypt.dll”* hints to encryption potentially being used by the specimen.

DetectItEasy confirms the nim language compiler assumptions:

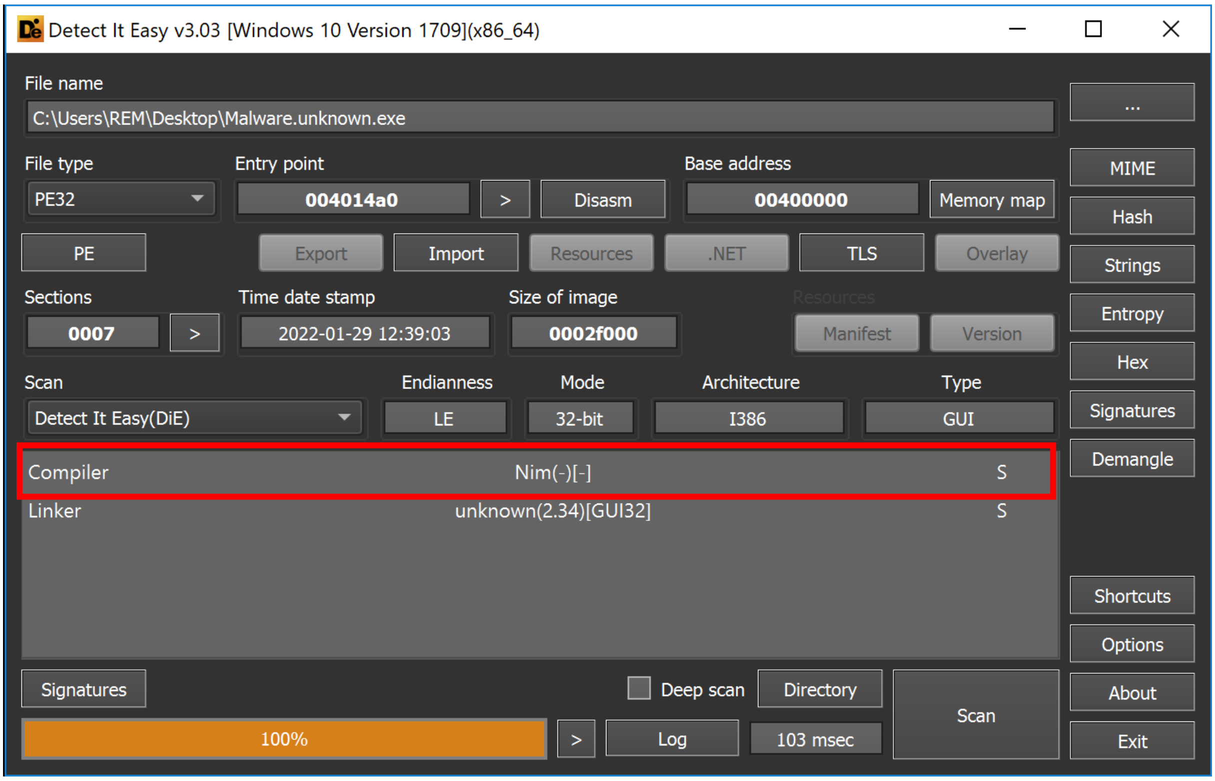


Figure 5 - DetectItEasy analysis result

Simple de-obfuscation of the last three strings from *floss* shows three hostnames:

- http://hey[.]youup[.]local
- Auth[.]ns[.]local
- [.]cosmofurbootsemporium[.]local (note the leading ".")

```
// Obfuscated strings (FLOSS output)
obfuscated_strings = ["hwtwtwpw:w/w/whwewyw.wywowuwuwpw.wlwowcwawlw",
                      "axuxtxhx.xnxsx.xlxoxcxaxlx",
                      ".BcBoBsBmBoBsBfBuBrBbBoBoBtBsBeBmBpBoBrBiBuBmB.BlBoBcBaBlB"]

// Obfuscation characters to remove
obfuscation_characters = ["w", "x", "B"]

// Removing obfuscation characters
[obfuscated_strings[i].replace(obfuscation_characters[i],"") for i in range(len(obfuscated_strings))]

// Clear-text urls
['http://hey.youup.local', 'auth.ns.local', '.cosmosfurbootsemporium.local']
```

Figure 6 - De-obfuscation of strings contained in specimen

Conclusion:
Basic static analysis points to this being an unpacked nim 32bit Windows binary.
Initial working hypothesis is that the binary has no functionality to spawn or inject processes but potentially has networking capabilities (via runtime DLL loading).

The de-obfuscated strings point to HTTP server (http://hey[.]youup[.]local) being involved – the "auth[.]ns[.]local" hostname might be an indicator for DNS traffic to be of interest. This is further substantiated by the usage of the root-domain "[.]cosmofurbootsemporium[.]local", which lacks the subdomain needed to complete the full hostname.

A filename pointing to a jpeg image ("cosmo.jpeg") on the user's desktop has been identified as a possible host-based indicator. At this point, it's unclear, if that jpeg will be uploaded / downloaded or serves any other purpose.

To verify the hypothesis, and obtain more insights into the specimen's behavior, basic dynamic analysis will be performed next.

Dynamic analysis

Running the specimen with administrative rights on a fresh snapshot of *Flare VM* shows the specimen running for about 15 seconds, without spawning any client processes. After 15 seconds it simply quits. The *FlareVM* has no internet access.

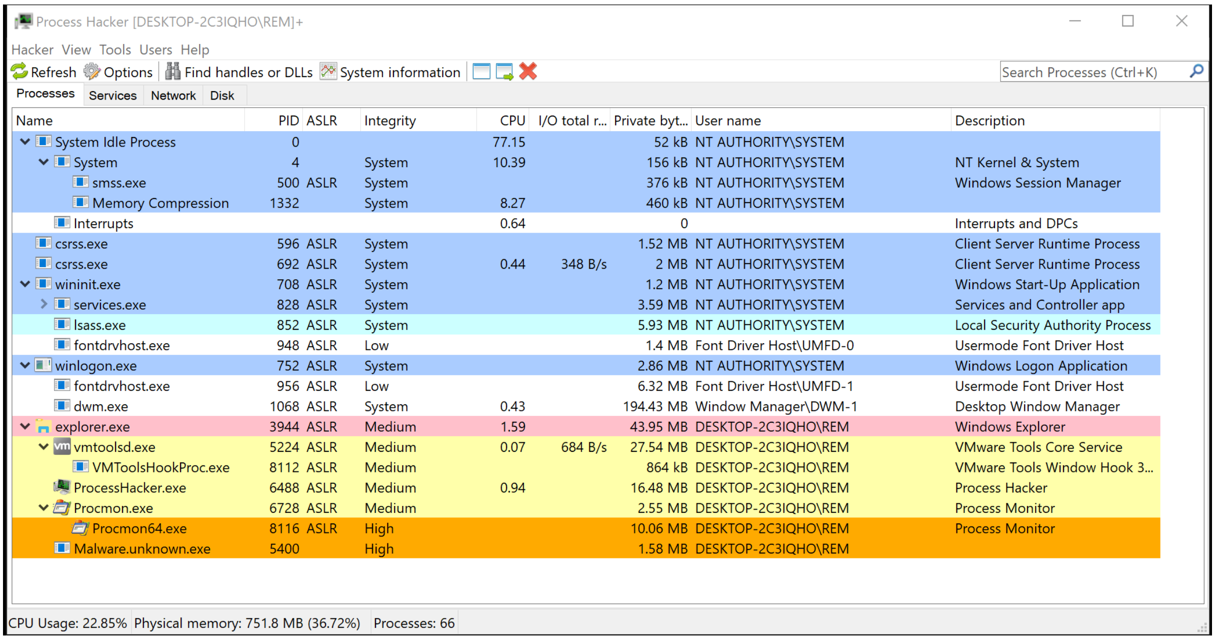


Figure 7 - Process Hacker output when running the specimen

The process-tree view of *Process Monitor* confirms the 15 seconds runtime and the lack of any child-processes.

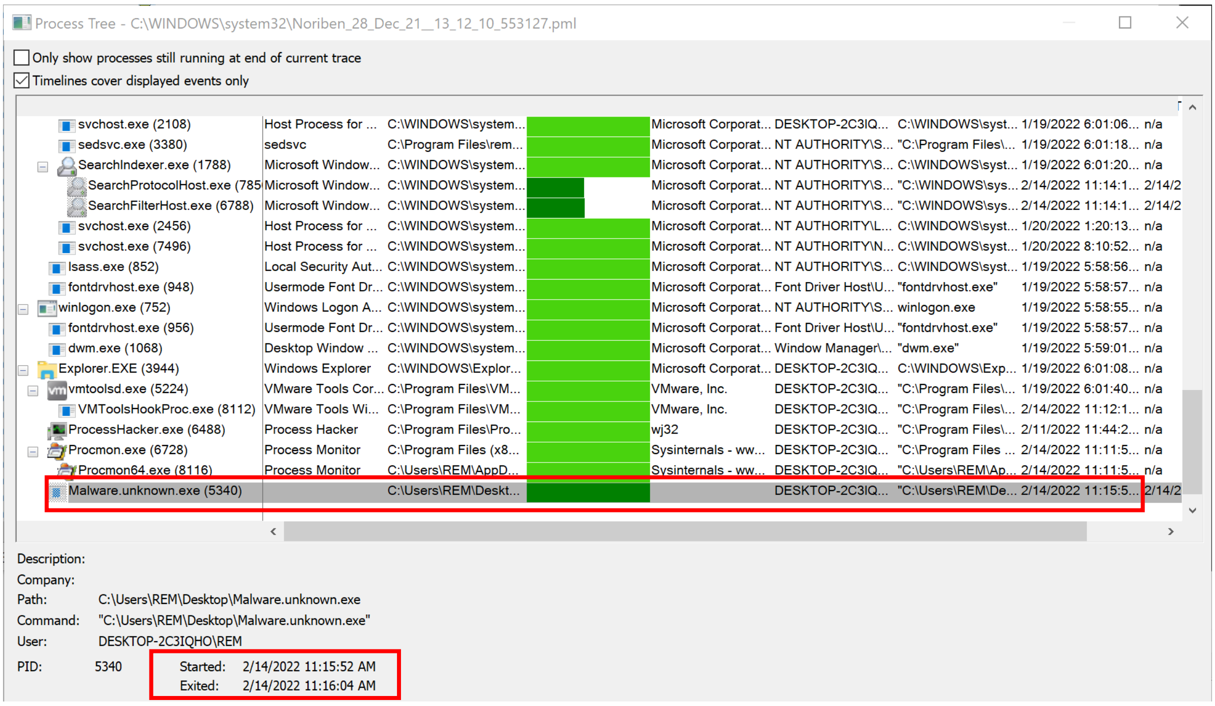


Figure 8 - Process Monitor tree-view output

So far, neither the "Desktop\cosmo.jpeg", nor the two hostnames (http://hey[.]youup[.]local, [.]cosmosfurbootsemporium[.]local) have been accounted for.

Step 1 (FlareVM without network access):

- Add a file named "cosmo.jpeg" to the desktop
- Re-run the specimen with administrative rights (still no internet access)
- Fille the cosmo.jpeg file with a series of "a" characters for easier recognition in network traffic, etc.

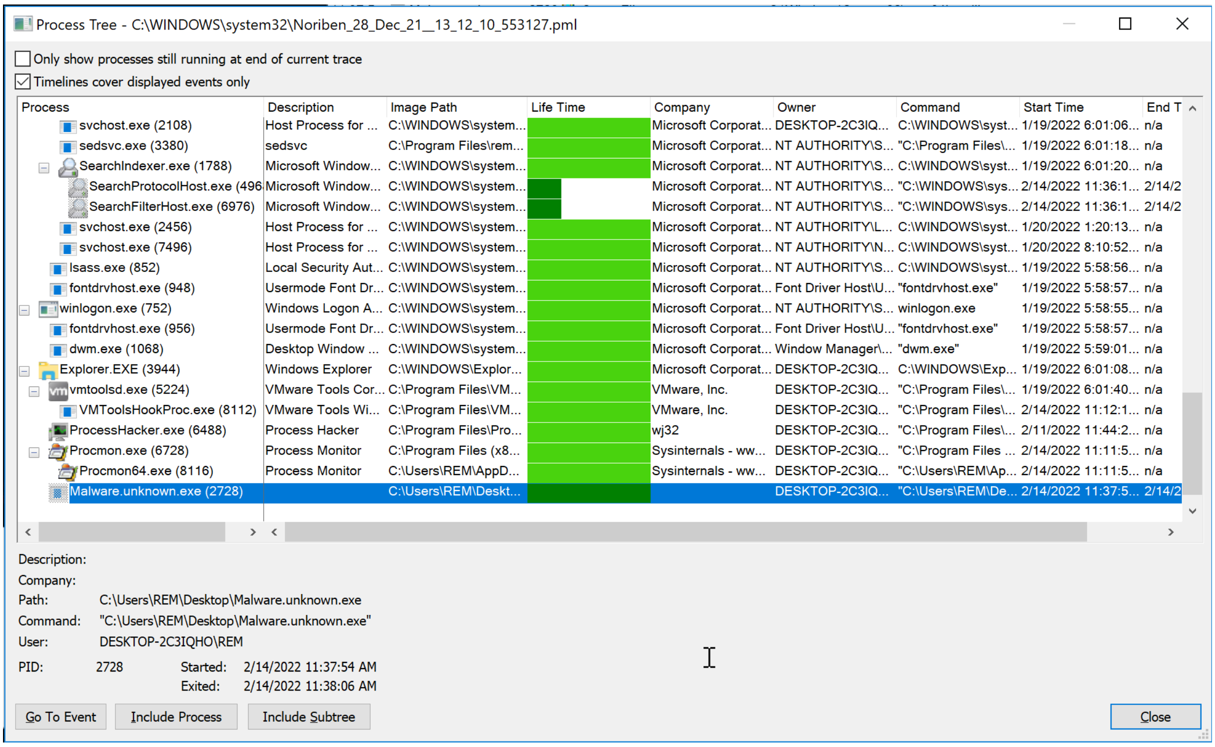


Figure 9 - Process Monitor tree for specimen with "cosmo.jpeg" present

- Runtime reduced to about 12 seconds, then the specimen exits
- No child processes spawned
- Contents of "cosmo.jpeg" unchanged after running the specimen again

Step 2 (FlareVM connected to REMnux VM for simulated internet access – HTTP, DNS):

- Enabling *inetsim* (including DNS) on *REMnux* to simulate network connectivity

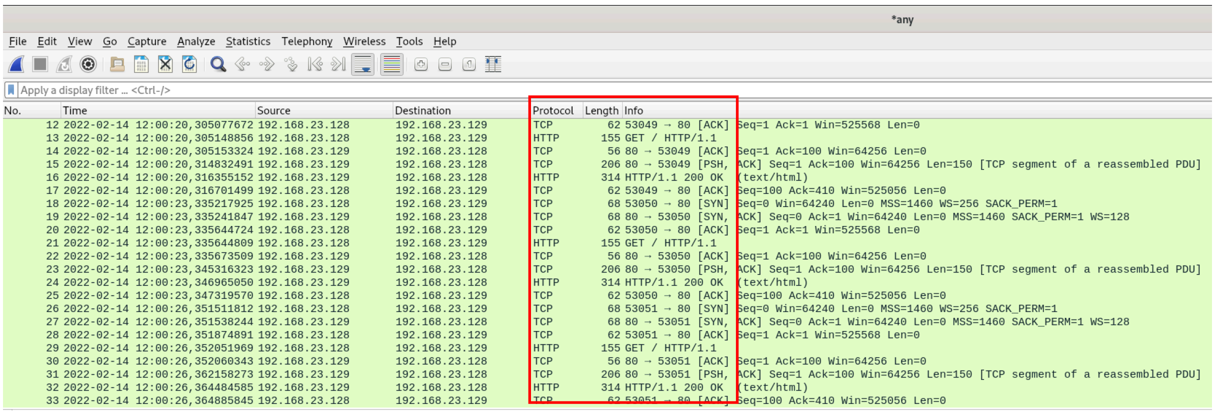
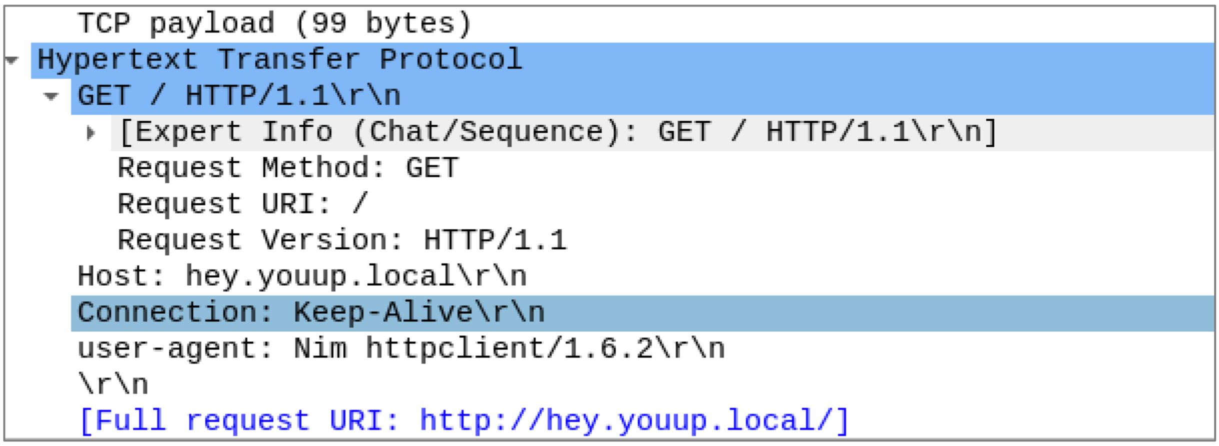


Figure 10 - Specimen beaconing to hey[.]youup[.]local (Wireshark)

- The network traffic consists of the same number of bytes transmitted every three seconds – the request header doesn't contain any data other than a GET request to hey[.]youup[.]local:



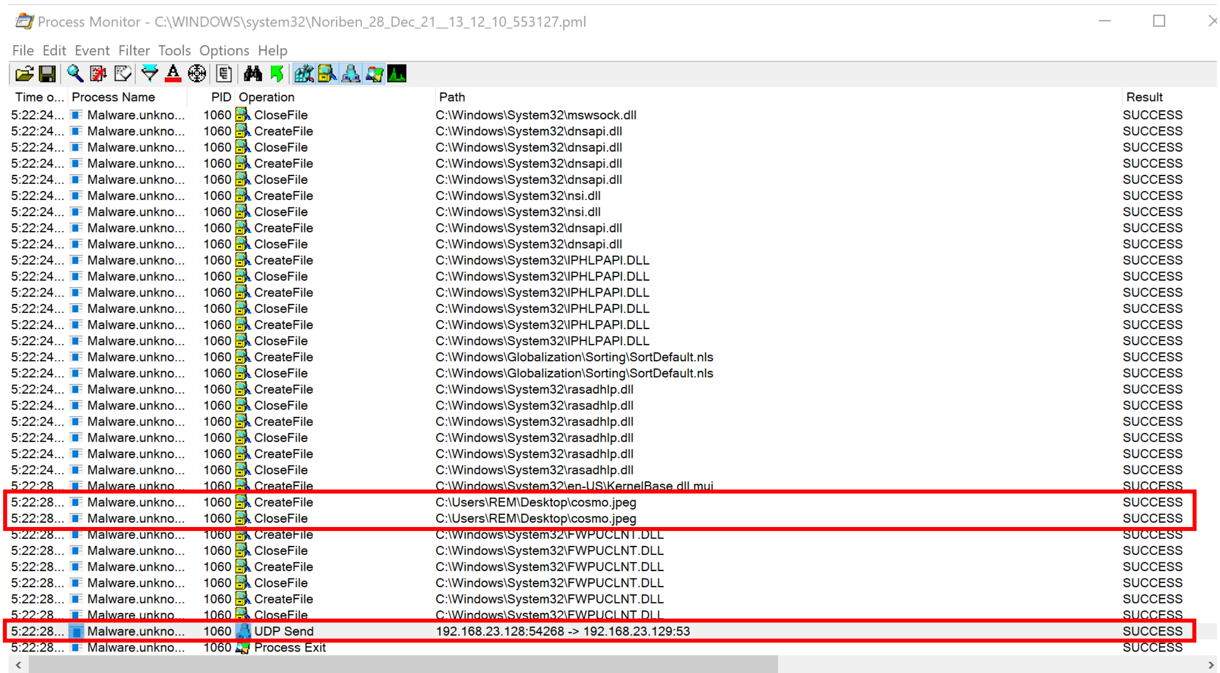
```
    TCP payload (99 bytes)
Hypertext Transfer Protocol
    GET / HTTP/1.1\r\n
      [Expert Info (Chat/Sequence): GET / HTTP/1.1\r\n]
      Request Method: GET
      Request URI: /
      Request Version: HTTP/1.1
    Host: hey.youup.local\r\n
    Connection: Keep-Alive\r\n
    user-agent: Nim httpclient/1.6.2\r\n
    \r\n
    [Full request URI: http://hey.youup.local/]
```

Figure 11 - GET request details for the "beacon" traffic (Wireshark)

- The specimen continues to run until terminated via *ProcessHacker*
- "cosmo.jpeg" remains unchanged while the specimen runs
- No DNS traffic to auth[.]ns[.]local has been observed
- No child processes have been spawned

Step 3 (Remove HTTP connectivity, continue with DNS only):

- Restart *Inetsim* on *REMnux* without HTTP/HTTPS reponses configured
- Run *fakedns* on REMnux to monitor DNS traffic
- Flush DNS resolver cache on *FlareVM*
- The specimen now creates traffic on port 53 – which indicates DNS traffic



Time o...	Process Name	PID	Operation	Path	Result
5:22:24...	Malware.unkno...	1060	CloseFile	C:\Windows\System32\mswsock.dll	SUCCESS
5:22:24...	Malware.unkno...	1060	CreateFile	C:\Windows\System32\dnsapi.dll	SUCCESS
5:22:24...	Malware.unkno...	1060	CloseFile	C:\Windows\System32\dnsapi.dll	SUCCESS
5:22:24...	Malware.unkno...	1060	CreateFile	C:\Windows\System32\dnsapi.dll	SUCCESS
5:22:24...	Malware.unkno...	1060	CloseFile	C:\Windows\System32\dnsapi.dll	SUCCESS
5:22:24...	Malware.unkno...	1060	CreateFile	C:\Windows\System32\nsi.dll	SUCCESS
5:22:24...	Malware.unkno...	1060	CloseFile	C:\Windows\System32\nsi.dll	SUCCESS
5:22:24...	Malware.unkno...	1060	CreateFile	C:\Windows\System32\dnsapi.dll	SUCCESS
5:22:24...	Malware.unkno...	1060	CloseFile	C:\Windows\System32\dnsapi.dll	SUCCESS
5:22:24...	Malware.unkno...	1060	CreateFile	C:\Windows\System32\IPHLPAPI.DLL	SUCCESS
5:22:24...	Malware.unkno...	1060	CloseFile	C:\Windows\System32\IPHLPAPI.DLL	SUCCESS
5:22:24...	Malware.unkno...	1060	CreateFile	C:\Windows\System32\IPHLPAPI.DLL	SUCCESS
5:22:24...	Malware.unkno...	1060	CloseFile	C:\Windows\System32\IPHLPAPI.DLL	SUCCESS
5:22:24...	Malware.unkno...	1060	CreateFile	C:\Windows\System32\IPHLPAPI.DLL	SUCCESS
5:22:24...	Malware.unkno...	1060	CloseFile	C:\Windows\System32\IPHLPAPI.DLL	SUCCESS
5:22:24...	Malware.unkno...	1060	CreateFile	C:\Windows\Globalization\Sorting\SortDefault.nls	SUCCESS
5:22:24...	Malware.unkno...	1060	CloseFile	C:\Windows\Globalization\Sorting\SortDefault.nls	SUCCESS
5:22:24...	Malware.unkno...	1060	CreateFile	C:\Windows\System32\rasadhlp.dll	SUCCESS
5:22:24...	Malware.unkno...	1060	CloseFile	C:\Windows\System32\rasadhlp.dll	SUCCESS
5:22:24...	Malware.unkno...	1060	CreateFile	C:\Windows\System32\rasadhlp.dll	SUCCESS
5:22:24...	Malware.unkno...	1060	CloseFile	C:\Windows\System32\rasadhlp.dll	SUCCESS
5:22:24...	Malware.unkno...	1060	CreateFile	C:\Windows\System32\rasadhlp.dll	SUCCESS
5:22:24...	Malware.unkno...	1060	CloseFile	C:\Windows\System32\rasadhlp.dll	SUCCESS
5:22:28...	Malware.unkno...	1060	CreateFile	C:\Windows\System32\en-US\KernelBase.dll.mui	SUCCESS
5:22:28...	Malware.unkno...	1060	CreateFile	C:\Users\REM\Desktop\cosmo.jpeg	SUCCESS
5:22:28...	Malware.unkno...	1060	CloseFile	C:\Users\REM\Desktop\cosmo.jpeg	SUCCESS
5:22:28...	Malware.unkno...	1060	CreateFile	C:\Windows\System32\FWPUCLNT.DLL	SUCCESS
5:22:28...	Malware.unkno...	1060	CloseFile	C:\Windows\System32\FWPUCLNT.DLL	SUCCESS
5:22:28...	Malware.unkno...	1060	CreateFile	C:\Windows\System32\FWPUCLNT.DLL	SUCCESS
5:22:28...	Malware.unkno...	1060	CloseFile	C:\Windows\System32\FWPUCLNT.DLL	SUCCESS
5:22:28...	Malware.unkno...	1060	CreateFile	C:\Windows\System32\FWPUCLNT.DLL	SUCCESS
5:22:28...	Malware.unkno...	1060	CloseFile	C:\Windows\System32\FWPUCLNT.DLL	SUCCESS
5:22:28...	Malware.unkno...	1060	UDP Send	192.168.23.128:54268 -> 192.168.23.129:53	SUCCESS
5:22:28...	Malware.unkno...	1060	Process Exit		SUCCESS

Figure 12 - ProcessMonitor with DNS traffic and access to cosmo.jpeg

- The specimen exits after a few seconds without spawning any sub-processes
- Observing DNS traffic captured by *fakedns* shows a DNS lookup for a subdomain of “[.]cosmosfurbootsemporium[.]local”:

```
remnux@remnux:~/malware/PMAT$ sudo fakedns
fakedns[INFO]: dom.query. 60 IN A 192.168.23.129
fakedns[INFO]: Response: win1710.ipv6.microsoft.com -> 192.168.23.129
fakedns[INFO]: Response: hey.youup.local -> 192.168.23.129
fakedns[INFO]: Response: YWFhYWFhYWFhYWFhYWFhYWE=.cosmosfurbootsemporium.local -> 192.168.23.129
```

Figure 13 - fakends output for specimen running without access to the beacon-URL

- Wireshark shows the unsuccessful attempt to connect to hey[.]youup[.]local (1)
- Wireshark also shows the DNS traffic to resolve a domain named “YWFhYWFhYWFhYWFhYWFhYWE=[.]cosmosfurbootsemporium[.]local” (2)

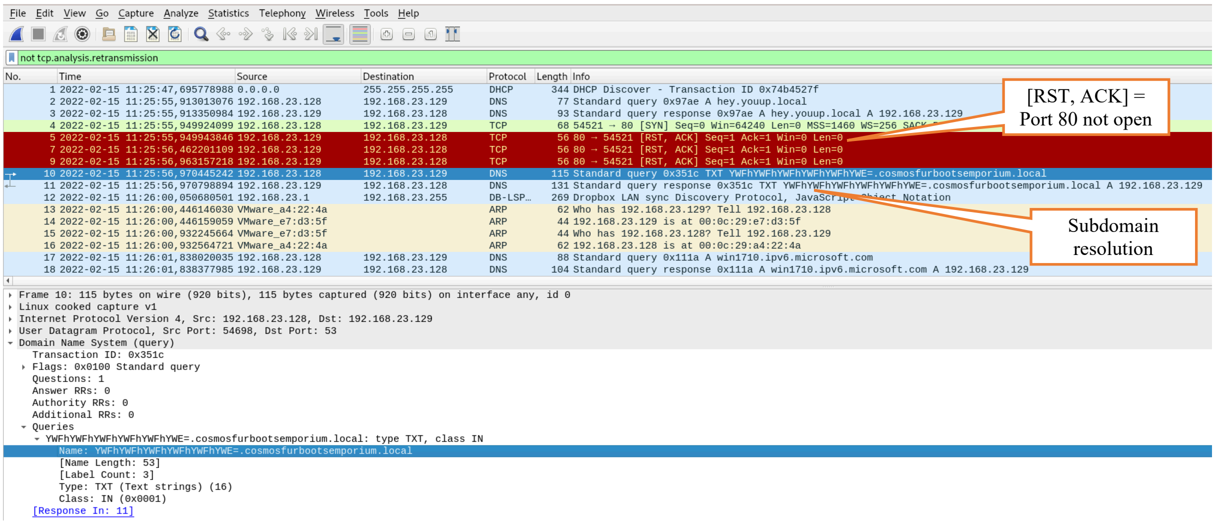


Figure 14 - Wireshark capture of subdomain DNS resolution

- The name of the subdomains is base64 encoded and decodes to “aaaaaaaaaa” – which is the content of our fictitious “*cosmo.jpeg*” file

base64 -d

```
remnux@remnux:~/malware/PMAT$ echo "YWFhYWFhYWFhYWFhYWFhYWE=" | base64 -d
aaaaaaaaaaaaaaaaa
```

Figure 15 – base64 decoded subdomain

- The contents of the “*cosmo.jpeg*” file remain unchanged
- To assess if the base64 encoded “a…a” string actually is the content of the *“cosmo.jpeg”*, an alternative version of *“cosmo.jpeg”* will be used for a re-run of the specimen:

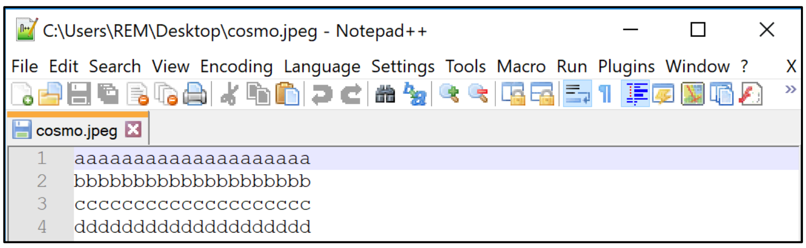


Figure 16 - Modified "cosmo.jpeg" for verification

- Re-running the specimen with the above contents now yields two DNS requests for subdomains of the “[.]cosmofurbootsemporium[.]local”

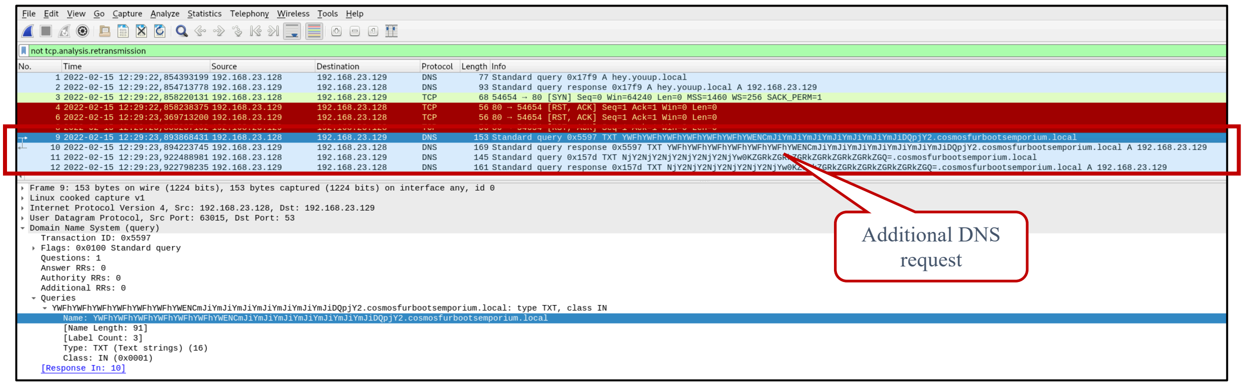


Figure 17 - Wireshark traffic running the specimen on updated "cosmo.jpeg"

- Concatenating the subdomain strings and decoding them reveals the content of the modified *“cosmo.jpeg”*

base64 -d

```
remnux@remnux:~/malware/PMAT$ echo
"YWFhYWFhYWFhYWFhYWFhYWFhYWENCmJiYmJiYmJiYmJiYmJiYmJiYmJiDQpjY2NjY2NjY2NjY2NjY2NjY2NjYw0KZGRkZGRkZGRkZGRkZGRkZGRkZGQ=" | base64 -d
aaaaaaaaaaaaaaaaaaaa
bbbbbbbbbbbbbbbbbbbb
cccccccccccccccccccc
```

Figure 18 - base64 decoded subdomain request strings

Conclusion:

Observed behavior during dynamic analysis:

- The specimen exfiltrates data from a file (with hard-coded filename) via DNS traffic
- Exfiltration happens via resolution of subdomains of the *“[.]cosmofurbootsemporium[.]local”*
- The data is base64 encoded for transmission and broken up into a separate chunk after 62 characters
- No additional processes have been observed in *ProcessMonitor* or *ProcessHacker*
- The specimen checks for a successful TCP connection to http://hey[.]youup[.]local and **if it can connect on port 80, does not exfiltrate data**

YARA Rules

Based on the findings outlined above, the following Yara rule can be used to detect the specimen:

```
rule malware_unknown {
    meta:
        description = "Detects the Malware.Unknown.exe provided as part of the PMAT course"
        md5 = "812a7c7eb9d7a4332b9e166aa09284d7"
        sha1 = "ec0d565afe635c2c7863b2a05df8a49c58b703a3"
        filename = "Malware.unknown.exe.malz"
        author = "Dirk F."

        Block = true
        Log = true
        Quarantine = false

    strings:
        $PE_Magic_Byte = "MZ"

        $malware_user_agent = "httpclient/1.6.2"
        $malware_exfil_file = "Desktop\\cosmo.jpeg"
        $malware_kill_switch_url = "hwtwtwpw:w/w/whwewyw.wywowuwuwpw.wlwowcwawlw"
        $malware_exfil_domain = ".BcBoBsBmBoBsBfBuBrBbBoBoBtBsBeBmBpBoBrBiBuBmB.BlBoBcBaBlB"

    condition:
        $PE_Magic_Byte at 0 and
        all of ($malware*)
}
```

Findings summary

- The specimen is coded in the nim language (32 bit)
- The malware sample performs DNS exfiltration
- The malware exfils a hard-coded picture, named “cosmo.jpeg” from the user’s desktop
- The specimen contacts a webserver at http://hey[.]youup[.]local
- If a connection to the webserver URL is successful, the malware **does not execute**, effectively making this a WannaCry-like kill switch
- If the kill switch URL isn’t reachable, the malware exfiltrates the content of “cosmo.jpeg” in base64 encoded chunks by trying to resolve the base64 encoded text as subdomain of “[.]cosmofurbootsemporium[.]local”
- To recreate the original file content of “cosmo.jpeg”, the DNS server listening on auth[.]ns[.]local would have to concatenate all subdomain-requests for “[.]cosmofurbootsemporium[.]local” and base64-decode the final text into an image file

Observed indicators of compromise

Type of ICO	Indicator	Description
Host-based	Desktop\cosmo.jpeg	File being exfiltrated via DNS
Network-based	http://hey[.]youup[.]local	“Kill-Switch” domain
Network-based	auth[.]ns[.]local	DNS Server for subdomain resolution
Network-based	[.]cosmofurbootsemporium[.]local	Domain-name used for data exfiltration

Figure 19 - Indicators of compromise found during analysis