Practical Malware Analysis & Triage
Malware Analysis Report

SikoMode Malware

Jan 2022 | TimDel | v1.3

Table of Contents

Table of Contents **2**

Executive Summary **3**

High-Level Technical Summary **4**

Malware Composition **6**
- unknown.exe 6
- passwrd.txt 6

Basic Static Analysis **7**
- Hashes 7
- Strings analysis 7
- PeStudio 8

Basic Dynamic Analysis **9**
- TCPView 9
- Procmon 10
- Wireshark 10
- ProcessHacker 11

Advanced Static Analysis **12**
- Main routine functions 12
  - stealStuff() 12
  - houdini() 12

Advanced Dynamic Analysis **18**

Indicators of Compromise **20**
- Network Indicators 20
- Host-based Indicators 20

Rules & Signatures **22**

Appendices **23**
- Yara Rules 23
- Callback URLs 23

Executive Summary

SHA256 hash	3aca2a08cf296f1845d6171958ef0ffd1c8bdfc3e48bdd34a605cb1f7468213e

SikoMode2.0 is a data exfiltration malware written in Nim discovered on Jan 11th 2022. He is able to evade most antiviruses as of today with a score of 4/68 on VirusTotal.com. The payload is a portable executable meant for x64 Windows.
The malware has no persistence system, this means that a reboot will stop its execution and prevent further exfiltration of datas.
It has been named SikoMode by its creator as we discovered in advanced static analysis, it is also the second iteration of a previously known malware, but with a new signature.

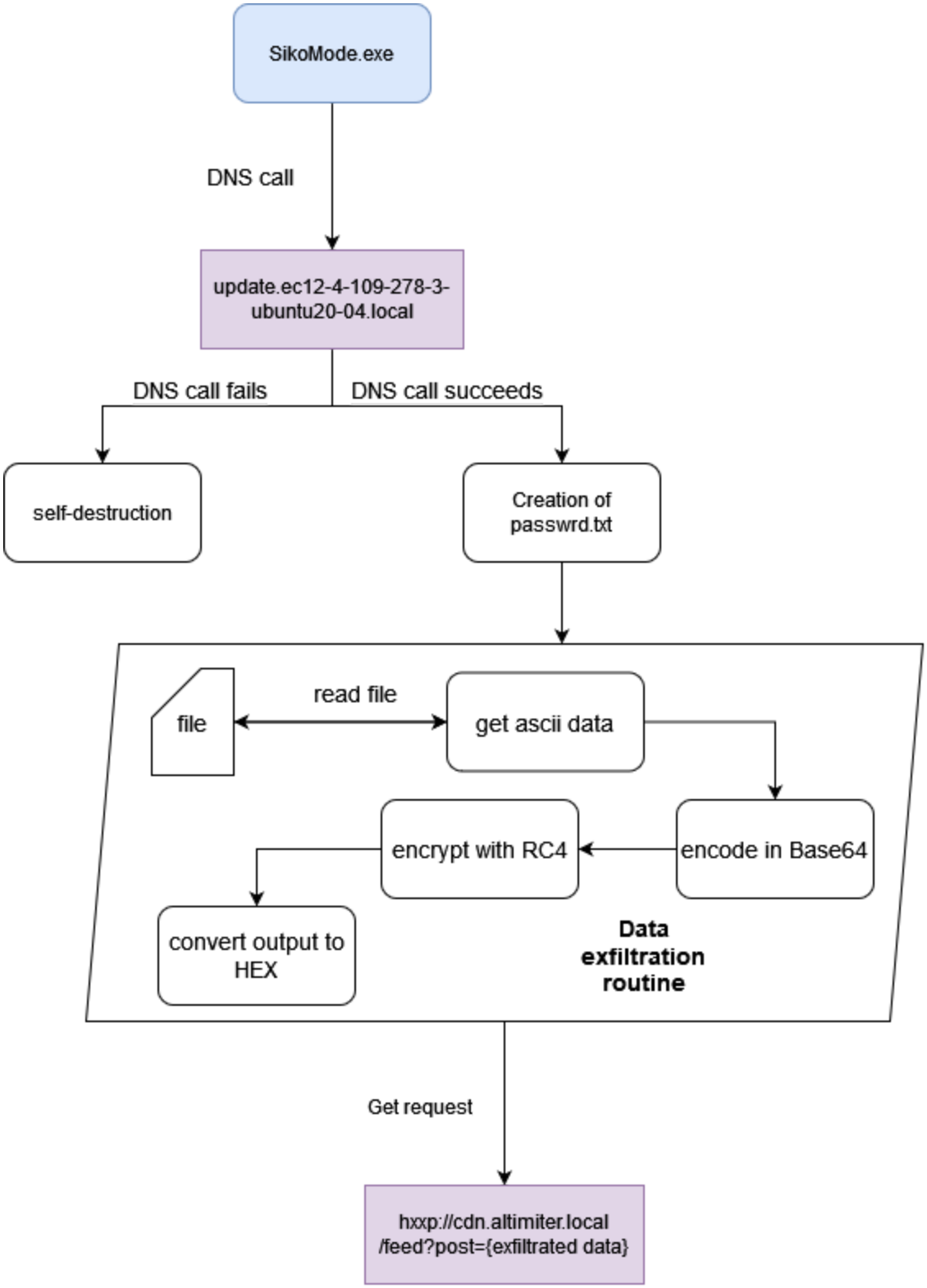
Despite his capacity to evade antivirus, symptoms of infection are blatant :

- creation of the file passwrd.txt located on C:\Users\Public containing the encryption key
- Frequent calls to hxxp://cdn.altimiter.local

YARA signature rules are attached in Appendix A. Malware samples and hashes have been submitted to VirusTotal for further examination.

High-Level Technical Summary

SikoMode2.0 first makes a DNS call to update.ec12-4-109-278-3-ubuntu20-04.local, if the request fails, the program self-destructs. If the DNS call succeeds, the data exfiltration loop will start by creating the file password.txt on C:\Users\Public\ and then use the key in this file to encrypt the data through the RC4 algorithm.
Then the encrypted data will be sent through get requests to hxxp://cdn.altimiter.local/feed?post={data in hex}.
The data on the victim's computer is never altered and none of his data is moved over the wire, only an encrypted copy of the data is sent as requests to a server. The get requests can however be intercepted, or found in logs and parsed to recreate exfiltrated files remotely.
After finishing his extraction routine, the malware self-destructs.


SikoMode.exe
DNS call
update.ec12-4-109-278-3-ubuntu20-04.local
DNS call fails
DNS call succeeds
self-destruction
Creation of passwrd.txt
file
read file
get ascii data
encrypt with RC4
encode in Base64
convert output to HEX
Data exfiltration routine
Get request
hxxp://cdn.altimiter.local/feed?post={exfiltrated data}

Malware Composition

SikoMode2.0 consists of the following component:

File Name	SHA256 Hash
unknown.exe	3aca2a08cf296f1845d6171958ef0ffd1c8bdfc3e48bdd34a605cb1f7468213e
passwrd.txt	1eebfcf7b68b2b4ffe17696800740e199acf207afb5514bc51298c2fe7584410

unknown.exe

The executable that was collected on the victim's machine. It contains the whole malware and only need passwrd.txt and a successful callback to his killswitch DNS to start his exfiltration loop.

passwrd.txt

A plain text file containing the key used to encrypt the data before sending it

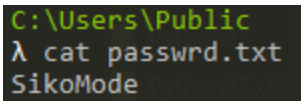


fig 1: content of passwrd.txt.

Basic Static Analysis

Hashes

	unknown.exe
SHA256	3aca2a08cf296f1845d6171958ef0ffd1c8bdfc3e48bdd34a605cb1f7468213e
MD5	b9497ffb7e9c6f49823b95851ec874e3

Strings analysis

In fig 2.1 we can spot the hardcoded url used to exfiltrate data and the use of the nim http client. This figure also proves to us that the malware is written in nim.

```
httpclient.nim
httpError
@unsupported http version
@invalid http version, `
@https
@httpclient.nim(1144, 15) `false`
@httpclient.nim(1082, 13) `not url.contains({'\r', '\n'})` url shouldn't contain any newline characters
@http://cdn.altimiter.local/feed?post=
@Nim httpclient/1.6.2
```

fig 2.1: excerpt from floss showing nim httpclient usage

In fig 2.2 we can see that the location of passwrd.txt is hardcoded in the malware.

```
password
@C:\Users\Public\passwrd.txt
passwrd__sikomode_14
```

fig 2.2: excerpt from floss showing passwrd.txt

In fig 2.3 we have a list of function names that will be useful for advanced analysis as nim is notoriously hard to reverse engineer without knowing first the functions.

```
λ cat floss.txt |grep sikomode
@msikomode.nim.c
ds_rename_handle__sikomode_56
checkKillSwitchURL__sikomode_25
unpackResources__sikomode_17
request__sikomode_208
get__sikomode_201
getContent__sikomode_194
ds_open_handle__sikomode_53
ds_deposite_handle__sikomode_88
houdini__sikomode_51
stealStuff__sikomode_130
sikomodeDatInit000
homeDir__sikomode_13
res__sikomode_263
passwrd__sikomode_14
uAgent__sikomode_15
currDir__sikomode_16
```

fig 2.3: excerpt from floss showing potential function names

PeStudio

looking at PeStudio blacklisted strings found in unknown.exe we can suspect that the malware finds its process id through the WinAPI to then terminate itself and self-destruct.

ascii	19	0x00020FC6	x	-	GetCurrentProcessId
ascii	18	0x00020FDC	x	-	GetCurrentThreadId
ascii	19	0x000210AE	x	-	RtlAddFunctionTable
ascii	22	0x000210D8	x	-	RtlLookupFunctionEntry
ascii	16	0x0002112C	x	-	TerminateProcess
ascii	14	0x00021188	x	-	VirtualProtect
ascii	6	0x00021356	x	-	getenv
ascii	19	0x0006A78D	x	-	GetCurrentProcessId
ascii	16	0x0006A7F9	x	-	TerminateProcess
ascii	22	0x0006A80A	x	-	RtlLookupFunctionEntry
ascii	18	0x0006A821	x	-	GetCurrentThreadId
ascii	14	0x0006A901	x	-	VirtualProtect
ascii	19	0x0006A9E2	x	-	RtlAddFunctionTable
ascii	6	0x0007C170	x	-	getenv
ascii	19	0x00085D95	x	-	GetCurrentProcessId
ascii	16	0x00085E18	x	-	TerminateProcess

fig 3: excerpt from strings on PEStudio

Basic Dynamic Analysis

Killing the program at any point during his process will launch the self-deletion sequence and result in loss of the sample.
However, the malware demonstrated limited self-destruction capabilities, as killing the program, or not responding to his killswitch URL while having the malware open in a disassembler for example, will stop him from self-deleting.

TCPView

Figures 4.1 and 4.2 show that unknown.exe opens TCP sockets on every local port one by one and uses some of them to make get requests through http.

Remote Port	Create Time	Module Name	Sent Packets	Recv Packets	Sent Bytes	Recv Bytes
80	11/01/2022 19:35:21	unknown.exe				
80	11/01/2022 19:35:22	unknown.exe				
80	11/01/2022 19:35:23	unknown.exe	1	2	237	408
80	11/01/2022 19:35:24	unknown.exe				
80	11/01/2022 19:35:25	unknown.exe	1	2	237	408
80	11/01/2022 19:35:26	unknown.exe				
80	11/01/2022 19:35:27	unknown.exe	1	2	237	408

fig 4.1: unknown.exe sends packets through http

State	Local Address	Local Port
Close Wait	66.0.0.4	1131
Close Wait	66.0.0.4	1132
Close Wait	66.0.0.4	1133
Close Wait	66.0.0.4	1134
Close Wait	66.0.0.4	1135
Close Wait	66.0.0.4	1136
Close Wait	66.0.0.4	1137
Close Wait	66.0.0.4	1138
Close Wait	66.0.0.4	1139

fig 4.2: the malware is opening tcp sockets on every local port one by one

Procmon

Unknown.exe creates the file and then reads the content of the file. passwrd.txt and the password comes from the malware and isn't downloaded from the internet.

fig 5.1 unknow.exe is creating and reading passwrd.txt

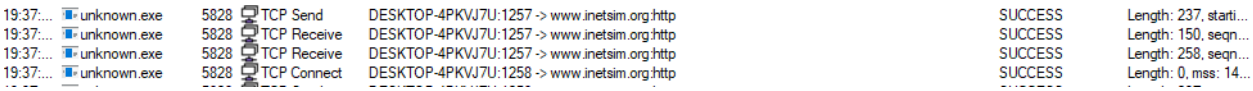


fig 5.2 unknown.exe is opening TCP sockets

Wireshark

From our InetSim machine, we can observe calls to the killswitch domain, failure to resolve will initiate self-destruction. We can also observe the get request and see that the data transferred is HEX.

fig 6.1: DNS request to the killswitch domain

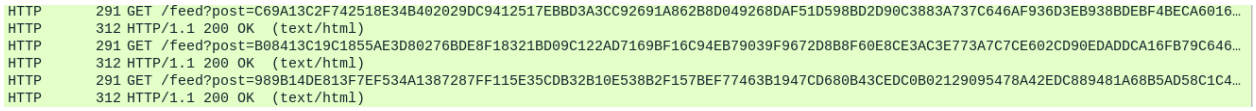
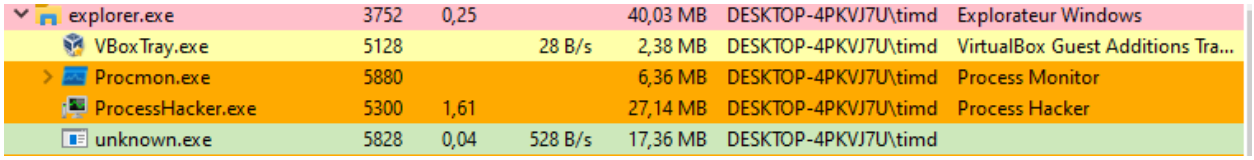


fig 6.2: Get requests containing exfiltrated data

ProcessHacker

unknown.exe has no additional behavior on the system, no subprocess and no other malicious process has been identified.



explorer.exe	3752	0,25		40,03 MB	DESKTOP-4PKVJ7U\timd	Explorateur Windows
VBoxTray.exe	5128		28 B/s	2,38 MB	DESKTOP-4PKVJ7U\timd	VirtualBox Guest Additions Tra...
Procmon.exe	5880			6,36 MB	DESKTOP-4PKVJ7U\timd	Process Monitor
ProcessHacker.exe	5300	1,61		27,14 MB	DESKTOP-4PKVJ7U\timd	Process Hacker
unknown.exe	5828	0,04	528 B/s	17,36 MB	DESKTOP-4PKVJ7U\timd	

fig 7: unknown.exe doesn't spawn subprocesses

Advanced Static Analysis

In fig 8.1 we observe that the function checkKillswitch() is used as a conditional to then call houdini() or launch the main routine (fig 8.2). Even if the condition is checked, the main routine also finishes with a call to the function houdini which self-delete the malware. Disassembling led us to discover more in terms of encryption as we discovered the use of a base64 encoding function in the stealStuff() function.

Main routine functions

checkKillSwitchURL() -> calls the killswitch domain and return a bool

unPackResources() -> creates the file passwrd.txt

stealStuff()

fig 8.3-4-5-6

It is the main function of the malware, it first reads a given file, then converts the ascii stream into base64 and then encrypts in RC4 with the use toRC4() which after encrypting converts the output into HEX.

houdini()

fig 8.6 and 9

This function is used by the malware to self-delete itself. We can compare the calls of houdini() in fig 8.6 with the function self_delete_bin.nim found on offensive nim repository[1]. As we can see this is the exact same sequence of calls which confirm that houdini() is the self-delete function.

[1] https://github.com/byt3bl33d3r/OffensiveNim/blob/master/src/self_delete_bin.nim

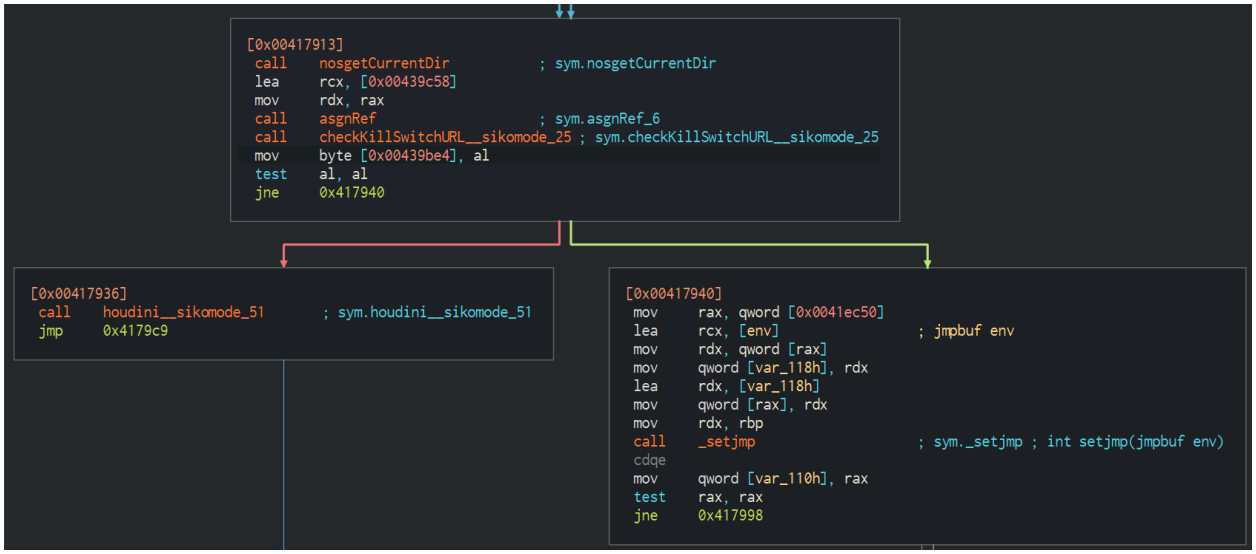


fig 8.1: killswitch URL condition

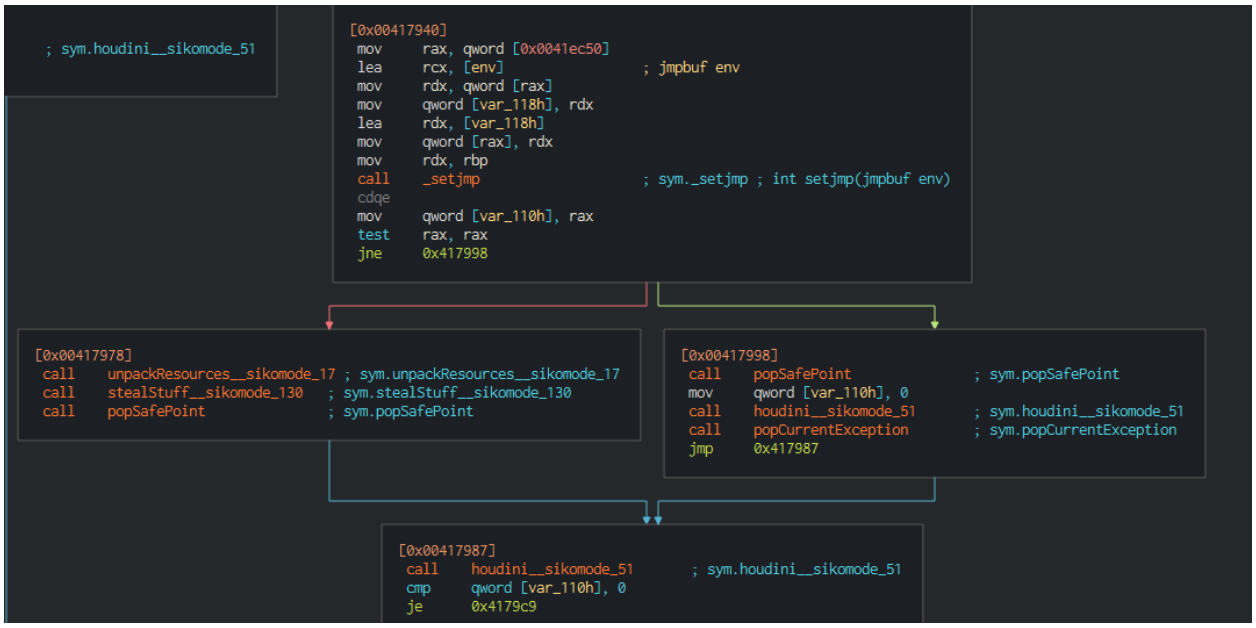


fig 8.2: complete routine

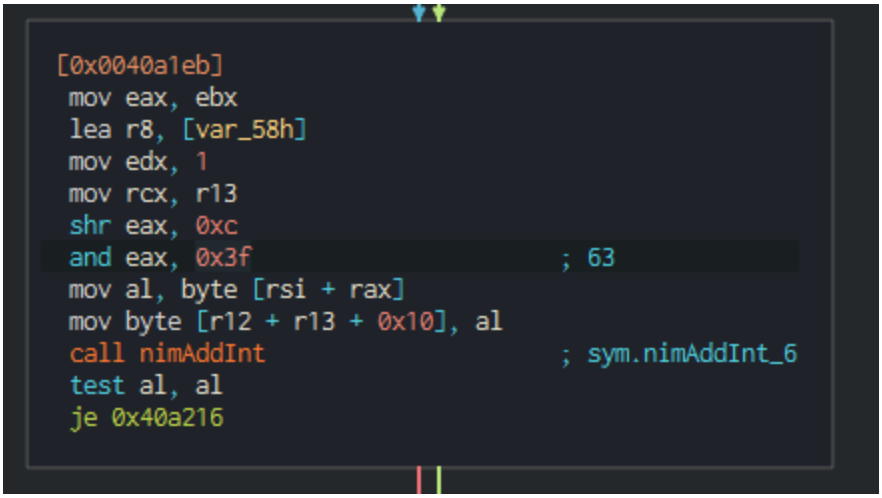


fig 8.3: base 64 encoding assembly code

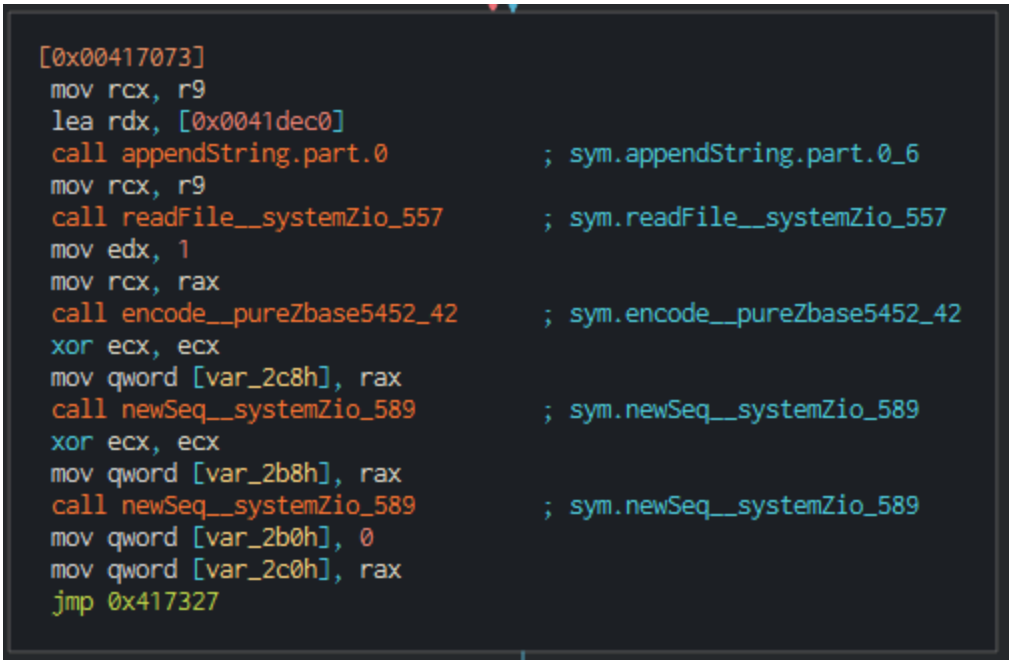


fig 8.4: in stealStuff(), these are the first functions called

fig 8.5: in stealStuff(), encryption to RC4

fig 8.6 in toRC4(), output converted to HEX

fig 8.6: in houdini(), self delete sequence

```
echo "[*] Attempting to rename file name"
if not ds_rename_handle(hCurrent).bool:
    echo "[-] Failed to rename to stream"
    quit(QuitFailure)

echo "[*] Successfully renamed file primary :$DATA ADS to specified stream, closing initial handle"
CloseHandle(hCurrent)

hCurrent = ds_open_handle(addr wcPath[0])
if hCurrent == INVALID_HANDLE_VALUE:
    echo "[-] Failed to reopen current module"
    quit(QuitFailure)

if not ds_deposite_handle(hCurrent).bool:
    echo "[-] Failed to set delete deposition"
    quit(QuitFailure)
```

fig 9: excerpt from self_delete_bin.nim on offensive nim repository

Advanced Dynamic Analysis

As we have already understood the main parts of the malware in advanced static analysis, we'll make sure that our assumptions were true.
Comparing the content of houdini() in Cutter disassembler and x64dbg makes us confident in the fact that this is the right function. As letting the function return will kill the program.

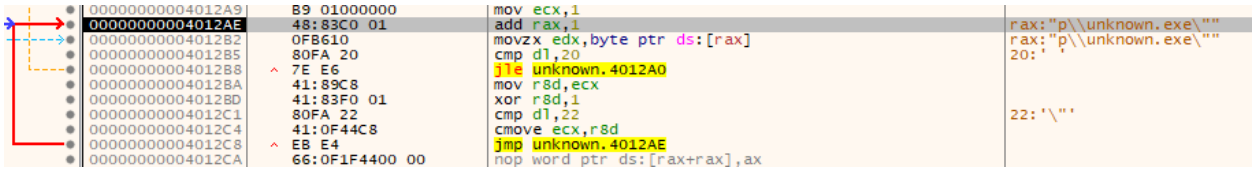


fig 10.1: unknown.4012AE get the filepath of the malware

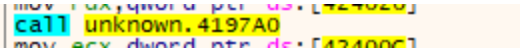


fig 10.1: function suspected to be houdini()

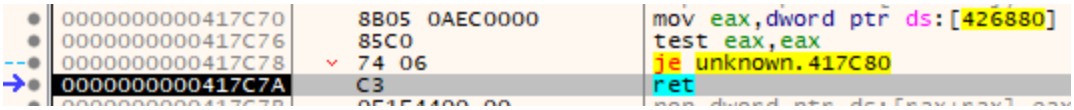


fig 10.2: content of said function

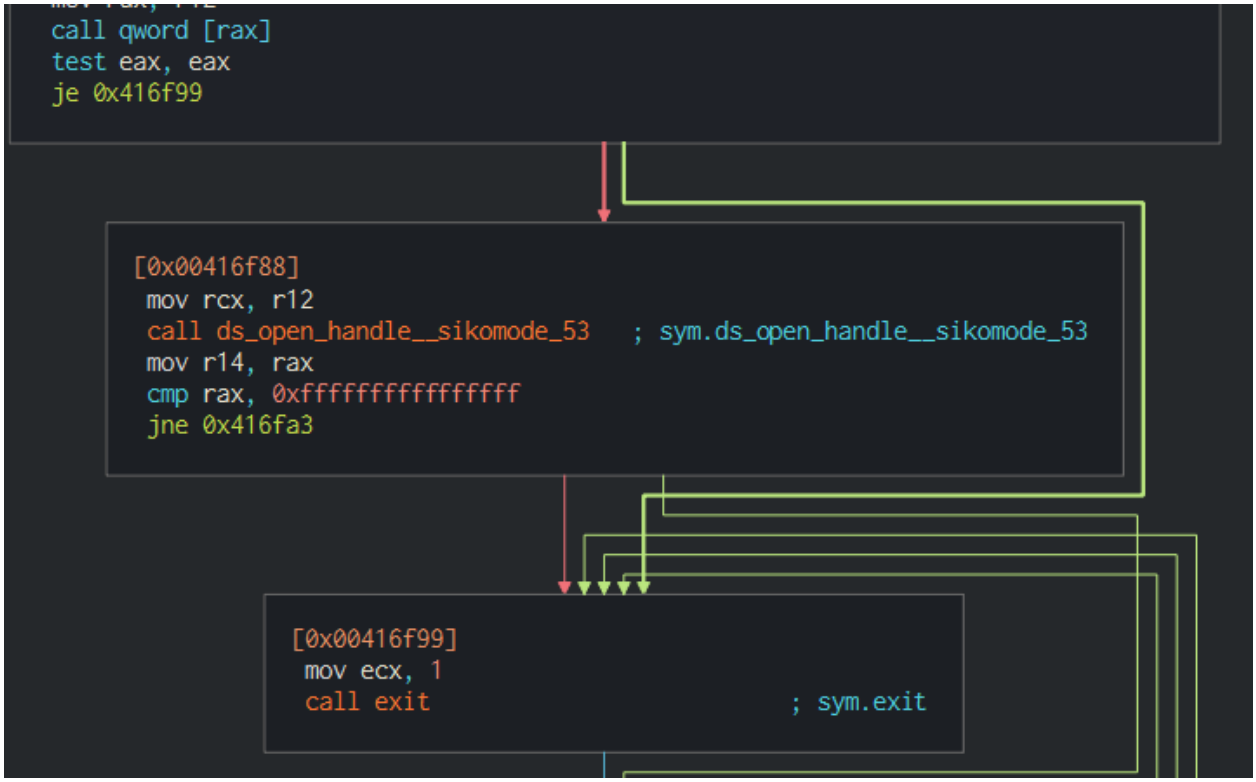


fig 11: excerpt from houdini()

Indicators of Compromise

The full list of IOCs can be found in the Appendices.

Network Indicators

1. DNS query to `update.ec12-4-109-278-3-ubuntu20-04.local`
2. Numerous get request to `hxxp://cdn.altimiter.local/feed?post=`

Fig 12: WireShark Packet Capture of killswitch query

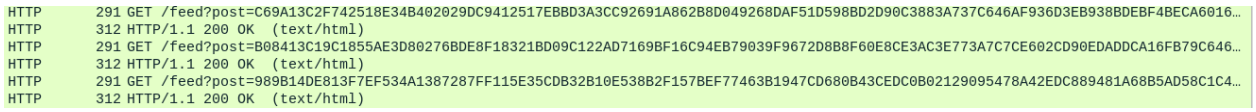


Fig 13: WireShark Packet Capture of data being extracted through http

Host-based Indicators

1. Creation of passwrd.txt in C:\Users\Public

2. opening of TCP sockets on every ports starting from a random local port and incrementing

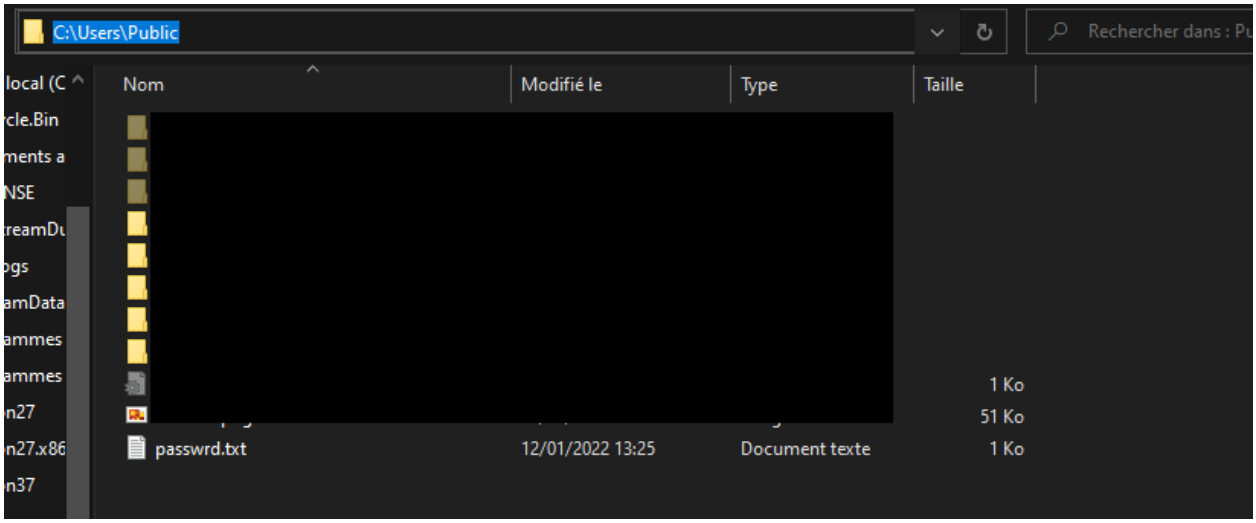


fig 14: password.txt file in C:\Users\Public

Process Name	Process ID	Protocol	State	Local Address	Local Port
unknown.exe	4488	TCP	Close Wait	66.0.0.4	14041
unknown.exe	4488	TCP	Close Wait	66.0.0.4	14042
unknown.exe	4488	TCP	Close Wait	66.0.0.4	14043
unknown.exe	4488	TCP	Close Wait	66.0.0.4	14044
unknown.exe	4488	TCP	Close Wait	66.0.0.4	14049
unknown.exe	4488	TCP	Close Wait	66.0.0.4	14050
unknown.exe	4488	TCP	Close Wait	66.0.0.4	14051
unknown.exe	4488	TCP	Close Wait	66.0.0.4	14052
unknown.exe	4488	TCP	Close Wait	66.0.0.4	14053

fig 15 : TCP sockets captured through TCPVIEW

Rules & Signatures

A full set of YARA rules is included in Appendix A.

As shown in the report, the malware doesn’t show itself to the victim like a ransomware, but the IOCs are easy to spot thanks to the hardcoded strings that reveal the url and the file and path to passwrd.txt.
Even if the malware is very poorly detected by antivirus, its main functions are not encrypted which adds more signatures.

Appendices

A. Yara Rules

Full Yara repository located at: https://github.com/TimDel44/PMAT-report

```
rule Yara_SikoMode{
    meta:
        last_updated = "2022-01-12"
        author = "TimDel"
        description = "Yara rule for SikoMode2.0"

    strings:
        // Fill out identifying strings and other criteria
        $lang = "nim"
        $PE_magic_byte = "MZ"
        $password = "C:\\Users\\Public\\passwrd.txt" ascii
        $url = "http://cdn.altimiter.local/feed?post=" ascii
        $sus_function_houdini= "houdini__sikomode" ascii
        $sus_function_stealstuff ="stealStuff__sikomode" ascii

    condition:
        // Fill out the conditions that must be met to identify the binary
        $PE_magic_byte at 0 and
        ($url and $password) or ($sus_function_houdini and
        $sus_function_stealstuff) and $lang

}
```

B. Callback URLs

Domain	Port
hxxp://cdn.altimiter.local	80
update.ec12-4-109-278-3-ubuntu20-04.local	53