NoaBot Botnet - Sandboxing with ELFEN and Analysis

nikhilh-20.github.io/blog/noabot_botnet/

Metadata

SHA256: `b5e4c78705d602c8423b05d8cd758147fa5bcd2ac9a4fe7eb16a07ab46c82f07`
VT link

Table of Contents

Family Introduction

`NoaBot` is a Mirai-based botnet and possesses most of the original Mirai botnet's capabilities. Its source code contains noticeable differences like the spreader is based in SSH and not Telnet. Akamai detected the `NoaBot` campaign in early 2023.

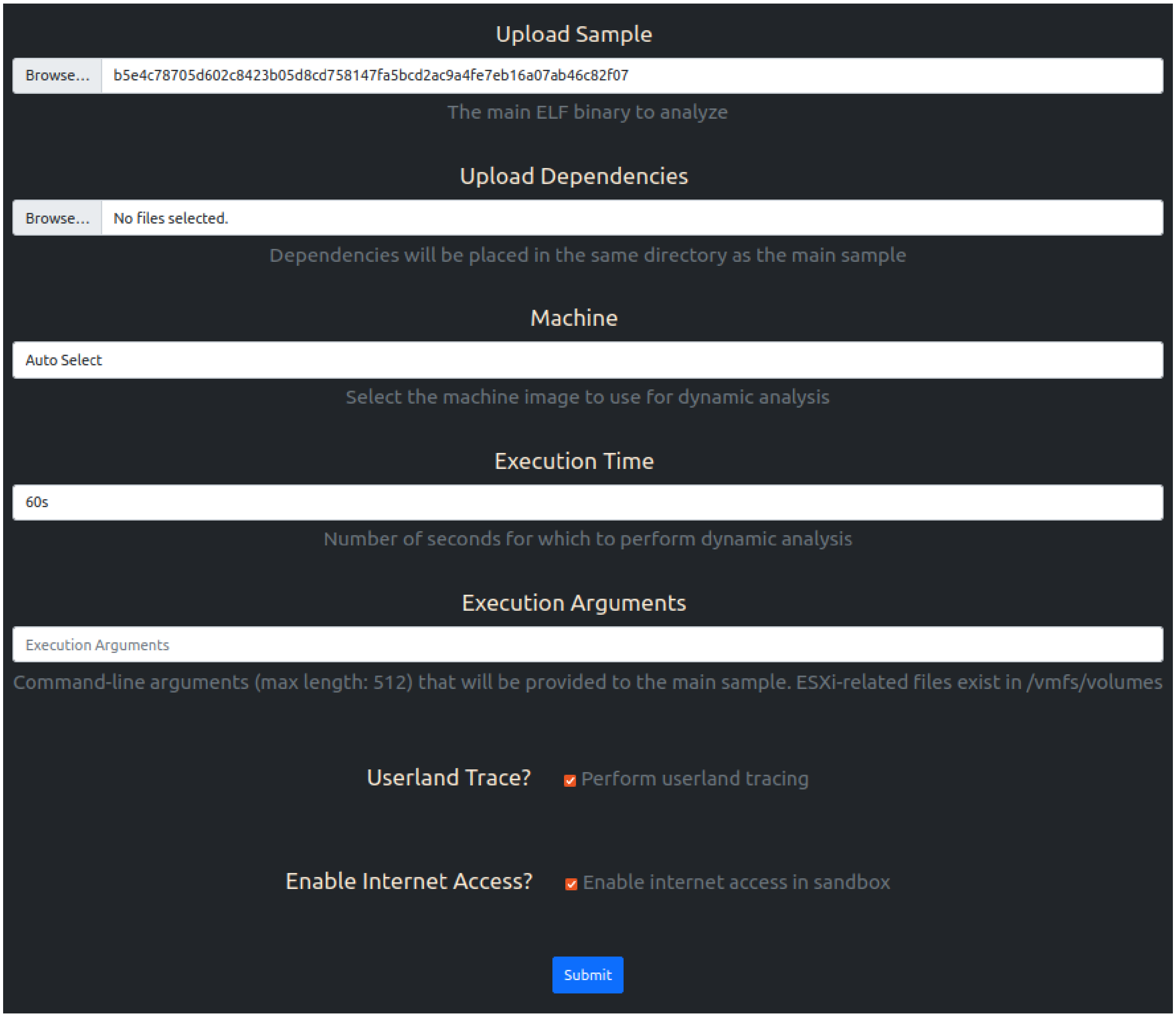
The sample analyzed in this post is an ELF executable targeted towards the MIPS 32-bit, little-endian architecture.

Sandboxing with ELFEN

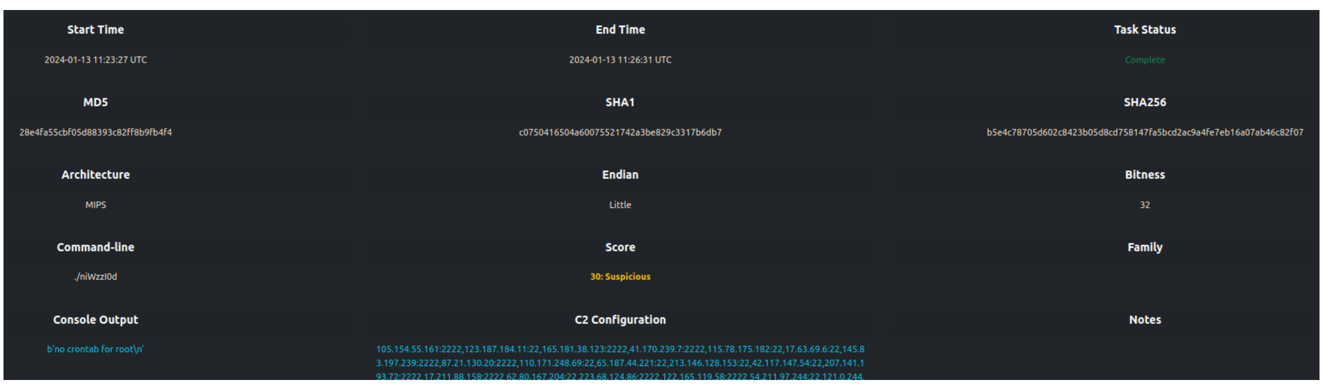
Generally, a malware analyst performs sandboxing early in their workflow. The purpose of sandboxing is to quickly get a general idea of the malware sample's capabilities - does it communicate over the network or encrypt files or establish persistence, etc. This information is useful in determining the next steps in the analysis workflow. I built the ELFEN sandbox to analyze Linux malware (file type: `ELF`) and provide this information. It is open-source and easy to set up.

Detonation

Unless it is known, a sample is usually submitted to a sandbox without any command-line arguments.



The analysis result summary is shown in the snap below:



uClibc Compilation

The sample is compiled with uClibc, and more specifically, with a version between `v0.9.21`-`v0.9.33.2` as evidenced by the string, `npxXoudifFeEgGaACSncs[`. ELFEN detects this open-source library usage.

Yara:open_source_libs:Linux_OpenSource_uClibc_1	10	Detects possible usage of code from uClibc between v0.9.21-v0.9.33.2	T1588.002: Obtain Capabilities: Tool	Nikhil Hegde <ka1do9>

Brute-Forcing Credentials

ELFEN generates process memory dumps during detonation. Besides extracting printable strings from the dumps, ELFEN also applies Yara rules on them. Some in-memory strings in the analysis hint at credentials brute-forcing

danielle
rodney
vutsr&%$#2
admin!@#123
qj o
nsBaseUrl
gretchen
vd{x&%$
gdf{
admin@321

ELFEN detects the presence of well-known password patterns through a Yara rule.

MemYara:generic:Generic_BruteForceCredentials	30	Detects presence of well-known password patterns	T1110.001: Brute Force: Password Guessing	Nikhil Hegde <ka1do9>

Persistence through Cron

The sample establishes persistence through a cron job that runs the sample every time the system reboots. The crontab file per user is located under the directory, `/var/spool/cron/crontabs`. ELFEN detects it as a dropped file and makes it available to the user for downloading. In this case, the sample also sets up command-line parameters when it runs through the cron job.

```
$ cat root
@reboot ./8zpeVaQk "$mimic|fuck" noa
```

ELFEN traces the crontab invocation and detects it:

<table>
<tr><td>14:38:20.962005 UTC</td><td>161</td><td>b'8zpeVaQk'</td><td>execve</td><td colspan="2">exec_path: b''
arg1: b''
arg2: b'(crontab -l; printf \'@reboot ./8zpeVaQk "$mimic|fuck" noa\\n\') | crontab -'</td></tr>
<tr><td>Process:CrontabExecve</td><td>30</td><td colspan="2">Detects usage of crontab</td><td>T1053.003: Scheduled Task/Job: Cron</td><td>Nikhil Hegde <ka1do9></td></tr>
</table>

Accessing Secrets

The sample looks for a variety of secret information such as bash history, SSH private keys and user accounts information. Curiously, the sample does not seem to do anything (read/write) with the found files. *A gap in tracing?* Nevertheless, an analyst can likely make the assumption that the secret information is leveraged in some manner.

ELFEN detects this behavior:

FileOps:BashHistoryAccess	30	Detects access to .bash_history file that contains Bash shell commands history	T1552.003: Unsecured Credentials: Bash History	Nikhil Hegde <ka1do9>
FileOps:SSHPrivateKeysAccess	30	Detects access to SSH private keys	T1552.004: Unsecured Credentials: Private Keys	Nikhil Hegde <ka1do9>
FileOps:UserAccountsInfoAccess	10	Detects access to /etc/passwd file that contains user accounts information	T1003.008: OS Credential Dumping: /etc/passwd and /etc/shadow	Nikhil Hegde <ka1do9>

Accessing Bash History

The sample looks for `.bash_history` files at various locations. This file records a history of the commands that a user has entered in the Bash shell. ELFEN traces this behavior.

16:33:55.654521 UTC	165	b'nginx'	open	file_path: b'/root/.bash_history' flags: 0	4101
16:33:55.656703 UTC	165	b'nginx'	open	file_path: b'/root/.ssh/id_rsa' flags: 0	4101
16:33:55.669735 UTC	165	b'nginx'	open	file_path: b'/root/.bash_history' flags: 0	4101
16:33:55.673843 UTC	165	b'nginx'	open	file_path: b'/root/.ssh/id_ed25519' flags: 0	4101
16:33:55.674727 UTC	165	b'nginx'	open	file_path: b'/root/.bash_history' flags: 0	4101
16:33:55.674915 UTC	165	b'nginx'	open	file_path: b'/root/.ssh/id_dsa' flags: 0	4101
16:33:55.676199 UTC	165	b'nginx'	open	file_path: b'/root/.bash_history' flags: 0	4101
16:33:55.676387 UTC	165	b'nginx'	open	file_path: b'/usr/sbin/.bash_history' flags: 0	-2
16:33:55.676480 UTC	165	b'nginx'	open	file_path: b'/bin/.bash_history' flags: 0	-2
16:33:55.676524 UTC	165	b'nginx'	open	file_path: b'/dev/.bash_history' flags: 0	-2
16:33:55.676547 UTC	165	b'nginx'	open	file_path: b'/bin/.bash_history' flags: 0	-2
16:33:55.676583 UTC	165	b'nginx'	open	file_path: b'/var/spool/mail/.bash_history' flags: 0	-2
16:33:55.677391 UTC	165	b'nginx'	open	file_path: b'/var/www/.bash_history' flags: 0	-2
16:33:55.677547 UTC	165	b'nginx'	open	file_path: b'/var/.bash_history' flags: 0	-2
16:33:55.683532 UTC	158	b'nginx'	fcntl	fd: 3 cmd: 4102 arg: 0	
16:33:55.683547 UTC	158	b'nginx'	fcntl	fd: 4 cmd: 4102 arg: 4294967295	
16:33:55.687050 UTC	165	b'nginx'	open	file_path: b'/home/.bash_history' flags: 0	-2

Accessing SSH Private Keys

The sample looks for user SSH private keys for multiple algorithms: `RSA`, `DSA` and `Ed25519`. These keys are used for authenticating the user over SSH. ELFEN traces this behavior.

16:33:55.653685 UTC	165	b'nginx'	readlink	file_path: b'/proc/168/exe' buffer: b'/root/guild/tHGhQIHC'	20
16:33:55.654072 UTC	165	b'nginx'	open	file_path: b'/etc/passwd' flags: 0	4101
16:33:55.654521 UTC	165	b'nginx'	open	file_path: b'/root/.bash_history' flags: 0	4101
16:33:55.656703 UTC	165	b'nginx'	open	file_path: b'/root/.ssh/id_rsa' flags: 0	4101
16:33:55.669735 UTC	165	b'nginx'	open	file_path: b'/root/.bash_history' flags: 0	4101
16:33:55.673843 UTC	165	b'nginx'	open	file_path: b'/root/.ssh/id_ed25519' flags: 0	4101
16:33:55.674727 UTC	165	b'nginx'	open	file_path: b'/root/.bash_history' flags: 0	4101
16:33:55.674915 UTC	165	b'nginx'	open	file_path: b'/root/.ssh/id_dsa' flags: 0	4101

Accessing User Accounts Information

The sample looks for the `/etc/passwd` file. This contains information about user accounts on the system. Note that benign executables access this file as well during runtime. However, context is important. The sample also accesses other secrets, so access to `/etc/passwd` should not be ignored. ELFEN traces this behavior.

16:33:55.654072 UTC	165	b'nginx'	open	file_path: b'/etc/passwd' flags: 0	4101

Process Name Change

The sample changes its process name to masquerade as a benign process. Specifically, the new process name can be one of many popular utilities such as `mongod`, `nginx`, `smbd`, `sshd`, etc. ELFEN traces and detects this behavior.

18:27:30.741831 UTC	158	b'HVpZ9arv'	prctl	option: 15 arg2: b'smbd' arg3: None arg4: None arg5: None

Process:NameChange	30	Detects process name change through prctl()	T1036: Masquerading	Nikhil Hegde <ka1do9>

Network Communications

Scanning through SSH

The sample scans ports `22` and `2222` (popular alternate port for SSH) for over 4000 IPv4 addresses. ELFEN traces this behavior. The original Mirai botnet spread through Telnet. Researchers at Akamai reported that NoaBot uses SSH.

18:27:38.649658 UTC	168	b'smbd'	connect	fd: 599 family: 0 ip: 140.69.21.212 port: 22	-149
18:27:38.652998 UTC	168	b'smbd'	connect	fd: 600 family: 0 ip: 102.31.149.177 port: 22	-149
18:27:38.654992 UTC	168	b'smbd'	connect	fd: 601 family: 0 ip: 180.216.119.8 port: 22	-149
18:27:38.656706 UTC	168	b'smbd'	connect	fd: 602 family: 0 ip: 158.246.71.32 port: 22	-149
18:27:38.658983 UTC	168	b'smbd'	connect	fd: 603 family: 0 ip: 63.238.0.94 port: 22	-149
18:27:38.659993 UTC	158	b'smbd'	socket	domain: 2 type: 1 protocol: 0	4102
18:27:38.664741 UTC	158	b'smbd'	connect	fd: 4102 family: 0 ip: 8.8.8.8 port: 53	0
18:27:38.671217 UTC	168	b'smbd'	connect	fd: 604 family: 0 ip: 182.164.119.71 port: 22	-149
18:27:38.672426 UTC	168	b'smbd'	connect	fd: 605 family: 0 ip: 148.102.115.241 port: 22	-149
18:27:38.675391 UTC	168	b'smbd'	connect	fd: 606 family: 0 ip: 34.141.159.238 port: 22	-149

ELFEN also captures network traffic into a PCAP and makes it available to the user for downloading. If the remote port is accepting connections, the sample sends a malformed SSH packet early in the SSH handshake. It contains the string, `hi`.

679	0.463945	10.0.2.15	149.162.20.22	TCP	50816 → 22 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=311117903 TSecr=0 WS=
1152	0.749972	149.162.20.22	10.0.2.15	TCP	22 → 50816 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460
1153	0.750105	10.0.2.15	149.162.20.22	TCP	50816 → 22 [ACK] Seq=1 Ack=1 Win=64240 Len=0
1549	1.028154	149.162.20.22	10.0.2.15	SSH	Server: Protocol (SSH-2.0-OpenSSH_5.3)
1550	1.028244	10.0.2.15	149.162.20.22	TCP	50816 → 22 [ACK] Seq=1 Ack=22 Win=64219 Len=0
11563	6.682330	10.0.2.15	149.162.20.22	SSH	Client: Encrypted packet (len=3)
11564	6.682940	149.162.20.22	10.0.2.15	TCP	22 → 50816 [ACK] Seq=22 Ack=4 Win=65535 Len=0
16592	13.449190	10.0.2.15	149.162.20.22	TCP	50816 → 22 [FIN, ACK] Seq=4 Ack=22 Win=64219 Len=0
16593	13.449718	149.162.20.22	10.0.2.15	TCP	22 → 50816 [ACK] Seq=22 Ack=5 Win=65535 Len=0

```
Frame 11563: 57 bytes on wire (456 bits), 57 bytes captured (456 bits)
Ethernet II, Src: RealtekU_12:34:56 (52:54:00:12:34:56), Dst: 52:55:0a:00:02:02 (
Internet Protocol Version 4, Src: 10.0.2.15, Dst: 149.162.20.22
Transmission Control Protocol, Src Port: 50816, Dst Port: 22, Seq: 1, Ack: 22, Le
SSH Protocol
[Malformed Packet: SSH]
  [Expert Info (Error/Malformed): Malformed Packet (Exception occurred)]
    [Malformed Packet (Exception occurred)]
    [Severity level: Error]
    [Group: Malformed]

0000  52 55 0a 00 02 02 52 54  00 12 34 56 08 00 45 00   RU····RT ··4V··E·
0010  00 2b dd ff 40 00 40 06  a7 06 0a 00 02 0f 95 a2   ·+··@·@· ········
0020  14 16 c6 80 00 16 a8 9f  8b 92 00 1a 5e 17 50 18   ········ ····^·P·
0030  fa db 3d c3 00 00 68 69  00                        ··=···hi ·
```

I observed that the sample does not send its SSH identification string first, as is usual in a normal SSH handshake. Instead, it waits for the server to send its identification string. It then replies with the malformed SSH packet.

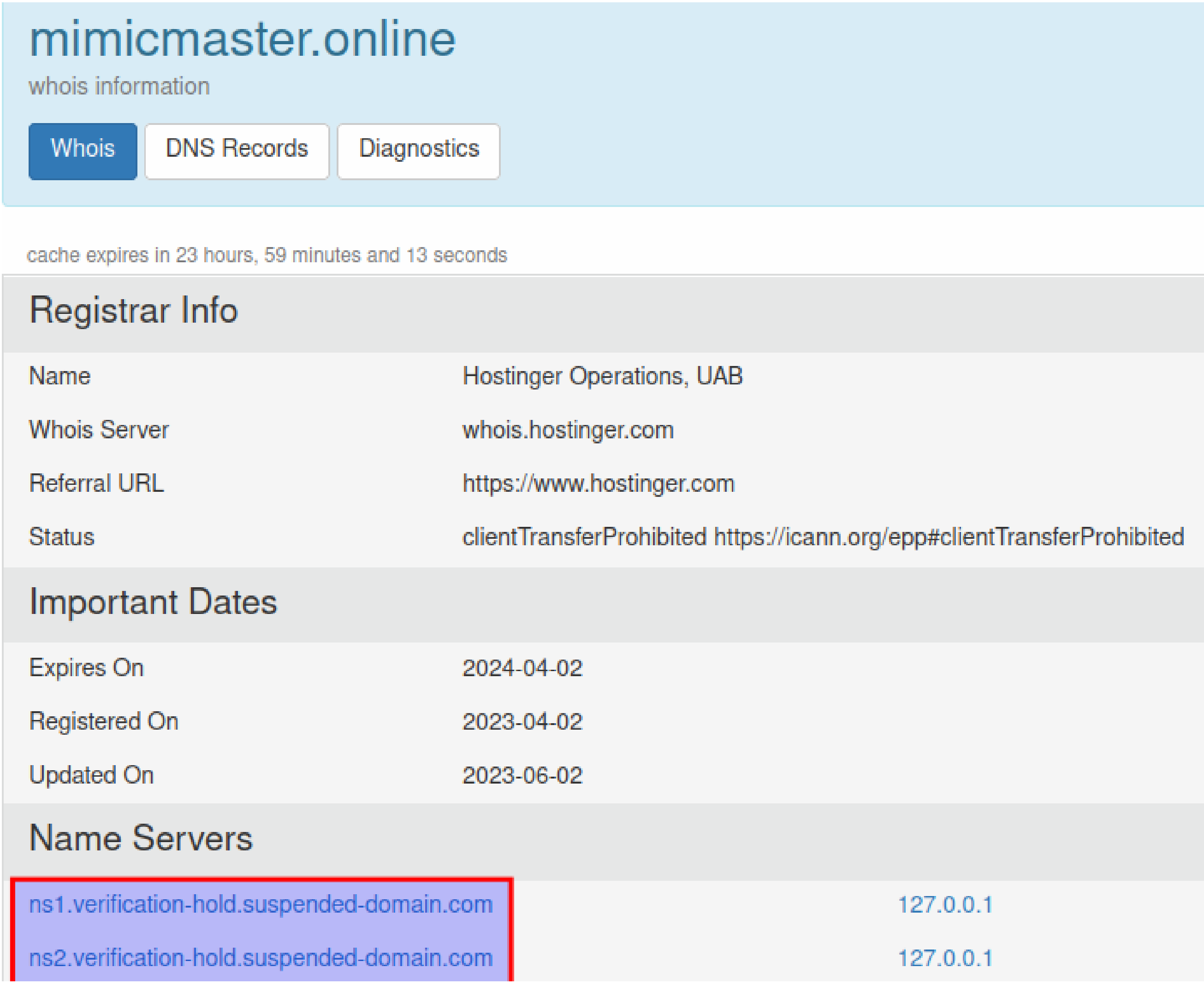
My hypothesis is that the sample is trying to capture the server SSH identification string. Perhaps, to check if it's vulnerable to a known exploit. It then sends the malformed SSH packet (the specific string, `hi` is irrelevant) to possibly avoid triggering any timeouts or `RST` packets from the server which may draw suspicion on server-side defenses. As seen in the snap above, the connection gracefully terminates with a `FIN-ACK-ACK` packet sequence.

C2 Domain

The sample reaches out to its C2, `mimicmaster[.]online`, which is currently unavailable.

1	0.000000	10.0.2.15	8.8.8.8	DNS	78	Standard query 0xf2b0 A mimicmaster.online
55	0.055191	8.8.8.8	10.0.2.15	DNS	78	Standard query response 0xf2b0 Refused A mimicmaster.online

From its Whois records, it can be seen that the domain is currently suspended.



The last known IPv4 address for the domain was `185[.]193.126.118` as seen on VT.

Last DNS records

Record type	TTL	Value
A	14400	185.193.126.118
+ CAA	14400	letsencrypt.org
+ CAA	14400	comodoca.com
+ CAA	14400	letsencrypt.org
+ CAA	14400	digicert.com
+ CAA	14400	globalsign.com
+ CAA	14400	digicert.com
+ CAA	14400	comodoca.com
+ CAA	14400	globalsign.com
NS	21600	ns1.dns-parking.com
NS	21600	ns2.dns-parking.com
+ SOA	3600	ns1.dns-parking.com

ELFEN performs protocol analysis on the captured network traffic. At this point, only DNS protocol analysis is supported.

Timestamp	Query domain	Query Type	Query Class	Response Type	Response Class	Response TTL (in seconds)	Response Data
08:54:47.637825	mimicmaster.online	A	IN	None	None	None	None

Summary

The NoaBot is yet another Mirai-based botnet, except it has notable differences in its capabilities like the SSH spreader. The main goal of this analysis was to demonstrate the usage of the ELFEN sandbox to quickly get insights into a given malware sample.

ELFEN supports features such as:

- Analysis and detection of Linux malware targeting x86-64, ARMv5, MIPS and PowerPC architectures.
- Tracing files, processes, network-related syscalls and `libc` string-related functions.
- PCAP capture and protocol analysis.
- Memory dumps and capturing dropped files
- and more!

If you've not already, give ELFEN a try!

References