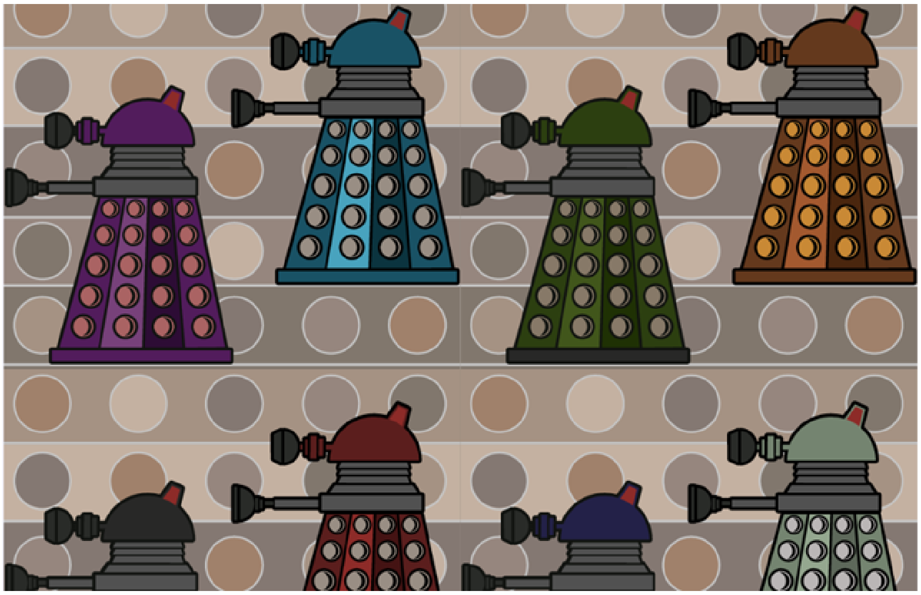
New Linux/Rakos threat: devices and servers under SSH scan (again)

welivesecurity.com/2016/12/20/new-linuxrakos-threat-devices-servers-ssh-scan/

December 20, 2016



ESET's Peter Kálnai and Michal Malik report on a new Linux/Rakos threat – devices and servers are under SSH scan again.

Apparently, frustrated users complain more often recently on various forums about their embedded devices being overloaded with computing and network tasks. What these particular posts have in common is the name of the process causing the problem. It is executed from a temporary directory and disguised as a part of the Java framework, namely ".javaxxx". Additional names like ".swap" or "kworker" are also used. A few weeks ago, we discussed the recent Mirai incidents and Mirai-connected IoT security problems in The Hive Mind: When IoT devices go rogue and all that was written then still holds true.

Attack vector

The attack is performed via brute force attempts at SSH logins, in a similar way to that in which many Linux worms operate, including Linux/Moose (which spread by attacking Telnet logins) – also referenced here – as analyzed by ESET since last year. The targets include both embedded devices and servers with an open SSH port and where a very weak password has been set. The obvious aim of this trojan is to assemble a list of unsecured devices and to have an opportunity to create a botnet consisting of as many zombies as possible. The scan starts with not too extensive list of IPs and spreads incrementally to more targets. Only machines that represent low-hanging fruit from the security perspective are compromised. Note that victims reported cases when they had had a strong password but they forgot their device that had online service enabled and it was reverted to a default password after a factory reset. Just a couple of hours of online exposure was enough for such a reset machine to end up compromised!

Analysis

The malware is written in the Go language and the binary is usually compressed with the standard UPX tool. The awkward thing was that the function names were stripped from the binary in the usual way, but they are still present in a special section anyway. With the help of a script by RedNaga Security that maps symbols back to their respective function in the IDA Pro disassembling software, the whole analysis was simplified to reviewing the features that function names suggested, like main_loadConfig, main_startLocalHttp, main_Skaro_Upgrade, main_IPTarget_checkSSH etc. There are strings like "Skaro" and "dalek" in the binary. The author(s) possibly had in mind a connection to a fictional planet in the science fiction television series Doctor Who from whence the Daleks originated.

As a first step, [Linux/]Rakos loads its configuration via standard input (stdin) in YAML format. The configuration file contains information like lists of C&Cs, all the credentials that are tried against its targets, and internal parameters:

```
—
version: 30
logging: no
generation: 0

skaro:
  ips:
    – “193.0.178.151”
    – “46.8.44.55”
    – “5.34.183.231”
    – “185.82.216.125”
    – “195.123.210.100”
  ping: 60

checkers:
– “http://193.0.178.151”
  – “http://46.8.44.55”
  – “http://5.34.183.231”
  – “http://185.82.216.125”
  – “http://195.123.210.100”
  – “http://httpbin.org/ip”

userpass: [
“root:qwerty”,
“user:admin”,
“ubnt:ubnt”,
“root:12345”,
“guest:1234”,
“root:1111”,
“test:test”,
“support:password”,
“admin:1”,
“test:test123”,
“manager:manager”,
“fax:fax”,
“service:service”,
“root:letmein”,
“sales:sales”,
“guest:guest”,
“shell:sh”,
“enable:system”,
“user:password”,
“backup:backup”,
“ftpuser:ftpuser”,
“admin:password123”,
“monitor:monitor”,
“bin:bin”,
“root:ikwb”,
“admin:manager”,
“oracle:oracle”,
“test:12345”,
“bob:bob”,
“user1:1234”,
“root:1234”,
“root:system”,
PlcmSpIp:PlcmSpIp”,
“user1:123456”,
“1:1”,
“root:1”,
“support:123456”,
“nagios:nagios”,
“demo:demo”,
“admin:1111”,
“PlcmSpIp:PlcmSpIp”,
“pos:pos”,
“support:12345”,
“root:baseball”,
“guest:12345”,
“admin:1234”,
“apache:apache”,
“root:123456”,
“adam:adam”,
“root:alpine”,
“tester:retset”,
“root:raspberry”,
“pi:raspberry”,
“administrator:1234”,
“admin:abc123”,
“admin:qwerty”,
“root:openelec”,
“admin:admin1234”,
“shipping:shipping”,
“ftpuser:asteriskftp”,
“operator:operator”,
]
```

The full plain text Linux/Rakos configuration is available on ESET’s Github: https://github.com/eset/malware-ioc/tree/master/rakos.

Following this, it starts a **local** HTTP service available at http://127.0.0.1:61314. There are two reasons why this is installed: the first is as a cunning method for the future versions of the bot to kill the running instances regardless of their name by requesting http://127.0.0.1:61314/et; second, it tries to parse a URL query for parameters “ip”, “u”, “p” by requesting http://127.0.0.1:61314/ex. The purpose of this /ex HTTP resource is still unclear at the time of writing and it seems not to be referenced elsewhere in the code.

The bot also creates a web server listening on all interfaces. In the early versions, it was listening on TCP port 13666, but now the port is picked randomly from 20,000 to 60,000. Sending a remote request to the device on this port returns the response …

```
{“origin”:”192.168.18.1″}
```

… where the IP address corresponds to the client side. This output is in the same format as the public test server http://httpbin.org with /ip request. On the side running Linux/Rakos, one might see the following logged to stdout:

```
“{2016/11/21 09:02:03 INFO StartChecker.func1 ▶ 001 check: 192.168.18.1}”
```

Next, it sends an initial HTTP request containing important information about the victim device to https://{C&C}/ping. The data sent may appear as follows (some fields have been edited):

```
{
“arch”: “amd64”,
“config”: 30,
“fork”: 0,
“generation”: 0,
“ip”: “192.168.18.1”,
“origin”: “unknown”,
“password”: “shipping”,
“services:”: {“http”:{“addr”:”192.168.18.1:80″,”available”:false,”running”:false},
“dns”:{“addr”:””,”available”:false,”running”:false},
“checker”:{“addr”:”192.168.18.1:22418″,”available”:false,”running”:true}},
“stats”: {
    “cnt”: “load: 0 scan: 0 bless: 0 sm:0 ins: 0 mem: 2692k”,
    “cpu”: “1 x Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz 3591Mhz”,
    “facts”: “host: ubuntu pid: 10219 uid: 0 args: [/tmp/.javaxxx]”,
    “load”: “1.14 0.45 0.17”,
    “mem”: “592MB / 1983MB free (35.21% used)”},“uptime”: 514,
“username”: “shipping”,
“uuid”: “ab-97-b1-d5-2d-8f”,
“version”: 706
}
```

The main feature of this bot is its scanning of the SSH service on various IP addresses, which are obtained from the C&C server by asking for the list located at https://{C&C}/scan. This list seems to be modified frequently. The previous versions of the trojan also scanned for the SMTP service, but the attackers have disabled this feature in current build. We speculate that this feature might be under further development together with additional network scanning features.

The main attack is performed as follows: if one of the username:password pairs from the configuration file results in a successful login to one of the target devices connection to target is successful, two commands are run on that newly-accessed victim (id, uname -m), and other checks are performed and their results reported. Finally the binary checks whether if it is possible to upload to the new victim and does so if the answer is affirmative. We simulated an attack locally with two targets picked, 127.0.0.1 and 127.0.0.100 (originally, the attackers try 200 simultaneous targets every 10 seconds). Suppose the bot fails to connect to the first one which it then marks as FORGET, while the latter one is successful with the INSTALL notice (a SSH connection was established with the correct shipping:shipping login credentials; also note that the uploaded executable is deleted immediately after execution):

1_skaro_logs

Moreover, the backdoor is capable of:

- updating the configuration file (from https://{C&C}/upgrade/vars.yaml)
- upgrading itself

No unequivocally malicious activities that might be expected, like DDoS attacks or spam spreading (yet!), are implemented. However, sending back the IP address, username and password allows the attackers to do anything they want with the machine afterwards. Together with the foul language used in the code, we think it is unlikely that this is just an invasive but innocent experiment or an unfortunate exercise in academic research.

There are reports online about the compromises. For example, one from August 23rd, 2016, may be found on Pastebin. The table below contains the output of running “lsof –n” on the guilty process. Note that the IP address ranges tried by SSH attempts seem random:

CMD	PID	USER	FD	TYPE	SIZE/OFF	NODE	NAME

CMD	PID	USER	FD	TYPE	SIZE/OFF	NODE	NAME	
.javaxxx	2773	root	txt	REG	5822568		/tmp/.javaxxx	(deleted)
.javaxxx	2773	root	5u	IPv6	0t0	TCP	*:13666	(LISTEN)
.javaxxx	2773	root	6u	IPv4	0t0	TCP	127.0.0.1:61314	(LISTEN)
.javaxxx	2773	root	7u	IPv4	0t0	TCP	192.168.88.210:59958->66.209.103.211:ssh	(ESTABLISHED)
.javaxxx	2773	root	10u	IPv4	0t0	TCP	192.168.88.210:57370->139.196.21.134:ssh	(ESTABLISHED)
.javaxxx	2773	root	13u	IPv4	0t0	TCP	192.168.88.210:52507->148.75.167.198:ssh	(SYN_SENT)
.javaxxx	2773	root	16u	IPv4	0t0	TCP	192.168.88.210:54746->208.9.162.70:ssh	(SYN_SENT)
.javaxxx	2773	root	17u	IPv4	0t0	TCP	192.168.88.210:54533->148.75.167.191:ssh	(SYN_SENT)
.javaxxx	2773	root	18u	IPv4	0t0	TCP	192.168.88.210:51856->139.196.20.79:ssh	(ESTABLISHED)
.javaxxx	2773	root	19u	IPv4	0t0	TCP	192.168.88.210:57210->208.9.162.95:ssh	(SYN_SENT)
.javaxxx	2773	root	24u	IPv4	0t0	TCP	192.168.88.210:45946->148.75.167.99:ssh	(SYN_SENT)
.javaxxx	2773	root	25u	IPv4	0t0	TCP	192.168.88.210:55928->208.9.162.25:ssh	(SYN_SENT)
.javaxxx	2773	root	27u	IPv4	0t0	TCP	192.168.88.210:50041->139.196.21.177:ssh	(ESTABLISHED)

Mitigation and cleanup

The trojan isn’t able to maintain persistence after the system is rebooted. Instead, available devices may be compromised repeatedly.

The steps needed to clean up after a compromise are as follows:

- connect to your device using SSH/Telnet,
- look for a process named .javaxxx,
- run commands like netstat or lsof with -n switch to confirm that it is responsible for unwanted connections,
- (voluntarily) collect forensic evidence by dumping the memory space of the corresponding process (with gcore for example). One could also recover the deleted sample from /proc with cp /proc/{pid}/exe {output_file}
- the process with the -KILL

Needless to say that victims have to secure their SSH credentials and have to do that after every factory reset.

We also prepared a plugin for Volatility Framework called vf_ioc_linux_rakos_a.py that would detect indicators of compromise if a whole memory dump supported by this framework is acquired. Moreover, it extracts from the malicious process space data such as configuration or information sent to the C&C. It is available here: https://github.com/eset/malware-ioc/tree/master/rakos

Conclusion

We have presented here another example of a Linux backdoor spreading through a well-known channel. It seems worthwhile for attackers to write new pieces of malicious software to misuse loopholes in the current state of network security. Our advice is this: Don’t build walls around your devices from sticks and straws, but from bricks and stones. The internet is a windy place.

Special thanks to Marc-Étienne Léveillé.

IoCs

Samples

The malware binary is removed after successful execution therefore there are not many samples collected.

SHA1	First seen	Arch	Size	Version
f80836349d6e97251030190ecd30dda0047f1ee6	2016-08-17	EM_X86_64	7 360 928 B	688
def04ec688ac6b41580dd3a6e78445b56536ba34	2016-09-27	EM_X86_64	1 606 936 B	694

SHA1	First seen	Arch	Size	Version
3435ca5505ce8dfe8e1b22e0ebd4f41c60050cc0	2016-09-27	EM_X86_64	1 613 292 B	695
e53c73fe6a552eab720e7ee685ea4e159ebd4fdd	2016-09-27	EM_X86_64	1 613 292 B	697
c93bddd9cdb4f2e185b54a4931257954e25e7c37	2016-09-28	EM_X86_64	1 614 436 B	698
14af6254d9ca310b4d52778d050cb8dd7a5de1d8	2016-10-21	EM_MIPS	4 095 740 B	???
c54d50025d9f66ce2ace3361a8626aee468d94ba	2016-11-09	EM_386	1 697 592 B	700
36b2fffe98f517355425797fc242f2cb82271c0c	2016-11-21	EM_386	1 875 844B	706
E46E8E5E823EB0466981AFB7683FD918D6FE78A9	2016-12-16	EM_386	1 876 888 B	708
0492E5C07C1426AF9CE73AD33E00A3FD8477C6C2	2016-12-16	EM_386	5 688 388 B	711

C&C Servers

217.12.208.28
217.12.203.31
193.169.245.68
46.8.44.55
195.123.210.100
5.34.183.231
5.34.180.64
185.82.216.125
185.14.30.78
185.14.29.65
185.20.184.117

20 Dec 2016 - 03:00PM

Sign up to receive an email update whenever a new article is published in our Ukraine Crisis – Digital Security Resource Center

Newsletter

Discussion