Kaiji: New Chinese Linux malware turning to Golang

intezer.com/blog/research/kaiji-new-chinese-linux-malware-turning-to-golang/

May 4, 2020

Written by Paul Litvak - 4 May 2020

It is not often that you see a botnet's tooling written from scratch. The Internet of things (IoT) botnet ecosystem is relatively well-documented by security specialists. New threat actors are generally discovered quickly due to the inherent noise caused by DDoS operations, both in terms of infecting new machines and conducting operations. Simply, it is difficult to hide such overt activities. Most DDoS actors do not invest resources in creating custom tooling, unless they require specific capabilities, and resort to using well-known botnet implants (e.g. Mirai, BillGates).

In late April we identified a new botnet campaign with definitive Chinese origins,targeting servers and IoT devices via SSH brute forcing. While most attackers derive their implants from popular and well-tested sources such as open source (e.g., Mirai) or blackmarket toolsets (e.g., BillGates), this botnet utilizes its own custom implant, which

MalwareMustDie named **Kaiji** based on one of the function names**.** The botnet was built from scratch using the Golang programming language, which is rare in the IoT botnet landscape.

Technical Analysis

Kaiji spreads exclusively via SSH brute forcing by targeting the root user only. Accessing root is important to its operation since some DDoS attacks are only available via crafting custom network packets. In Linux, custom network packets are only given to a privileged user such as root.

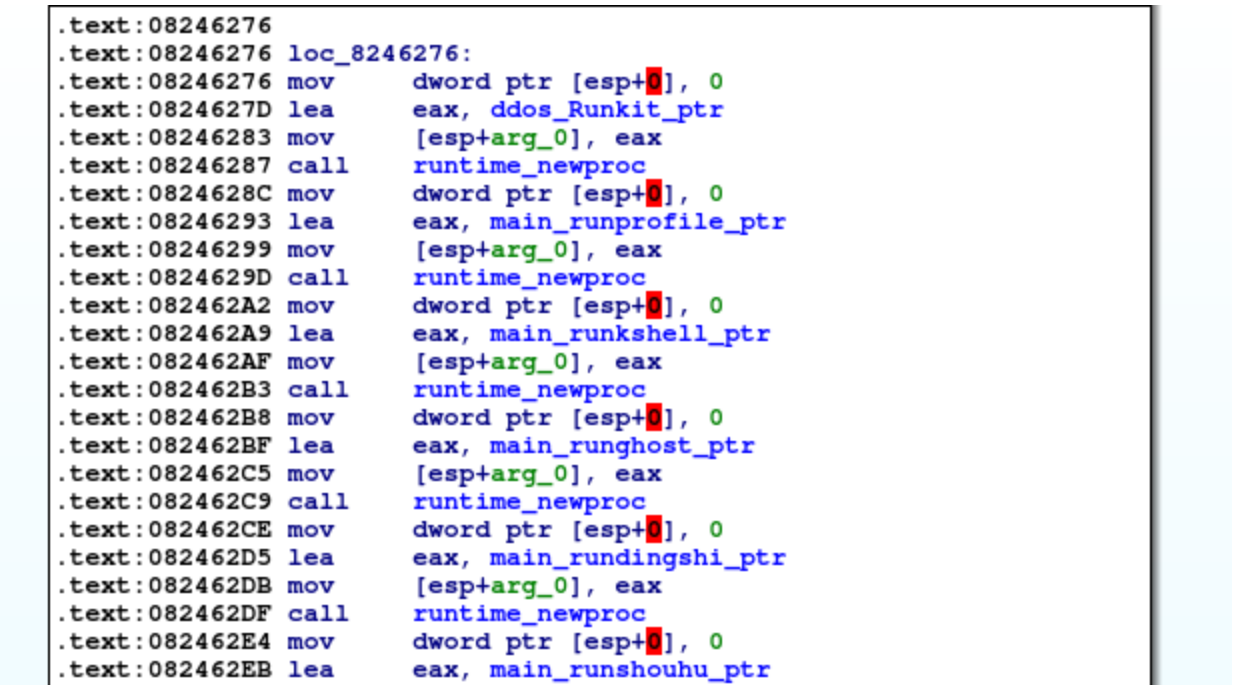
Once a SSH connection is established, a bash script is executed which sets up the environment for the malware:

A /usr/bin/lib directory is created and then Kaiji is installed under the filename ‘netstat’, ‘ps’, ‘ls’, or some other system tool name.

Kaiji has simple features. It consists of an arsenal of multiple DDoS attacks such as ipspoof and synack attacks, an ssh bruteforcer module to continue the spread, and another ssh spreader which relies on hijacking local SSH keys to infect known hosts which the server has connected to in the past.

Despite the Kaiji file being stripped, we were able to restore function names using IDAGolangHelper. This technique works by retrieving function definitions embedded within the Golang binary which are not removed by the *strip* command.

Once the malware is executed, it copies itself to */tmp/seeintlog* and launches a second instance which commences its malicious operations. Each operation is implemented within its own goroutine:



```
.text:08246276
.text:08246276 loc_8246276:
.text:08246276 mov     dword ptr [esp+0], 0
.text:0824627D lea     eax, ddos_Runkit_ptr
.text:08246283 mov     [esp+arg_0], eax
.text:08246287 call    runtime_newproc
.text:0824628C mov     dword ptr [esp+0], 0
.text:08246293 lea     eax, main_runprofile_ptr
.text:08246299 mov     [esp+arg_0], eax
.text:0824629D call    runtime_newproc
.text:082462A2 mov     dword ptr [esp+0], 0
.text:082462A9 lea     eax, main_runkshell_ptr
.text:082462AF mov     [esp+arg_0], eax
.text:082462B3 call    runtime_newproc
.text:082462B8 mov     dword ptr [esp+0], 0
.text:082462BF lea     eax, main_runghost_ptr
.text:082462C5 mov     [esp+arg_0], eax
.text:082462C9 call    runtime_newproc
.text:082462CE mov     dword ptr [esp+0], 0
.text:082462D5 lea     eax, main_rundingshi_ptr
.text:082462DB mov     [esp+arg_0], eax
.text:082462DF call    runtime_newproc
.text:082462E4 mov     dword ptr [esp+0], 0
.text:082462EB lea     eax, main_runshouhu_ptr
```

There exist 13 central goroutines which are important for the implant's operation. Many of these functions are named in an English representation of Chinese words. We have highlighted the most interesting functions and added a translation from Chinese to relevant functions:

doLink routine:

Decrypt C2 addresses, register the newly infected server to one of the command servers and launch the doTask and RotKit goroutines.

Incidentally, some of the C2 addresses are decrypted through a chain of three encryption schemes, while another C2 address is simply encoded in base64:

```
mov     [esp+120h+var_120], eax
mov     eax, dword_84722C4
mov     ecx, off_84722C0 ; "MS52ZXJzaW9uZGF5Lnl
mov     [esp+120h+var_11C], ecx
mov     [esp+120h+var_118], eax
call    encoding_base64__Encoding_DecodeString
```

```
mov     [esp+54h+var_40], ebx
call    ddos_DecryptCFB
mov     eax, [esp+54h+var_3C]
mov     ecx, [esp+54h+var_38]
mov     edx, [esp+54h+var_34]
mov     [esp+54h+arg1], eax
mov     [esp+54h+var_50], ecx
mov     [esp+54h+var_4C], edx
mov     eax, [esp+54h+var_C]
mov     [esp+54h+var_48], eax
mov     eax, [esp+54h+var_28]
mov     [esp+54h+var_44], eax
mov     eax, [esp+54h+var_24]
mov     [esp+54h+var_40], eax
call    ddos_DecryptCBC
mov     eax, [esp+54h+var_3C]
mov     ecx, [esp+54h+var_38]
mov     edx, [esp+54h+var_34]
mov     [esp+54h+arg1], eax
mov     [esp+54h+var_50], ecx
mov     [esp+54h+var_4C], edx
mov     eax, [esp+54h+var_10]
mov     [esp+54h+var_48], eax
mov     eax, [esp+54h+var_30]
mov     [esp+54h+var_44], eax
mov     eax, [esp+54h+var_2C]
mov     [esp+54h+var_40], eax
call    ddos_DecryptECB
```

On the left, C2 base64 decoding. On the right, C2 decryption.

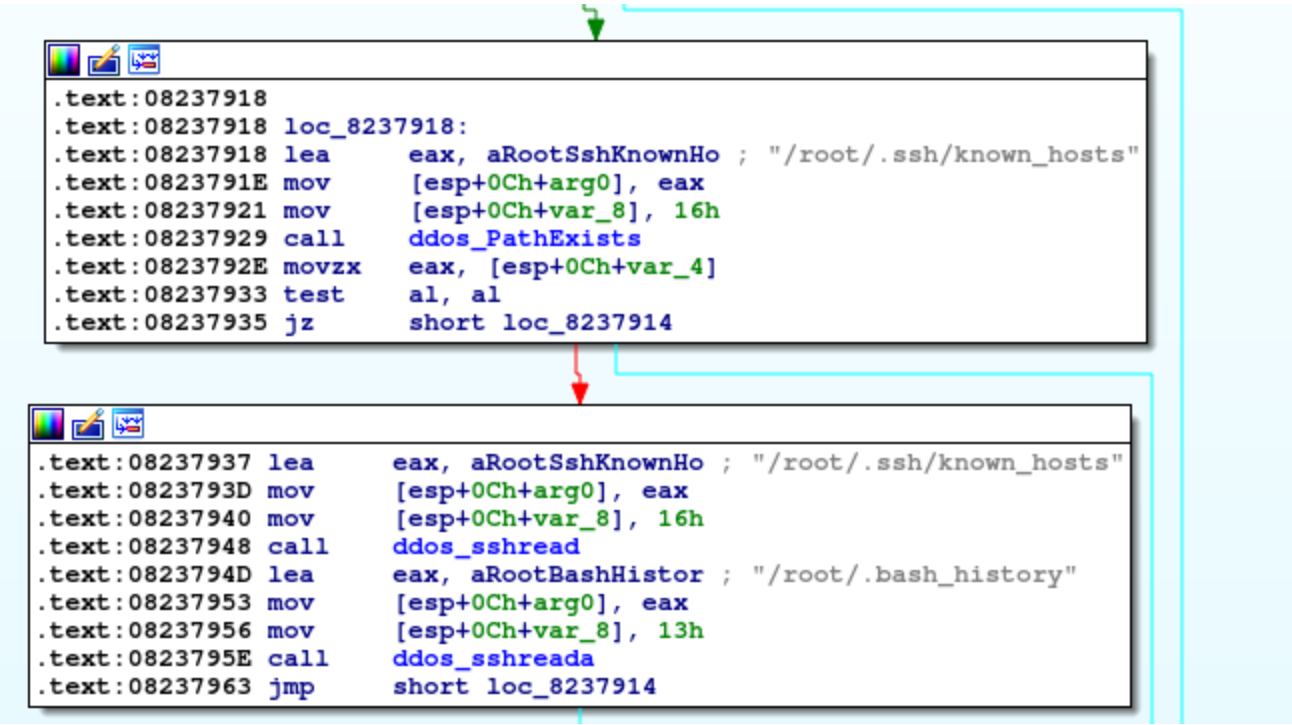
The binary contained four command server hostnames, two of which were resolved to localhost since they were registered. The only hostname which worked was operational for two weeks before failing to respond.

main_doTask:
Fetches commands from the C2. These include:
• DDoS instructions
• SSH bruteforce instructions, including host range and a password to attempt login
• Run shell command
• Replace C2 servers
• Delete itself and remove all persistence

For DDoS operations, a target and an attack technique are retrieved.

Attacks include:
• Two TCPFlood implementations (one with raw sockets)
• Two UDPFlood implementations (one with raw sockets)
• IPSpoof attack
• SYNACK attack
• SYN attack
• ACK attack

ddos_Rotkit:
Tries to connect to known hosts through existing SSH RSA keys or IPs found in bash history:



main_runkshell:
Install persistence through rc.d and Systemd services:

Systemd (/etc/systemd/system/linux.service):

```
[Unit]
Description=
[Service]
Type=forking
ExecStart=/boot/System.img.config
ExecReload=/boot/System.img.config
ExecStop=/boot/System.img.config
[Install]
WantedBy=multi-user.target
```

rc.d (/etc/rc.d/init.d/linux_kill):

```
#!/bin/sh
### BEGIN INIT INFO
#chkconfig: 2345 10 90
#description:System.img.config
# Default-Start: 2 3 4 5
```

```
# Default-Stop:
### END INIT INFO
/boot/System.img.config
exit 0
```

main_runghost: Install persistence through /etc/profile.d (/etc/profile.d/linux.sh)
main_rundingshi (漢字: run timing): Install persistence through crontab
main_runganran (漢字: run infection): Another persistence technique, backdoor the SSH init script /etc/init.d/ssh to call the rootkit on startup
main_runshouhu (漢字: run surgery): Copy the rootkit to */etc/32679* and run it every 30 seconds
main_runkaiji (漢字: run boot): Install more persistence init.d files, e.g.: /etc/init.d/boot.local
ddos_rdemokill: Check the CPU usage machine periodically and kill if CPU usage exceeds 85%. This can inadvertently kill unrelated processes. Interestingly, this function refers to the rootkit as a demo

In our own sandbox we observed that the rootkit tends to invoke itself too many times, leaving the machine gasping for memory:

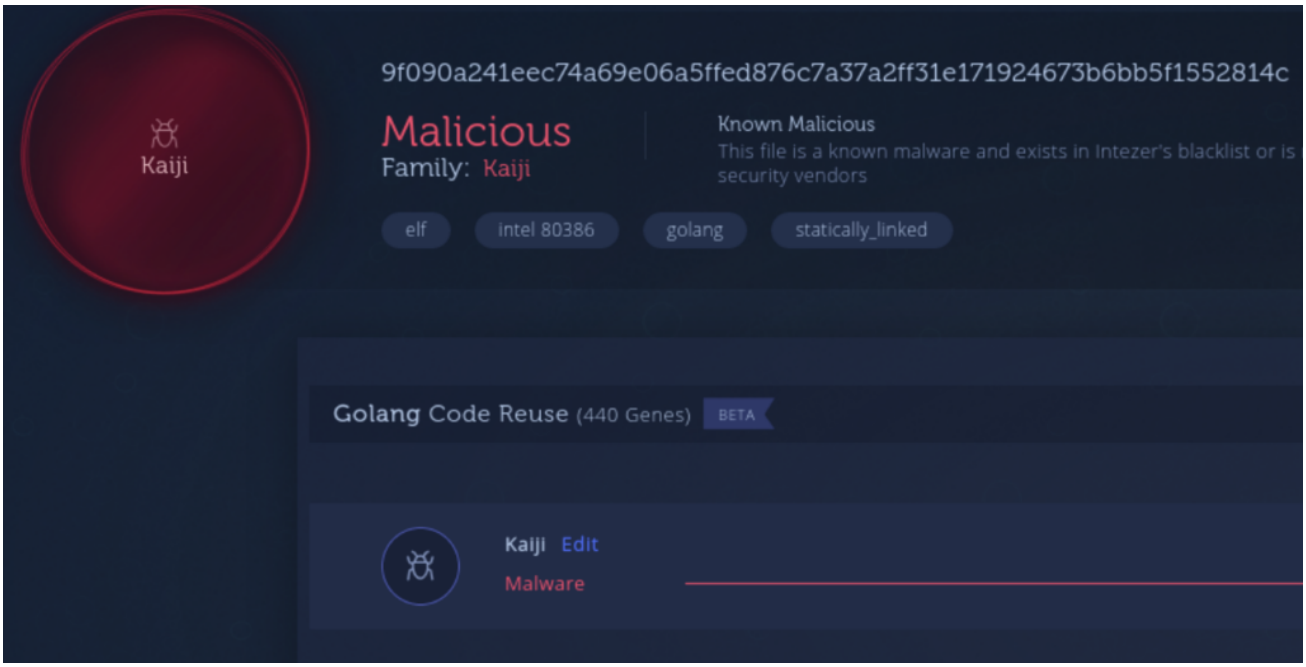
```
 46336 pts/0    00:00:00 seeintlog <defunct>
 46337 pts/0    00:00:00 ls
 46342 pts/0    00:00:00 seeintlog <defunct>
 46343 pts/0    00:00:00 ls
 46348 pts/0    00:00:00 seeintlog <defunct>
 46349 pts/0    00:00:00 ls
 46354 pts/0    00:00:00 seeintlog <defunct>
 46355 pts/0    00:00:00 ls
 46360 pts/0    00:00:00 seeintlog <defunct>
 46361 pts/0    00:00:00 ls
 46366 pts/0    00:00:00 seeintlog <defunct>
 46369 pts/0    00:00:00 ls
 46372 pts/0    00:00:00 seeintlog <defunct>
 46373 pts/0    00:00:00 ls
 46379 pts/0    00:00:00 seeintlog <defunct>
 46380 pts/0    00:00:00 ls
 46385 pts/0    00:00:00 seeintlog <defunct>
 46386 pts/0    00:00:00 ls
 46391 pts/0    00:00:00 seeintlog <defunct>
 46392 pts/0    00:00:00 ls
 46397 pts/0    00:00:00 seeintlog <defunct>
 46398 pts/0    00:00:00 ls
 46403 pts/0    00:00:00 seeintlog <defunct>
 46519 pts/3    00:00:00 ps
paul@ubuntu:~$ ps -A | grep seeintlog | wc
    989    4944   44495
```

This, together with the fact that the C2 was operational only temporarily, and the presence of a ‘demo’ string, led us to believe that this is an early version still in testing.

Conclusion

It is rare to see a botnet written from scratch, considering the tools readily available to attackers in blackmarket forums and open source projects. In this post we have uncovered a new DDoS operation in its early stages that was written from scratch. This is another confirmation of an interesting trajectory noted by vendors such as Palo Alto that malware developers are turning to modern languages such as Golang for their operations.

The Kaiji samples are now indexed in Intezer Analyze. Powered by our new Golang cross platform code connections, users will be able to easily spot if this threat actor switches to Windows.



IOCs

4e8d4338cd3b20cb027a8daf108c654c10843e549c3f3da6646ac2bb8ffbe24d
9198853b8713560503a4b76d9b854722183a94f6e9b2a46c06cd2865ced329f7
98aee62701d3a8a75aa19028437bc2d1156eb9bfc08661c25db5c2e26e364dca
0ed0a9b9ce741934f8c7368cdf3499b2b60d866f7cc7669f65d0783f3d7e98f7
f4a64ab3ffc0b4a94fd07a55565f24915b7a1aaec58454df5e47d8f8a2eec22a
9f090a241eec74a69e06a5ffed876c7a37a2ff31e171924673b6bb5f1552814c
370efd28a8c7ca50275957b47774d753aabb6d7c504f0b81a90c7f96c591ae97
357acbacdb9069b8484f4fdead1aa946e2eb4a505583058f91f40903569fe3f3
cu.versiondat[.]xyz

1.versionday[.]xyz
www.aresboot[.]xyz
www.6×66[.]com
www.2s11[.]com

Paul Litvak
Paul is a malware analyst and reverse engineer at Intezer. He previously served as a developer in the Israel Defense Force (IDF) Intelligence Corps for three years.