Malware Report | Cryptojacker

Kənan Rəsulov

Date: 22.02.2025

Task: As a part of my internship in Intern Intelligence, my second task is is to analyze a malware sample in sandbox environment both static and dynamically. In this task, I used a **Linux Virtual Machine** to observe the behaviour of the sample.

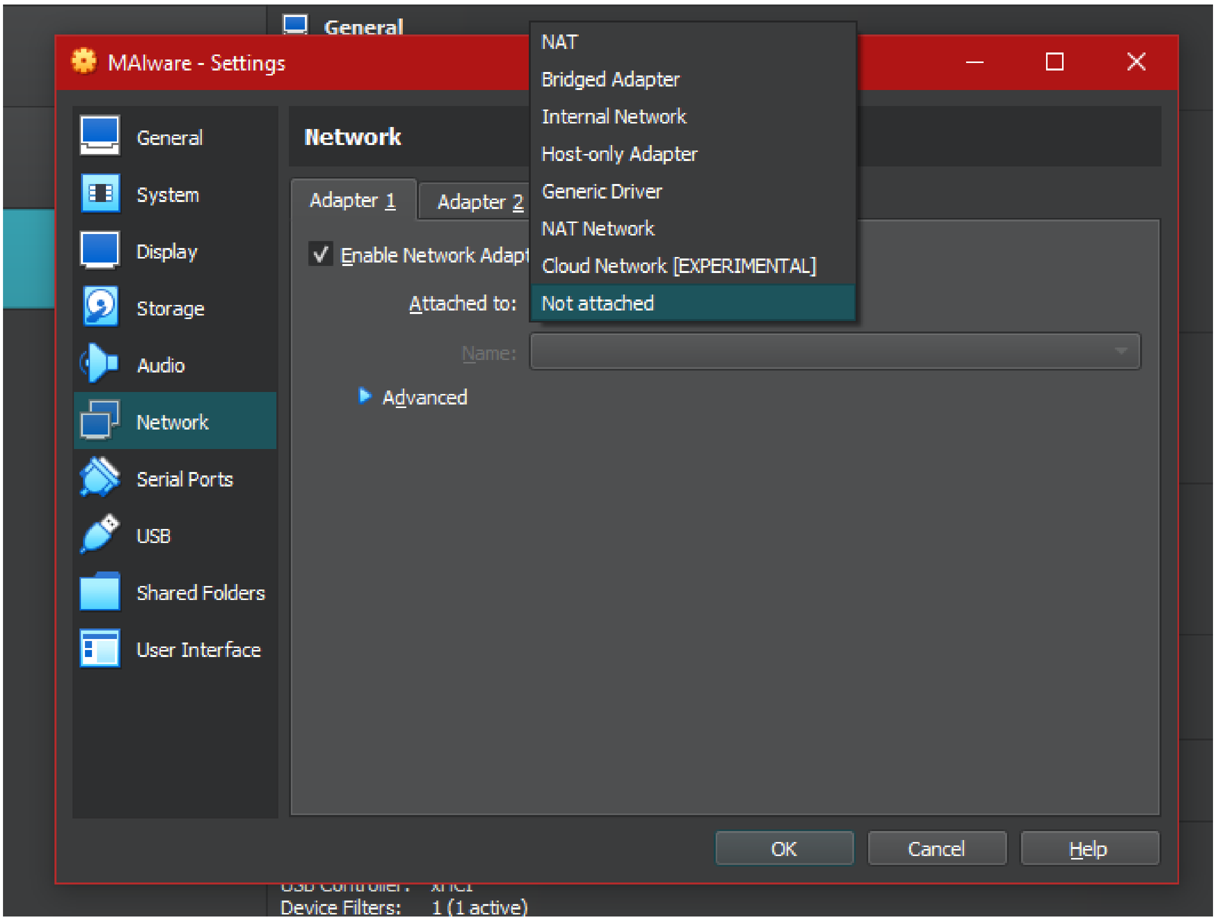
Source of the malware sample:

https://github.com/MalwareSamples/Linux-Malware-Samples

Table of contents

Getting the environment ready

First of all, if we are going to test a malware, we should create a proper lab environment to prevent our main OS get infected. First step for this I created a clone of my original Kali Linux VM. And configured its network settings.

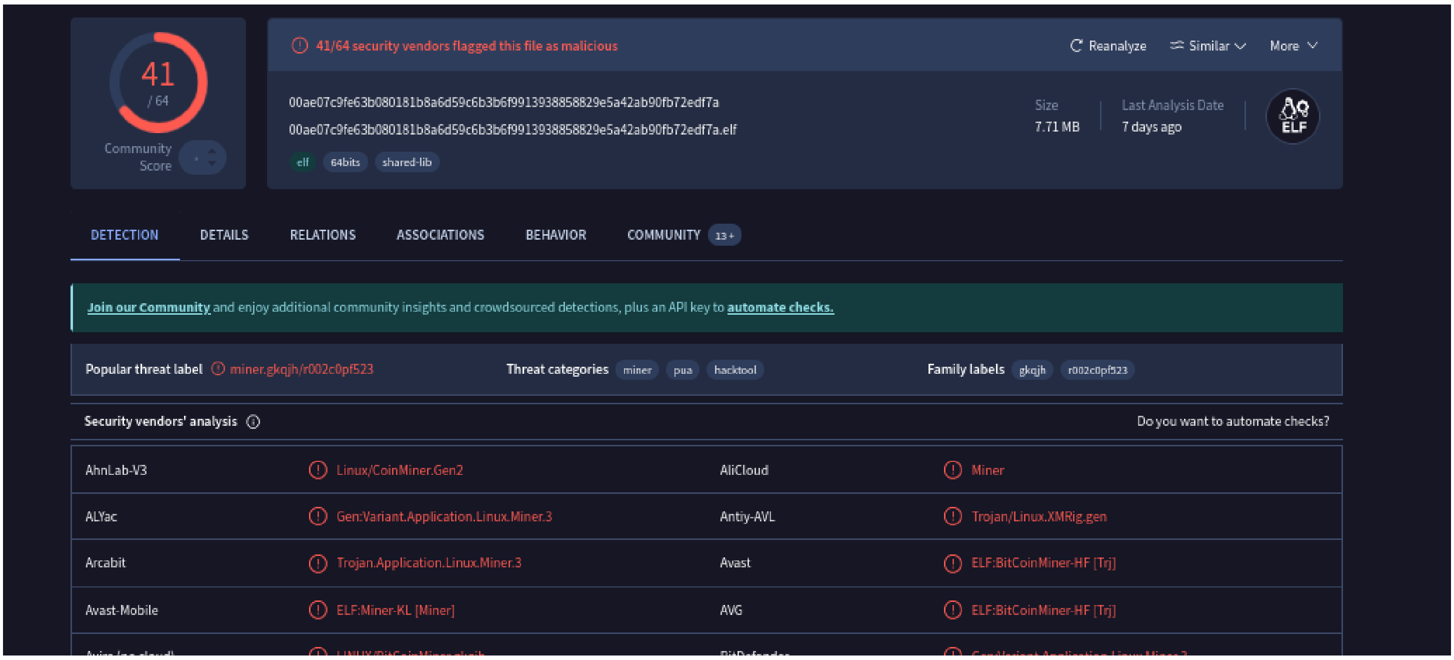


So we deattached it from network to prevent spread. Let's start our machine and start our testing. I also created a directory called '*Malware_analysis*' and stored malware in sub directory called '*samples*'

Static analysis

I started my analysis by checking the sample's type and found that this is an ELF binary (ELF (Executable and Linkable Format) is the standard file format for executables, object code, shared libraries, and core dumps in Linux and other Unix-like operating systems)

Then I checked its **SHA-256** and **SHA-512** hashes for testing it in Virus Total



It seems like we are dealing with a **Crypto Miner** (Cryptojacker) which is detected by 41 security vendors. I also checked its strings to find suspicious rows exposing its purpose

```
┌──(root㉿kali)-[~/Malware-Analysis/logs]
└─# strings samples/malware_sample >> logs/malware_strings
```

```
  GNU nano 8.0                         malware_strings
N3fmt2v712system_errorE
St13runtime_error
N3fmt2v712format_errorE
NdP#
TfAX
\$~s
cryptonight/0
cn/0
cryptonight
cryptonight/1
cn/1
cryptonight-monerov7
cryptonight_v7
cryptonight/2
cn/2
cryptonight-monerov8
cryptonight_v8
cryptonight/r
cn/r
cryptonight_r
cryptonight/fast
cn/fast
cryptonight/msr
cn/msr
cryptonight/half
cn/half
cryptonight/xao
cn/xao
cryptonight_alloy
cryptonight/rto
cn/rto
```

We saw **cryptonight_r, cryptonight/fast, cryptonight_alley** etc. strings in the output and this also proves this executable has crypto miner inside it

Dynamic analysis

After I finished static analysis, I continued with analyzing it dynamically with executing it and monitoring its behaviour. To do that I first used tool called *strace* to observe system calls

```
┌──(root㉿kali)-[~/Malware-Analysis/samples]
└─# strace -f -e trace=all ./malware_sample >> strace.txt
execve("./malware_sample", ["./malware_sample"], 0x7ffe35840598 /* 54 vars */) = 0
brk(NULL)                               = 0x5650f7957000
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f2998668000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=105710, ...}) = 0
mmap(NULL, 105710, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f299864e000
close(3)                                = 0
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libpthread.so.0", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\0\0\0\0\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0644, st_size=14408, ...}) = 0
mmap(NULL, 16400, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f2998649000
mmap(0x7f299864a000, 4096, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1000) = 0x7f299864a000
mmap(0x7f299864b000, 4096, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x2000) = 0x7f299864b000
mmap(0x7f299864c000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x2000) = 0x7f299864c000
close(3)                                = 0
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/librt.so.1", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\0\0\0\0\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0644, st_size=14552, ...}) = 0
mmap(NULL, 16400, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f2998644000
mmap(0x7f2998645000, 4096, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1000) = 0x7f2998645000
mmap(0x7f2998646000, 4096, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x2000) = 0x7f2998646000
mmap(0x7f2998647000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x2000) = 0x7f2998647000
close(3)                                = 0
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libdl.so.2", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\0\0\0\0\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0644, st_size=14408, ...}) = 0
mmap(NULL, 16400, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f299863f000
mmap(0x7f2998640000, 4096, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1000) = 0x7f2998640000
mmap(0x7f2998641000, 4096, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x2000) = 0x7f2998641000
mmap(0x7f2998642000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x2000) = 0x7f2998642000
close(3)                                = 0
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libm.so.6", O_RDONLY|O_CLOEXEC) = 3
```

Analyzing these system calls we can see sample tries to connect somewhere and uses obfuscation to fool the OS. Next I used *ltrace* to monitor library calls

```
┌──(root㉿kali)-[~/Malware-Analysis/samples]
└─# ltrace ./malware_sample >> ltrace.txt
__cxa_atexit(0x558e6898a2d0, 0x558e691b5580, 0x558e691a9008, 0x7fc5f72f0680)      = 0
__cxa_atexit(0x558e6899d030, 0x558e691b55f0, 0x558e691a9008, 0)                   = 0
__cxa_atexit(0x558e6899d030, 0x558e691b5600, 0x558e691a9008, 32)                  = 0
__cxa_atexit(0x558e689b82f0, 0x558e691b5680, 0x558e691a9008, 64)                  = 0
__cxa_atexit(0x558e68d9c650, 0x558e691b56a0, 0x558e691a9008, 96)                  = 0
__cxa_atexit(0x558e689b99f0, 0x558e691b58e0, 0x558e691a9008, 128)                 = 0
__cxa_atexit(0x558e689b97b0, 0x558e691b5920, 0x558e691a9008, 160)                 = 0
__cxa_atexit(0x558e689c4ac0, 0x558e691b5940, 0x558e691a9008, 192)                 = 0
strlen("x")                                                                        = 1
malloc(2)                                                                          = 0x558e6aeb72a0
memcpy(0x558e6aeb72a0, "x\0", 2)                                                   = 0x558e6aeb72a0
__cxa_atexit(0x558e689b82f0, 0x558e691b5990, 0x558e691a9008, 0x558e6aeb7278)      = 0
strlen("x")                                                                        = 1
malloc(2)                                                                          = 0x558e6aeb72c0
memcpy(0x558e6aeb72c0, "x\0", 2)                                                   = 0x558e6aeb72c0
__cxa_atexit(0x558e689b82f0, 0x558e691b5980, 0x558e691a9008, 0x558e6aeb7278)      = 0
strlen("127.0.0.1")                                                                = 9
malloc(10)                                                                         = 0x558e6aeb72e0
memcpy(0x558e6aeb72e0, "127.0.0.1\0", 10)                                          = 0x558e6aeb72e0
__cxa_atexit(0x558e689b82f0, 0x558e691b59a0, 0x558e691a9008, 0x2e302e302e373231)  = 0
strlen("default")                                                                  = 7
memcmp("default", "default", 7)                                                    = 0
__cxa_atexit(0x558e689e2020, 0x558e691b6d60, 0x558e691a9008, 0x40000000)          = 0
__cxa_atexit(0x558e68a0b830, 0x558e691b8120, 0x558e691a9008, 0xa1fa35ef)          = 0
strlen("application/json")                                                         = 16
malloc(17)                                                                         = 0x558e6aeb7300
memcpy(0x558e6aeb7300, "application/json", 16)                                     = 0x558e6aeb7300
__cxa_atexit(0x558e68d9c650, 0x558e691b8220, 0x558e691a9008, 0x558e6aeb7300)      = 0
strlen("Content-Type")                                                             = 12
memcpy(0x558e691b8210, "Content-Type", 12)                                         = 0x558e691b8210
__cxa_atexit(0x558e68d9c650, 0x558e691b8200, 0x558e691a9008, 0x2d746e65746e6f43)  = 0
strlen("content-type")                                                             = 12
memcpy(0x558e691b81f0, "content-type", 12)                                         = 0x558e691b81f0
```

In this output we saw some interesting lines:

__cxa_atexit()

This function registers a callback to be executed when the program exits. Malware might use this to clean up traces, remove files, or trigger final payloads.The different memory addresses like 0x558e6898a2d0 represent function pointers, which could hint at suspicious behavior depending on what functions they point to.

strlen()

Measures the length of a string. Commonly used to process hostnames, IP addresses, or C2 commands.Notably, strlen("127.0.0.1") = 9 suggests that the malware is referencing 127.0.0.1, which could be a test or fallback IP address.

malloc()

Allocates memory on the heap. Malware often uses dynamic memory allocation to store payloads, data exfiltration buffers, or decrypted code.The allocation of small chunks like malloc(2) for "x" or malloc(10) for 127.0.0.1 suggests storing simple strings, possibly for communication.

memcpy()

Copies data from one location to another. This could be used for assembling network packets, obfuscating data, or manipulating sensitive information.

Strings Like:

"127.0.0.1" – A local IP, which could be used for testing or tunneling. However, malware often replaces this with a real C2 server during deployment. "application/json" and "Content-Type" – Suggest the malware might communicate over HTTP, possibly sending or receiving JSON payloads. "text/plain" – Might indicate different payload formats or fallback communication methods.

Next tool I am used is called *tcpdump*, which dumps the network traffic. I used it to create a **.pcap** file to analyze it further.

```
┌──(root㉿kali)-[~/Malware-Analysis/samples]
└─# ./malware_sample
[2025-02-22 07:38:14.871] unable to open "/root/Malware-Analysis/samples/config.json".
[2025-02-22 07:38:14.871] unable to open "/root/.xmrig.json".
[2025-02-22 07:38:14.871] unable to open "/root/.config/xmrig.json".
[2025-02-22 07:38:14.871] no valid configuration found, try https://xmrig.com/wizard

┌──(root㉿kali)-[~/Malware-Analysis/samples]
└─# sudo tcpdump -i eth0 -w capture.pcap
tcpdump: listening on eth0, link-type EN10MB (Ethernet), snapshot length 262144 bytes
^C69 packets captured
74 packets received by filter
0 packets dropped by kernel
```

Then I opened the file with Wireshark; "wireshark capture.pcap"

Apply a display filter ... <Ctrl-/>

No.	Time	Source	Destination	Protocol	Length	Info
26	2.871298	192.168.0.103	239.255.255.250	SSDP	376	NOTIFY * HTTP/1.1
27	2.871299	192.168.0.103	239.255.255.250	SSDP	433	NOTIFY * HTTP/1.1
28	2.871299	192.168.0.103	239.255.255.250	SSDP	421	NOTIFY * HTTP/1.1
29	2.871406	192.168.0.103	239.255.255.250	SSDP	431	NOTIFY * HTTP/1.1
33	4.535089	192.168.0.106	239.255.255.250	SSDP	167	M-SEARCH * HTTP/1.1
59	24.999916	192.168.0.106	239.255.255.250	SSDP	167	M-SEARCH * HTTP/1.1
1	0.000000	192.168.0.108	34.107.221.82	TCP	66	35736 → 80 [ACK] Seq=1 Ack=1 Win=249 Len=0 TSval=2139171581 TSec…
41	10.243780	192.168.0.108	34.107.221.82	TCP	66	[TCP Dup ACK 1#1] 35736 → 80 [ACK] Seq=1 Ack=1 Win=249 Len=0 TSv…
53	20.479759	192.168.0.108	34.107.221.82	TCP	66	[TCP Dup ACK 1#2] 35736 → 80 [ACK] Seq=1 Ack=1 Win=249 Len=0 TSv…
68	30.721137	192.168.0.108	34.107.221.82	TCP	66	[TCP Dup ACK 1#3] 35736 → 80 [ACK] Seq=1 Ack=1 Win=249 Len=0 TSv…
30	3.651028	192.168.0.108	34.107.243.93	TLSv1.2	105	Application Data
32	3.729547	192.168.0.108	34.107.243.93	TCP	66	38462 → 443 [ACK] Seq=40 Ack=40 Win=249 Len=0 TSval=1812573229 T…
38	9.655946	192.168.0.108	34.107.243.93	TLSv1.2	105	Application Data
40	9.734457	192.168.0.108	34.107.243.93	TCP	66	38462 → 443 [ACK] Seq=79 Ack=79 Win=249 Len=0 TSval=1812579234 T…
48	15.657141	192.168.0.108	34.107.243.93	TLSv1.2	105	Application Data
50	15.735262	192.168.0.108	34.107.243.93	TCP	66	38462 → 443 [ACK] Seq=118 Ack=118 Win=249 Len=0 TSval=1812585235…
55	21.661120	192.168.0.108	34.107.243.93	TLSv1.2	105	Application Data
57	21.739527	192.168.0.108	34.107.243.93	TCP	66	38462 → 443 [ACK] Seq=157 Ack=157 Win=249 Len=0 TSval=1812591239…
63	27.664363	192.168.0.108	34.107.243.93	TLSv1.2	105	Application Data
65	27.742466	192.168.0.108	34.107.243.93	TCP	66	38462 → 443 [ACK] Seq=196 Ack=196 Win=249 Len=0 TSval=1812597242…
36	8.878237	TPLink_3c:d3:79	Broadcast	ARP	60	Who has 192.168.0.107? Tell 192.168.0.1
37	9.596162	fe80::c8cd:5c1f:cc3…	ff02::2	ICMPv6	62	Router Solicitation

We see some suspicious packets here:

1. Frames 41, 53, and 68 show TCP Dup ACK #1, #2, and #3 from 192.168.0.108 to 34.107.221.82 on port 80 (HTTP).

Why it's suspicious:

Multiple duplicate ACKs indicate packet loss or potential network manipulation. Since it's an external IP (34.107.221.82), this could suggest disrupted communication with an external server, possibly a

C2 server. Port 80 is often used for malware communication since it easily bypasses firewalls due to being a common HTTP port.

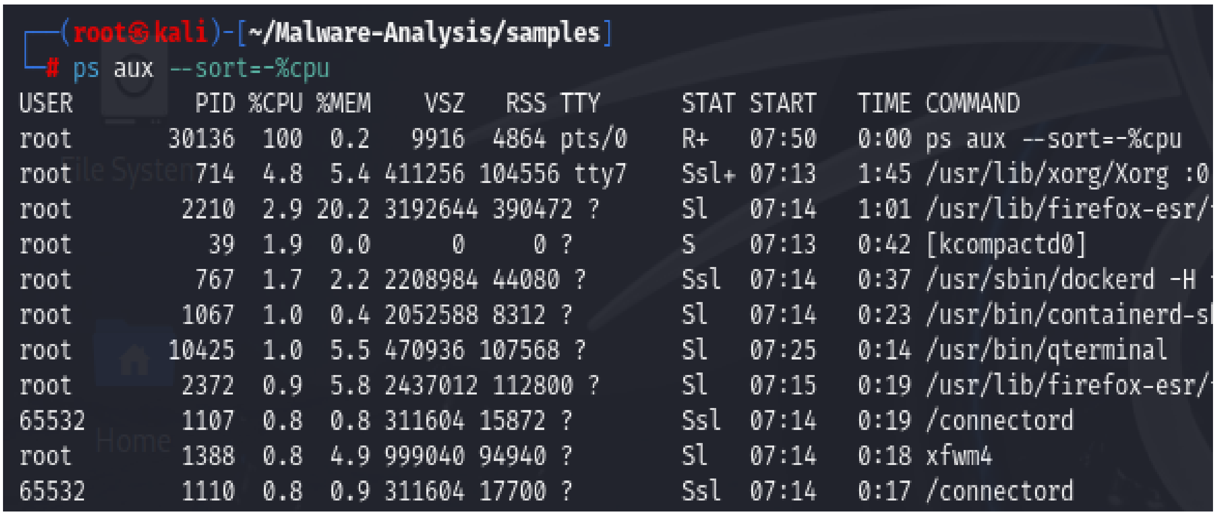
2. Unusual External Communication

Traffic between 192.168.0.108 and 34.107.243.93 over TCP 443 (TLS/SSL) in frames 30-65 is consistent, but consider:

Why it's suspicious:

Frequent short-length TLS packets could suggest beaconing behavior, where malware sends periodic "heartbeat" signals to a C2 server. Repeated small packets without much data can indicate malware maintaining persistence or awaiting commands.

In the end, I analyzed the processes with *ps* command, but there was just only one process called **/connectord** which was executed in the root directory.



```
┌──(root㉿kali)-[~/Malware-Analysis/samples]
└─# ps aux --sort=-%cpu
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root     30136  100  0.2   9916  4864 pts/0    R+   07:50   0:00 ps aux --sort=-%cpu
root       714  4.8  5.4 411256 104556 tty7    Ssl+ 07:13   1:45 /usr/lib/xorg/Xorg :0
root      2210  2.9 20.2 3192644 390472 ?      Sl   07:14   1:01 /usr/lib/firefox-esr/
root        39  1.9  0.0      0     0 ?        S    07:13   0:42 [kcompactd0]
root       767  1.7  2.2 2208984 44080 ?       Ssl  07:14   0:37 /usr/sbin/dockerd -H
root      1067  1.0  0.4 2052588 8312 ?        Sl   07:14   0:23 /usr/bin/containerd-s
root     10425  1.0  5.5 470936 107568 ?       Sl   07:25   0:14 /usr/bin/qterminal
root      2372  0.9  5.8 2437012 112800 ?      Sl   07:15   0:19 /usr/lib/firefox-esr/
65532     1107  0.8  0.8 311604 15872 ?        Ssl  07:14   0:19 /connectord
root      1388  0.8  4.9 999040 94940 ?        Sl   07:14   0:18 xfwm4
65532     1110  0.8  0.9 311604 17700 ?        Ssl  07:14   0:17 /connectord
```