Connecting Kinsing malware to Citrix and SaltStack campaigns

By susannah.matt@redcanary.com

Published: 2020-07-22 · Archived: 2026-04-05 22:20:58 UTC

Kinsing, a malware family distributed by the H2Miner botnet, is known for targeting Linux-based infrastructure systems including Docker container hosts, Redis instances, and (in recent weeks) Salt servers.

Up to this point, the security firm Intezer and an individual known as "Blackorbird" on Twitter have published evidence linking Kinsing to a Salt exploitation campaign from early May 2020.

Red Canary has uncovered additional evidence linking the Kinsing malware family to Salt server attack campaigns, and, in turn, how these attacks seem to relate to the Citrix ADC/Netscaler exploits from earlier in the year. This post includes a comparison of publicly available research from various sources, some manual analysis of binaries collected from VirusTotal, and additional research based on observations across the environments that Red Canary monitors.

If you give a monkey a typewriter and infinite time, he'll eventually produce some groundbreaking threat intelligence.

Establishing Kinsing

Docker & Redis host exploitation

One of the better-documented H2Miner/Kinsing campaigns occurred from December 2019 to March 2020 and was documented by Aqua Security. In this campaign, the H2Miner botnet exploited misconfigured Docker Engine API ports to deploy a compromised Docker container. The entry point for the compromised container would download and execute a script, `d.sh`.

Consistent with other opportunistic miner threats documented here and elsewhere, the `d.sh` script attempted to stop and remove other potential cryptocurrency miners from victim systems in addition to some security tools. Finally, the script downloaded a Golang-based remote administration tool (RAT) and established persistence via a cron job. This cron persistence would repeat the infection process and keep the RAT running.

Eventually, the malware would attempt to execute a script for lateral movement. To achieve this, a script would enumerate all SSH known hosts and potential keys from `known-hosts`, `authorized-keys`, `.bash-history`, and other files. Once a list of hosts and keys are gathered, the script issues `ssh` commands to move laterally around the environment and deploy itself.

Golang RAT

The primary malware deployed during Docker host exploitation was a Golang-based RAT named `kinsing`. In the deployment scripts, the RAT was even named `kinsing` during downloads for execution. It bears mentioning

that while `kinsing` is a tool used by the H2Miner cryptocurrency miner botnet, the RAT itself is separate from the mining component. The RAT uses the go-resty, gopsutil, osext, and diskv Go libraries for multiple tasks related to command and control (C2), process monitoring, and execution.

Researchers from Alibaba Cloud analyzed Kinsing in January 2020 as part of an investigation into the H2Miner botnet and tied it to the exploitation of vulnerable Redis servers. In their analysis, they identified multiple function symbols, including some with the the following names:

- `backconnect`
- `checkHealth`
- `connectForSocks`
- `downloadAndExecute`
- `getMinerPid`
- `isMinerRunning`
- `runTask`
- `masscan`

Analysis by Lacework revealed an additional point of interest: the Kinsing malware often contains the complete play of *Hamlet* by William Shakespeare as a measure to slow analysis.

Trend Micro's analysis indicated that Kinsing supported several commands from C2 with data reported back in the C2 URL with special formatting. In these cases, it appeared that these C2 URL fields included:

- `/get`
- `/h`
- `/getT`
- `/l`
- `/o`
- `/r`
- `/s`
- `/mg`
- `/ms`

In each campaign where Kinsing is used, the RAT binary contains a hardcoded filename where it will deploy a XMRIG Monero miner binary.

In our analysis at Red Canary, we noticed an additional point of interest: Kinsing's `masscan` functionality is implemented by unpacking a `firewire.sh` shell script containing code to install `libpcap` dependencies and execute masscan as the process name `firewire`. The command line for this execution followed this pattern:

```
firewire -iL <input list> --rate <rate> -p<ports> -oL <output list> 2>/dev/null
```

Deployed XMRIG

In both the Redis and Docker host exploitation campaigns the Kinsing RAT went on to deploy and execute an XMRIG Monero miner process. In the case of the Docker host campaign, it was named `kdevtmpfsi`.

During our analysis, the binary contained all the properties we'd expect of an unprotected XMRIG executable. Multiple binary strings matched XMRIG usage options in plaintext as well as expected version numbers.

```
 -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
-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)
-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 5%% (5 minutes in 100 minutes)
--donate-over-proxy=N control donate over xmrig-proxy feature
--no-cpu disable CPU mining backend
-t, --threads=N number of CPU threads
-v, --av=N algorithm variation, 0 auto select
--cpu-affinity set process affinity to CPU core(s), mask 0x3 for cores 0 and 1
--cpu-priority set process priority (0 idle, 2 normal to 5 highest)
--cpu-max-threads-hint=N maximum CPU threads count (in percentage) hint for autoconfig
--cpu-memory-pool=N number of 2 MB pages for persistent memory pool, -1 (auto), 0 (disable)
--cpu-no-yield prefer maximum hashrate rather than system response/stability
--no-huge-pages disable huge pages support
--asm=ASM ASM optimizations, possible values: auto, none, intel, ryzen, bulldozer
--randomx-init=N threads count to initialize RandomX dataset
--randomx-no-numa disable NUMA support for RandomX
--randomx-mode=MODE RandomX mode: auto, fast, light
--randomx-1gb-pages use 1GB hugepages for dataset (Linux only)
--randomx-wrmsr=N write value (0-15) to Intel MSR register 0x1a4 or do nothing (-1) (Linux only)
-S, --syslog use system log for output messages
-l, --log-file=FILE log all output to a file
--print-time=N print hashrate report every N seconds
--no-color disable colored output
--verbose verbose output
-c, --config=FILE load a JSON-format configuration file
-B, --background run the miner in the background
-V, --version output version information and exit
-h, --help display this help and exit
--dry-run test configuration and exit
--export-topology export hwloc topology to a XML file and exit
XMRig 5.4.0
```

```
built on Dec 22 2019 with GCC
failed to export hwloc topology.
hwloc topology successfully exported to "%s"
Usage: xmrig [OPTIONS]
```

In addition to this confirmation of XMRIG's presence, the adversary included plaintext configuration data embedded into the binary. Often when the official XMRIG releases are used, an operator supplies a `config.json` file with all the information needed to make the miner work. In the case of this binary, the contents of `config.json` were embedded, disclosing details we can use for tracking the H2Miner campaigns. While the configuration was fairly generic, one piece of information stood out: the Monero wallet address.

```
 "background": true,
"donate-level": 0,
"cpu": true,
"colors": false,
"opencl": false,
"pools": [
"coin": "monero",
"algo": null,
"url": "xmr-eu1.nanopool.org",
"user": "46V5WXwS3gXfsgR7fgXeGP4KAXtQTXJfkicBoRSHXwGbhVzj1JXZRJRhbMrvhxvXvgbJuyV3GGWzD6JvVMuQwAXxLZmTWkb",
"pass": "mine",
"tls": false,
"keepalive": true,
"nicehash": false
```

Linking Kinsing to SaltStack

Golang RAT + Monero miner

In May 2020, security vendors linked Kinsing to an additional campaign: one exploiting SaltStack CVE-2020-11651 and CVE-2020-11652. Initial reports were posted via Twitter and were later followed by reporting from various sources. Intezer took the lead on this documentation, demonstrating through an analysis of code similarities that the samples seen in SaltStack exploits were related to the Kinsing RAT. In this section, I want to walk through some of the similarities between the Kinsing RAT used in the SaltStack campaign relative to malware used in previous campaigns because it will prove useful to further tie Kinsing to additional exploitation activity.

First, in observing the TTPs involved for the SaltStack campaign, we can determine the affected SaltStack services were exploited to download and execute scripts that are similar to the function and structure of previous Kinsing campaigns. The actor shows the tendency to use single-letter names on their shell scripts, move laterally via SSH, manipulate cron jobs for persistence, and perform `md5sum` hash checking to verify malware integrity before execution.

In the execution of the malware, two binaries showed up: a Golang-based RAT and a XMRIG miner. This matches previous Kinsing campaigns.

Matching Monero wallet address

One of the strongest ties between the SaltStack campaign and the previous H2Miner/Kinsing campaigns is the presence of a shared Monero wallet address:

```
46V5WXwS3gXfsgR7fgXeGP4KAXtQTXJfkicBoRSHXwGbhVzj1JXZRJRhbMrvhxvXvgbJuyV3GGWzD6JvVMuQwAXxLZmTWkb
```

This wallet address appeared in miner samples from the Docker host, Redis, and SaltStack compromises. As in previous samples, the ones from the SaltStack campaign contained an embedded JSON-formatted configuration file for XMRIG.

This does not necessarily indicate that all the campaigns were performed by the same actor, but it does suggest that the funds from all the campaigns made a stop in the same account/wallet before distribution. After entering this wallet, the funds could funnel through a laundering operation or the operators could convert it to different currencies for distribution.

The play's the thing: Hamlet in the binary strings

Another piece of evidence tying the SaltStack campaign to previous H2Miner/Kinsing campaigns was the presence of the text of *Hamlet* by William Shakespeare, in its entirety, in the RAT's binary strings. This additional text bloated the RAT binary to 15MB in size and provided some entertainment during analysis of the malware. In all reality, the padding didn't prove to slow down analysis greatly. Its inclusion could reflect the adversary's desire to increase the malware's file size, as antivirus configurations sometimes exclude files above a specified size.

Embedded "masscan" script in binary

Another piece of familiar evidence appeared in the `salt-store` Golang RAT binary from the SaltStack campaign: an embedded shell script that incorporated `masscan` functionality. As with the previous campaigns, the embedded shell script deployed dependencies, referred to the `masscan` binary as "firewire," and called `masscan` with the exact same command-line arguments.

Linking Kinsing to Citrix ADC

Golang RAT + Monero miner

In January 2020, the year started off with several malware families conducting campaigns against Citrix Application Delivery Controller (ADC) devices via CVE-2019-19781. While Kinsing and H2Miner have not been formally tied to an ADC campaign, we've found enough evidence in both public research and our own analysis to assess with high confidence that Kinsing malware was used in the ADC campaign.

During one campaign, a Golang RAT and Monero miner component appeared in Citrix ADCs. An awesome blog post from IronNet discusses these findings in detail. First, IronNet dove into analysis of a malicious `nspps`

Golang binary masquerading as a legitimate ADC process named `nsppe`. This malware contained capabilities to encrypt C2 traffic, use SOCKS proxies, execute commands, run a XMRIG miner, and run `masscan`.

When analyzing the C2 protocol used by [nspps](), IronNet noted these network POST requests:

- `/h`
- `/get`
- `/getT`
- `/l`
- `/o`
- `/r`
- `/s`

The article includes a detailed summary of these calls, which when compared to C2 calls documented by Trend Micro show evidence that `nspps` and Kinsing malware both use similar, if not the exact same, C2 protocols.

Analysis also determined `nspps` contained incredibly similar tasking capabilities:

- `backconnect`
- `download_and_exec`
- `exec`
- `masscan`
- `redis_brute`
- `scan`
- `socks`
- `update`

In our own analysis, Red Canary found that numerous binary strings were shared across the `nspps` binary and previous Kinsing malware samples. The entirety of *Hamlet* did not appear in this binary, but we did find other similarities.

In the Citrix ADC campaign, `nspps` deployed a XMRIG miner named `netscalerd`. This binary was named to again masquerade as a system process as ADCs were once named Citrix Netscaler devices. As with former Kinsing campaigns, numerous strings within the binary showed an embedded `config.json` configuration file and the expected usage instructions for an XMRIG binary.

```
 "background": false,
"donate-level": 0,
"cpu": true,
"colors": false,
"opencl": false,
"pools": [
"coin": "monero",
"algo": null,
"url": "xmr-eu1.nanopool.org",
"user": "46V5WXwS3gXfsgR7fgXeGP4KAXtQTXJfkicBoRSHXwGbhVzj1JXZRJRhbMrvhxvXvgbJuyV3GGWzD6JvVMuQwAXxLZmTWkb",
```

```
"pass": "mine",
"tls": false,
"keepalive": true,
"nicehash": false

-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
-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)
-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 5%% (5 minutes in 100 minutes)
--donate-over-proxy=N control donate over xmrig-proxy feature
--no-cpu disable CPU mining backend
-t, --threads=N number of CPU threads
-v, --av=N algorithm variation, 0 auto select
--cpu-affinity set process affinity to CPU core(s), mask 0x3 for cores 0 and 1
--cpu-priority set process priority (0 idle, 2 normal to 5 highest)
--cpu-max-threads-hint=N maximum CPU threads count (in percentage) hint for autoconfig
--cpu-memory-pool=N number of 2 MB pages for persistent memory pool, -1 (auto), 0 (disable)
--cpu-no-yield prefer maximum hashrate rather than system response/stability
--no-huge-pages disable huge pages support
--asm=ASM ASM optimizations, possible values: auto, none, intel, ryzen, bulldozer
--randomx-init=N threads count to initialize RandomX dataset
--randomx-no-numa disable NUMA support for RandomX
--randomx-mode=MODE RandomX mode: auto, fast, light
--randomx-1gb-pages use 1GB hugepages for dataset (Linux only)
--randomx-wrmsr=N write custom value (0-15) to Intel MSR register 0x1a4 or disable MSR mod (-1)
--randomx-no-rdmsr disable reverting initial MSR values on exit
-S, --syslog use system log for output messages
-l, --log-file=FILE log all output to a file
--print-time=N print hashrate report every N seconds
--no-color disable colored output
--verbose verbose output
-c, --config=FILE load a JSON-format configuration file
-B, --background run the miner in the background
-V, --version output version information and exit
-h, --help display this help and exit
--dry-run test configuration and exit
XMRig 5.5.0
```

```
built on Jan 11 2020 with GCC
Usage: xmrig [OPTIONS]
```

Matching Monero wallet address

While examining the XMRIG embedded `config.json` file, you probably spotted what we did: the Monero wallet address `46V5WXwS3gXfsgR7fgXeGP4KAXtQTXJfkicBoRSHXwGbhVzj1JXZRJRhbMrvhxvXvgbJuyV3GGWzD6JvVMuQwAXxLZmTWkb` appeared in the Citrix ADC miner sample. This wallet address also featured prominently in the other Kinsing campaigns. While this isn't definitive evidence for attribution, it proved to be an excellent way to tie the campaigns together since searching VirusTotal Intelligence for this wallet address helped us map out the related samples.

Embedded "masscan" script in binary

As with other Kinsing samples, `nspps` contained an embedded script to deploy dependencies and run `masscan`. In this script, the binary was named "firewire" and also used the same command-line arguments found in other samples.

Linking Kinsing to F5 BIG-IP exploitation

In June 2020, F5 issued guidance concerning a vulnerability discovered within their BIG-IP network appliances. CVE-2020-5902 for these devices allows remote code execution without authentication and has subsequently allowed adversaries to exploit F5 systems for an initial foothold into networks. While monitoring a VirusTotal LiveHunt YARA rule for Kinsing, we found a new version uploaded under the name "bigip." As of this writing, there is no public documentation around Kinsing/H2Miner exploiting F5 appliances but this binary name could be an indication. This binary matched the previously mentioned function names for Kinsing malware and also contained the complete text of Hamlet. In addition, around the same time another XMRIG sample with the Kinsing Monero wallet address appeared on VirusTotal under the name "bigipdaemon." This matches the same pattern we observed before: Golang RAT, then XMRIG miner.

Attribution implications and complications

The Monero wallet connection

The Monero wallet address shared across all samples mentioned is what originally drew our attention to further analyze Kinsing campaigns. It was our first piece of evidence but it wasn't strong enough to stand on its own. The largest implication of this shared wallet address is that the campaigns may be related to the same actor or group. While we can't assert that the same exact people used Kinsing during the campaigns, we feel there is enough evidence to indicate that the Kinsing RAT has been used across additional campaigns than previously documented.

Threat intelligence should be always questioned, so let's take a moment to examine this assumption: should a shared wallet address indicate the same actor? What other possible explanations exist? Cryptocurrencies have enabled the rapid movement of funds via the internet, and Monero in particular has introduced privacy features

that complicate the money trail. We know from the miner configurations that funds went into this wallet, but it remains an opaque structure to us and we cannot know where the funds go afterward. In the physical world, this would be the equivalent to dropping money into an anonymous bank account. We generally assume that the account belongs to someone, but there's no way to know if the account belongs to a person, group, or a proxy holding the funds in escrow for later distribution.

In the ideal case, the same actors across these campaigns would be the owner of the account. In the worst case, the account would be the first stop in a chain of transactions to hide the origin of the funds.

Conclusions about capabilities

During our analysis of capabilities and campaigns, we made one large assumption: that the H2Miner botnet operating Kinsing has exclusive access to Kinsing source code. This is the simplest explanation, as we have not found evidence suggesting the source code for Kinsing is available publicly on the internet, and it has not been used widely enough to indicate that Kinsing is a malware-as-a-service offering. Alternate explanations to challenge this assumption include the possibility that Kinsing code is shared across multiple actors or groups, but this possibility would also need to take into account shared access for the Monero wallet address. Any shared code access would also require access to the funds gained.

The first and most obvious conclusion we can draw about capabilities is the fact that one or more actors have used Kinsing across Linux and FreeBSD platforms. Previous to this assertion there was no evidence that Kinsing or its operator botnet H2Miner had targeted FreeBSD devices. This capability suggests that the Kinsing operators are familiar with the ability of Golang to cross-compile for multiple platforms and that they are either familiar with the operating system components of enterprise network devices or can quickly learn them. This is an important conclusion to draw in 2020 and beyond because of the increasing number of remote code execution vulnerabilities for enterprise network appliances, including those for Palo Alto and F5 Big-IP devices. In addition, Kinsing operators are familiar with the ability to rapidly weaponize emerging remote code execution exploits for opportunistic use.

It's important to note that these threats are not going away and will seek opportunities to exploit any devices connected to the public internet. The safest way to mitigate these threats is to patch and segment your network, protecting the management interfaces of network devices from unauthorized access.

Related Articles

Subscribe to our blog

You'll receive a weekly email with our new blog posts.
