SafePay: The new kid on the block

By DCSO CyTec Blog

Published: 2025-05-27 · Archived: 2026-04-06 01:00:05 UTC

12 min read

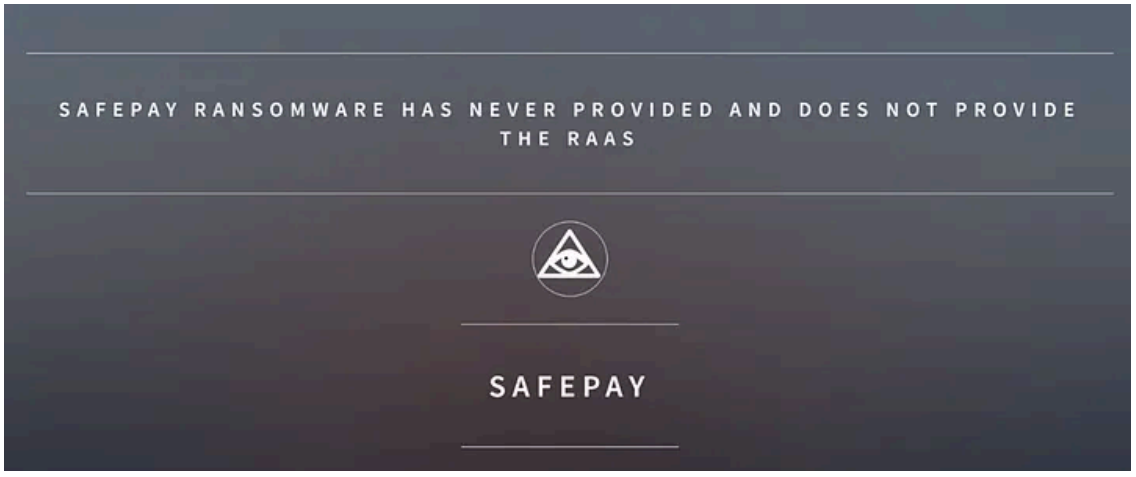
May 27, 2025

Earlier this year, DCSO was made aware of a security incident at one of our clients that was part of a **ransomware campaign by the actor SafePay**. While there is limited reporting on the group available, **DCSO uncovered additional new information on the ransomware variant in the incident response case.**

Executive summary:

- SafePay focusing on Germany and US, utilizing the double-extortion scheme (data theft and encryption)
- The ransomware does not shy away from contacting victims directly (e.g. via phone calls) to increase pressure
- SafePay ransomware shares similarities with other ransomware strains, however, we believe that this is due to inspiration not because of same origin
- We provide tooling for configuration extraction, aiding the analysis of the ransomware characteristics

Press enter or click to view image in full size

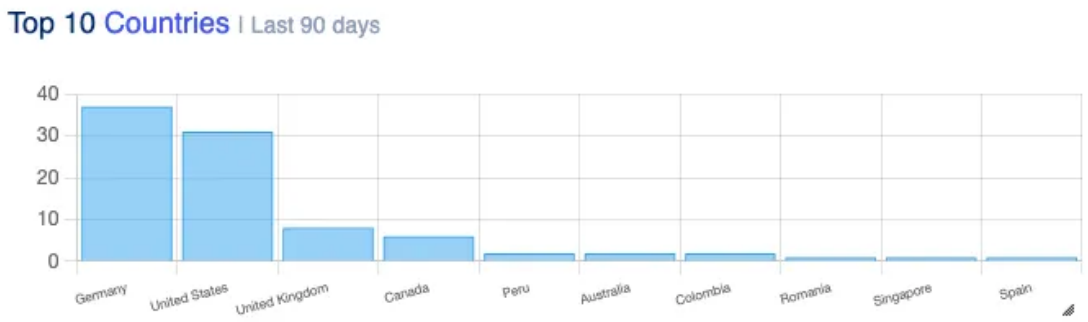


SafePay Leak Site

Blog post authored by Johann Aydinbas, Bennet Conrads, Moaath Oudeh and Denis Szadkowski

Background

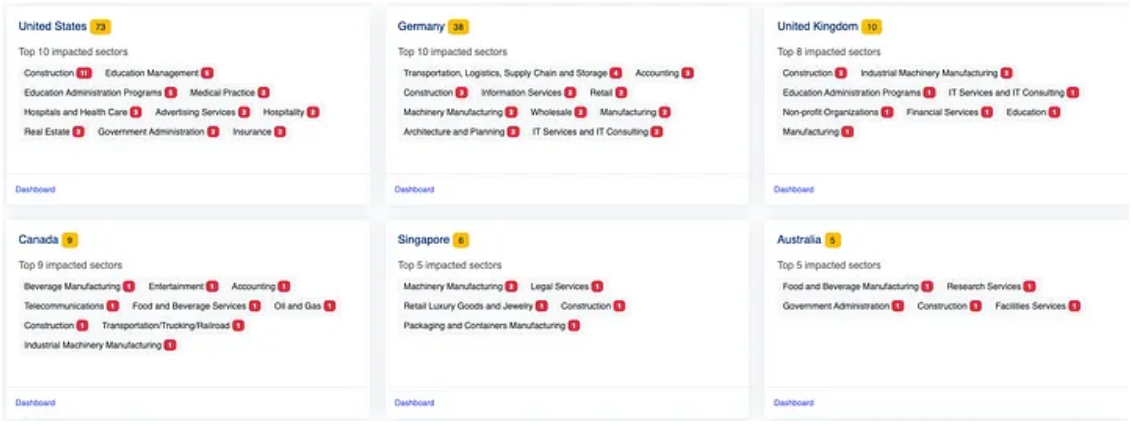
The SafePay ransomware group is a relatively new group, first appearing on our radar in November 2024. During that period, the group [was first reported on by researchers at Huntress](). The group follows a double-extortion scheme, both exfiltrating data and encrypting it on victim machines using their own SafePay ransomware.



Recent focus of SafePay group on Germany and the US — Source: ecrime.ch

At the time of writing, SafePay lists 169 victims on their leak site, with the targets predominantly based in Central Europe and North America, and a low number of targets located in Asia. The main focus of the group appears to be Germany and the United States, with **Germany having seen multiple batches of victims** listed recently and **currently making up almost 18% of the victims**, taking up the spot of most targeted in recent additions. The leak site has been updated in early May 2025 with the latest data for download having been posted on April 24th, showing fairly recent activity of the group.

Press enter or click to view image in full size



Victim country distribution of SafePay group — Source: ecrime.ch

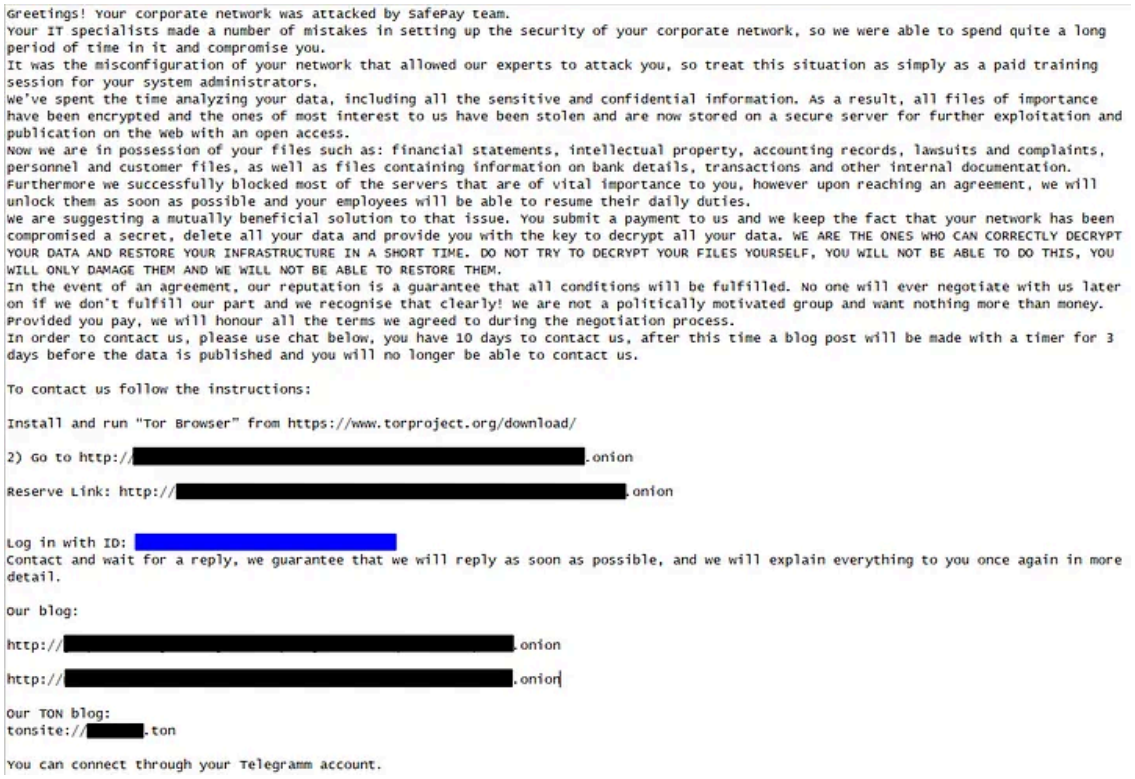
Their leak site features a headline stating that "*SafePay Ransomware has never and does not provide the RaaS*". RaaS (ransomware as a service) systems usually delegate some tasks or compromises to partners of affiliates either for a monthly subscription or one-time fee to use the ransomware. Other variants of the RaaS revenue model include affiliate programs in which affiliates who compromise networks share profits from the extortion with the developers of the ransomware.

SafePay ransomware

To encrypt victim data, SafePay employs a custom ransomware strain of the same name. Encrypted files exhibit the characteristic ".safepay" extension while the ransom note is named "readme_safepay.txt" correspondingly.

Compared to earlier reporting, the SafePay ransomware introduced a victim ID which is provided in their updated ransom note and used to log in to their portal to initiate contact.

Press enter or click to view image in full size



```
Greetings! Your corporate network was attacked by SafePay team.
Your IT specialists made a number of mistakes in setting up the security of your corporate network, so we were able to spend quite a long
period of time in it and compromise you.
It was the misconfiguration of your network that allowed our experts to attack you, so treat this situation as simply as a paid training
session for your system administrators.
We've spent the time analyzing your data, including all the sensitive and confidential information. As a result, all files of importance
have been encrypted and the ones of most interest to us have been stolen and are now stored on a secure server for further exploitation and
publication on the web with an open access.
Now we are in possession of your files such as: financial statements, intellectual property, accounting records, lawsuits and complaints,
personnel and customer files, as well as files containing information on bank details, transactions and other internal documentation.
Furthermore we successfully blocked most of the servers that are of vital importance to you, however upon reaching an agreement, we will
unlock them as soon as possible and your employees will be able to resume their daily duties.
We are suggesting a mutually beneficial solution to that issue. You submit a payment to us and we keep the fact that your network has been
compromised a secret, delete all your data and provide you with the key to decrypt all your data. WE ARE THE ONES WHO CAN CORRECTLY DECRYPT
YOUR DATA AND RESTORE YOUR INFRASTRUCTURE IN A SHORT TIME. DO NOT TRY TO DECRYPT YOUR FILES YOURSELF, YOU WILL NOT BE ABLE TO DO THIS, YOU
WILL ONLY DAMAGE THEM AND WE WILL NOT BE ABLE TO RESTORE THEM.
In the event of an agreement, our reputation is a guarantee that all conditions will be fulfilled. No one will ever negotiate with us later
on if we don't fulfill our part and we recognise that clearly! We are not a politically motivated group and want nothing more than money.
Provided you pay, we will honour all the terms we agreed to during the negotiation process.
In order to contact us, please use chat below, you have 10 days to contact us, after this time a blog post will be made with a timer for 3
days before the data is published and you will no longer be able to contact us.

To contact us follow the instructions:

Install and run "Tor Browser" from https://www.torproject.org/download/

2) Go to http://[redacted].onion

Reserve Link: http://[redacted].onion


Log in with ID: [redacted]
Contact and wait for a reply, we guarantee that we will reply as soon as possible, and we will explain everything to you once again in more
detail.

Our blog:

http://[redacted].onion

http://[redacted].onion

Our TON blog:
tonsite://[redacted].ton

You can connect through your Telegramm account.
```

Updated ransom note with victim ID in blue

The ransom note appears to be original and not taken or copied from other ransomware incidents as it features unique wording. An interesting statement in the note is the group stating that they "[…] are not a politically motivated group and want nothing more than money". Groups, e.g. LockBit, have insisted on being apolitical, while other ransomware groups have explicitly announced political objectives or official support of government goals as was the case with the Conti ransomware group announcing their "full support" for the Russian government in 2022.

A detailed analysis of the SafePay ransomware follows below.

Field Observations

During the IR engagement earlier this year, DCSO's Incident Response Team (D.I.R.T.) observed several notable findings, related to SafePay ransomware TTPs:

- The SafePay ransomware group tried to increase pressure on the affected client by making direct phone calls after encrypting the environment, aiming to coerce a faster response or payment.
- There was a 25-day gap between the initial access — achieved through password spraying against the VPN gateway — and the first discovery activities, which may suggest that the ransomware group was either preoccupied with other operations, operating with limited resources or relying on initial access brokers.

- The remote workstation “WIN-3IUUOFVTQAR” used by the attackers to access the environment matches the one identified in the intrusion investigated by Huntress, which indicates bad operational security (opsec) practices by the attackers.
- The collection activities were conducted in a targeted manner, focusing on critical business data; the attackers used SharpShares to identify accessible file shares, employed WinRAR to compress the collected data, and successfully exfiltrated 450 GB of data by unknown means.
- The attackers actively searched for backup solutions in the affected environment and encrypted them, in addition to deleting Volume Shadow Copies (VSC), all in an effort to inhibit recovery activities and maximize the impact of the attack.
- All encryption activities happened within virtual machines (VMs), although the attackers were in the possession of the necessary privileges to perform encryption of the VMs on the hypervisor level. This could indicate that the SafePay ransomware group at the time of writing was not in the possession of a ransomware variant compatible with hypervisors such as VMware ESXi.
- It took the SafePay ransomware group 26 days from initial access to obtain Domain Admin privileges, with minimal observable activity during the first 25 days post-compromise; once Domain Admin was achieved, they completed data collection, exfiltration, and encryption within just two days.

All in all, D.I.R.T. rates the overall level of sophistication of the SafePay ransomware group as intermediate, given the extended periods of inactivity post-compromise, reliance on publicly available tools, lack of advanced evasion techniques, bad opsec practices, and the apparent unavailability of a VMware ESXi ransomware variant to encrypt VMs directly from the hypervisor.

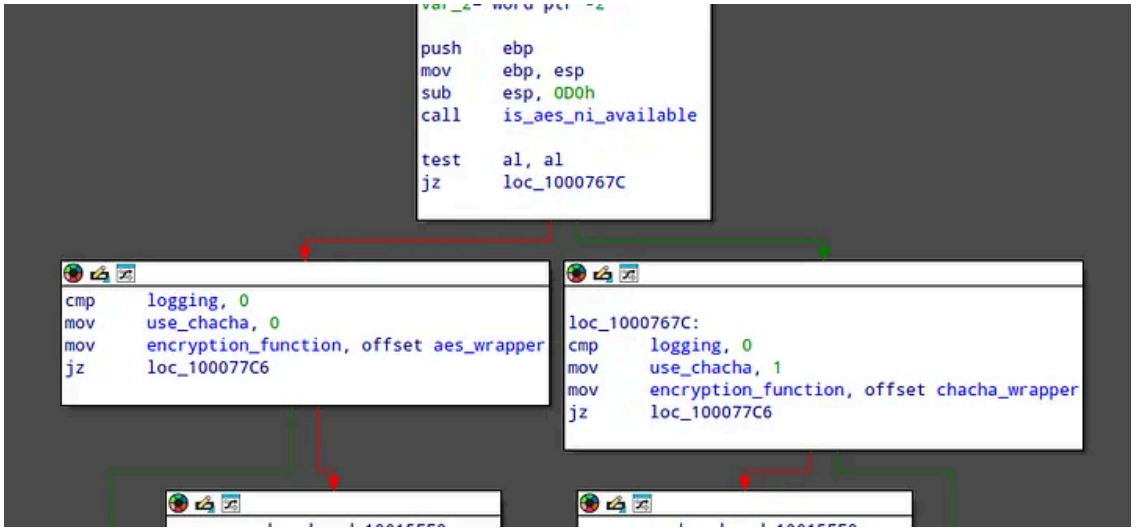
In-Depth Analysis of SafePay Ransomware

The SafePay ransomware is written in C and built around Overlapped I/O, which is the Windows solution for asynchronous I/O. Files are enumerated and encrypted using multiple separate threads, the specific number depending on the processor count.

Successful execution of the ransomware requires passing the victim ID as key phrase via command line argument `-pass`. It is used to decrypt the internal configuration data, such as what processes to kill or directories to skip. We have written a config extraction tool which is described below in more detail.

File encryption uses a symmetric cipher to encrypt an alternating stream of 1MB chunks, while Elliptic Curves are used to encrypt the key material. The exact number of chunks encrypted/skipped is controlled by specifying an encryption level via the `-enc <1-10>` command line switch, providing a way to perform so-called *partial encryption* of files for the purpose of speed.

Press enter or click to view image in full size



SafePay choosing the symmetric cipher at runtime

The choice of symmetric cipher depends on the availability of AES-NI — if the CPU supports AES instructions, SafePay will pick AES-CBC, otherwise ChaCha20 is used to encrypt files. This appears to be an upgrade compared to previous reporting by Huntress. In addition, we noticed SafePay no longer has a Cyrillic language killswitch as documented by Huntress.

SafePay encrypts every file using a separate key which is generated using `RtlGenRandom` (aka `SystemFunction036`). An 80 byte meta data blob is attached to each encrypted file with the following format:

```
[8 byte file size]
[32 byte ECC master public key]
[32 byte file ECC public key]
[1 byte encryption level]
[1 byte is ChaCha used?]
[6 byte unused]
```

For research purposes and testing we have reimplemented the decryption algorithm for a modified ECC master key. You can find the Python script on our GitHub.

Comparison to other ransomware

The SafePay ransomware has similarities to a multitude of other ransomware families. Notable is the architectural similarity to LockBit3 (LockBit Black), with both ransomware strains using Overlapped I/O, a very similar state machine to encrypt files, and similar import resolution structure and data structures used to manage files to encrypt.

However, based on our research, **we believe SafePay to be independently developed, but likely influenced by existing ransomware.**

During our analysis, we have extracted SafePay's cryptographic primitives and used the most excellent ransomware tooling repository by rivitna to compare it to other ransomware:

- SafePay uses SHA512 as its key derivation function. This is a rare occurrence and only shared with Hive, DarkBit and Inc. ransomware
- SafePay uses a specific CRC32 polynomial 0x4c11db7 for import resolution — the same specific polynomial is used in Proxima and Babuk, though both use a differing starting value (SafePay: 0xFF, Proxima/Babuk: 0xFFFFFFFF)
- SafePay uses MurmurHash, also used by Conti
- For ECC, SafePay uses Curve25519 which is not uncommon, same as the usage of ChaCha20 and AES-CBC
- LockBit3 (LockBit Black) uses ChaCha20 as well, but employs RSA instead of ECC

Based on this we believe SafePay might have been developed independently from any source or builder leak but likely may have taken inspiration from other ransomware strains.

Reversing Obfuscated Stack Strings with Ghidra

While analyzing the binary, we encountered a common obfuscation technique known as *stack strings*. These are strings constructed on the stack at runtime to evade static detection. We decided to try out Ghidra's recently improved Python scripting add-on (PyGhidra) to automate the decoding.

What are stack strings?

Stack strings are strings constructed at runtime by placing individual bytes directly onto the stack, rather than defining them as static string literals. This technique is often used by malware authors to evade detection and hinder static analysis.

```
uVar2 = 0;
obf_str[0] = 0x1b;
obf_str[1] = 0x1f;
obf_str[2] = 0xe;
obf_str[3] = 0x18;
obf_str[4] = 0xe;
obf_str[5] = 0x16;
obf_str[6] = 0x4f;
obf_str[7] = 0x4f;
obf_str[8] = 0x5c;
obf_str[9] = 0x17;
obf_str[10] = 0x1c;
obf_str[0xb] = 0x1d;
obf_str[0xc] = 0x76;
do {
  obf_str[uVar2] = obf_str[uVar2] ^ (byte)uVar2 ^ *kernel32dll ^ 0x37;
  uVar2 = uVar2 + 1;
} while (uVar2 < 0xd);
```

Figure 1 — Example stack string deobfuscation

Figure 1 illustrates an example of this type of string obfuscation. In this case, the string is initialized using raw byte values, which are later resolved through a while loop. The loop performs the following steps for each byte:

- XORs the byte with its index in the array.

- Then XORs the result with the first character of the kernel32.dll header (which is “M”, since Windows PE files begin with “MZ”).
- Finally, XORs the result with a constant byte specific to the string (in this case, 0x37).

After deobfuscation, the resulting string is revealed to be “advapi32.dll”, which is then stored in the *obf_str* variable.

How to find the stack strings via pattern matching

Since the binary uses XOR operations to obfuscate strings, one effective method of detection is to look for recognizable instruction patterns.

```
                    LAB_10001880                                    XREF[1]:
10001880 8a 44 0d c0     MOV        AL,byte ptr [EBP + ECX*0x1 + obf_str[1]]
10001884 32 c1           XOR        AL,CL
10001886 32 02           XOR        AL,byte ptr [EDX]
10001888 34 37           XOR        AL,0x37
1000188a 88 44 0d c0     MOV        byte ptr [EBP + ECX*0x1 + obf_str[1]],AL
```

Figure 2 —

In Ghidra, you can search for specific byte patterns in the disassembly. Wildcard bytes can be represented using *??*, allowing for more flexible matching. For example, the pattern *32 ?? 32 ?? 34 ??* proved useful in identifying the XOR-based obfuscation sequences across the binary.

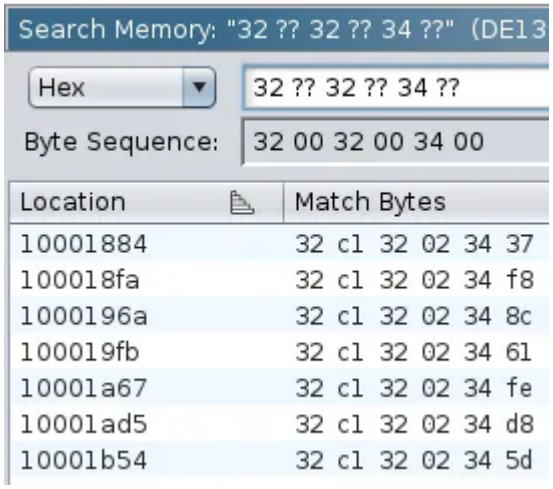


Figure 3 —

By reviewing the results, we noticed a recurring instruction pattern surrounding the obfuscated strings. Typically, these sequences began with:

```
MOV EDX, dword ptr [0x10015ff8]
```

And ended with a conditional jump instruction, such as:

```
JC <VALUE>
```

Upon further analysis, we discovered two additional variants of the starting instruction:

```
MOV EAX, [0x10015ff8]
MOV ECX, dword ptr [0x10015ff8]
```

Automating the task

While analyzing the binary, it quickly became evident that there were over a hundred obfuscated stack strings. Manually resolving each one would be extremely time-consuming and inefficient, so we decided to automate it.

Since our analysis was being conducted in Ghidra, we leveraged a relatively new feature called **PyGhidra**. Unlike earlier versions that only supported Java or Jython, PyGhidra allows you to write and run Ghidra scripts using standard Python — making scripting more accessible and flexible.

On a high level, the script works like this:

- Traverse the binary and identify all stack strings based on known instruction patterns.
- Simulate each snippet and resolve the obfuscated strings
- Save the decoded results for further analysis

To locate stack strings using above identified patterns, we used Ghidra's built-in APIs. For emulating and executing the snippets, we used **Unicorn**, a lightweight, multi-platform CPU emulator based on QEMU. It allowed us to simulate execution of individual instructions in isolation without running the actual binary.

Finally, the script was executed using Ghidra's **headless mode** via PyGhidra.

Figure 4 — Example output

Results

The script successfully identified **117 unique stack strings** and resolved **116 of them**. One string failed during emulation due to a more complex instruction sequence that Unicorn couldn't handle out-of-the-box.

Handling emulation constraints

Since Unicorn doesn't have access to Ghidra's memory layout, it couldn't resolve the address 0x10015FF8 (which points to the kernel32.dll during runtime) during emulation. To work around this, we patched the instruction to load the expected value — 'M' (the first character of the "MZ" header) — directly into the register. This allowed the emulation to continue and the string to be resolved correctly.

You can find the full script on our GitHub repository, including instructions on how to run it in headless PyGhidra and configure Unicorn.

Extracting the config:

The SafePay binary contains a built-in configuration that guides its behavior — such as which files or directories to encrypt, which system locations to avoid as well as the ransom note.

However, this configuration isn't stored in plaintext. Instead, it's encrypted within the binary, making it invisible during inspection.

After analyzing the binary's decryption logic, we were able to reverse-engineer how the config is retrieved at runtime. Using this understanding, we developed a script that **extracts and decrypts the config directly from the binary**, without needing to execute it.

The script relies on **two inputs**:

1. **The pass key (aka victim ID):**
   This key can be found in the ransom note and is used by the victim to login into the portal. It is also specified as the -pass parameter, which is provided to the malware at runtime by the attacker.
2. **The corresponding binary:**
   This needs to be the binary that generated the ransom note.

The script locates the encrypted blob, applies the correct decryption algorithm, and outputs the configuration components in a readable format.

Layout of the config data:

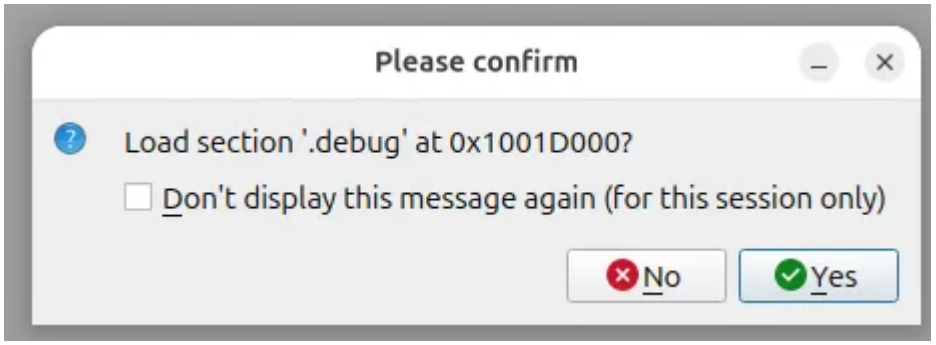
The encrypted configuration is stored in its own dedicated section within the binary. In our sample, this section was named **.debug**

The layout of the section is as follows:

- **First 4 bytes:** A **MurmurHash** checksum. This hash is used to verify if the data was decrypted successfully.
- **Next 4 bytes:** A **length field** indicating the size of the encrypted config payload. This length usually matches the number of remaining bytes in the section.

- **Remaining bytes:** The **encrypted config data** itself.

***Note to IDA users**: Likely due to the name .debug, IDA doesn't load the section into memory by default which can lead to confusion analyzing the binary as the entire section will be missing and memory references point to non-existing data. This can be fixed by checking "manual load" when first loading the binary, followed by confirming loading of the .debug section:*



Manual load popup for the .debug section

Decrypting the config

The malware uses a combination of cryptographic and hashing algorithms to decrypt the data. These include:

- **SHA-512** — to derive the encryption key and nonce
- **ChaCha20** — the stream cipher used to encrypt/decrypt the data
- **MurmurHash** — to validate the integrity of the decrypted config

To decrypt the config, the malware follows a specific sequence:

- **Hash the pass key** using SHA-512.
- **Take the first 56 bytes** of the SHA-512 hash:
- The **first 32 bytes** are used as the **ChaCha20 key**.
- The **remaining 24 bytes** serve as the **nonce**.
- **Initialize the ChaCha20 cipher** with the key and nonce.
- **Decrypt the payload** using the cipher.
- **Verify the result** by comparing its **MurmurHash** (seed: 0xFFFFFFFF, result must be unsigned) to the hash stored at the beginning of the config section.

What's inside the config?

Once decrypted, the configuration reveals various parameters used by the ransomware to control its behavior:

- **Default mutex**
- **Ransom file extension**
- **Ignored extensions**
- **Ignored files**
- **Ignored directories**
- **Processes to terminate**

- **Services to terminate**
- **The ransom note**

You can find the config decryptor on our GitHub repository, including instructions on how to run it.

Conclusion

SafePay is a relatively new ransomware actor that appeared around October 2024. They use a custom ransomware that we believe is independently developed but possibly influenced by other existing ransomware families. Recently, they seem to focus their attacks on Germany and the United States.

We analyzed the ransomware in detail, extracted the cryptographic primitives in use and wrote tooling that aid in understanding the ransomware, which we want to share with the community.
