vb 2025 BERLIN

24 - 26 September, 2025 / Berlin, Germany

DEMYSTIFYING THE PLAYBOY RAAS

Gijs Rijnders

Dutch National Police, The Netherlands

gijs.rijnders@politie.nl

ABSTRACT

The Dutch National Police was able to seize infrastructure of the Playboy ransomware operation hosted in the Netherlands after receiving a tip from an intelligence source. The Cyber Threat Intelligence team, responsible for tracking and analysing threats against the police organization, reverse engineered and analysed the toolchain containing the builder, encryptors and decryptors.

The Playboy toolchain consists of a key generator to generate a new keypair for an affiliate, as well as encryptors and decryptors for various operating systems and CPU architectures. The variety of support for different environments suggests that Playboy is highly likely not created by a single developer. This paper provides a detailed analysis of these tools, explains what logic they implement, and how they are related.

Playboy is at least partially based on the leaked Babuk ransomware source code. This conclusion was reached by the Cyber Threat Intelligence team after writing Yara rules for patterns in the Playboy malware samples, and performing retrohunts on *VirusTotal*. Since the leaked Babuk code has been used by many cybercriminals, it is difficult to pinpoint who might be behind Playboy.

INTRODUCTION

Ransomware has been one of the most prolific forms of cybercrime in recent years. The problem keeps growing, with a new operation seeing the light almost every month. Ransomware attacks can inflict substantial damage, and individuals and organizations in the Netherlands become victims too. Moreover, ransomware groups pose a cyber threat to the Dutch National Police.

The Cyber Threat Intelligence team at the Dutch National Police, from here on CTI, is responsible for investigating cyber threats that target the police organization. Their primary objective is protecting law enforcement information and infrastructure, together with the Security Operations Centre (SOC). CTI provides threat intelligence to the SOC and helps them strengthen their defences. Furthermore, CTI has the capability to analyse complex malware when it poses a threat to the police organization or Dutch society. CTI also keeps a close eye on ransomware group activities and tracks new victims every day. The estimated number of victims recorded worldwide per month are depicted in Figure 1.

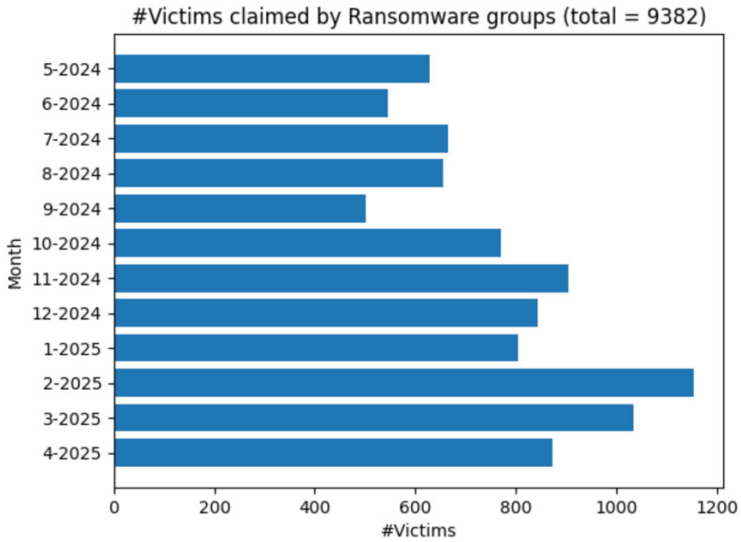

Figure 1: Estimated number of ransomware victims per month worldwide in the last year.

The numbers shown represent victims claimed by ransomware groups. Hundreds of them around the world are claimed by cybercriminals every month. Even though the tracking of victims is done by an automated system that can return some false positives, it shows that ransomware is a very prevalent form of cybercrime. Considering the number of victims and the severity of the damage, the Dutch National Police treats this form of cybercrime as a serious issue. It combats ransomware through disruptive infrastructure takedowns and the No More Ransom initiative, which aims to help victims recover their encrypted files for free [1].

In January 2025, CTI received intelligence that a Virtual Private Server (VPS) destined for cybercrime was being hosted in the Netherlands. The cybercrime investigation teams collaborated with the hosting provider with the aim of taking control of the infrastructure and seizing the data. Once seized, sophisticated malware was found on the VPS. The malware turned out to be a toolchain for building ransomware encryptors and decryptors, belonging to a new RaaS (ransomware-as-a-service) operation called Playboy.

Playboy gives us a good example of the tools and workflows ransomware operators use behind the curtains to facilitate their cybercriminal operations. This paper sheds light on the Playboy RaaS from the perspective of the seized software toolchain. It explains how such a ransomware operation works, and dives into the technical details of the encryptor.

RANSOMWARE-AS-A-SERVICE

Ransomware has evolved a lot over the years to be more effective and scalable, and as a result, almost every group today goes by the ransomware-as-a-service, or RaaS model. In this model, the ransomware group operates a platform that provides the ransomware encryptor and decryptor to other cybercriminals, who are the ones that actually choose and breach their victims. These so-called 'affiliates' pay a share of the ransom payout to the ransomware group for their services. RaaS is essentially a business model where cybercriminals that lack programming skills can get their ransomware operation up and running affordably. Examples of famous ransomware groups that have operated the RaaS model are LockBit, Hive and DarkSide [2].

RaaS groups exist in various forms and complexities. Some have a leak page where victims are named and shamed if they refuse to pay the ransom. Others also provide a portal where affiliates can log in and manage aspects of victims, payments, and even build ransomware samples for new victims. Ransomware-as-a-service is a multi-billion-dollar business, and hence an effective form of cybercrime.

The Playboy toolchain contains all the tools necessary to facilitate affiliates in their ransomware attacks. The operator can build encryptors and decryptors for affiliates with their own encryption keys, and therefore, Playboy fits well in the RaaS model.

TOOLCHAIN

The seized VPS contained a directory called 'Software' with several executables, a batch file and a configuration file. This directory contained the following files.

- **Key generator**: a command-line tool called 'keygen.exe' that generates a new pair of private and public elliptic curve keys and stores them as 'priv.bin' and 'pub.bin'.
- **Key replacer tool**: a command-line tool called 'replace.exe' that reads 'priv.bin' and 'pub.bin' from disk and places them in the ransomware executables. The private key will be placed in the decryptor, and the public key in the encryptor.
- **Build script**: a *Windows* batch script called 'build.bat' that executes the key generator and key replacer tool in sequence.
- **Encryptors**: a series of precompiled ransomware executables that perform the encryption on various architectures. Playboy supports *Win32* and *VMware ESXi* operating systems, as well as NAS devices running *Linux* with an x86-64 or ARM architecture.
- **Decryptors**: a series of precompiled executables that perform decryption on the same operating systems and architectures as the encryptors.
- **Config file**: a configuration file in JSON format called 'config.json' that contains parameters for the encryptor.
- **Builder tool**: a command-line tool that takes the parameters specified in 'config.json' and configures them in the encryptor executables.

The workflow for creating a new ransomware encryptor and decryptor for an affiliate is depicted in Figure 2. The ransomware operator first runs 'build.bat', which in turn executes the key generator and replacer tool. The encryptors and decryptors for all available architectures now contain the newly generated keys. The operator can now change the parameters in 'config.json' to their liking, and execute the builder tool. The ransomware is ready to use after this step.

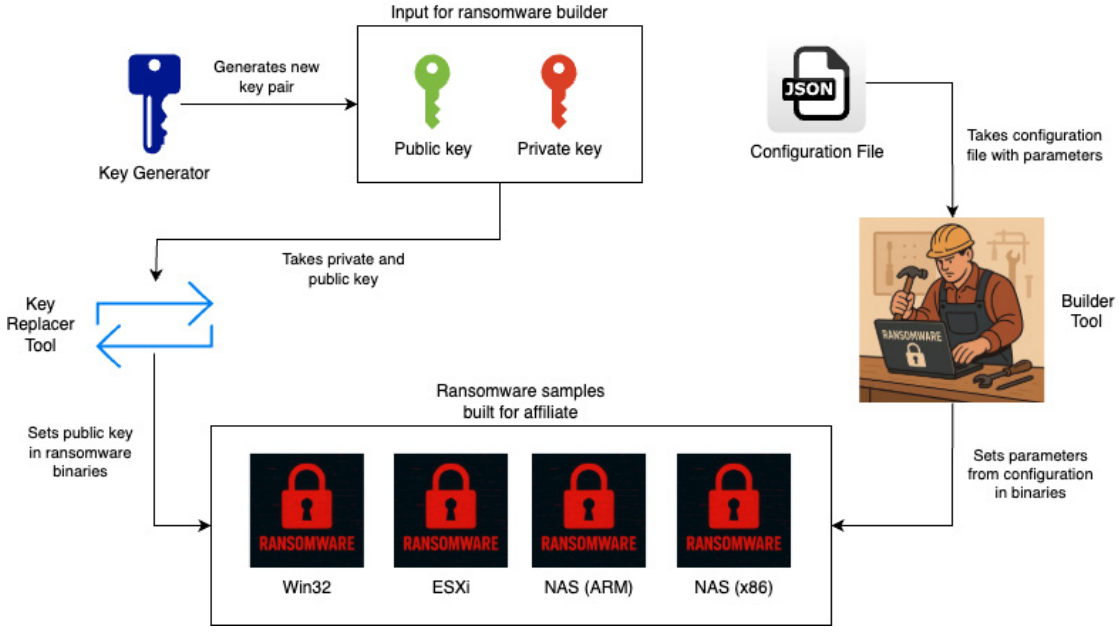

Figure 2: The software components in the Playboy toolchain and how they are related.

The configuration file contains several parameters in JSON format. For example, it allows the operator to specify the ransom note and some aspects of the behaviour of the encryptor. Table 1 describes all configurable parameters.

Parameter	Default value	Description
skip_lan	True	Specifies whether the encryption of network drives should be skipped.
skip_local	False	Specifies whether local drive encryption should be skipped.
note_content	Listed in Figure 3	Contents of the ransom note as presented to the victim by the encryptor.
change_desktop _wallpaper	True	Specifies whether the desktop wallpaper should be changed after encryption. The wallpaper is rendered from text in the code indicating that the victim's files are encrypted.
self_delete	True	Specifies whether encryptor should be automatically deleted on next reboot.
restart_system	False	Specifies whether infected system should be rebooted automatically after encryption.
wipe_free_space	False	Specifies whether the encryptor should attempt to wipe free space on local drives.
running_one	True	Specifies whether a mutex should be created and tested to ensure only one instance of the encryptor can run at a given time.
pass_protect	False	Enables one to specify a password on the command line for the encryptor. This password must match what is configured by the operator during build time.

Table 1: Detailed description of configurable parameters for the encryptor.

```
PlayBoy LOCKER\r\nHi!\r\nYour files have been stolen and encrypted. We are ready to
publish your stolen data on our blog\r\nYou can buy our decrypt service, to decrypt
your files and avoid data leakage.\r\nWe are waiting for you here!
```

Figure 3: Ransom note contents as configured in 'config.json'.

The parameters from 'config.json' are parsed by the builder and stored in the encryptor at prepared placeholders in the resources. These resources are referenced as RC_DATA with hard-coded identifiers 101 through 111, allowing for easy config extraction from encryptor samples.

ENCRYPTOR

The Playboy RaaS provides encryptors for various operating systems and architectures. The *Win32* variant is written in C++ and compiled with heavy optimizations. Upon startup, the encryptor performs some basic debugger presence checks, and parses specified command-line switches. For example, it allows a password to be specified to protect execution of the encryptor. Furthermore, the attacker can specify a path to narrow down the encryption process, and a username and password to probe network drives. Finally, the encryptor supports a debug mode, which can be enabled using the '-debug' switch. Once enabled, it outputs verbose log messages about its behaviour.

One of the most important steps ransomware encryptors must take is to create a mutex and check for its existence. This ensures only one instance of the encryptor can run at a given point in time. The name of the mutex can be used to detect the presence of the Playboy encryptor on the victim's system, and its value is: 'Global\\EncryptorSingleInstance'. Multiple running instances could interfere with each other's operations and result in irrecoverable files. While mutual exclusivity seems essential for ransomware, Playboy implemented this as optional. Albeit enabled by default, it is configurable using the 'running_one' parameter in the builder configuration file.

The next step the encryptor executes is deleting volume shadow copies. This is a simple technique commonly employed by ransomware to make sure the victim cannot use shadow copies to recover encrypted files. There are multiple ways of doing this, but Playboy uses the command listed in Figure 4 to do so.

```
cmd.exe /c vssadmin delete shadows /all /quiet
```

Figure 4: Shell command used to delete volume shadow copies on Windows.

Other tasks the encryptor executes before starting the encryption process are killing running processes that might keep sensitive files open, such as *Microsoft Office*, *SQL Server*, browsers and mail clients. If files that are targeted by the encryptor are kept open by other applications, encryption can fail. A whitelist of system files and directories is used by the encryptor to make sure the infected system is not bricked and the victim can still run the decryptor.

The NAS variant is written in Go and contains statically linked libraries for cryptography. The NAS variant is a *Linux* ELF binary file, and it contains a whitelist of files and directories to exclude from encryption. This whitelist is depicted in Figure 5, and the presence of the paths 'home/httpd' and below suggest that the operator targets QNAP devices with its NAS variant.

```
v19 = (*(int (__golang **)(int))(a3 + 24))(a4);
if ( internal_stringslite_Index(v19, v22, (int)"/proc", 5) >= 0
  || internal_stringslite_Index(a1, a2, (int)"/boot", 5) >= 0
  || internal_stringslite_Index(a1, a2, (int)"/sys", 4) >= 0
  || internal_stringslite_Index(a1, a2, (int)"/run", 4) >= 0
  || internal_stringslite_Index(a1, a2, (int)"/dev", 4) >= 0
  || internal_stringslite_Index(a1, a2, (int)"/etc", 4) >= 0
  || internal_stringslite_Index(a1, a2, (int)"/home/httpd", 11) >= 0
  || internal_stringslite_Index(a1, a2, (int)".system/thumbnail", 17) >= 0
  || internal_stringslite_Index(a1, a2, (int)".system/opt", 11) >= 0
  || internal_stringslite_Index(a1, a2, (int)".config", 7) >= 0
  || internal_stringslite_Index(a1, a2, (int)".qpkg", 5) >= 0 )
```

Figure 5: Whitelist of files and directories in NAS x86 binary indicating QNAP devices are the target.

FILE ENCRYPTION PROCESS

File encryption starts after all initialization work is complete. The *Win32* encryptor obtains a list of logical drives mounted on the victim's system using the GetLogicalDriveStrings API [3]. If a path is specified via the command-line switch, that one is used instead.

The *Win32* encryptor first removes the read-only attribute for every targeted file. The file is then renamed using the MoveFileExW API, and the extension '.PLBOY' appended. The *Linux* and *ESXi* encryptors append a different suffix: '.plboy' or '.plboyMetric'. Files are renamed beforehand, and therefore encryption is done in-place. The original file is opened with the CreateFileW API using the GENERIC_READ and GENERIC_WRITE flags. Plaintext blocks are read from the file and encrypted. The file pointer is then changed back to the start of the block so the encrypted data is written back at the correct position.

I/O completion ports

File I/O can be implemented in multiple ways on the *Win32* operating system. The simplest way is using synchronous I/O with the API functions ReadFile and WriteFile in sequence on a single thread. While this is easy to understand and manage, it does not perform very well on a large number of I/O operations. Therefore, ransomware authors have increasingly turned to asynchronous I/O using completion ports [4] in recent years. The Sodinokibi ransomware was one of the first, dating back as far as 2019 [5].

The Playboy ransomware also implements its file I/O using completion ports. A completion port is first created using the CreateIoCompletionPort API. The encryptor then creates a thread pool with the sole purpose of processing I/O requests from the encryptor. Those threads call the GetQueuedCompletionStatus API repeatedly to poll for new events, and dispatch those to worker threads accordingly. The operating system usually sends messages to a completion port when I/O operations are completed. However, you can also queue your own requests using the PostQueuedCompletionStatus API. In this case, you can specify a user-defined parameter and use the completion port as mechanism to efficiently route file encryption tasks through the newly created thread pool. This is commonly how ransomware encryptors use the completion ports, and Playboy does this as well.

The asynchronous file encryption process is depicted in Figure 6. The encryptor starts on a single thread by enumerating the contents of a root directory, such as a logical drive root. When a subdirectory is encountered, it is visited as well. When a file is not included in the whitelist, and marked for encryption, the encryptor queues a message in the completion port. A polling loop then picks up the message, removes it from the queue, and dispatches file encryption to one of the workers. By routing the encryption logic this way, threads spend a lot less time waiting for blocking I/O operations than they would when using synchronous I/O functions.

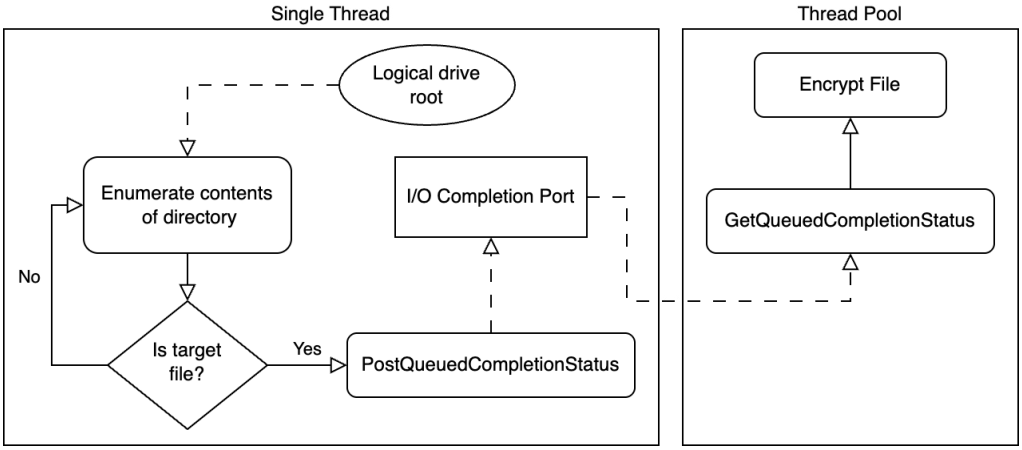

Figure 6: Process of asynchronous I/O using completion ports in Playboy encryptor.

Encryption algorithms

The *Win32* Playboy encryptor uses the Curve25519 elliptic curve and HC-128 [6] algorithms to encrypt files. Figure 7 shows the first steps of the encryption phase. A cryptographically secure secret key is generated for each file using the CryptGenRandom API. To use the secret key as a point on the Curve25519 curve, the secret key is clamped [7] directly after.

```
if ( !CryptGenRandom(hCryptProv, 0x20u, keys.FileSecretKey) )// Generates secret key here
  goto LABEL_57;
keys.FileSecretKey[0] &= 0xF8u;                 // Secret key trimming
keys.FileSecretKey[31] = keys.FileSecretKey[31] & 0x3F | 0x40;
curve25519_point_multiply(footer.FooterFileKey, keys.FileSecretKey, &base_point);
curve25519_point_multiply(&keys, keys.FileSecretKey, AttackerPublicKey);
memset(&footer.HashLength, 0, 20);
*(_OWORD *)footer.sha256_state = sha_constants_0;
*(_OWORD *)&footer.sha256_state[16] = sha_constants_1;
*(_OWORD *)&footer.sha256_state[32] = sha_constants_2;
*((_QWORD *)&v17 + 1) = *((_QWORD *)&sha_constants_3 + 1);
*(_OWORD *)&footer.sha256_state[48] = sha_constants_3;
compute_sha256(&footer, &keys, 0x20u);          // Derive encryption key from shared secret.
```

Figure 7: Elliptic curve Diffie-Hellman for file encryption in Playboy encryptor.

A new 32-byte secret key, K_f, is generated for every file the encryptor processes. Furthermore, the attacker creates a pair of secret key, K, and public key, P, for each victim. The public key, P, is embedded in the encryptor, and K in the decryptor. A per-file public key, P_f, is calculated by multiplying K_f with the Curve25519 base point, which is 9. This public key, P_f, is stored in the footer of the encrypted file. At this point, the attacker cannot yet decrypt the file using K. To do so, P must be applied to K_f. The decryptor will then compute $K * P_f = K_f$, multiplying the secret key, K, of the attacker with P_f stored in the file footer. This scheme is called Elliptic-Curve Diffie-Hellman (ECDH) [8] and is often used in ransomware as an alternative to RSA. The per-file encryption key, K, is then hashed with SHA-256 and the result is used to encrypt blocks in the file using the HC-128 symmetric encryption algorithm.

The other encryptors use Curve25519 as well, but the symmetric encryption algorithms differ. The *Linux* NAS encryptors implement ChaCha20 [9] instead of HC-128. Moreover, the *ESXi* encryptor is likely based on the Babuk source code and implements the SOSEMANUK stream cipher [10].

Encrypted file footer

It is a common practice for ransomware encryptors to append a footer to every encrypted file. The decryptor will need the secret key to decrypt the contents of the file. Furthermore, some ransomware strains calculate integrity checks like HMAC and checksums such as CRC32 to test whether the decryption was successful. Magic values or static markers are common too. These are used by the decryptor to test whether the targeted file is actually encrypted by the corresponding encryptor. Those values must be included in the encrypted file too, and the footer is a straightforward place to store them. Few ransomware strains use a header instead of a footer, because the entire file would need to be rewritten. A footer can safely be appended without changing the remainder of the file.

All Playboy encryptors were analysed, and they all implemented a different symmetric encryption algorithm. The footer also contains different things in every variant. The footer created by the *Win32* encryptor is the most extensive, with a size of 72 bytes, and its contents are described in Table 2.

Offset	Size	Description
0	32 bytes	Secret key required to decrypt the file.
32	4 bytes	CRC32 checksum of secret key written to the file footer. The *Win32* decryptor code best shows the purpose of the checksum in Figure 9.
36	4 bytes	Unknown value.
40	32 bytes	Static marker value: ‘Tis coolis diffuse very andomly!’.

Table 2: Detailed description of footer stored by Win32 encryptor.

The *ESXi* encryptor only stores the secret key for the encrypted file in the footer. The NAS encryptor appends a static marker value ‘\xAB\xBC\xCD\xDE\xEF\xF0’ after the secret key, making the footer 38 bytes in size. Static marker values are a common way for ransomware decryptors to test whether a file should be decrypted. The Playboy *Win32* decryptor also does this, as shown in Figure 8. It searches to the end of the file minus the footer size, reads from there, and tests for the marker on offset 40, and if the marker is not present, it will not touch the file.

```
SetFilePointerEx(FileW, (LARGE_INTEGER)(FileSize.QuadPart - 72), 0, FILE_BEGIN);
ReadFile(FileW, Buffer, 72u, &NumberOfBytesRead, 0);
if ( Buffer[10] != ' siT'
  || Buffer[11] != 'looc'
  || Buffer[12] != 'd si'
  || Buffer[13] != 'uffi'
  || Buffer[14] != 'v es'
  || Buffer[15] != ' yre'
  || Buffer[16] != 'odna'
  || Buffer[17] != '!ylm' )
{
```

Figure 8: Static marker check by Playboy Win32 decryptor.

The *Win32* encryptor computes the CRC32 checksum of the secret key as written to the file footer. The decryptor in turn uses this checksum to determine whether the file footer has not been corrupted. Figure 9 lists the decompiled code for the CRC32 check in the decryptor. It shows that if the checksum is not equal to the expected value, it spawns a message box with an error. An interesting observation here is that the initial CRC32 value is non-standard. Translated to ASCII it reads 'dong', which could be a joke.

```
checksum = 'gnod';
i = 64;
do
{
  c = (unsigned __int8)*ptr++;
  checksum = crc32_table[c ^ HIBYTE(checksum)] ^ (checksum << 8);
  --i;
}
while ( i );
if ( Buffer[8] != checksum )
{
  MessageBoxW(0, this, L"Key broken!", 0);
  goto LABEL_32;
}
```

Figure 9: Code that computes CRC32 of key embedded in footer and raises error if it is incorrect.

WHO IS PLAYBOY?

The Playboy toolchain contains encryptors and decryptors for *Win32*, *Linux* and *VMware ESXi*, and therefore, the operators are unlikely to be newcomers. Moreover, the different implementations in those encryptors and decryptors suggest that they might be written by different authors, or copied from another ransomware group.

To investigate more, CTI created the Yara rules listed in the Indicators of Compromise section of this paper and performed a retrohunt on *VirusTotal*, with the goal of finding related samples. With the most sophisticated implementation, the *Win32* encryptor seems to be Playboy's primary tool. Therefore, a retrohunt was performed on the NAS x86 encryptor first.

The Yara rule 'Playboy_Linux_Encryptor' looks for the marker pattern in the code. It was executed first and detected a very similar sample with SHA-256: a52c87e1e8483ad75d0fc6344828426ce071439daf49864844ce7fc18eeea32f. The filename 'e_nas_linux_amd64' also has similarities with the naming convention used by Playboy, whose NAS encryptors are named 'e_nas_x86' and 'e_nas_arm'. The similar sample found by CTI is a *Linux* NAS encryptor that has the LockBit 3.0 ransom note shown in Figure 10 embedded.

```
qmemcpy(
  data,
  "~~~ LockBit 3.0 the world's fastest ransomware since 2019~~~\n"
  "\n"
  ">>>> Your data are stolen and encrypted\n"
  "\n"
  ">>>> What guarantees that we will not deceive you? \n"
  "\n"
  "\tWe are not a politically motivated group and we do not need anything other than your money. \n"
  "    \n"
  "\tIf you pay, we will provide you the programs for decryption and we will delete your data. \n"
  "\tLife is too short to be sad. Be not sad, money, it is only paper.\n"
  "    \n"
  "\tIf we do not give you decrypters, or we do not delete your data after payment, then nobody will pay us in the "
  "future. \n"
  "\tTherefore to us our reputation is very important. We attack the companies worldwide and there is no dissatisfi"
  "ed victim after payment.\n"
  "    \n"
  "\tYou can obtain information about us on twitter https://twitter.com/hashtag/lockbit?f=live\n"
  "    \n"
  "\n"
  ">>>> you can contact us in mail or tox.\n"
  "\t\n"
  "\tTox ID LockBitSupp: CCF12BE4CF49692C4AF295EA10DE1910EDC0CA55E1359B9F66506A0496645412A16FB3D61602\n"
  "\tmail Support: xndxaxia@proton.me\n"
  "\t\n"
  ">>>> Your personal DECRYPTION ID: B7568014A48684D6D525F3F3722638C4\n"
  "\n"
  ">>>> Warning! Do not DELETE or MODIFY any files, it can lead to recovery problems!\n"
  "\n"
  ">>>> Warning! If you do not pay the ransom we will attack your company repeatedly again!\n"
```

Figure 10: Ransom note embedded in e_nas_linux_amd64, referring to LockBit 3.0 and LockBitSupp.

The retrohunt results suggest that the *Linux* encryptor might originate from the LockBit 3.0 ransomware operation, a.k.a. LockBit Black. However, when comparing the code, e_nas_linux_amd64 does not noticeably share any patterns. Moreover, the retrohunt also returned a sample of an encryptor with indicators of Babuk, and therefore, it is more likely the *Linux* encryptor originates from Babuk.

Writing an encryptor and decryptor for *ESXi* requires different knowledge than for *Win32* and *Linux*, and therefore, it is possible that those are developed by different authors. CTI created the Yara rule 'Playboy_ESXi_Encryptor' for the *ESXi* encryptor, and executed a retrohunt with it too. A few dozen other malware samples in the *VirusTotal* corpus matched this Yara rule, and those strongly suggested that the Playboy *ESXi* encryptor is based on the Babuk source code. A few different patterns have been tested in a Yara rule for the *Win32* encryptor, but no similar samples were found.

The source code of the Babuk ransomware builder was leaked a few years ago. Even though this ransomware operation has long been retired, its malware is still widely being used. Furthermore, the retrohunts on *VirusTotal* also returned names of several other well-known ransomware strains. This indicates that Playboy is not the only ransomware operation that leverages the Babuk source code. Some even use it to impersonate well-established names. This has happened before with LockBit [11], where the imposter leverages LockBit's reputation to extort victims into paying the ransom.

CTI was able to reverse engineer the Playboy ransomware encryptors, write Yara rules for them and hunt for similar samples on *VirusTotal*. Other ransomware investigations previously conducted by other researchers could provide information on the threat actor potentially behind Playboy. If Playboy shares a significant portion of its code with another, well-known ransomware operation, it indicates that Playboy is probably not entirely new. The *Linux* and *ESXi* encryptors in the Playboy RaaS are likely based on the leaked Babuk source code. However, many other ransomware operations have done so too in the past, and therefore, it is difficult to attribute the Playboy RaaS to any threat actor based on code similarity.

The analysis of the Playboy ransomware toolchain was performed statically. Some differences in the file encryption algorithm of the various encryptors has been statically identified, but no dynamic analysis was performed. Therefore, CTI has not determined whether the encryptors actually function properly.

CONCLUSION

The dismantling of the Playboy ransomware infrastructure, following swift collaboration between law enforcement and cybersecurity professionals, highlights the growing importance of proactive threat intelligence and public-private partnerships in combating cybercrime. Through reverse engineering and detailed analysis of the toolchain, it became evident that Playboy's design and functionality are significantly influenced by the leaked Babuk ransomware source code. The modular architecture – supporting multiple operating systems and CPU architectures – points to a more distributed development effort, likely involving multiple actors or affiliates, rather than a single threat actor.

Despite successful seizure of its infrastructure, the reuse of Babuk code and the adaptability of the Playboy ransomware family demonstrate the persistent challenges in attribution and the broader risks posed by source code leaks in the cybercriminal ecosystem. This case underscores the need for continued vigilance, robust malware detection through techniques like YARA-based retrohunting, and cooperation across borders to prevent the re-emergence or evolution of similar threats. Moving forward, cybersecurity defenders must stay agile and anticipate how threat actors repurpose leaked tools to develop new variants and extend their reach across critical infrastructure and law enforcement domains.

REFERENCES

[1] No More Ransom. https://www.nomoreransom.org/en/index.html.

[2] Baker, K. Ransomware as a Service (RaaS) Explained How It Works & Examples. CrowdStrike. 30 January 2023. https://www.crowdstrike.com/en-us/cybersecurity-101/ransomware/ransomware-as-a-service-raas/.

[3] Microsoft. GetLogicalDriveStringsW function (fileapi.h). https://learn.microsoft.com/en-us/windows/win32/api/fileapi/nf-fileapi-getlogicaldrivestringsw.

[4] Microsoft. I/O Completion Ports. https://learn.microsoft.com/en-us/windows/win32/fileio/i-o-completion-ports.

[5] Tiwari, R.; Koshelev, A. Taking a deep dive into Sodinokibi ransomware. Acronis. 3 July 2019. https://www.acronis.com/en-eu/blog/posts/sodinokibi-ransomware/.

[6] Stream Ciphers HC-128 and HC-256. https://personal.ntu.edu.sg/wuhj/research/hc/index.html.

[7] Madden, N. What's the Curve25519 clamping all about? https://neilmadden.blog/2020/05/28/whats-the-curve25519-clamping-all-about/.

[8] Bernstein, D.J. A state-of-the-art Diffie-Hellman function. https://cr.yp.to/ecdh.html.

[9] Bernstein, D.J. The ChaCha family of stream ciphers. https://cr.yp.to/chacha.html.

[10] Dela Cruz, A.; Gelera, B.; De Guzman, M.; Sto.Tomas, W. New Linux-Based Ransomware Cheerscrypt Targeting ESXi Devices Linked to Leaked Babuk Source Code. Trend Micro. 25 May 2022. https://www.trendmicro.com/en_nl/research/22/e/new-linux-based-ransomware-cheerscrypt-targets-exsi-devices.html.

[11] NP AV. Ransomware Gangs Impersonate LockBit to Intimidate Victims and Leverage AWS in Latest Attacks. 24 October 2024. https://blogs.npav.net/blogs/post/ransomware-gangs-impersonate-lockbit-to-intimidate-victims-and-leverage-aws-in-latest-attacks.

INDICATORS OF COMPROMISE

All indicators of compromise found in the Playboy RaaS investigation are listed in this section.

Yara rules

```
rule Playboy_Linux_Encryptor
{
 meta:
  author = "Gijs Rijnders"
  description = "Linux encryptor Playboy"
  target_entity = "file"
 strings:
  $pattern = { C6 84 24 ?? 00 00 00 AB C6 84 24 ?? 00 00 00 BC C6 84 24 ?? 00 00 00 CD C6 84 24 ?? 00 00
00 DE C6 84 24 ?? 00 00 00 EF C6 84 24 ?? 00 00 00 F0 }
  $a0 = { C6 84 24 ?? 00 00 00 AB }
  $a1 = { C6 84 24 ?? 00 00 00 BC }
  $a2 = { C6 84 24 ?? 00 00 00 CD }
  $a3 = { C6 84 24 ?? 00 00 00 DE }
  $a4 = { C6 84 24 ?? 00 00 00 EF }
  $a5 = { C6 84 24 ?? 00 00 00 F0 }
  $chacha20 = "chacha20"
  $golang = "golang"
 condition:
  ($pattern or all of ($a*)) and $chacha20 and $golang and (uint32(0x0) == 0x464c457f)
}

rule Playboy_ESXi_Encryptor
{
 meta:
  author = "Gijs Rijnders"
  description = "ESXi encryptor Playboy"
  target_entity = "file"
 strings:
  $sosemanuk_unum32 = { 00 00 00 00 13 CF 9F E1 26 37 97 6B 35 F8 08 8A }
  $fopen = "fopen"
  $rb = "r+b"
  $vmdk = ".vmdk"
  $vmem = ".vmem"
  $vswp = ".vswp"
  $vmsn = ".vmsn"
 condition:
  all of them and (uint32(0x0) == 0x464c457f)
}
```

SHA-256

Hash	Description
c2faabcd0f2a08bdda2bb78594aa2e7b8791dbef8a81025710ebea3d9eeabe6a	Builder.exe
4b89e887f8655552f262c2631a30dabae877c98343ff5319410d397533fc8d86	*ESXi* encryptor
c4461514857dd73a9facbb53566d83a7e4dd9be8475ab93bd655a4a851d32aed	*Linux* (NAS) encryptor for ARM
d816be88fb691acbe3bea3d75dd7578d9eb52e2129ef4c125f2293de5e6f4406	*Linux* (NAS) encryptor for x86
adf375ebd28651b88f5c6b3cdd453c739e18a9a50a7263b17f3fbae87380f2aa	*Win32* encryptor
838ef806fe284bdf81eb29e924eb049f5d8364ace1e554d56e98e91ce02c6c0e	Build.bat
d3a525c8efb4b15e2ae3b64ac079bb65577c8ebde7230bb0c76b21673e60f0e1	Babuk *ESXi* encryptor
a52c87e1e8483ad75d0fc6344828426ce071439daf49864844ce7fc18eeea32f	Lockbit *Linux* NAS encryptor x64
204ae746f0ea2981e27af0082e18237d5b92ec786304cb34e53c4a1ba82a7747	*Win32* decryptor
e4ab7e7855faab81f26964220bd00760b395405a8f8b3b7e83470c4782ec65ea	*ESXi* decryptor
f049f4e625151f241f1c1b2ca77dc00548910edbe3122717509c1f4adb37e4cf	*Linux* (NAS) ARM decryptor
a6c8f9026bbff82c9ebbd58832a660045d6f123104a153705524b232b6defd03	*Linux* (NAS) x86 decryptor