8COM
CYBER SECURITY

High-End Social Engineering Meets Malware:

Analysis of a Targeted Attack Campaign Against Law Firms

High-End Social Engineering Meets Malware: Analysis of a Targeted Attack Campaign Against Law Firms

In November 2025, a sophisticated social engineering campaign emerged targeting law firms and legal organizations in Germany. Unlike conventional phishing operations, this campaign demonstrated an exceptionally high level of operational investment and tradecraft maturity, combining identity theft, real-time voice communication, and multi-stage technical deception to compromise human resources departments.

The attack campaign is characterized by several distinguishing features that set it apart from commodity malware distribution:

Targeted Victim Selection

The threat actors specifically targeted HR departments within German law firms, exploiting the predictable workflow of resume review and candidate screening. To date, all confirmed victims have been located in Germany, suggesting either a geographically constrained campaign or an initial testing phase targeting a specific region before broader deployment. By focusing on organizations that routinely process job applications, the attackers leveraged institutional trust in hiring procedures to reduce suspicion and increase the likelihood of successful execution.

Identity Theft and Impersonation

Rather than using fictitious personas, the attackers appropriated the professional identities of real individuals. Victims reported receiving applications that referenced authentic LinkedIn profiles, accurate employment histories, and verifiable professional credentials. In at least one confirmed case, the individual whose identity was stolen could be contacted directly and confirmed they had no involvement in the application process.

This level of reconnaissance suggests the threat actors invested significant effort in target profiling, potentially harvesting publicly available professional data from LinkedIn, corporate websites, and other open-source intelligence (OSINT) platforms to construct convincing false identities.

AI-Enhanced Voice Impersonation

Multiple HR representatives who engaged with the attackers via phone reported that the conversations felt unusually natural and professionally appropriate. The caller‘s voice characteristics, tone, and communication style matched the demographic and professional background suggested by the resume. In several instances, victims stated that the individual „sounded exactly like someone with that background would sound.“

This observation, combined with recent intelligence on the commercial availability of AI-powered voice cloning platforms, suggests the possible deployment of synthetic voice technology. As documented in Group-IB‘s „Weaponized AI“ whitepaper (2026), voice impersonation services are now accessible for as little as $10–$3,000 per month, offering real-time speech-to-text and text-to-speech capabilities integrated with large language models (LLMs).

These platforms enable attackers to clone target voices using minimal audio samples scraped from social media or public recordings and synthesize entire conversations using fully automated AI-generated speech.

While definitive attribution to AI voice cloning cannot be confirmed without access to call recordings, the behavioral profile of the attacks aligns closely with documented AI-assisted social engineering techniques currently circulating in underground forums.

Operational Sophistication and Fail-Safe Design

A defining characteristic of this campaign is the extraordinarily high operational investment in maintaining plausibility, even in failure scenarios. The attack sequence incorporates multiple defensive layers designed to preserve the illusion of a legitimate job application if the victim becomes suspicious or the malware fails to execute.

Perhaps most notably, the malware implements an adaptive execution strategy that fundamentally alters its behavior based on the security posture of the target system. Stage 2 of the payload performs real-time detection of enterprise endpoint detection and response (EDR) solutions, specifically checking for the presence of:

- Microsoft Defender for Endpoint (MsSense.exe)
- CrowdStrike Falcon (CSFalconService.exe)
- F-Secure (fshoster64.exe)
- Kaspersky (avp.exe)

If any of these security products are detected, the malware immediately aborts malicious execution and displays only the decoy PDF resume. No password dialog is presented, no payload is decrypted, and the victim is left with what appears to be a standard document that has been successfully opened.

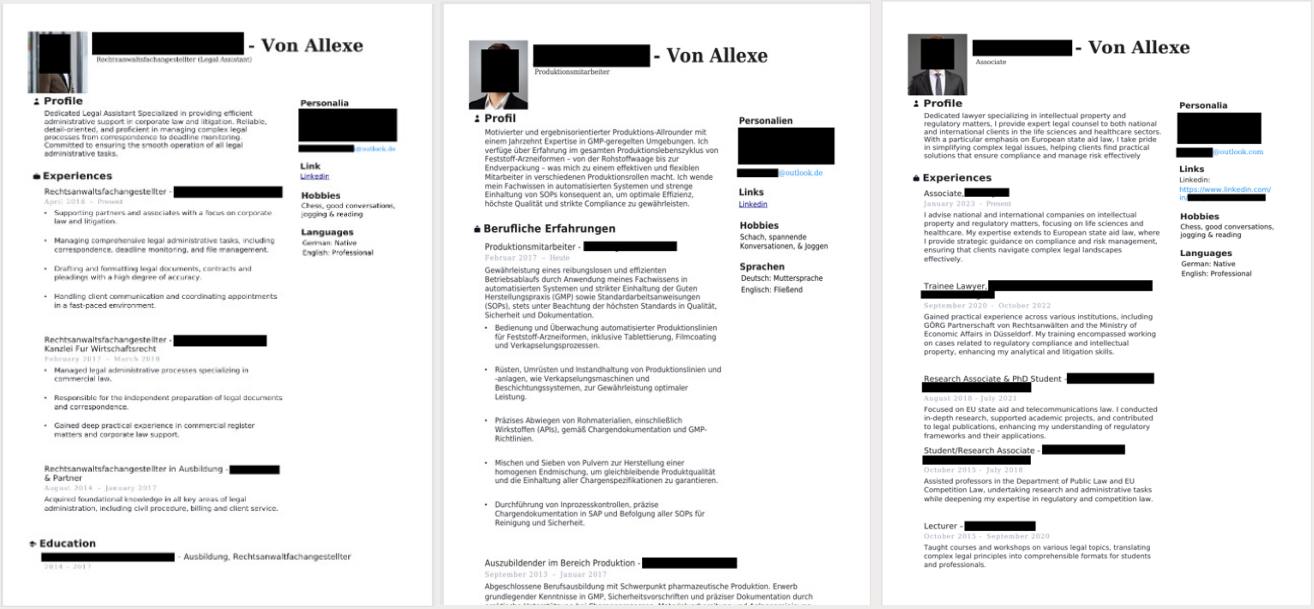


Figure 1: PDF resume, displayed as decoy (redacted for privacy reasons)

If no EDR solution is detected, the malware proceeds to display an Adobe Reader-style password prompt. The victim is then expected to ask the „applicant", still on the phone, to request the password. This could be a crucial step that signals the attacker that the critical execution is about to happen. This password is not arbitrary; it serves as the cryptographic key to decrypt Stage 3 of the malware. Without the correct password provided via phone, the payload cannot be unlocked.

This conditional execution model demonstrates several sophisticated design principles:

1. Environment-aware execution: The malware adapts its behavior based on security telemetry, ensuring it only executes in environments where detection risk is minimized
2. Human-in-the-loop decryption: By requiring telephonic password transmission, the attackers retain operational control over which systems are compromised and can abort the attack if the victim exhibits suspicion
3. Forensic evasion: In environments with robust EDR coverage, no malicious code is ever decrypted or executed, leaving minimal forensic artifacts for incident response teams to analyze
4. Plausible deniability: If confronted, the attacker can claim they simply sent a password-protected resume and had no knowledge of any malicious payload

Additional fail-safe mechanisms include:

5. Valid digital code signature: The malicious executable is signed with a legitimate certificate, reducing security warnings and bypassing basic trust verification
6. Decoy PDF delivery: After the malware executes (or if EDR is detected), a genuine (unencrypted) PDF resume is displayed, allowing the victim to review the „applicant's" qualifications
7. Identity persistence: If the victim later investigates, they encounter a real person with a matching LinkedIn profile, reinforcing the perception of legitimacy
8. Multi-channel coordination: The simultaneous use of voice (phone call) and digital (email) communication channels mirrors standard business practice and reduces cognitive red flags

This approach demonstrates a clear understanding of defensive security practices, incident response workflows, and victim psychology. By ensuring that every observable artifact (from the phone conversation to the displayed resume) appears authentic, and by actively preventing execution in hardened environments, the attackers significantly reduce the likelihood of detection, attribution, and incident escalation.

Campaign Objectives

The combination of targeted victim selection, identity theft, potential AI voice synthesis, adaptive execution logic, and fail-safe operational design suggests a threat actor with strategic objectives beyond opportunistic malware distribution. The geographic concentration on German law firms may indicate sector-specific intelligence gathering, access to privileged legal communications, or targeting of high-value clients represented by these organizations. The campaign's emphasis on persistence, stealth, and believability indicates that the attackers prioritize:

- Sustained access over immediate financial gain
- Enterprise compromise rather than individual targeting
- Operational security to enable long-term intelligence gathering or network positioning
- Selective targeting by limiting execution to systems without enterprise-grade security monitoring

This profile is consistent with advanced persistent threat (APT) tradecraft, where initial access serves as a gateway to broader organizational compromise, data exfiltration, or supply chain infiltration.

The technical analysis that follows examines the multi-stage malware payload delivered through this campaign, revealing a sophisticated execution chain designed to evade endpoint detection, establish command-and-control communication, and maintain persistent access to compromised systems.

Initial Infection Vector and Social Engineering

The infection chain begins with a sophisticated social engineering campaign targeting human resources departments of law firms and legal organizations. Unlike conventional phishing operations, this attack combines voice-based social engineering with technical deception to bypass traditional security awareness training.

The attack unfolds through a carefully orchestrated sequence of social and technical manipulation designed to exploit organizational trust and routine business processes.

Attack Sequence

Initial Contact

The threat actor initiates contact via phone call to the HR department, posing as a job applicant interested in employment opportunities within the targeted organization. During this conversation, the attacker establishes rapport with the HR representative while simultaneously sending a resume via email using a OneDrive share link. This dual-channel approach, voice communication paired with email delivery, creates a sense of legitimacy and reduces suspicion.

The use of OneDrive, a trusted enterprise collaboration platform, adds an additional layer of credibility to the attack. Files delivered through legitimate cloud storage services often bypass email attachment scanning and are less likely to trigger security alerts.

Technical Deception

The shared file, presented as a standard PDF resume, is in fact a digitally signed Windows executable file masquerading as an Adobe PDF document. The malicious binary is a modified version of Sysinternals DiskView, a legitimate Microsoft utility, repurposed to serve as the initial infection vector.

Key characteristics of the malicious file:

- Valid digital signature to bypass security warnings
- Adobe PDF icon to visually deceive the victim
- Modified version of legitimate Sysinternals DiskView
- File extension: .EXE (often hidden by default Windows settings)

When executed, the malware displays an Adobe Reader-style password dialog, claiming the resume is password-protected. This UI element is designed to appear entirely legitimate, matching the visual design patterns users expect from genuine PDF readers.

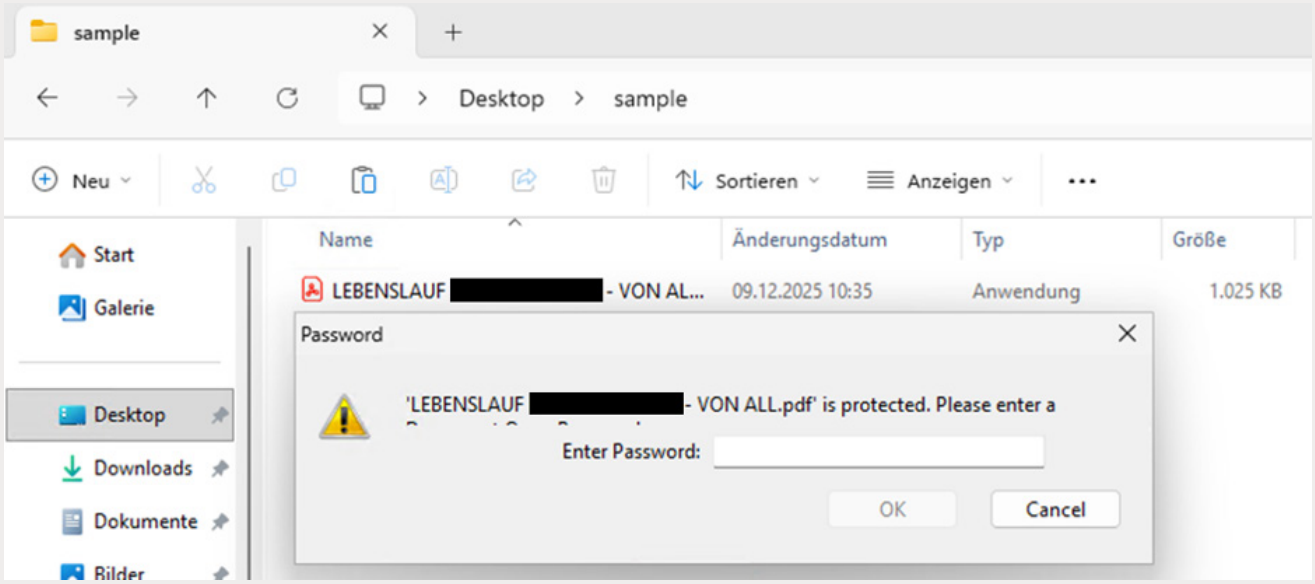


Figure 2: Password prompt displayed to victim

Credential Harvest via Social Engineering

The HR representative, believing they are accessing a legitimate password-protected resume, asks the „applicant“, who is still on the phone at this point, to request the password. The threat actor provides the password, which the victim then enters into the fake dialog.

This action triggers the malware‘s execution chain while simultaneously opening a genuine, unencrypted PDF resume as a decoy. The decoy PDF serves a critical function: it reinforces the illusion of normalcy. From the victim‘s perspective, they successfully opened a password-protected resume and can now review the applicant‘s qualifications. This visual distraction keeps the victim engaged while malicious activity proceeds silently in the background.

No exploit or vulnerability is leveraged at this stage. The attack relies entirely on user execution combined with social engineering, a technique that remains highly effective in enterprise environments where resume reviews and document sharing are routine operations.

Stage 0: Modified and Signed Sysinternals DiskView

Initial static analysis confirms that the malicious executable presents itself as „Sysinternals DiskView“ and is signed with a valid certificate issued by MOUNI MEDIA PRIVATE LIMITED.

Code Signing Analysis

The binary carries a valid digital signature with the following attributes:

Certificate Details:

- **Signer Name:** MOUNI MEDIA PRIVATE LIMITED
- **Email:** anjitmounimedia@gmail.com
- **Signature Algorithm:** sha256RSA
- **Certificate Validity:** October 29, 2026 05:16:12 UTC
- **Certificate Fingerprint:** `70e6a96e5ac6736029d1030e005f7a905a27c855`

Despite the valid signature, comparative analysis with the legitimate Sysinternals DiskView binary reveals substantial modifications to the executable structure.

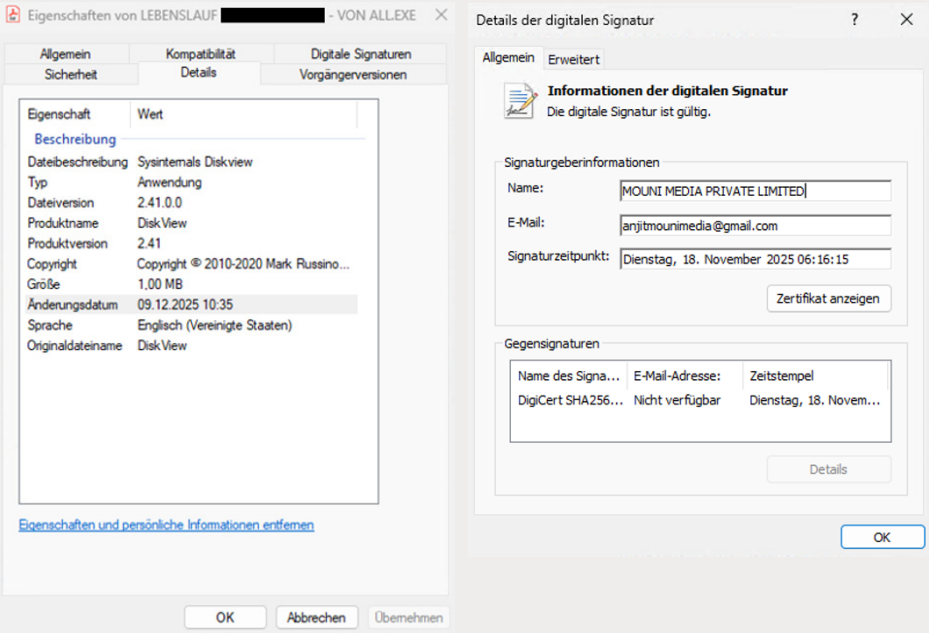


Figure 3: File details and digital signature information

Binary Structure Comparison

Structural analysis using PE Bear reveals significant differences between the legitimate DiskView and the malicious variant:

Original DiskView:

- `.text` section: Standard code segment
- `.rdata` section: Read-only data
- `.data` section: Initialized data
- `.rsrc` section: Standard application resources
- `.reloc` section: Relocation table present
- **Total sections:** 5

Modified Malicious DiskView:

- `.text` section: Modified code segment containing unpacker stub
- `.rdata` section: Largely unmodified
- `.data` section: Modified to support unpacker
- `.rsrc` section: **Dramatically increased in size**
- `.reloc` section: **Removed completely**
- **Total sections:** 4

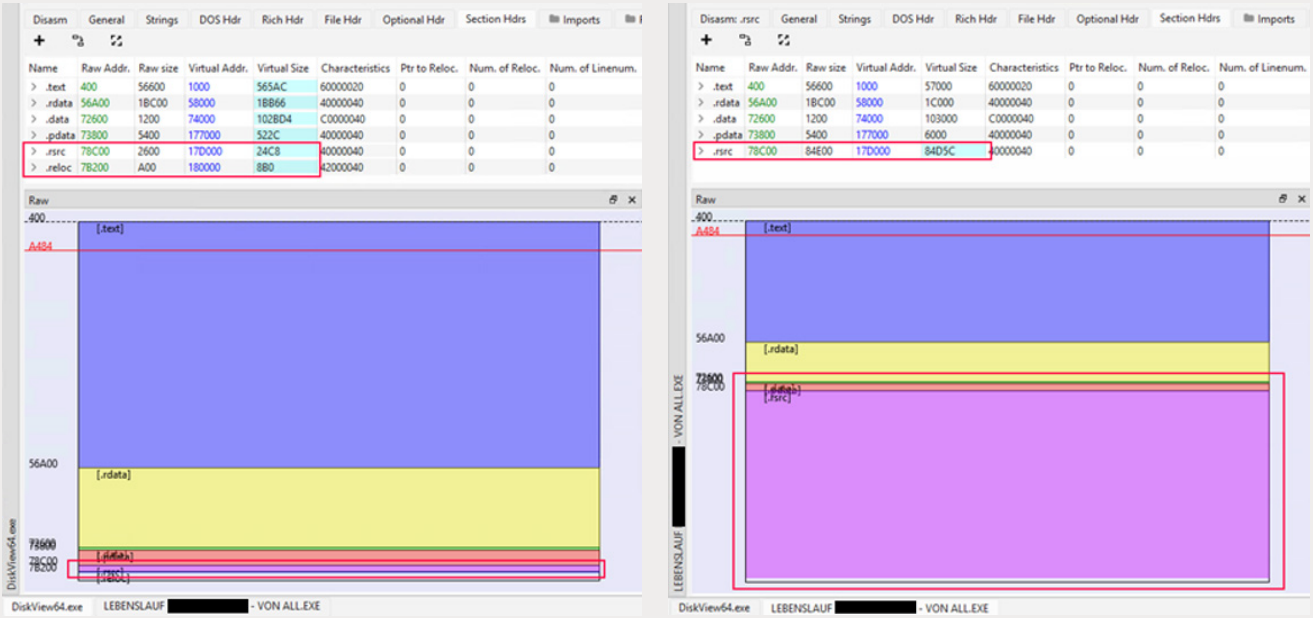


Figure 4: Original DiskView (left) vs. Modified Malicious DiskView (right) (PE Bear)

The removal of the `.reloc` section indicates that the malicious binary has been compiled with a fixed base address and does not support Address Space Layout Randomization (ASLR) rebasing, a deliberate choice to simplify the unpacker implementation.

Entropy Analysis and Payload Detection

The entropy analysis of the different sections of the PE file using Detect It Easy (visible in Figure 4) reveals a critical indicator of malicious modification. While both executables contain a `.rsrc` section, the malicious variant shows an entropy spike near-maximum entropy values (approaching 8.0) in the upper half region of the `.rsrc` section. This indicates that the first half of the `.rsrc` section is likely unencrypted (in a cryptographical sense), while the other half seems encrypted.

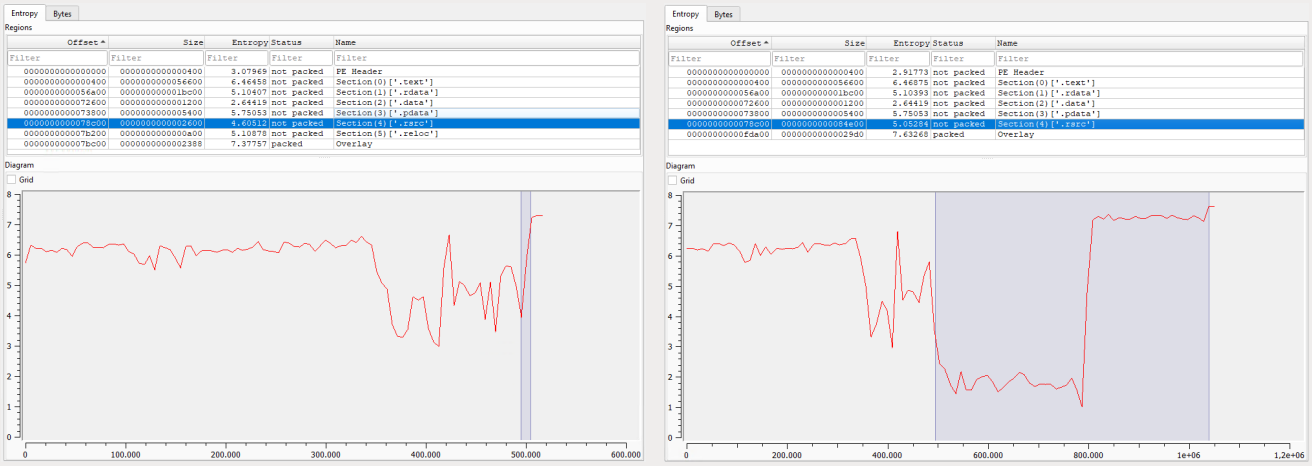


Figure 5: Original DiskView .rsrc Entropy (left) vs Malicious DiskView .rsrc Entropy (right) (Detect it easy)

Modified WinMain Function

The decompilation of the modified binary reveals that a part of the original DiskView functionality has been completely replaced. When both executables are decompiled and the WinMain functions are compared, it becomes evident that the original DiskView „-accepteula“ command-line parameter-check function has been replaced by an unpacker function that serves also as the entry point for Stage 1 execution.

Figure 6: Original DiskView

Figure 7: „LEBENSLAUF“ Sample

The code indicates that execution immediately diverts to the unpacker routine, confirming that no original DiskView code is executed.

Stage 1 Unpacking Routine

The unpacker routine operates by loading data byte-by-byte from the `.rsrc` area, which begins at **`0x14017D000`** and ends at **`0x140202000`** (see Figure 8). The decryption process uses simple XOR operations using a dynamically generated lookup table.

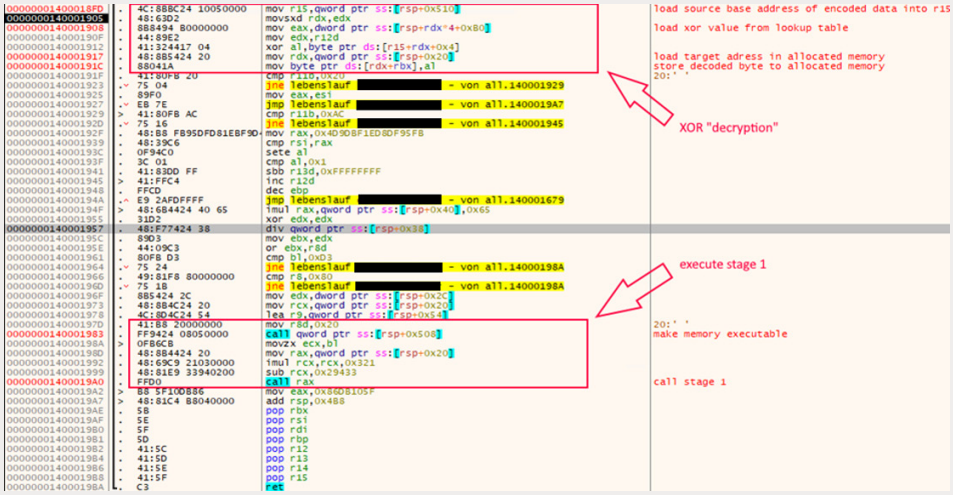


Figure 8: The stage 1 unpacking routine

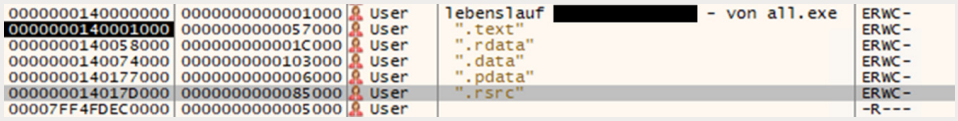


Figure 9: Memory Map location of .rsrc

Unpacking Process:

1. **Source Location:** `0x1401C8988` (`.rsrc` section base address + offset)
2. **Size Extraction:** The first DWORD at this location (`0x2F31F` = 192799 bytes) specifies the encrypted payload size
3. **Lookup Table Generation:** A 256-byte XOR key table is constructed on the stack using complex arithmetic operations
4. **Memory Allocation:** `VirtualAlloc` is called to allocate RW memory
5. **XOR Decryption Loop:** The payload is decrypted byte-by-byte XORed with bytes from the Lookup Table
6. **Memory Execution Protection:** VirtualProtect is called to make memory executable
7. **Execution Transfer:** Call the entry function of stage 1 at the start of the allocated memory

The unpacking routine uses a position dependent XOR scheme where each byte's XOR key is derived from both its position and values extracted from the lookup table. This provides a basic obfuscation layer that prevents static signature detection of the Stage 1 payload.

Key Observations:

- The Windows loader process never encounters the decrypted Stage 1 code, as it is only materialized in memory at runtime
- The `.rsrc` section's high entropy makes the payload location immediately evident to entropy-based detection tools
- The unpacker stub in the `.text` section contains no legitimate DiskView code paths
- The absence of the `.reloc` section indicates the binary is not position-independent and requires a fixed base address

This design confirms that the legitimate DiskView binary has been weaponized as a dropper, as it has been minimally modified: the original code remains largely intact, but a single malicious function has been inserted to unpack and execute a payload stored in a newly added .rsrc section. The icon has been replaced with an Adobe PDF appearance to enhance social engineering effectiveness, and the binary has been re-signed with a different but valid code signing certificate, allowing it to maintain trust indicators while serving as a multi-stage loader.

Stage 1: Build Obfuscated API Dispatch Table and Extract Stage Two

Following successful execution of Stage 0, control transfers to the decrypted payload located in RWX memory at base address 0xE00000. This stage implements a multi-phase initialization sequence designed to establish a foundation for subsequent payload execution while evading static and dynamic analysis.

Memory Layout and Entropy Analysis

Figure 10 visualizes the memory entropy of Stage 1, depicting the decoder routine and encrypted payload starting at offset 0x9AE3. The entropy visualization clearly demarcates the decoder routines (low entropy regions) and the encrypted payload (high entropy regions).

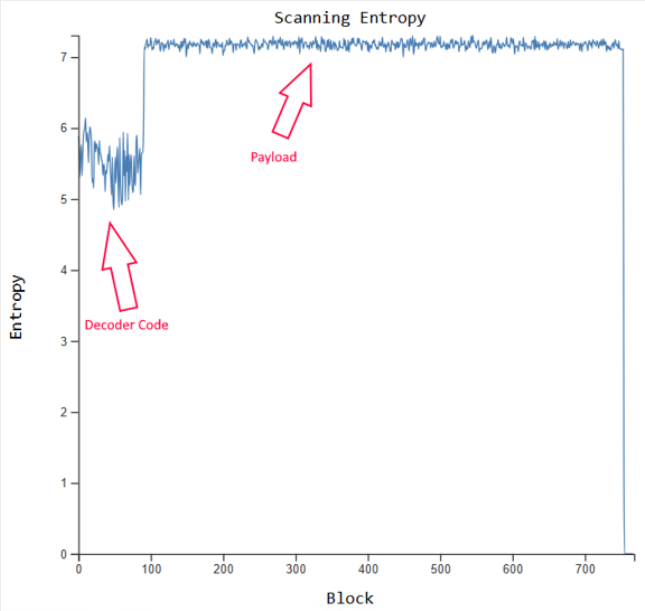
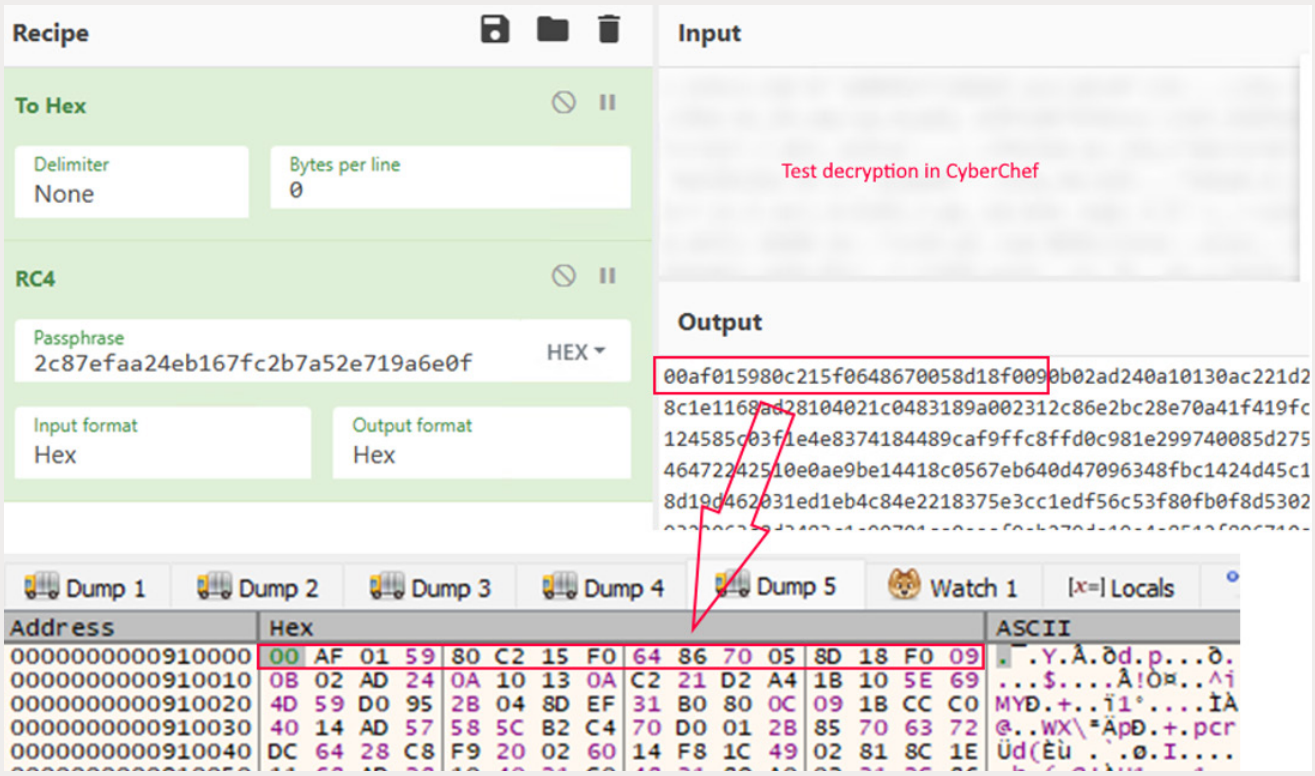


Figure 10: Stage 1 Memory Entropy, visualizing decoder and encrypted payload starting at 0x5AE3

RC4 Decryption with Hardcoded Key

The payload memory is decrypted using an RC4 implementation that employs a hardcoded 16-byte key: `2C 87 EF AA 24 EB 16 7F C2 B7 A5 2E 71 9A 6E 0F`

The RC4 implementation follows the standard algorithm, which can be verified using CyberChef (shown in Figure 10).



aPLib Decompression

Following successful RC4 decryption, the decrypted memory contains compressed data. Figure 11 demonstrates the special data length handling that is characteristic of aPLib's compression format.

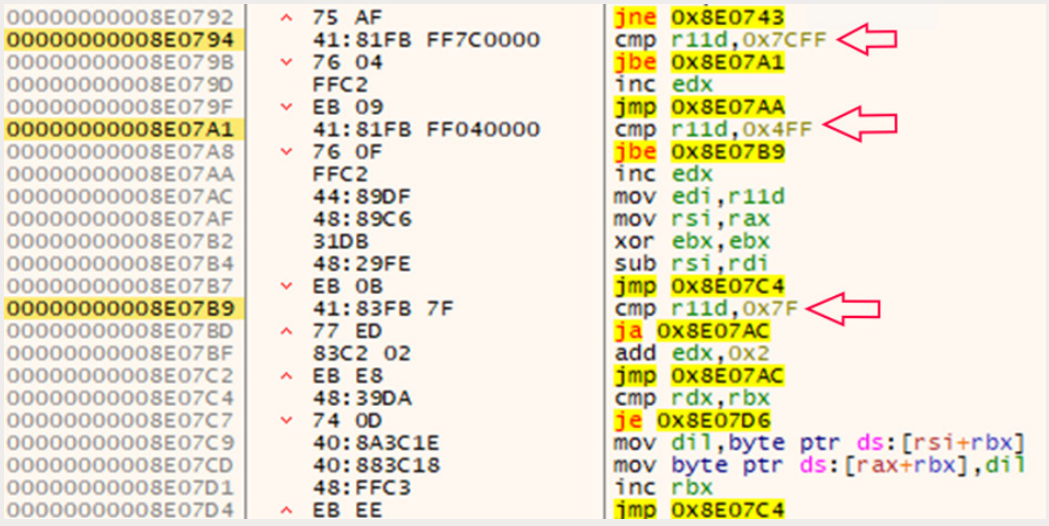


Figure 11: Special data length handling, unique in aPlib decompression[1]

aPLib is a highly efficient LZ77-based compression algorithm frequently employed in malware due to its very tiny decoder size (~200 bytes), high compression ratio and fast decompression speed.

[1] https://github.com/secretsquirrel/the-backdoor-factory/blob/master/aPLib/src/depack.c#L138

MurmurHash2-Based API Resolution
PEB-Walking and DLL Enumeration

Stage 1 implements a classic PEB walking technique to enumerate the loaded DLLs of the current process. Figure 12 illustrates the iteration through loaded DLLs and the explicit search for ntdll.dll.

The DLL names are read character-by-character and validated through an obfuscated validation routine:

Figure 12: Iteration through loaded DLLs and search for ntdll.dll

This obfuscation serves to hinder static analysis, as the DLL name cannot be identified through simple string comparison. The use of mathematical operations (division/modulo) instead of string operations makes automated detection significantly more difficult.

Upon successfully identifying ntdll.dll the code extracts the DllBase address for subsequent export enumeration.

MurmurHash2 Comparison and API Storage

The core of the API resolution mechanism is a MurmurHash2 implementation (Figure 13 shows the seed calculation, while Figure 14 displays the complete implementation). This non-cryptographic hash function is used to efficiently match desired API names against ntdll exports.

The offset to the ntdll.dll is then used to iterate over all the exported functions and their function names are hashed with MurmurHash2 with seed 0x514EFB3D. These hashes are then used to find the function pointers to functions needed later in stage 2.

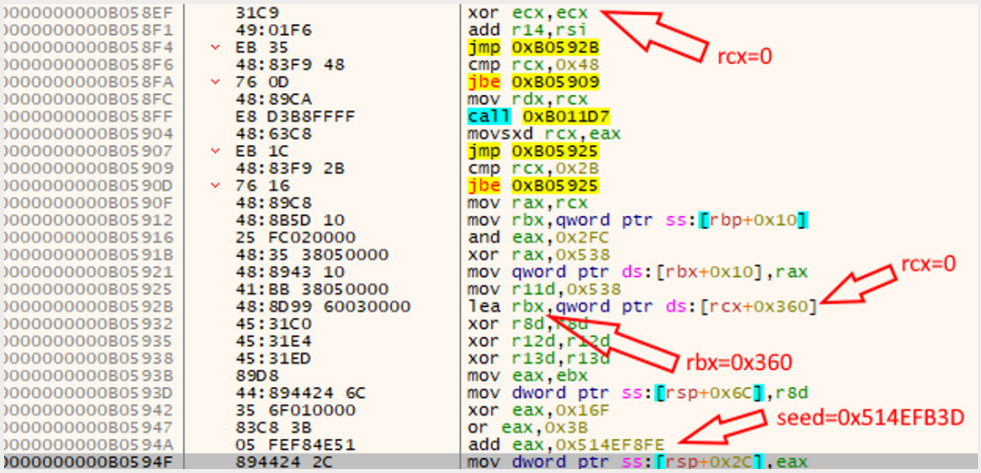


Figure 13: MurmurHash seed calculation

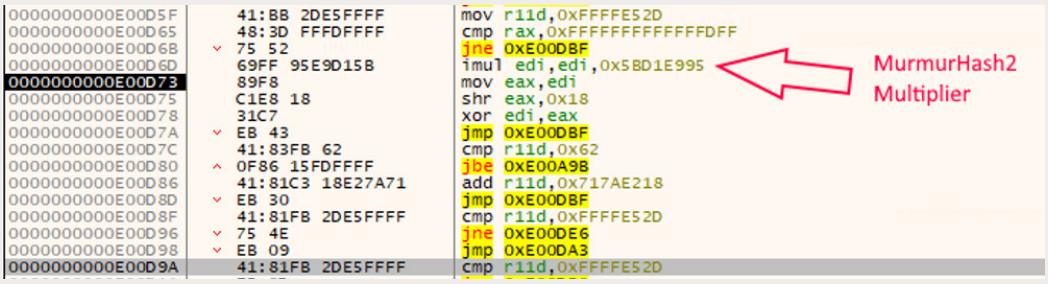


Figure 14: MurmurHash2 implementation[2]

XOR-Obfuscated API Dispatch Table

Figure 15 illustrates the construction of the XOR-obfuscated API dispatch table. Stage 1 takes the resolved API addresses and subjects them to a **four-stage XOR transformation** before writing them to the jump table. The obfuscated addresses are written as **executable x64 instructions** into the jump table:

Figure 15: Part of the xor obfuscated API call dispatch table

At runtime, these values are transformed back into valid API addresses through the inverse XOR operations. This creates a self-modifying code structure that evades static analysis.

Control is then transferred to the Stage 2 entry point.

[2] https://github.com/explosion/murmurhash/blob/master/murmurhash/MurmurHash2.cpp#L76

Stage 2: Environment Check and Password-Based Decryption

EDR Detection and Evasion

Stage 2 begins with an environment check that acquires the current process list using `ZwQuerySystemInformation` and iterates through it to detect the presence of the following endpoint security solutions:

- Microsoft Defender for Endpoint (`MsSense.exe`)
- CrowdStrike Falcon (`CSFalconService.exe`)
- F-Secure (`fshoster64.exe`)
- Kaspersky (`avp.exe`)

If any of these EDR systems are detected, Stage 2 extracts the decoy resume PDF to `%APPDATA%\Local\Temp` and opens it with the default PDF viewer, avoiding further malicious execution in monitored environments.

Command-Line Parameter Processing

If no EDR systems are detected, the code checks for an additional command-line parameter and validates whether it is a 32-character hexadecimal string. The processing follows two paths:

- Non-hexadecimal string: The second command-line parameter is passed to a key derivation function (KDF) that generates a 16-byte key
- Hexadecimal string: The string is converted from hexadecimal back to 16-byte binary format

This 16-byte key is then used to RC4-decrypt the encrypted Stage 3 payload, as shown in Figure 16.

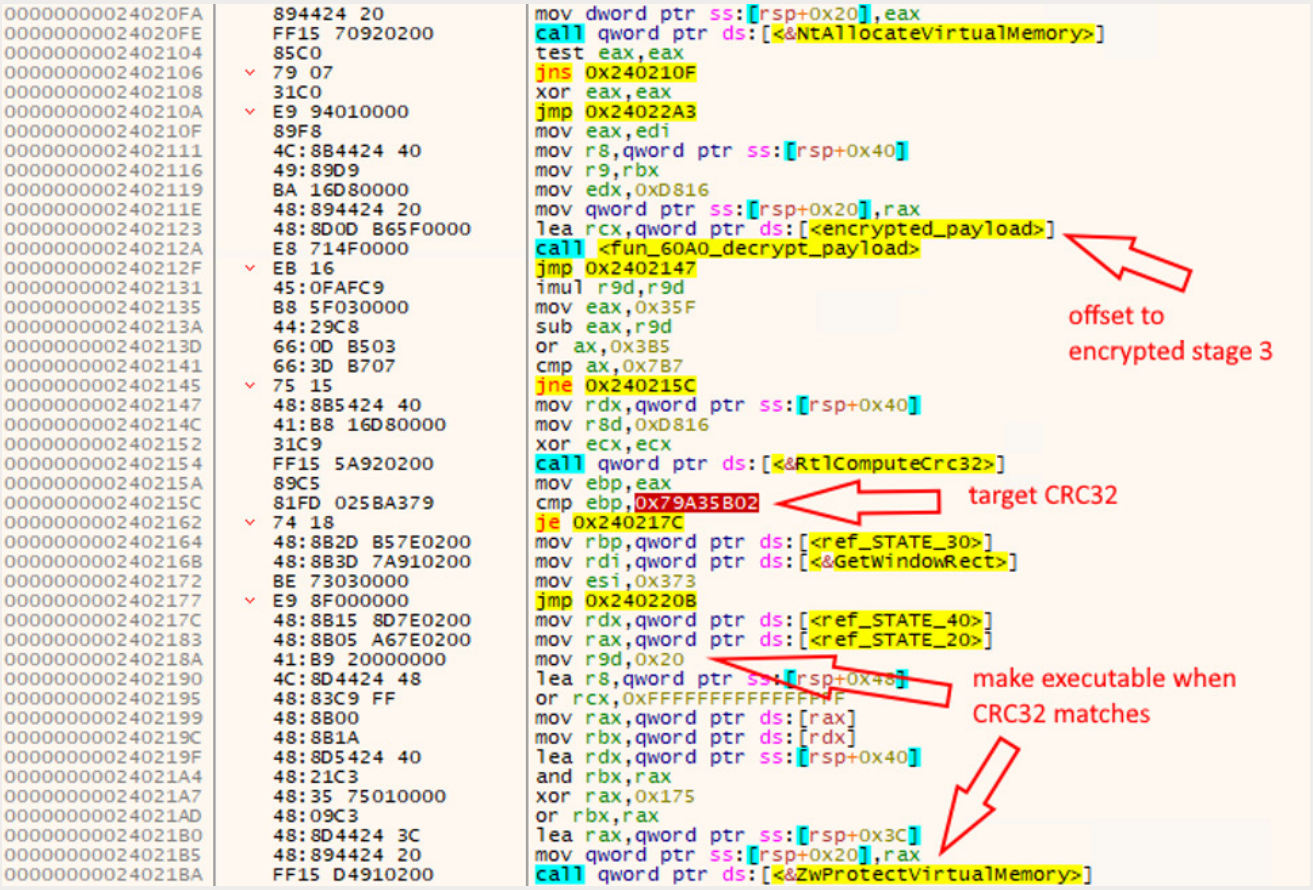


Figure 16: RC4 decryption and CRC32 check of stage 3

Password Dialog Functionality

When the executable is launched via double-click (without command-line parameters), the program displays the password dialog shown in Figure 2.

As illustrated in Figure 17, the password dialog's `WndProc` callback function[3] reads the entered password and passes it to the key derivation function (labeled fun_03E0_calc_hash_to_rsp+68 in the disassembly). To verify the password, the payload is decrypted with the derived key and validated against the expected CRC32 checksum.

[3] https://learn.microsoft.com/de-de/windows/win32/api/winuser/nc-winuser-wndproc

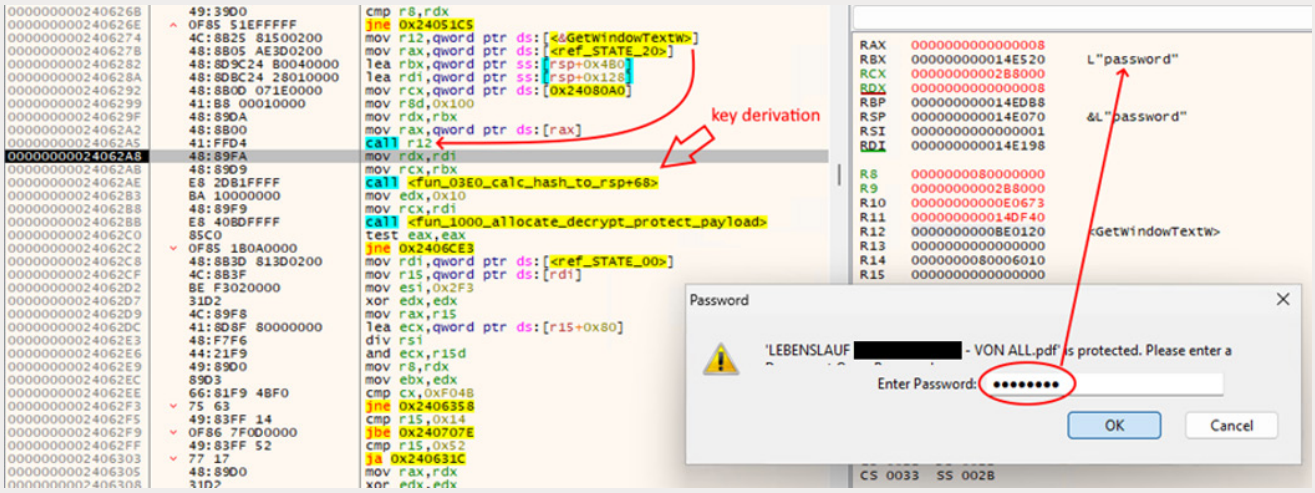


Figure 17: WndProc Callback function, processing the entered password

Password Recovery Through Reverse Engineering

At this point, we needed to determine the correct password. With no available intelligence on this sample or known victims remembering the password, we employed an AI-assisted coding approach to develop a brute-force application. The most critical step was accurately reproducing the key derivation function.

Key Derivation Function Reconstruction

To reverse-engineer the KDF, we executed the key derivation function multiple times with different inputs and recorded complete execution traces in x64dbg. Then we exported the traces as CSV files and provided the traces to Anthropic's Claude AI for analysis.

After several iterations and refinements, Claude successfully produced code that generated identical 16-byte keys for all test inputs, perfectly replicating the original implementation.

Brute-Force Strategy

With the reconstructed KDF, we developed a brute-force application that reads candidate passwords from a dictionary text file, performs key derivation on each candidate, RC4-decrypts the dumped Stage 3 payload and finally validates the result against the known CRC32 checksum.

Given that the attackers transmitted passwords verbally over the phone, we made several reasonable assumptions about the password's characteristics. First, the password would not be overly complex, as it needed to be easily communicated and understood during a phone conversation. Additionally, it must be straightforward to articulate verbally without ambiguity.

Finally, we hypothesized that it likely functioned as a „pseudo-password" designed to protect application documents rather than provide cryptographic security, suggesting a more user-friendly approach than a truly random cryptographic key.

Based on these assumptions, we selected a German dictionary as our base wordlist and applied systematic transformations to generate candidates. These transformations included case variations such as uppercase, lowercase, and mixed case combinations. We also appended numeric suffixes ranging from 2 to 6 digits, as well as common special character suffixes that are frequently used in user-generated passwords.

After several days of computation, the brute-force attack successfully recovered the correct password.

```
$ ./decrypt_bruteforce2.exe /c/Users/user/dev/stage3_enc.bin --dictionary
/c/Users/user/dev/german_dict.txt --suffix 6 --charset 0123456789!.?
========================================
Dictionary Suffix Testing
========================================
File: c:/Users/user/dev/stage3_enc.bin (55318 bytes)
Target CRC32: 0x79A35B02
Testing...
========================================

Input: 'Sonne1124'
  Key:   3159ECB4E865F5F5126C9983F8E839BE
  CRC32: 0x79A35B02 (target: 0x79A35B02) MATCH!
  Saved: decrypted_payload.bin

  First 64 bytes:
    0000: 41 56 41 55 49 BD 1A 75 78 C6 86 F8 CE 63 41 54
    0010: 49 BC D4 BC 77 2F 54 71 52 C2 55 48 BD 93 95 E4
    0020: A2 33 6D EE 57 57 48 89 CF 56 53 48 83 EC 60 B8
    0030: 35 27 F8 84 31 D2 48 89 44 24 30 48 8B 4C 24 30

========================================
Result: Match found!
----------------------------------------
```

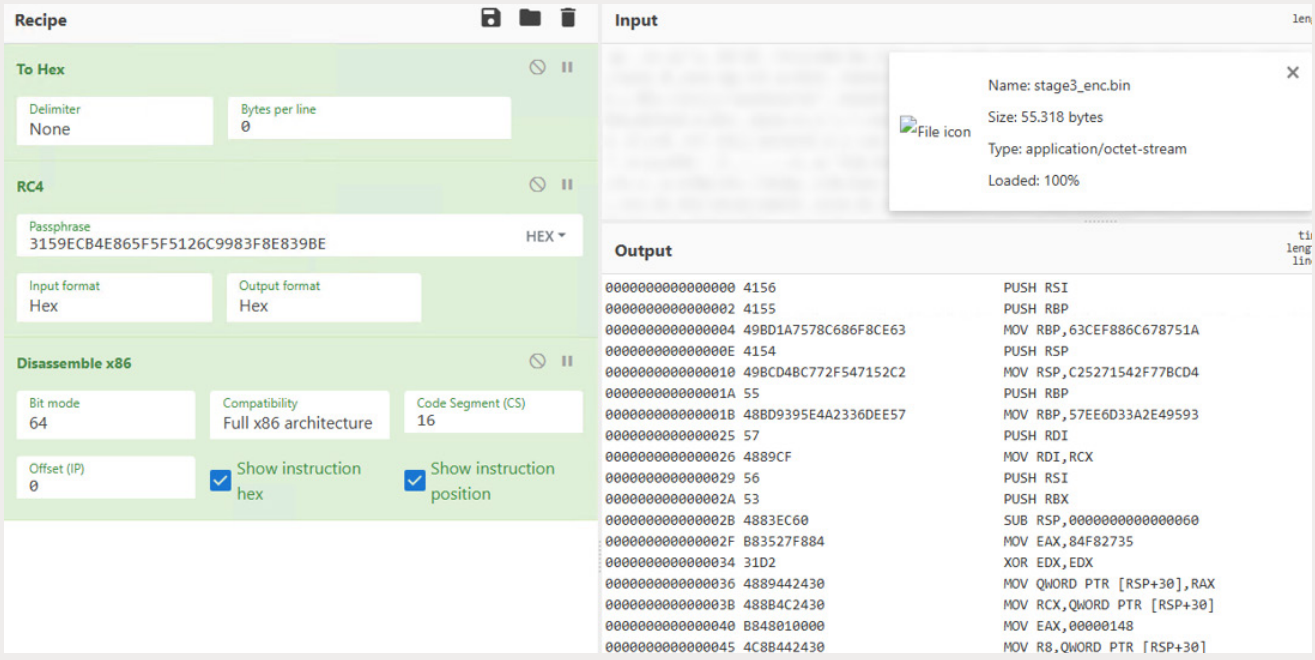


Figure 18: RC4 decryption with the found key results in valid x86 machine instructions

With the Stage 3 payload now decrypted, execution proceeded to the next stage of the malware.

Stage 3: Secondary Unpacking and Extended API Resolution

Stage 3 follows an architecture nearly identical to Stage 1, implementing a recursive unpacking pattern. The stage begins by performing RC4 decryption on an embedded Stage 4 payload, followed by aPLib decompression to extract the next executable code.

The RC4 key is hardcoded: `85 20 66 77 49 D4 D9 7B 48 F1 3A 18 CC 14 62 C7`

Figure 19 shows the key location in memory, while Figure 20 shows our RC4 validation with this key.

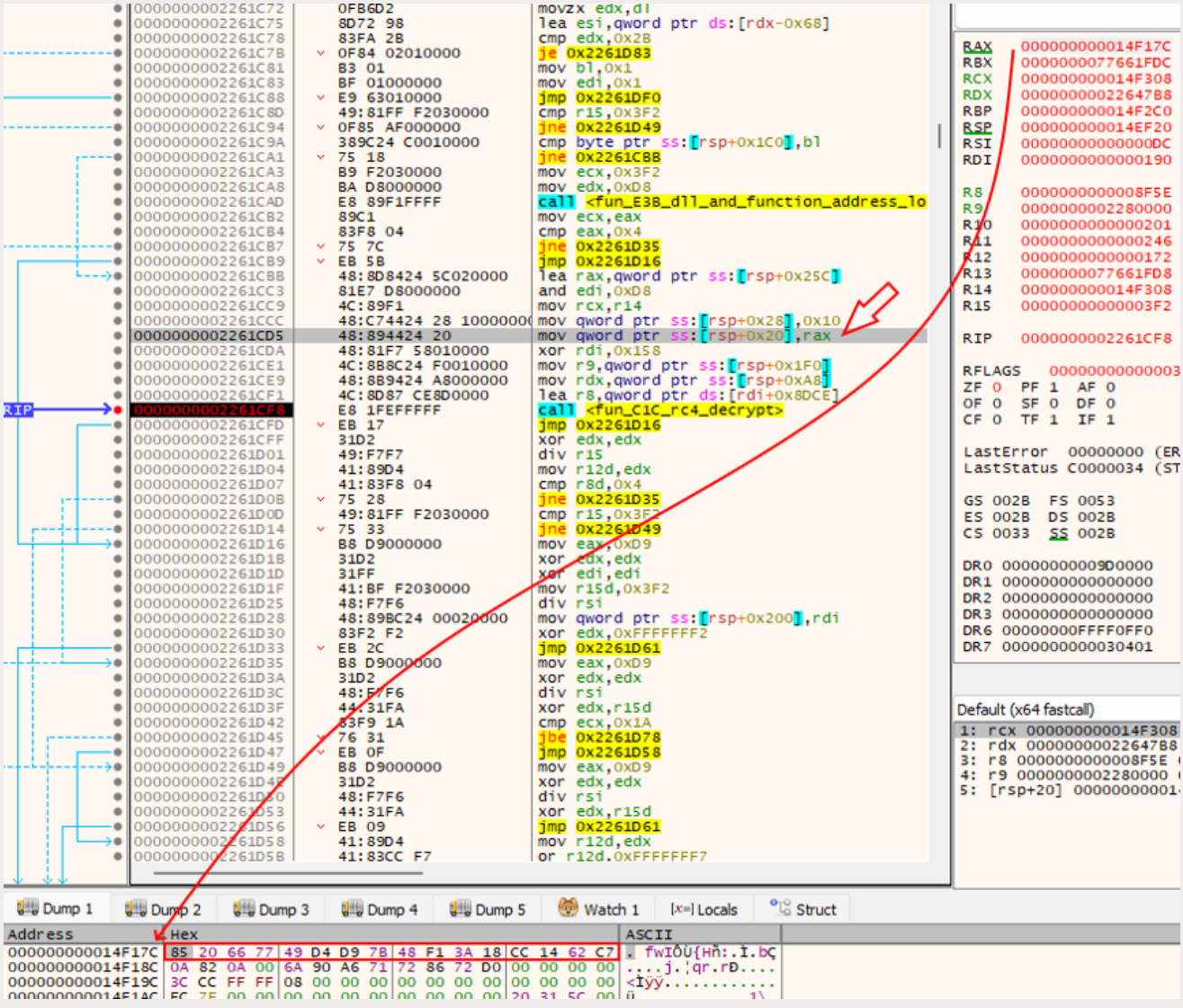


Figure 19: Key identification for stage 4 decryption

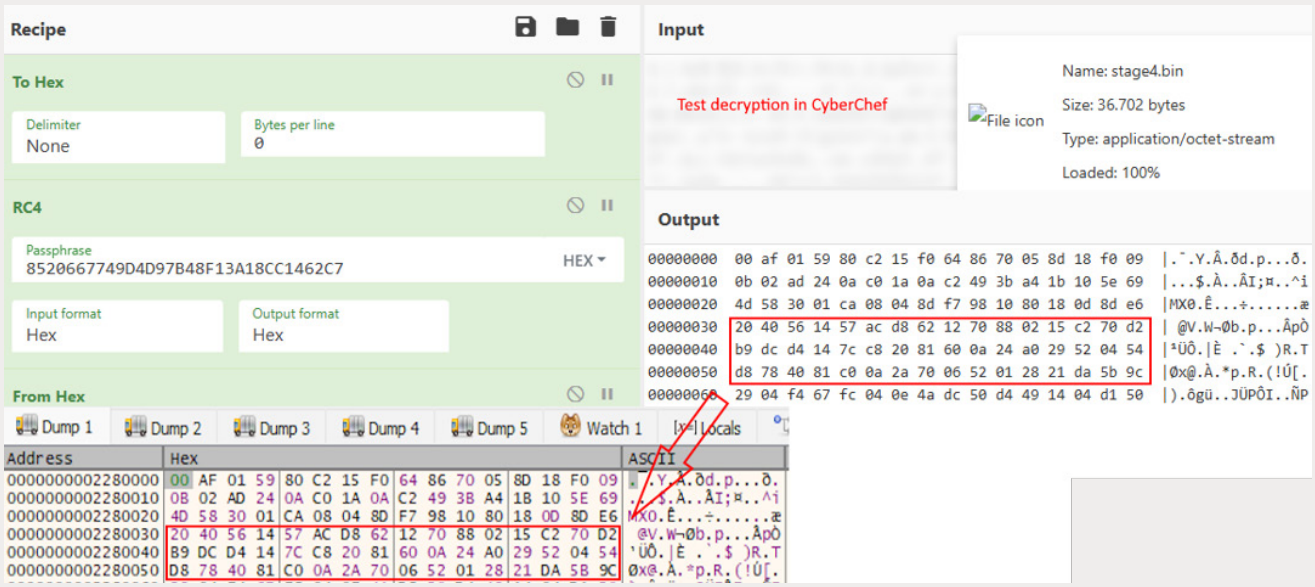


Figure 20: RC4 decryption validation shows the same result

DLL Loading and API Enumeration

Following decompression, Stage 3 dynamically loads additional Windows libraries to expand its API capabilities beyond the bootstrap functions resolved in Stage 1. The stage employs the same MurmurHash2-based API resolution technique to enumerate and resolve function addresses from the newly loaded DLLs.

The loaded libraries include ole32.dll for COM functionality, enabling the malware to interact with Windows Component Object Model interfaces through functions such as CoCreateInstance and CoSetProxyBlanket.

Additional ntdll.dll functions are resolved, significantly expanding the syscall-level API set available to the malware. These include critical memory management functions like NtAllocateVirtualMemory, NtFreeVirtualMemory, and NtProtectVirtualMemory, as well as process and thread manipulation APIs such as NtResumeProcess, NtSetContextThread, and RtlCreateUserThread. File system operations are enabled through NtCreateFile, NtReadFile, and NtWriteFile, while registry access is provided via NtOpenKey, NtQueryValueKey, and NtEnumerateValueKey.

Most notably, Stage 3 loads WINHTTP.dll to establish network communication capabilities. The resolved WinHTTP functions include WinHttpOpen, WinHttpConnect, WinHttpOpenRequest, WinHttpSendRequest, WinHttpReceiveResponse, WinHttpQueryHeaders, WinHttpQueryDataAvailable, and WinHttpReadData, providing a complete HTTP client implementation.

The inclusion of WinHttpSetOption suggests the malware configures specific connection parameters, potentially for proxy handling or TLS settings.

XOR-Obfuscated API Dispatch Table

Identical to Stage 1, all resolved API addresses undergo the four-stage XOR transformation before being written to a new jump table structure.

Control is then transferred to the Stage 4 entry point.

Stage 4: Download and Reflective PE Loading

Figure 21 shows Stage 4 maintaining entropy values consistently below 7, indicating the stage contains only unencrypted machine code without embedded encrypted payloads.

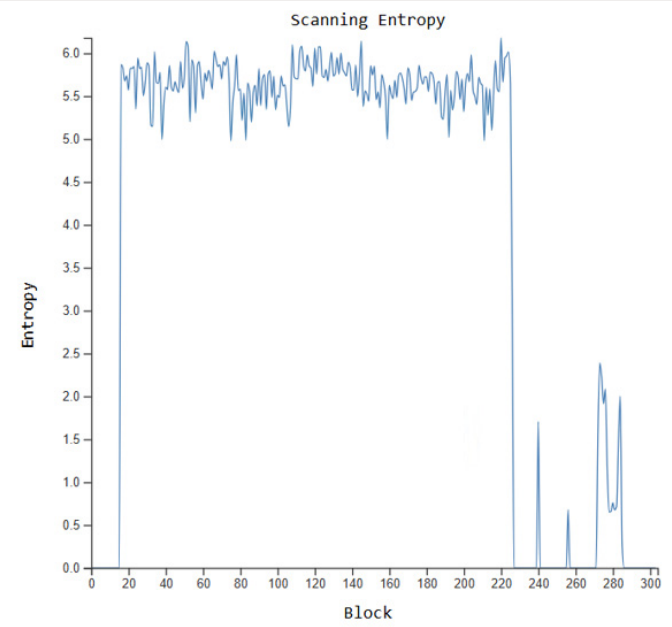


Figure 21: Entropy of Stage 4 memory area

Cryptographic Seed Generation

Stage 4 employs a dedicated routine to generate an 8-byte cryptographic seed that serves as the foundation for subsequent PRNG operations. The seed derivation combines multiple entropy sources to create a value that varies between executions.

The algorithm reads `TickCountMultiplier` and `SystemTime` from the `KUSER_SHARED_DATA` structure mapped at `0x7FFE0000`, providing a time-dependent component. This is combined with the `StandardError` handle obtained from the Process Environment Block's `ProcessParameters`, contributing process-specific entropy.

The resulting seed is stored in a global variable and functions as mutable state, each invocation of the PRNG consumes the current value and writes back a transformed result. Notably, the seed is not generated just once; the malware recalculates it at multiple points during execution.

System Fingerprint Generation

Beyond the cryptographic seed, Stage 4 constructs a 26-byte system fingerprint designed to uniquely identify the infected machine. This fingerprint is persisted in a global structure and later incorporated into the URL generation process.

The primary component derives from the current user's Security Identifier. Through direct calls to ZwOpenProcessToken and ZwQueryInformationToken, the malware retrieves the To-

kenUser information and extracts the three Domain SID SubAuthority values. These 12 bytes uniquely identify the Windows installation and remain constant across reboots, effectively binding the fingerprint to the victim's machine.

Additional entropy is gathered from the environment variables USERDOMAIN, USERNAME, and COMPUTERNAME. Each string is processed through a simple hash function that iterates over the characters, applying a multiply-and-add operation (imul by 0x25, then add the character value) to produce a 64-bit hash. The individual hashes are then combined through XOR and addition operations.

These components are mixed with several hardcoded magic constants through arithmetic transformations, producing a fingerprint that is both deterministic for a given system and practically unique across different installations. The structure also contains additional static bytes that appear to encode version or configuration information.

Persistence via AppData Installation and Run-Key

Stage 4 implements a conditional persistence mechanism based on how the sample was initially executed. When the malware is launched without command line arguments, indicating a first-run scenario triggered directly by the victim, the decryption key for Stage 3 is passed through the process environment variable `ARG`. In this case, the executable copies itself to `%AppData%\Roaming\Microsoft\DV\DiskView.exe`, establishing a foothold for subsequent executions.

Conversely, when the sample is started with the decryption key supplied as a command line argument, the `ARG` environment variable remains unset from Stage 2, and the copy procedure is bypassed entirely. This logic assumes that command line invocation indicates the malware is already running from its persisted location rather than being manually executed by the victim for the first time.

The persistence is established by leveraging the WMI `StdRegProv` provider to create a Registry Run key. Rather than using direct Registry API calls, the malware utilizes WMI's COM interfaces to write the autostart entry, providing an additional layer of indirection that may evade API monitoring focused on traditional registry functions.

The malware invokes `StdRegProv.SetStringValue()` with the following parameters

hDefKey	0x80000001 (HKEY_CURRENT_USER)
sSubKeyName	Software\Microsoft\Windows\CurrentVersion\Run"
sValueName	"DiskView"
sValue	"C:\Users\user\AppData\Roaming\Microsoft\DV\DiskView.exe" 3159ecb4e865f5f5126c9983f8e839be

Figure 22 shows the WMI call setting the `sValue` parameter, which contains both the malware executable path and a 32-character hexadecimal parameter. This parameter is the environment key set in Stage 2, required for decrypting Stage 3 payload.

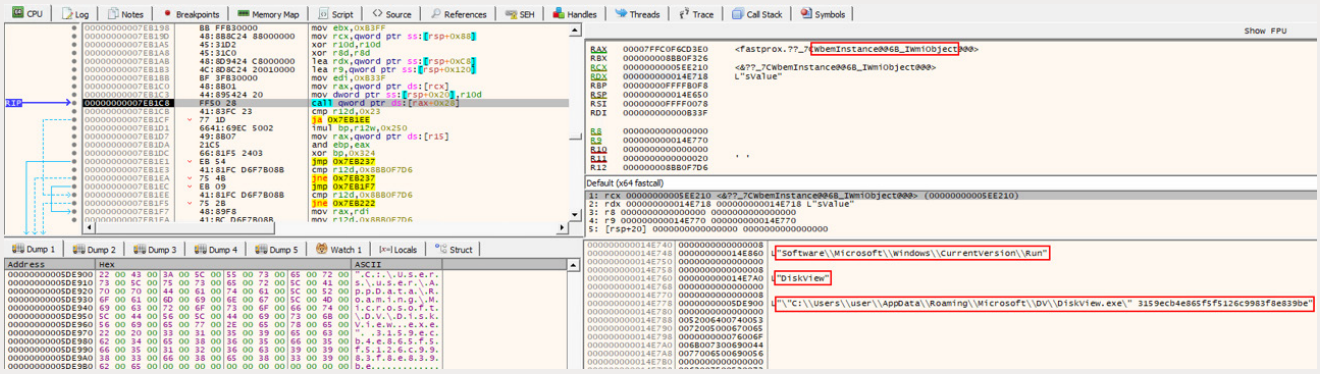


Figure 22: WMI call via `StdRegProv` to add new registry Run key for persistence

The resulting registry entry, visible in Figure 23, ensures the malware executes automatically at every user login while binding execution to the specific infected system.

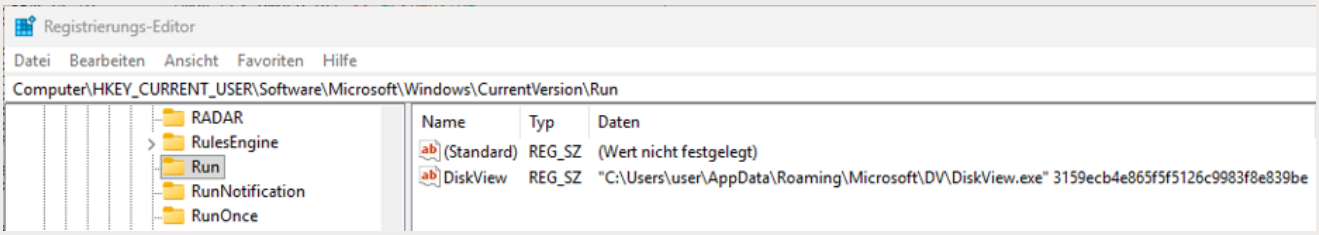


Figure 23: The added Run key, executing the copied sample, passing the decryption key for stage 3 as argument

The choice of `StdRegProv` over direct API calls (`RegSetValueEx`) demonstrates operational security awareness, as WMI-based registry modifications generate different telemetry patterns and may bypass endpoint detection rules focused on registry API hooking.

Check for ZSATunnel.exe

Following persistence establishment, the malware enumerates all running processes via `ZwQuerySystemInformation(SystemProcessInformation)` to detect explicitly the presence of `ZSATunnel.exe` (Figure 24).

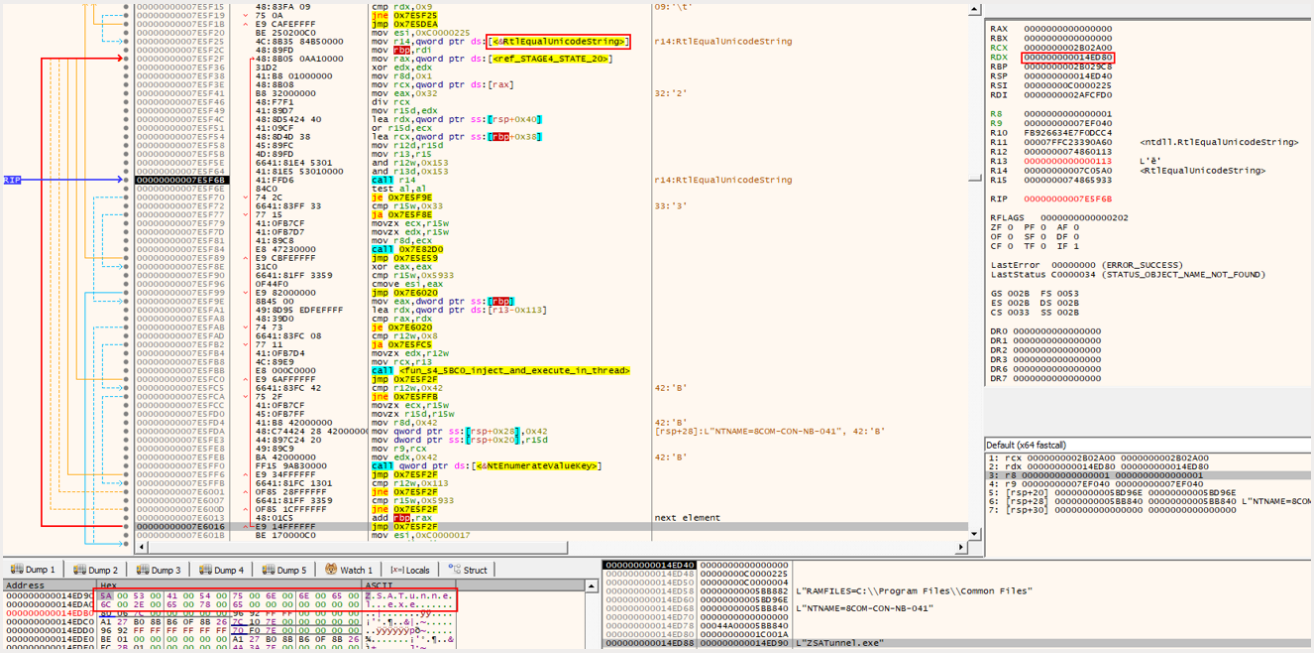


Figure 24: Iteration over process list and search for ZSATunnel.exe

When the malware detects `ZSATunnel.exe` in the running process list, it sets a global boolean flag in the binary's data section.

During the subsequent C2 communication setup, this flag is read back and used as an index into an array of C2 endpoints (Figure 25).

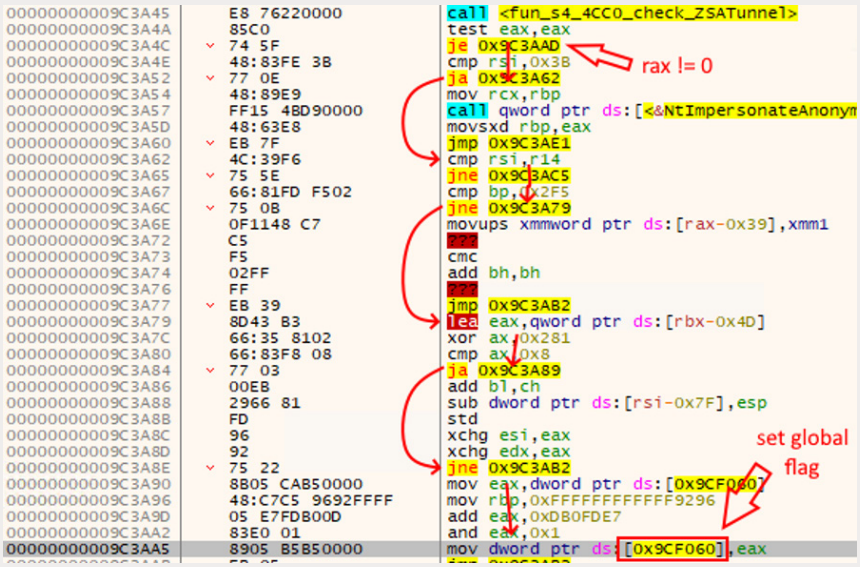


Figure 25: Setting the global flag after ZSATunnel.exe detection

If the flag is set, indicating the presence of Zscaler Client Connector, the malware selects a hardcoded IP address rather than the primary domain-based endpoint (see Figure 26).

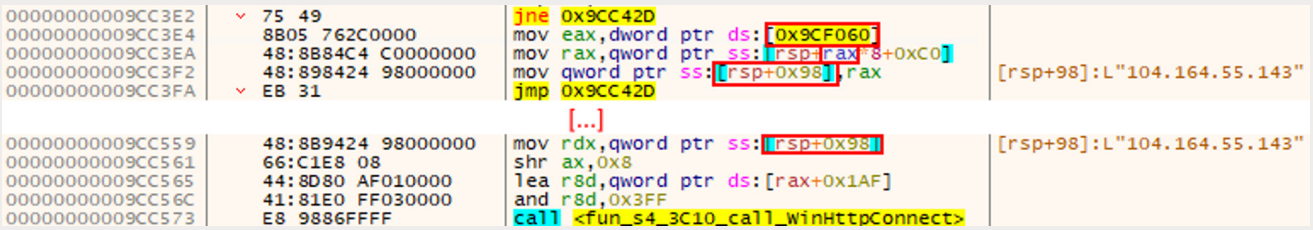


Figure 26: Using the global flag to use the plain IP address instead the DNS name to be passed to WinHttpConnect

This behavior strongly suggests that the threat actors are deliberately attempting to bypass Zscaler's DNS-based web filtering and SSL inspection policies by falling back to a raw IP address for C2 communication.

Single-Instance Enforcement via Named Event Objects

The malware implements a single-instance mechanism using a named event object with a dynamically generated GUID (e.g. `{67DAC128-FE43-4749-A247-F6FF08903052}`), visible in the session-specific `\Sessions\2\BaseNamedObjects\` namespace as shown in Figure 26. This ensures that only one instance runs per user session while allowing separate instances across different logon sessions. Figure 25 illustrates the detection logic: when `NtCreateEvent` returns `STATUS_OBJECT_NAME_EXISTS` (`0x40000000`), the duplicate instance calls `ZwSetEvent` to signal the original before terminating.

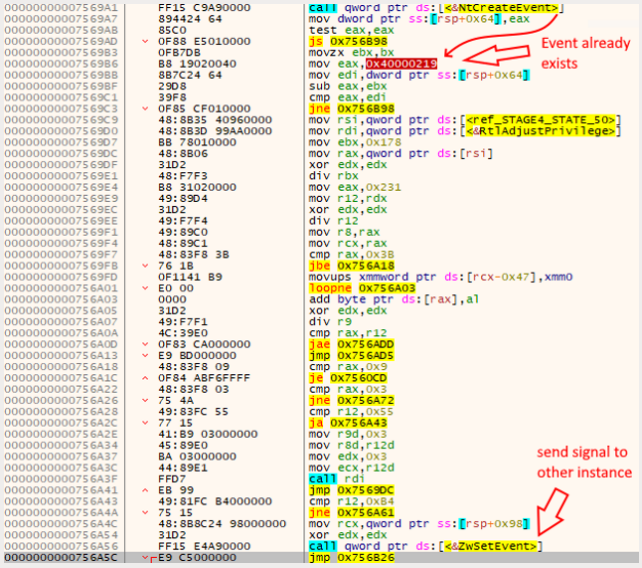


Figure 27: Named Event existence check

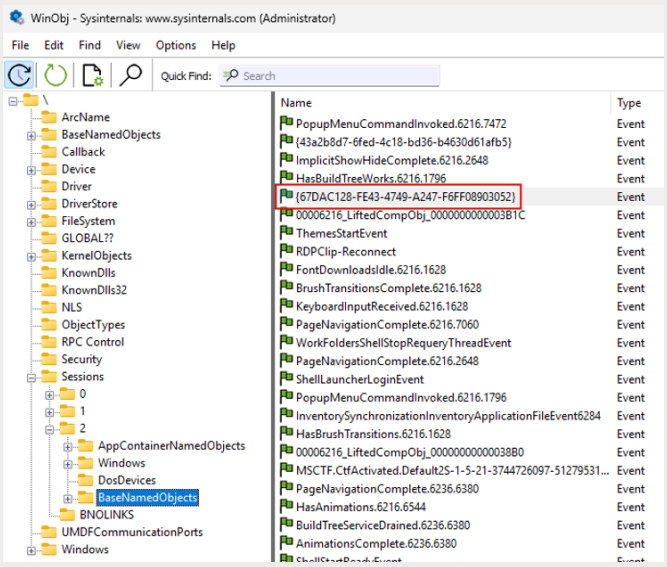


Figure 28: The created Event

C2 Communication Infrastructure

The malware generates unique URLs for each HTTP request using a custom pseudo-random number generator (PRNG). The algorithm combines system fingerprint data with an XORShift-based transformation that updates the seed after generating each URL character. Since the initial seed incorporates time-dependent values (such as `SystemTime` and process handles), every execution produces a different URL path, rendering static blocklist-based defenses ineffective.

The generated URL structure follows the pattern:

`e/rh/x1slvwn/hbnryr1/yenraboyy/m3a0w4a03wvx/ymxmzlsd/nzgk2q5ojsxeh1s/mvz.jpg`

This pattern mimics legitimate image requests to evade network-based detection.

The malware communicates with the C2 server using the WinHTTP API. The initial network request is an HTTP GET operation that downloads a payload with a `.jpg` extension. The downloaded content is encrypted using XTEA (Extended Tiny Encryption Algorithm).

The standard XTEA delta constant (0x9E3779B9) is present in the implementation, as shown in Figure 27, allowing initial algorithm identification.

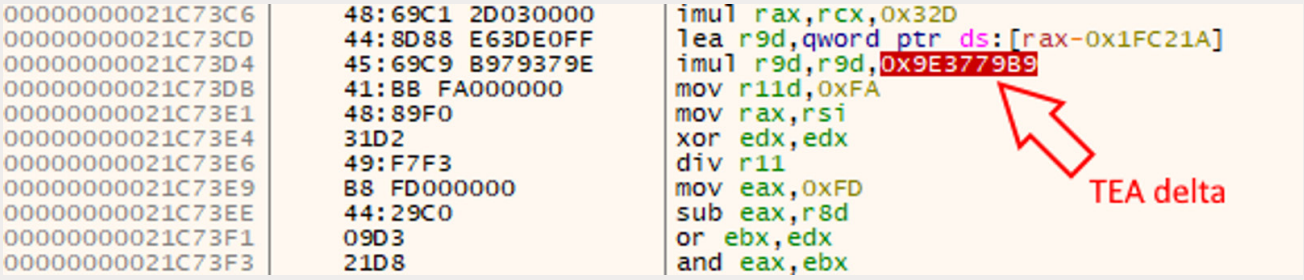


Figure 29: Part of the XTEA algortihm

Following decryption, the malware constructs another XOR-obfuscated API dispatch table for stage 5. Both the dispatch table and the decrypted payload memory regions are subsequently marked with executable permissions to prepare for execution.

Stage 5: Screen Capture and Exfiltration

During the analysis of stage 5, the attackers appeared to have dismantled their C2 infrastructure. We analyzed several similar samples and tested all extracted IP addresses and domains, including historical A-record entries, but were unable to locate any active servers.

Consequently, we only obtained a single stage 5 payload via HTTP GET during the initial discovery of C2 server addresses in stage 4. Since the HTTP downloader code explicitly disables SSL certificate validation, we configured a minimal Python web server to serve the previously downloaded „jpg“ payload and modified the system‘s hosts file to redirect the C2 domain to localhost. This setup allowed the malware to successfully download the payload, enabling behavioral analysis.

Screenshot Capture Mechanism

Stage 5 implements screen capture functionality using the Windows GDI32 and GDI+ libraries. The malware first creates a device context compatible with the screen using `CreateCompatibleDC` and allocates a bitmap buffer via `CreateCompatibleBitmap` sized to the screen dimensions. It then captures the screen content using `BitBlt` to copy the display buffer into the allocated bitmap. The captured bitmap is encoded to PNG format using GDI+ encoding functions, followed by base64-encoding of the PNG data. Finally, the base64-encoded screenshot is encrypted using the same XTEA implementation with a randomly generated key which is added to the header of the payload before transmission to the C2 server.

Figure 28 shows the state immediately after screenshot capture, with the base64-encoded PNG data stored in register R13.

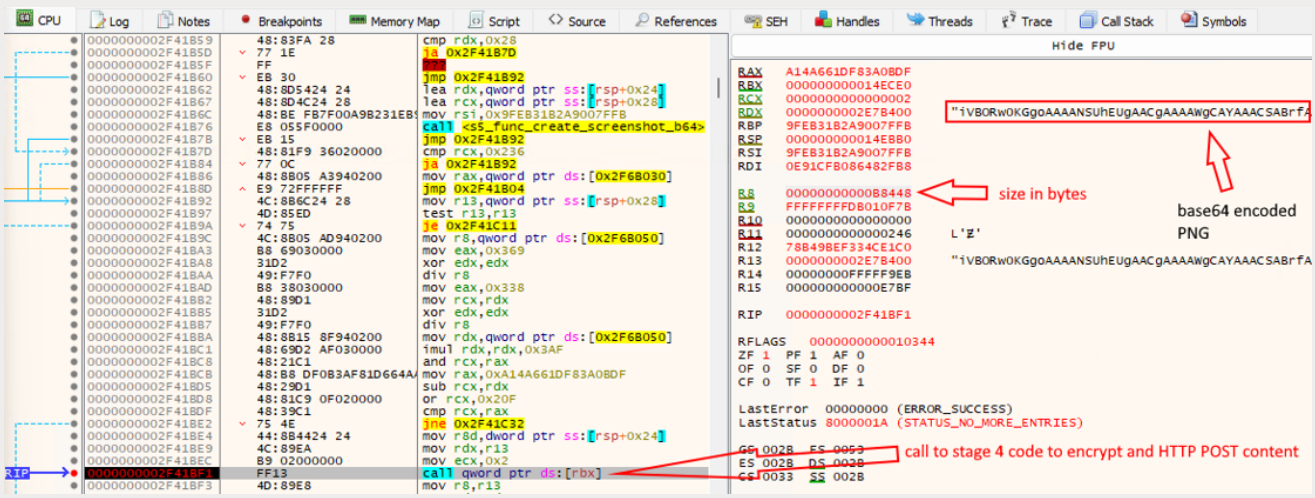


Figure 30: The screen capture is stored as base64-encoded string and encrypted and transferred in stage 4 code

Figure 29 demonstrates the decoding and rendering of this base64 string in CyberChef, confirming the screenshot content.

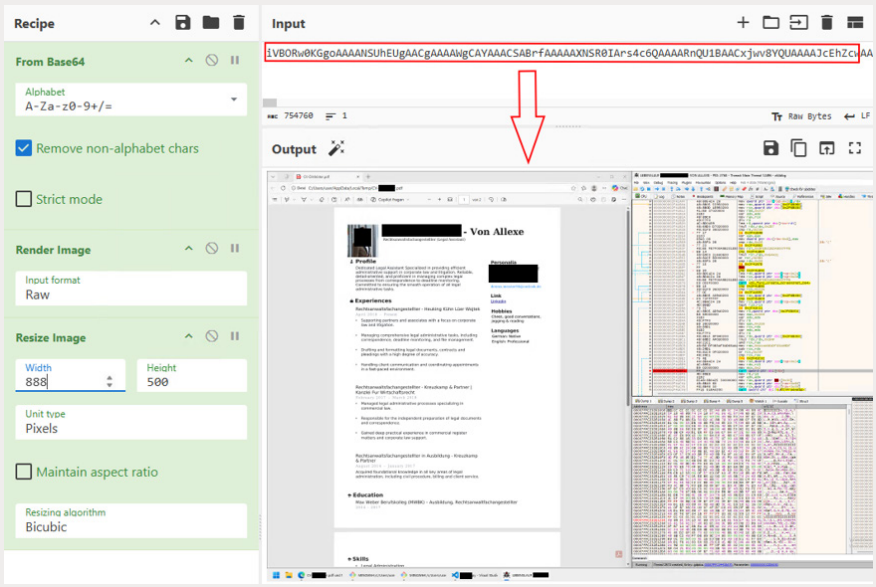


Figure 31: Rendering the screen capture in CyberChef shows the whole screen

The encrypted screenshot is then transmitted to the C2 server via an HTTP POST request by jumping back into the encryption and HTTP code of stage 4.

Conclusion

Our analysis concludes at this point due to the unavailability of the C2 infrastructure. Based on the observed behavior, we assess that the malware likely operates on a periodic polling mechanism, requesting new payloads at regular intervals (estimated every few minutes).

The screenshot exfiltration as the primary initial action is highly atypical and reveals significant insights into the attackers' operational model. Unlike conventional malware families, which typically execute immediate actions such as credential theft (infostealers), file encryption (ransomware), or automated command execution (commodity RATs), this malware implements a deliberate, manual reconnaissance-first approach. The threat actors appear to employ an attacker-in-the-loop operational model, where captured screenshots are reviewed to assess the value and context of each compromised system before committing additional resources.

This behavior pattern is characteristic of targeted intrusion operations rather than opportunistic cybercrime. The substantial development effort invested in this multi-stage loader, including custom XTEA encryption with non-standard modifications, sophisticated PRNG-based URL generation with system fingerprinting, multiple layers of XOR-obfuscated API dispatch tables, and the complete absence of automated post-exploitation actions, suggests a highly selective targeting strategy. The attackers are willing to sacrifice scale and automation for operational security and target discrimination.

The lack of immediate persistence mechanisms, credential harvesting, or lateral movement capabilities in the observed stages further reinforces this assessment. We hypothesize that subsequent payloads are deployed selectively based on the screenshot analysis, potentially including advanced post-exploitation frameworks such as Cobalt Strike, Sliver, or custom tooling tailored to high-value targets. This careful, staged approach minimizes the malware's footprint on non-critical systems while reserving sophisticated capabilities for confirmed targets of interest.

The operational sophistication, custom cryptographic implementations, and manual victim triage process indicate a well-resourced threat actor conducting focused intelligence collection or pre-positioning for targeted cyber operations. Organizations should treat any detection of this malware family as a potential indicator of targeted reconnaissance activity warranting immediate incident response and threat hunting procedures.

IOCs

Hashes

```
f847bda565a5f715c1833250c755dd609878a880b10b7ddde8e3245a8fbcd67f
0f5de795b2a1453dd87d64e4c683177c1ac98d559b4c9c255bbe8493ea10fabb
9993ae862e80930fc460454ed36f9811ab106eeb4731a6f319b3f9a09b284ae1
```

Domains

```
borsaplazma.com
ussuvsnation.com
```

IPs

```
194.58.47.198
104.164.55.143
```

User-Agent

```
Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/130.0.0.0 Safari/537.36 Edg/130.0.0.0
```

About 8COM

8COM is a managed security services provider founded in 2004 and headquartered in Neustadt an der Weinstraße, Germany. Its Security Operations Center (SOC), fully operated within Germany, is staffed by certified analysts who monitor IT and OT environments 24/7, detect threats at an early stage, and actively respond to incidents.

The SOC services are certified according to ISO 27001 based on the BSI IT-Grundschutz framework. With more than 120 employees – over half of whom work in round-the-clock SOC operations – 8COM combines deep technical expertise with extensive experience in operational incident response.
In addition, the company offers penetration testing and security awareness services, helping organizations and public authorities strengthen their resilience and maintain their ability to operate securely in the digital world.

For more information, visit: www.8com.de