Operation Neusploit: APT28 Uses CVE-2026-21509 | ThreatLabz

By Sudeep Singh, Roy Tay

Published: 2026-02-02 · Archived: 2026-04-05 19:44:07 UTC

Technical Analysis

In the following sections, ThreatLabz discusses the technical details of Operation Neusploit, including how the backdoors and stealers function and how they were deployed. We observed two variants of the attack chain. Both variants begin with a specially crafted RTF file that weaponizes CVE-2026-21509 and, after successful exploitation, downloads a malicious dropper DLL from the threat actor's server. There are two variants of this dropper DLL that deploy different components. We will discuss both the variants in the following sections.

Dropper Variant 1

The first dropper variant DLL is responsible for deploying a malicious Microsoft Outlook Visual Basic for Applications (VBA) project named *MiniDoor*. MiniDoor's primary goal is to steal the user's emails and forward them to the threat actor.

MiniDoor dropper DLL analysis

MiniDoor is a lightweight 64-bit DLL written in C++. The malicious functionality is implemented in the exported function: *UIClassRegister*. The DLL does not use code obfuscation and includes two variants of string decryption:

- Strings decrypted using a hardcoded 1-byte XOR key (0x3a).
- Encrypted strings prefixed with a 1-byte XOR key, which is then used to decrypt the strings.

Below are the key functionalities of this DLL.

- Creates a mutex with the static name *adjgfenkbe*.
- A 58-byte XOR key is first decrypted using a single-byte XOR key (*0x3a*). The decrypted string, *savntjkengkvnvblhfbegjbtnhkwrenvbjjnkhejhkwenrjvbejbrbrncbis*, is then used as a rolling XOR key to decrypt the Outlook VBA project stored (encrypted) inside the *.rdata* section of the DLL.
- Creates the directory structure *%appdata%\Microsoft\Outlook* recursively if it does not already exist.
- Writes the decrypted VBA project (*MiniDoor*) to *%appdata%\Microsoft\Outlook\VbaProject.OTM*.
- Sets the relevant Windows registry keys to downgrade Outlook security and allow the malicious project to load automatically each time Microsoft Outlook launches.

The table below shows the registry keys set by the dropper.

Subkey	Value Name	Value	Description
HKCU\Software\Microsoft\Office\16.0\Outlook\Security	Level	1	Enables all macros in Microsoft Outlook.
Software\Microsoft\Office\16.0\Outlook\Options\General	PONT_STRING	0x20	Disables the "Content Download Warning" dialog box.
Software\Microsoft\Office\16.0\Outlook	LoadMacroProviderOnBoot	1	Ensures macro provider loads when the Microsoft Outlook application starts.

Table 1: The registry keys set by the MiniDoor DLL dropper to steal email from Microsoft Outlook.

MiniDoor analysis

ThreatLabz named this VBA-based malware *MiniDoor,* as it appears to be a minimal version of *NotDoor* reported by Lab52. Similar to *NotDoor*, *MiniDoor* collects emails from the infected machine, but does not support the email-based commands implemented in *NotDoor*. Below are key functionalities of the Outlook VBA.

- Monitors the *MAPILogonComplete* event which occurs after the Outlook user has logged on. Once triggered, the macro sleeps for 6 seconds before iterating through four folders in the user's mailbox..
- Two pre-configured email addresses are hardcoded in the VBA macro by the threat actor:
  - ahmeclaw2002@outlook.com
  - ahmeclaw@proton.me
- The *SearchNewMessageAndHandle* method searches the following folders for existing emails.
  - Inbox
  - RssFeeds
  - Junk

    - Drafts
- The stealing functionality is implemented in the *ForwardEmail* method, which iterates over each folder's contents and, for each message that was not already forwarded:
    - Saves a local copy to *%TEMP%\temp_email.msg*.
    - Drafts a new email, attaches temp_email.msg, and sends the email to both configured recipient addresses.
- Sets the *DeleteAfterSubmit* property of the *mailItem* to *true* to ensure that no copy of the message is saved in the Sent folder after it is forwarded to the threat actor.
- For each Outlook message that is forwarded, the macro sets the *AlreadyForwarded* property to *Yes* to prevent the same message from being forwarded twice.
- Handles the *Application_NewMailEx* event (triggered when new emails are received) by forwarding the received email to the above-mentioned email addresses.

The complete *MiniDoor* macro code is available in the ThreatLabz GitHub repository.

Dropper Variant 2

In the second dropper variant, the infection chain is more complex and involves multiple stages. Similar to the first dropper variant, after successful exploitation of CVE-2026-21509, the attack chain downloads a tool that ThreatLabz named *PixyNetLoader*, which drops malicious components on the endpoint and sets up the Windows environment to start the infection chain.

PixyNetLoader analysis

The dropper DLL used in variant 2 of the attack chain is new and previously undocumented.

PixyNetLoader's string decryption mechanism is similar to the *MiniDoor* dropper DLL. Below are the key functionalities.

- Creates a mutex with the name *asagdugughi41*.
- Checks for the presence of *EhStoreShell.dll* at *%programdata%\USOPublic\Data\User\EhStoreShell.dll*.
- If *EhStoreShell*.dll is not found at location above, then the main dropper logic is invoked.
- All the embedded payloads are encrypted using a 0x47 byte long rolling XOR key: *shfioehh243t3dcwechortjbo6k7pjl8lop7ku45ht3u4grywefdyehriobjojko5k65iyh*. They are decrypted and dropped to the file system locations in the table below:

Location	**Size (in bytes)**
%programdata%\Microsoft OneDrive\setup\Cache\SplashScreen.png	0x39649
%programdata%\USOPublic\Data\User\EhStoreShell.dll	0x36200

Location	Size (in bytes)
%temp%\Diagnostics\office.xml	0xDE4

Table 2: Decrypted embedded payloads, including their file system drop locations and corresponding sizes.

- Uses *COM object hijacking* to establish persistence. *EhStorShell.dll* is the legitimate name for the Enhanced Storage Shell Extension DLL. By setting the Windows registry keys listed in the table below, PixyNetLoader ensures that the next-stage malicious shellcode loader DLL is loaded each time the *explorer.exe* process starts.

subKey	ValueName	Value
Software\Classes\CLSID\{D9144DCD-E998-4ECA-AB6A-DCD83CCBA16D}\InProcServer32	Null	%programdata%\USOPublic\Data\User\EhStoreShell.dll
Software\Classes\CLSID\{D9144DCD-E998-4ECA-AB6A-DCD83CCBA16D}\InProcServer32	ThreadingModel	Apartment

Table 3: Windows registry keys set by PixyNetLoader to ensure persistence.

- Executes the following command using the *CreateProcess* Windows API to set up a Windows scheduled task. This command leverages the previously dropped *office.xml* file to configure the scheduled task as shown below.

```
schtasks.exe /Create /tn "OneDriveHealth" /XML "%temp%\Diagnostics\office.xml"
```

The Windows scheduled task is named *OneDriveHealth* and configured to launch the command below exactly one minute after the task is registered. The OneDriveHealth scheduled task launches the following command:

```
%windir%\system32\cmd.exe
/c (taskkill /f /IM explorer.exe >nul 2>&1) & (start explorer >nul 2>&1) & (schtasks /delete /f /tn OneDriveHe
```

The complete *office.xml* Windows scheduled task configuration file is available in the ThreatLabz GitHub repository.

Shellcode loader analysis

The dropped DLL *EhStoreShell.dll* is loaded in the *explorer.exe* process. Its key functionality is to extract shellcode embedded using steganography in the file named *SplashScreen.png* (that was previously dropped) and execute it.

The string decryption in the *EhStoreShell.dll* is similar to the *MiniDoor* dropper DLL. In addition, all the API names are resolved at runtime using the DJB2 API hashing algorithm.

Below are the key functionalities:

- Loads the legitimate version of *EhStorShell.dll*.
- Resolves addresses for the following exports from the legitimate DLL:
    - DllCanUnloadNow
    - DllGetClassObject
    - DllRegisterServer
    - DllUnregisterServer
- Overwrites the export addresses in the malicious *EhStoreShell.dll* with the API addresses above to proxy the execution to the legitimate version of *EhStorShell.dll*. This is done to preserve the functionality of the COM service.

Conditional execution of malicious functionality

The *EhStoreShell.dll* executes its malicious logic only when both of the following conditions are met:

- Checks the host process that loaded the DLL. The malicious functionality is invoked only when the host process is *explorer.exe*. If the host process is not *explorer.exe*, then the code remains dormant.
- Checks whether the *Sleep()* API is short circuited (a common implementation used by several sandboxes) to detect the analysis environment. This check is implemented as shown below.
    - Calculates current timestamp by calling *std::chrono::steady_clock::now()*.
    - Calls *Sleep()* to delay execution by 3 seconds.
    - Calculates current timestamp again by calling *std::chrono::steady_clock::now()*.
    - If the difference between the current timestamp and the previous timestamp is greater than 2.9 seconds, only then it continues with the malicious activity. If the difference is less than 2.9 seconds, then the code assumes that the *Sleep()* API call has been tampered with.

PNG steganography and shellcode loader

Once all the checks pass, *EhStoreShell.dll* creates a new thread using *beginthreadex*. The thread start function performs the following actions:

- Decrypts the PNG path, *%programdata%\Microsoft OneDrive\setup\Cache\SplashScreen.png*, then expands environment variables to obtain the full file path.
- Uses steganography to extract the malicious shellcode from the PNG file.
    - Each pixel of the PNG image is represented by 4 bytes (1 byte per channel) for the red, green, blue, and alpha channels.
    - The Least Significant Bit (LSB) of each byte represents an encoded data bit, hence each byte of encoded data is stored within 8 bytes of image data (or 2 pixels)

    - The first 4 bytes of encoded data represents the payload size in little endian byte order and is followed by the cleartext payload itself.
- Creates a mutex named *dvyubgbqfusdv32*.

The complete code to extract the shellcode from the PNG file is available in the ThreatLabz GitHub repository.

The shellcode is executed by the *EhStoreShell.dll* via the following actions:

- Allocates executable memory using the native Windows API *NtAllocateVirtualMemory*.
- Copies the extracted shellcode into the newly allocated memory region.
- Transfers execution to the shellcode.

Shellcode analysis

The main purpose of this 64-bit shellcode is to load a .NET assembly embedded inside it. In order to load a managed assembly from native code, the shellcode uses the *CLR hosting* technique. Below are the key steps used to achieve managed code execution in-memory from unmanaged code.

- Loads the *mscoree.dll* and *oleaut32.dll* libraries.
- Initializes the .NET runtime by calling *CLRCreateInstance* (exported by *mscoree.dll*).
- Requests the *ICLRMetaHost* interface, selects the .NET version v4.0.30319, and initializes *ICorRuntimeHost* interface.
- Retrieves the application domain by calling I*CorRuntimeHost::GetDefaultDomain*, then queries this object to obtain the *_AppDomain* interface.
- Uses *SafeArrayCreate* and *SafeArrayAccessData* methods to copy 0xfc00 bytes of the embedded .NET assembly into the array.
- Calls *_AppDomain::Load_3* to load the .NET assembly passed via *SafeArray*, enabling in-memory execution of the .NET assembly.
- Retrieves the entrypoint of the .NET assembly and invokes it using *MethodInfo::Invoke_3*.

Covenant Grunt analysis

The embedded .NET assembly is a *Grunt* implant associated with the open source .NET *Covenant* C2 framework. In this sample, the implant uses the Filen API as a C2Bridge to communicate and receive tasks from the threat actor. This abuse of legitimate APIs was previously observed in other *Covenant Grunt* implants linked to APT28 by ThreatLabz and other researchers.

Strings in this sample are XOR-encoded with the hardcoded string *EIZ4EG2K8R* and then Base64-encoded. These include the domains for querying the Filen API, the Authorization Bearer Token, and Filen parent folder UUID (*fe644d8c-2601-46ea-bf7d-3db110aa08d4*).

Explore more Zscaler blogs
