

StrikeShark: a new campaign involving a custom SharkLoader and Cobalt Strike Beacon

SL securelist.com/strikes shark-campaign/120326

Fareed Radzi

June 24, 2026



Introduction

During our research of activity affecting a diplomatic organization in Indonesia, we uncovered a previously undocumented malware family that we have named **SharkLoader**. What initially appeared to be an isolated case quickly expanded into a broader campaign as we identified additional SharkLoader infections across multiple countries and sectors.

Our investigation revealed that SharkLoader serves as a loader designed to deploy Cobalt Strike Beacon on compromised systems. We observed the threat actor deploying SharkLoader through exploitation of internet-facing applications, including Microsoft Exchange, Microsoft SharePoint, and Openfire Server, as well as through malware-based delivery mechanisms.

Beyond the diplomatic entity in Indonesia, we identified related activity targeting government organizations in Taiwan, software development companies across multiple countries, and entities in other sectors located in Hong Kong, Lebanon, Syria, Colombia, North Macedonia, Nepal, Serbia, and more. The observed victimology suggests a campaign with broad geographic reach and a diverse target set rather than a narrow focus on a specific industry or region.

For now, we are tracking this activity as **StrikeShark**. Although the operators utilize several open-source post-compromise tools associated with Chinese-speaking developers, we have not identified direct code reuse, infrastructure overlap, or operational similarity to confidently attribute the activity to any known APT or cybercrime group. As a result, attribution remains preliminary and the campaign's ultimate objectives are still under research.

Initial infection

Our analysis of SharkLoader intrusions indicates that the threat actor employs multiple methods to gain initial access to victim environments. During our investigation, we observed two primary infection vectors: the exploitation of vulnerabilities in internet-facing applications and the deployment of custom dropper samples, some of which were disguised as legitimate software.

Exploitation of public-facing applications

In the incident affecting an Indonesian diplomatic entity, the threat actor exploited Microsoft Exchange vulnerabilities, including CVE-2021-26855 (ProxyLogon), to gain access to the target environment. Similar activity was observed in Taiwan, where software development organizations were compromised through exploitation of Openfire (CVE-2023-32315). In a separate incident affecting a Colombian organization, the threat actor exploited a GeoServer instance vulnerable to CVE-2024-36401.

Beyond these incidents, we identified additional exploitation activity targeting vulnerabilities in multiple internet-facing enterprise applications and network appliances including those listed below:

Remote Code Execution (RCE)

- Apache Shiro: CVE-2016-4437
- Hikvision Products: CVE-2021-36260
- Microsoft SharePoint: CVE-2021-27076
- Zimbra Collaboration Suite: CVE-2022-27925
- Microsoft Exchange Server: [CVE-2022-41082](#)
- F5 BIG-IP system: CVE-2023-46747
- Fortinet FortiOS: CVE-2024-21762
- React Server Components: [CVE-2025-55182](#)

Authentication Bypass

- Fortinet FortiOS: CVE-2022-40684

- Cisco IOS XE Web UI: CVE-2023-20198

As of the time of writing this article, we haven't obtained the exploits the attackers used. However, based on the vulnerabilities observed across multiple attacks, we assess with medium confidence that the threat actor primarily relies on publicly available proof-of-concept (PoC) exploits to gain initial access. All the vulnerabilities identified during our investigation have publicly available exploit code, including PoCs hosted on GitHub and other open-source platforms, suggesting the actor leverages existing offensive resources rather than develops custom exploit capabilities. The victim profile also indicates that the activity is largely opportunistic, affecting organizations across various industries, regions, and technology environments without a clear focus on a specific target set. Also, one of the IP addresses associated with the C2 domain was also observed conducting internet-wide scanning activity, potentially aimed at identifying and exploiting vulnerable internet-facing systems at scale.

Following exploitation, the attacker established persistence on compromised servers through the deployment of webshells. Although we were unable to recover the webshell files, a series of commands whose execution we observed in our telemetry along with the detection records of webshells strongly indicate their use for post-exploitation activities.

One of the earliest observed actions involved copying the legitimate Windows application *SystemSettings.exe* to a new location before executing it.

```
1 cd C:\Windows\ImmersiveControlPanel\  
2 copy SystemSettings.exe C:\ProgramData\  
3 cd C:\ProgramData\  
4 SystemSettings.exe
```

This application was later abused as part of a DLL sideloading chain used to launch SharkLoader, which in this scenario was hidden in the malicious *SystemSettings.dll* library. We suspect that this DLL along with malicious encrypted files, which we'll describe further, was uploaded through the webshell to the same directory as *SystemSettings.exe*.

In another case involving the exploitation of CVE-2021-27076, the threat actor launched *SystemSettings.exe* triggering the subsequent SharkLoader sideloading chain from different directories on the system, which suggests renewed operational activity in the victim environment. In some of the cases, they used security product vendor names as the directory names, allegedly to appear legitimate.

```
1 cd C:\ProgramData\KasperskyLab\  
2 dir  
3 .\SystemSettings.exe  
4 cd %APPDATA%  
5 dir  
6 cd kasperskylab  
7 dir  
8 .\SystemSettings.exe
```

Dropper-based distribution

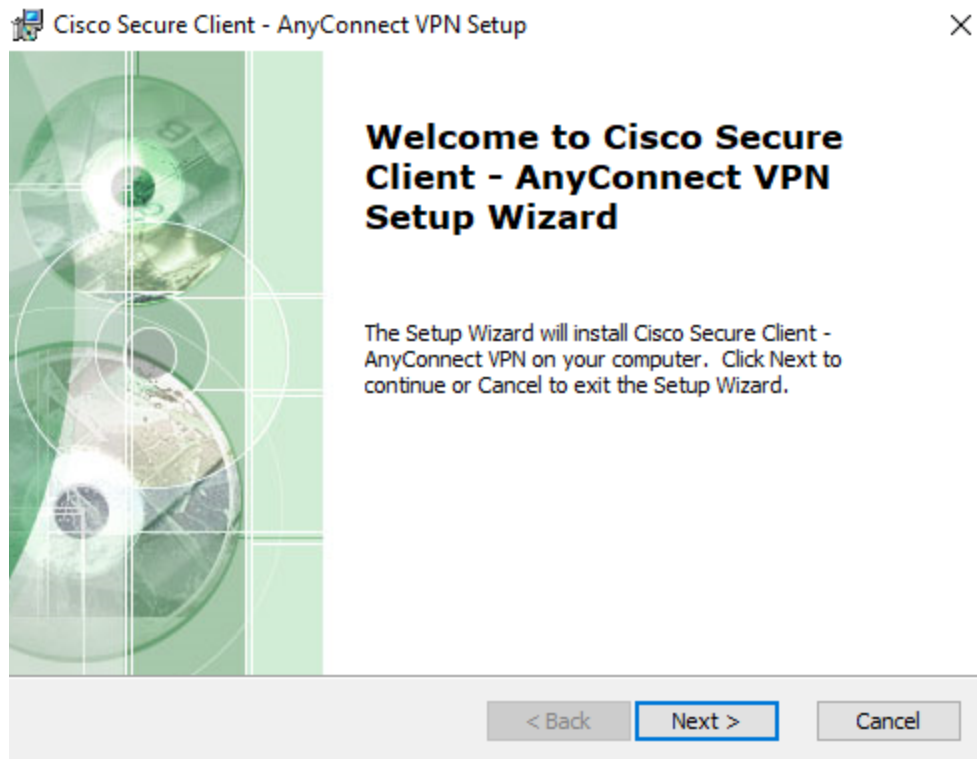
In several observed cases, the threat actor distributed SharkLoader through custom dropper executables masquerading as legitimate software installers or applications such as Google Update and Cisco AnyConnect. However, the exact delivery mechanism used to distribute these droppers remains unknown.

The observed dropper filenames include:

- GoogleUpdateStepup.exe
- AnyConnect-win-4.10.04071-predeploy-k9exe
- AutoUpdate.exe
- 319-pfd-8001-reva_traitement biologique_master.zip

In one of the samples we analyzed, the threat actor used a legitimate Cisco AnyConnect VPN installer as a lure. The custom dropper extracted zlib-compressed data embedded within its resource section, decompressed it into an MSI package, and wrote the file to %APPDATA%\reports\AnyConnect-win-4.msi. The MSI package was a legitimate Cisco AnyConnect VPN installer, which was subsequently executed via the ShellExecuteW API, making the user believe the custom dropper was a legitimate application.

While the Cisco AnyConnect installer was decompressed and executed, SharkLoader components were silently dropped into directories in %APPDATA% different from %APPDATA%\reports\ in the background, executing the malware loader once the installation process completes.



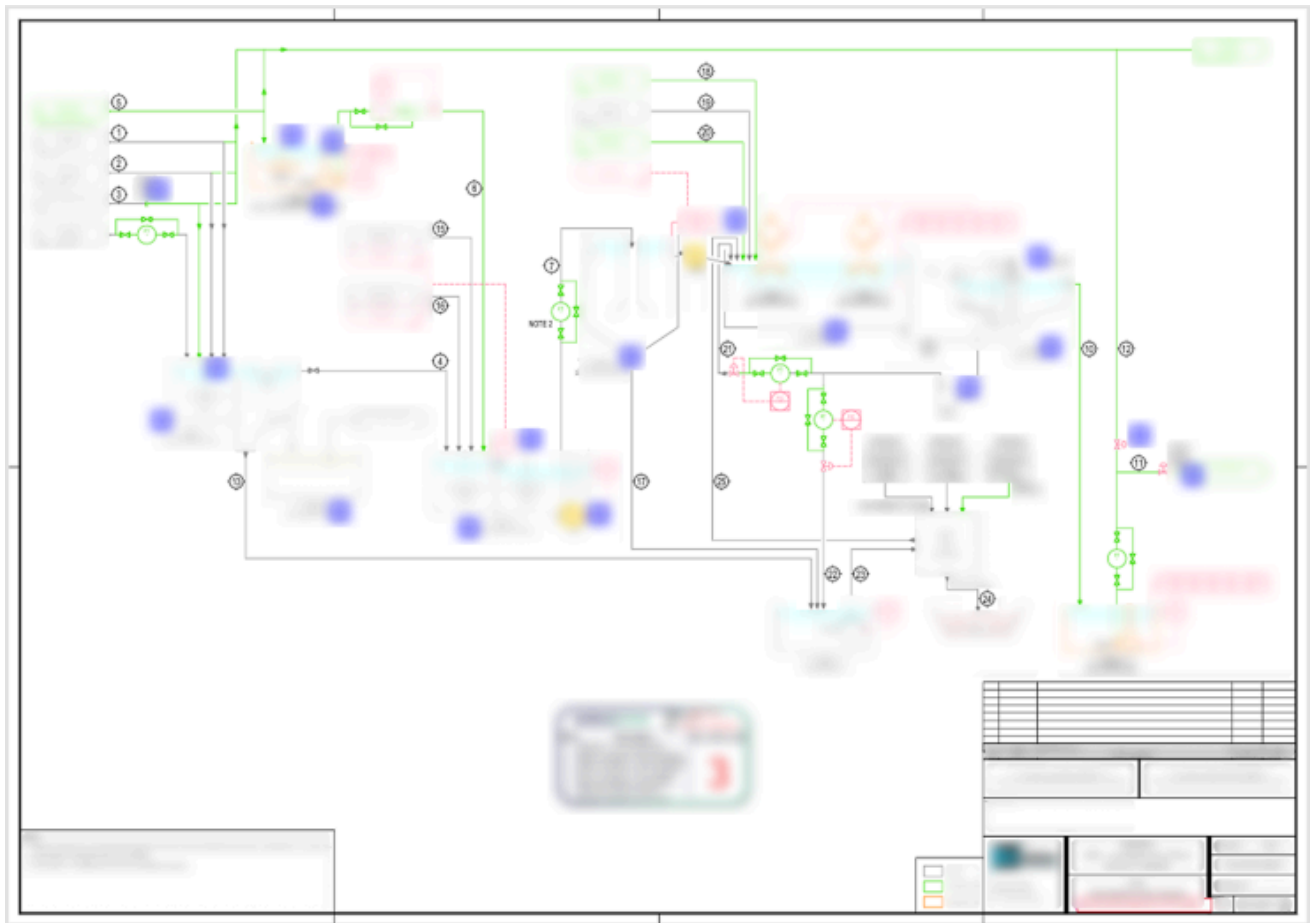
Malicious Cisco Secure Client installer

In addition to installer-themed lures, several SharkLoader droppers use decoy PDF documents to persuade victims to open the malicious file. However, not all samples employ this technique, as some droppers function solely as a delivery mechanism for SharkLoader without presenting any lure content.

Among the samples analyzed, most droppers write the decoy PDF to a subdirectory named `aswerf` within the `%TEMP%` directory, while others save the document directly to `%TEMP%`.

Analysing the sample shows the PDF files are stored within the dropper's resource section under the resource name `TELEMETRY` and are compressed with `zlib`. Upon execution, the dropper extracts and decompresses the embedded PDF, writes it to disk using the same filename as the dropper executable but with a PDF extension, and launches it via `cmd.exe /c` to display the decoy document to the victim.

The following are examples of PDF documents extracted and displayed by the droppers during the deployment of SharkLoader.



Lure document 1. The document appears to be related to a biological treatment process and was produced by an engineering consultant

QJ

中华人民共和国航天工业部部标准

QJ1003—86

液体火箭发动机设计程序

1986—09—01发布

1986—12—01实施

中华人民共和国航天工业部 批准

Lure Document 2. Translated title: Liquid Rocket Engine Design Program

In one dropper sample, discovered on a machine located in Lebanon (MD5: 1F65544978B8EA0E745E573B8EE9684B), the dropper extracts and decompresses `SystemSettings.dll` from zlib-compressed data embedded within the binary and writes it to `%APPDATA%\xwreg`. It also extracts and decompresses `DscCoreR.mui` and `SyncRest.dat` from resources named `VAULTSVCD` and `UMRDPRDAT`, respectively, and writes them to the same directory.

```

15: v2 = 0;
16: memset(pszPath, 0, 0x200u);
17: if ( Destination )
18: {
19:     sprintf(pszPath, L"%s\\%s", Destination, L"SystemSettings.dll");
20:     if ( PathFileExist(pszPath) )
21:         return 1;
22:     v4 = 2;
23:     v5 = operator new(0x200u);
24:     v6 = operator new(0x200u);
25:     v7 = v5;
26:     if ( v5 )
27:     {
28:         memset(v6, 0, v5);
29:         v7 = Decompress(v6, v5);
30:         if ( v7 )
31:         {
32:             while ( 1 )
33:             {
34:                 if ( v7 == 0xFFFFFFFF )
35:                     break;
36:                 v8 = v7;
37:                 v9 = operator new(0x10000 * v8);
38:                 if ( v9 )
39:                 {
40:                     if ( v9 )
41:                     {
42:                         break;
43:                     }
44:                     memset(v9, 0, v9);
45:                     v10 = Decompress(v9, v9);
46:                     if ( v10 )
47:                     {
48:                         goto LABEL_32;
49:                     }
50:                     sub_34000000(L"Error During Apply New Memory");
51:                 }
52:             }
53:         }
54:     }
55: }

```

The dropper extracts SystemSettings.dll from the binary and retrieves encrypted components from the resource section

The dropper then copies the legitimate SystemSettings.exe application from C:\Windows\ImmersiveControlPanel to the target location to facilitate DLL sideloading. Across other SharkLoader dropper samples analyzed, the malware components were observed being written to either %APPDATA%\xwreg or %APPDATA%\xgdf.

SharkLoader installation

SharkLoader is composed of multiple components that work together to load and execute the final implant, a Cobalt Strike Beacon.

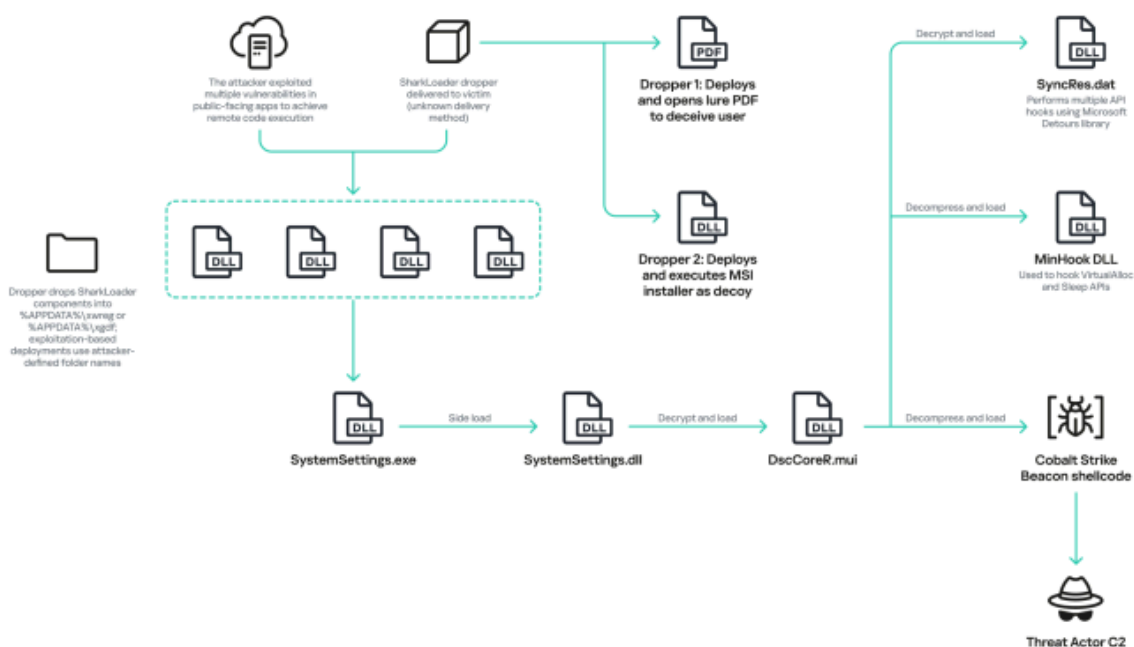
Filename	Description
SystemSettings.exe	Legitimate Windows application abused for DLL side-loading of the malicious DLL SystemSettings.dll.
SystemSettings.dll	Main malicious SharkLoader DLL responsible for the core loader functionality.
DscCoreR.mui	An encrypted module that contains an embedded Cobalt Strike Beacon and the MinHook library. This module loads SyncRes.dat, installs a couple of API hooks, and executes the Beacon directly in memory.
SyncRes.dat	An encrypted DLL that is used to install multiple API hooks.

While the majority of SharkLoader samples analyzed rely on the sideloading of SystemSettings.dll, other variants leverage alternative DLL side-loading targets, including msedge.dll, PrintDialog.dll, and miracastview.dll, each of them leveraging a corresponding legitimate application.

Across the different variants examined, the encrypted modules were also observed using a variety of filenames, including:

- 1 GameInputInboxs32.mui
- 2 diagerr.xml
- 3 NtfsLog.etl
- 4 Ignored.Dat
- 5 VistaCompat.nls

The SharkLoader execution flow is as follows:



SharkLoader infection chain observed in the StrikeShark campaign

In the dropper-based infections, after deploying all required SharkLoader components, the dropper creates two scheduled tasks through the Windows Task Scheduler COM interfaces. Task names:

- OneDrive Standalone Update Task-S-1-5-21-4165425321-4153752593-2322023643-1000

- MicrosoftUpdateTaskUserS-1-5-32-2456537112-101246289-228944324-1000

Both tasks are configured to execute the copied `SystemSettings.exe` from the malware's working directory (for example, `%APPDATA%\xwreg` or `%APPDATA%\xgdf`), triggering the side-loading of the malicious `SharkLoader DLL`.

The first scheduled task uses a time-based trigger that executes every five minutes, providing long-term persistence.

The second task is configured to execute every second, likely to ensure immediate execution of `SharkLoader` following deployment.

After a delay of approximately 1.5 seconds, the dropper removes the second scheduled task by using the Task Scheduler COM interfaces, leaving the first task in place to maintain persistence on the system.

SharkLoader DLL – Main implant

For the detailed analysis of the infection chain, we'll focus on the `SharkLoader` components deployed by a malicious dropper named 一种异常状况的截图（包括操作系统和输入法版本）.pdf.exe (MD5: 24FCEBDEECBA65004FDB0923763D74FD), which was identified in a campaign targeting a government entity in Taiwan.

Filename	MD5
SystemSettings.exe	D98F568496512E4F98670C61C97CB07A
SystemSettings.dll	AA3086BE652C8B20B0B29B2730D57119
DscCoreR.mui	A514D1BB62D7916475946FE7C07AC0AA
SyncRest.dat	9CBD560F820C95D7C38342CD558CB5C6

“PerfectDLL Hijacking” technique

Once the malicious DLL is loaded, `SharkLoader` implements a technique commonly referred to as “Perfect DLL Hijacking” and originally described by a security researcher named Elliot Killick on his [blog](#). The purpose of this technique is to bypass the Windows loader lock and safely create a malicious thread via the `CreateThread` API without risking a deadlock.

According to [Microsoft's Dynamic-Link Library Best Practices](#), the Windows loader holds a synchronization object known as the “loader lock” while executing the `DllMain` function. This

mechanism ensures that only one thread can perform DLL loading and initialization operations within a process at any given time. As a result, invoking APIs such as `CreateThread` or `LoadLibrary` from within `DllMain` can lead to deadlocks because the loader lock remains held throughout the execution of the function.

To avoid this issue, `SharkLoader` manipulates the process's internal loader state to release the loader lock before invoking `CreateThread` from the `DllMain` execution path. By doing so, it attempts to execute its malicious code without triggering the loader-related deadlocks that can occur when threads are created while the loader lock remains held.

```
16 // Resolve LdrpLoaderLock (internal loader lock address)
17 if ( ResolveLdrpLoaderLockAddress() )
18 {
19     // Resolve LdrpWorkInProgress (internal loader state variable)
20     pLdrpWorkInProgress = ResolveLdrpWorkInProgressAddress();
21     if ( pLdrpWorkInProgress )
22     {
23         // Release loader lock
24         LeaveCriticalSection(pLdrpLoaderLock);
25
26         // Signal loader completion event
27         fn_SetEvent(loaderCompletionEvent, &eventSignalData);
28
29         // Modify loader state
30         _InterlockedDecrement64(pLdrpWorkInProgress);
31
32         // Spawn malicious thread after loader lock released
33         hThread = CreateThread(0, 0, MainMalicious, 0, 0, &ThreadId);
34         hThread_1 = hThread;
35         if ( hThread )
36         {
37             WaitForSingleObject(hThread, 0xFFFFFFFF);
38             CloseHandle(hThread_1);
39             return 1;
40         }
41     }
42 }
```

Implementation of the Perfect DLL Hijacking technique to bypass the Windows Loader Lock

Based on the code, `SharkLoader` first resolves the addresses of several undocumented loader structures within `ntdll.dll`, including:

1. `LdrpLoaderLock`: the critical section object used by the Windows loader to synchronize module loading and initialization operations
2. `LdrpWorkInProgress`: an internal loader state variable that tracks whether module initialization is currently in progress

After locating these structures, `SharkLoader` forcefully releases the loader lock by invoking `LeaveCriticalSection` on `LdrpLoaderLock`. It then decrements the value of

LdrpWorkInProgress with InterlockedDecrement64, effectively marking the initialization process as complete.

Finally, the malware signals the loader completion event via SetEvent before creating a new thread to execute its malicious functionality. As a result, these actions manipulate the loader's internal state and cause Windows to treat the DLL initialization process as having completed successfully. This allows SharkLoader to continue execution after forcefully releasing the loader lock, despite still operating from within the DllMain execution path.

Decryption and loading of >DscCoreR.mui

As shown in the previous section, the loader creates a new thread after escaping the Windows loader lock. This thread subsequently spawns a second thread responsible for decrypting and reflectively loading the encrypted file, DscCoreR.mui.

The routine first reads the encrypted file into memory and extracts the first 16 bytes to use as the Blowfish decryption key. It then initializes the Blowfish cipher by using custom P-array and S-box constants embedded in the loader and decrypts the file in ECB mode with the extracted key. Once decryption is complete, the resulting PE file is reflectively loaded into memory and executed without being written to disk.

00000000:	1B 50 F9 2D CA 29 7F 66 09 0E B2 37 A6 E0 FD C0	P . -) . f . . . 7 . . .	Key
00000010:	1D 79 C0 FD D2 DF 14 B1 01 40 6A DC FB F1 EA A8	y @ j	
00000020:	09 AA 09 9E 1E 5C 65 B6 19 9F 46 89 39 B2 8A F3	 \ e . . . F 9 . . .	
00000030:	AD 09 47 68 BA 19 9D 15 EF A1 2D 81 5C 2D 3C 50	. . . 6 h - . \ - < P	
00000040:	62 1D 88 2D 57 4F 42 33 FA F4 27 80 5F 22 A9 A2	b . . - W 0 B 3 . . . ' . - "	
00000050:	9C AB E6 56 D4 66 AB 57 36 61 38 BA 47 67 CB A8	. . . V f W 6 a 8 G g . . .	
00000060:	86 E7 29 0F 73 F1 87 05 30 23 64 BD 8D C5 81 10	. . .) . s . . . 0 # d	
00000070:	C1 19 00 C6 82 A2 93 F7 0F 36 6A 49 BA B0 80 81	 6 j I	
00000080:	EE E9 FD 89 1C 1E FE F6 57 FC 80 5C F0 89 40 7B	 W . \ . _ @ {	Encrypted PE
00000090:	60 C2 48 B7 7A 26 AC 03 53 0A 2D FC B5 1C 0E E7	. . . H z & . S . -	
000000A0:	D0 AB D2 15 01 74 C0 8B 16 31 B4 6A 45 8D D6 F3	 t . . . 1 j E . . .	
000000B0:	3F 5E E3 1D 94 DE 8A D3 80 97 34 A7 6D F9 43 E7	? ^ 4 m C	
000000C0:	F6 FC 84 1E 1A 40 10 00 B3 79 C8 56 2C C1 C6 FB	 @ . y V , ...	
000000D0:	53 AC 81 8F 9F 76 06 89 A4 CA FD AD CB 19 F7 A5	S . . v	
000000E0:	8C C5 2D 14 EE 8E FA E4 E9 2C 6A 7E 65 4D 3D 80		

Structure of the encrypted DscCoreR.mui file containing the 16-byte Blowfish key bytes followed by the encrypted PE bytes

The decrypted DscCoreR.mui file is a packed PE file with its MZ header removed, likely as an anti-analysis measure. After decryption, SharkLoader processes the PE image by parsing its headers, allocating memory for the image, mapping its sections, applying relocations, resolving imported functions, and setting the appropriate memory protections. Once the in-memory PE loading process is complete, the main loader, SystemSettings.dll, transfers execution to the entry point of the mapped image, which contains the packer stub.

The stub then unpacks the protected code, invokes the DLL's DllMain function, and returns execution to SystemSettings.dll. Finally, SystemSettings.dll calls the exported function

SetUserProcessPriorityBoost from the mapped DLL, triggering execution of the fully unpacked next-stage DLL.

DscCoreR.mui and SyncRes.dat DLLs

Within the decrypted and unpacked DscCoreR.mui code, the malware proceeds to load and decrypt a second encrypted file, SyncRes.dat, before reflectively loading the resulting DLL into memory.

The mapped DLL installs multiple API hooks by using Microsoft Detours, which will be discussed in the next section.

After mapping and loading SyncRes.dat for API hooks, the DscCoreR.mui performs installation of the Vectored Exception Handler (VEH) and then creates a thread in a suspended state that is later used to execute the Cobalt Strike Beacon shellcode. Additionally, to facilitate additional API hooks, it decompresses and loads the [MinHook](#) library and uses it to install hooks on the VirtualAlloc and Sleep APIs.

The DscCoreR.mui then decompresses the Cobalt Strike Beacon shellcode into the memory region associated with the suspended thread and then the suspended thread is resumed, resulting in execution of the beacon.

Decryption and loading of SyncRes.dat

To decrypt SyncRes.dat, the malware extracts a 16-byte AES-128 key and a 16-byte initialization vector (IV) directly from the file itself. The first 16 bytes of the file contain the AES key, while the subsequent 16 bytes contain the IV. The remaining file content consists of AES-encrypted data, which is decrypted using the extracted key and IV. Once decrypted, the resulting data reveals a PE image with its MZ header removed, similar to DscCoreR.mui.

00000000:	72 7B 0C 79 44 FF 1B B0	72 7B 0C 79 44 FF 1B B0	r{ yD...r{ yD...	Key
00000010:	72 7B 0C 79 44 FF 1B B0	72 7B 0C 79 44 FF 1B B0	r{ yD...r{ yD...	IV
00000020:	53 12 2B CF 35 6B 7A BB 9C 11 D5 50 EF 85 20		S + 5k z...P...	Encrypted PE
00000030:	00 17 57 85 5F 77 49 B7 8A C7 B4 82 E6 E1 0B 2F		W _wI... /	
00000040:	34 E4 71 CF 15 84 DD 37 57 81 95 1E F5 40 EB AB		4 q...7W...@...	
00000050:	5E 55 26 0E EB 81 AB 65 E8 86 A7 12 9B 6A 47 F4		^U&...e...jG...	
00000060:	6C 6A 71 73 7E B8 42 43 B9 D3 F2 35 2C 76 5D 40		ljqs~ BC...5,v]@	
00000070:	6E 72 F2 F2 A0 B3 98 AB D5 B4 CA 9E 89 20 C0 6D		nr... ..m	
00000080:	F4 56 DC BD 54 02 86 58 60 F1 A7 37 03 5F DD A9		.V..T..X'. 7.._	
00000090:	A6 D9 19 94 18 4A 9D 6F 53 61 8F AD F4 6A 32 01		...J oSa...j2	
000000A0:	F4 29 DE 47 2B E2 CA 4E E9 28 E9 1A F5 91 E8 A8		.) G+..N.(.....	
000000B0:	C3 4C 70 8D C8 F2 32 A0 B4 89 FB BC BF 45 2A 11		Lp...2...E*	
000000C0:	EB 2C AA 55 88 22 92 C8 4C 7C 33 C5 78 48 11 06		., U"...L 3 xH...	
000000D0:	2E 7D B8 9B 13 7E EC A4 8F EB C0 0C AE 59 60 D4		.} ~...Y'	

Structure of the encrypted SyncRes.dat file showing the AES key, IV, and encrypted PE bytes

Similar to the decrypted DscCoreR.mui module, the decrypted SyncRes.dat file is also protected by an unknown custom packer. After decryption, the loader reflectively loads the PE

image before transferring execution to the module's entry point.

The entry point contains a packer stub responsible for unpacking the protected code in memory. Once the unpacking routine is complete, the malware invokes a specific exported function named `StartEngineData`, which serves as the primary execution routine of the third-stage DLL.

Before continuing with the `DscCoreR.mui` analysis, we will first discuss `SyncRes.dat`.

SyncRes.dat decrypted DLL: Multiple API hooks

The decrypted and unpacked `SyncRes.dat` DLL is primarily responsible for installing multiple Windows API hooks by using the Microsoft Detours library. After attaching all detour hooks, it calls `DetourTransactionCommitEx` to apply them in one commit.

The following table lists the hooked Windows APIs and their corresponding hook handler functions.

Hooked Windows APIs	Detour function description
CreateProcessA	• Saves all original <code>CreateProcessA</code> parameters for use in the parent process (PPID) spoofing routine. • Creates a new thread that executes the process creation routine responsible for PPID spoofing. Falls back to the original <code>CreateProcessA</code> if the thread creation fails. • Identifies an <code>svchost.exe</code> process that has the same security context as the current <code>SharkLoader</code> process. • Builds an extended startup attribute list to set the selected <code>svchost.exe</code> as the spoofed parent. • Calls the original <code>CreateProcessA</code> with the modified parent attribute. As a result, any new process created by the current process (primarily from the Cobalt Strike beacon) is spawned under <code>svchost.exe</code> instead of the current module process.

CreateProcessW	• Saves all original CreateProcessW parameters for use in the PPID spoofing routine, which is executed through an APC-based mechanism rather than a dedicated thread compared to the CreateProcessA API hook. • Schedules a delayed process creation (10 microseconds) through APC execution using CreateWaitableTimerW and SleepEx. The timer callback performs the svchost.exe PPID spoofing logic, similar to the CreateProcessA spoofing routine. As a result, new processes created via CreateProcessW by the current process (primarily from the Cobalt Strike beacon) are launched under svchost.exe through an APC-based execution mechanism
OpenProcessToken	• Once hooked, the malware initializes jitasm to construct a direct syscall stub for NtOpenProcessToken at runtime. • Invokes NtOpenProcessToken through the constructed direct syscall stub, redirecting the original API (OpenProcessToken) call flow.
AdjustTokenPrivileges	Redirects the API call to a direct NtAdjustPrivilegesToken syscall stub constructed by jitasm.
OpenProcess	Redirects the API call to a direct NtOpenProcess syscall stub constructed by jitasm.
WriteProcessMemory	Redirects the API call to a direct NtWriteVirtualMemory syscall stub constructed by jitasm.
NtCreateUserProcess	Redirects the API call to a direct NtCreateUserProcess syscall stub constructed by jitasm.

LoadLibraryA

- Redirects the API call to a function that resolves LdrLoadDll API using a ROR13-based API hashing algorithm.
- Uses the original parameters to invoke LdrLoadDll directly.
- If LdrLoadDll resolution or invocation fails, uses CreateTimerQueue and CreateTimerQueueTimer to schedule a 10-millisecond delayed execution of the original LoadLibraryA, with CreateEventW used for synchronization.

GetModuleHandleA

- Redirects the API call to a custom function that resolves the module base address through the following steps:
 - Enumerates loaded modules within the current process using CreateToolhelp32Snapshot, Module32FirstW, and Module32NextW.
 - Compares each enumerated module name with the module name provided in the API parameter.
 - Returns the module base address if a match is found.
- Falls back to the original GetModuleHandleA API if the custom resolution routine fails.

GetModuleHandleW

Similar approach to the GetModuleHandleA API hooks above.

GetProcAddress	• The original GetProcAddress parameters are passed to the hook handler. • The hook handler computes a Murmur32 hash of the requested function name. • The hook handler parses the module's PE structure and locates the export table. • Each exported function name is hashed using the same Murmur32 algorithm and compared against the previously generated hash. • If a hash match is found, the corresponding function address is returned. If no match is found, the call falls back to the original GetProcAddress.
LoadLibraryExA	The hook handler redirects the API call to its original address. In short, the hooked LoadLibraryExA calls the original LoadLibraryExA function.
VirtualAllocEx	Redirects the API call to a direct NtAllocateVirtualMemory syscall stub constructed by jitasm.
VirtualProtectEx	Redirects the API call to a direct NtProtectVirtualMemory syscall stub constructed by jitasm.
VirtualProtect	Redirects the API call to a direct NtProtectVirtualMemory syscall stub constructed by jitasm.
ResumeThread	Redirects the API call to a direct NtResumeThread syscall stub constructed by jitasm.
GetThreadContext	Redirects the API call to a direct NtGetContextThread syscall stub constructed by jitasm.

OpenThread	Redirects the API call to a direct NtOpenThread syscall stub constructed by jitasm.
NtCreateThread	Redirects the API call to a direct NtCreateThread syscall stub constructed by jitasm.
NtCreateThreadEx	Redirects the API call to a direct NtCreateThreadEx syscall stub constructed by jitasm.
NtQueueApcThread	Redirects the API call to a direct NtQueueApcThread syscall stub constructed by jitasm.
NtQueueApcThreadEx	Redirects the API call to a direct NtQueueApcThreadEx syscall stub constructed by jitasm.
ExpandEnvironmentStringsA	The detour redirects the API to a custom function that creates a new thread. That thread executes a routine that calls the ExpandEnvironmentStringsA API.
CreateFileMappingA	- The detour redirects the API call to a custom function that creates a new thread. Within the thread, it initializes thread-pool and timer objects, sets a threadpool timer for 10 ms and a waitable timer for 0.1 ms, then calls CreateFileMappingNumaA. - If thread creation fails, CreateFileMappingNumaA is called directly without creating a thread.
MapViewOfFile	The detour redirects the API call to a custom function that creates a new thread. The thread runs a similar thread-pool and timer setup to the previous function, resolves MapViewOfFileEx via GetProcAddress, calls it with zeroed arguments, and stores the return value.

UnmapViewOfFile	• The detour redirects the API to a function that tries to run the unmap (same API) in a new thread. • The thread creates an event and timer queue, schedules a callback after 10 ms to call UnmapViewOfFile and signal the event, then waits and cleans up. • If thread creation fails, it calls UnmapViewOfFile directly.
NtMapViewOfSectionEx	Redirects the API call to a direct NtMapViewOfSectionEx syscall stub constructed by jitasm.
NtCreateNamedPipeFile	Redirects the API call to a direct NtCreateNamedPipeFile syscall stub constructed by jitasm.
NtReadFile	Redirects the API call to a direct NtReadFile syscall stub constructed by jitasm.
NtWriteFile	Redirects the API call to a direct NtWriteFile syscall stub constructed by jitasm.
EtwEventWrite	The detour redirects EtwEventWrite to a stub that always returns 1, which prevents ETW logging.
EventWriteEx	The detour redirects EventWriteEx to a function that always returns 0, which prevents ETW logging.
EventWrite	The detour redirects EventWrite to a function that always returns 0, which prevents ETW logging.

Upon completing the installation of API hooks via the decrypted SyncRes.dat, the DscCoreR.mui DLL proceeds with the remaining functions, which are discussed below.

VEH registration and access violation handling

Following the installation of the API hooks, the malware registers a Vectored Exception Handler (VEH) to monitor exceptions generated during runtime. The handler specifically checks for access violation exceptions (0xC0000005). When such an exception occurs, it retrieves the faulting memory address from the exception record and calls `VirtualProtect` to restore read, write, and execute (RWX) permissions to the corresponding memory page before resuming execution.

During our analysis, no access violations were observed. It is possible that this mechanism is intended to handle access violations that may occur under specific runtime conditions.

Thread creation for Cobalt Strike Beacon execution

The malware creates a new thread in a suspended state that is intended to execute the Cobalt Strike Beacon shellcode. The thread entry point is configured to point to a memory buffer that will later contain the beacon shellcode.

At this stage, the buffer does not yet contain the actual Cobalt Strike Beacon shellcode. Instead, the thread is created in a suspended state so that the malware can prepare and inject the shellcode into the buffer before execution. Once the beacon payload has been written into the buffer, the malware resumes the suspended thread using the `ResumeThread` API, which triggers the execution of the Cobalt Strike beacon.

MinHook DLL, API hooking, and Cobalt Strike beacon

After creating the suspended thread for beacon execution, the malware decompresses a zlib-compressed MinHook PE file embedded within `DscCoreR.mui`. The MinHook library is used to install API hooks for the `VirtualAlloc` and `Sleep` functions. Once the MinHook DLL is decompressed and loaded into memory, the malware resolves the exported functions `MH_Initialize` and `MH_CreateHook`, which are then used to install hooks on the `VirtualAlloc` and `Sleep` APIs.

After the hooks are installed, the malware invokes a function that decompresses a zlib-compressed Cobalt Strike Beacon shellcode embedded within the malware. The function first decompresses the shellcode into a temporary buffer and then allocates executable memory using `VirtualAlloc` with RWX permissions. The decompressed beacon is subsequently copied into the allocated memory region.

Because the `VirtualAlloc` API has already been hooked at this stage, the hook handler captures the address and size of the allocated memory used to store the beacon shellcode. The hook records the addresses and sizes of the first three successful memory allocations

and stores these values in global variables to track specific memory regions allocated during execution. These tracked regions are associated with memory buffers used by the Cobalt Strike Beacon during runtime.

The second hook, on the Sleep API, is used when Cobalt Strike Beacon calls Sleep, such as during beacon sleep intervals. It temporarily modifies the memory protection of the tracked allocation regions by using VirtualProtect, changing their protection to PAGE_READWRITE (RW) before invoking the original Sleep function. After the sleep period ends, the malware restores the memory protection of those regions to PAGE_EXECUTE_READWRITE (RWX). This behavior suggests that the malware developer implemented this mechanism to evade memory scanning techniques that identify executable (RWX) code regions in memory.

Finally, after the API hooks are installed and the Cobalt Strike Beacon shellcode has been written to the thread buffer, the malware calls the ResumeThread API to resume the suspended thread and begin execution of the beacon.

Persistence mechanism

While the analyzed SharkLoader implant does not contain a built-in persistence mechanism especially when it comes to cases when it is dropped after the exploitation of a public-facing application, our investigations revealed that the threat actor employs several techniques to maintain access to compromised systems.

Registry Run key: In the incident that affected an organization in Hong Kong, the attacker manually created a registry Run key to launch SystemSettings.exe upon user login. The following command was used:

```
1 reg add HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Run  
/v "MFUpdate" /t REG_SZ /d "$appdata\Identities\SystemSettings.exe" /f
```

This technique allows the malware to automatically execute whenever the user logs in, ensuring persistent access.

Scheduled task: In the separate compromise that affected a diplomatic government entity in Indonesia, the attacker established persistence through a scheduled task configured to execute SharkLoader daily. The task, named "\Microsoft\Windows\Edge\Edgeupdate", was configured to run C:\ADriveLogs_Logs\SystemSettings.exe by using the following command:

```
1 Schtasks /create /s /u "" /p "" /ru "SYSTEM" /tn "\Microsoft\Windows\Edge\Edgeupdate"  
/sc DAILY /tr "C:\ADriveLogs_Logs\SystemSettings.exe /F"
```

Running the task with SYSTEM privileges ensures that SharkLoader executes even if no user is logged in.

Post-compromise activity

Following initial compromise and persistence, the attacker engaged in extensive reconnaissance and credential theft activities.

System information enumeration: The attacker initially gathered basic system information by using the following commands:

```
1 systeminfo  
2 ipconfig /all  
3 tasklist /svc
```

Post-exploitation tools: Our analysis revealed the use of several third-party post-exploitation tools, most of which are open-source and developed by Chinese-speaking developers. These tools included:

Tool name	Description
FScan	Network scanner tool with vulnerability exploitation modules
Searchall	Sensitive information search tool
Pillager	Information gathering tool

We also detected the use of SharpGPOAbuse by the threat actor, a tool designed to modify Group Policy Objects within Active Directory environments.

Active Directory enumeration: In the compromise affecting a diplomatic government entity in Indonesia, the attacker used both Cobalt Strike and a webshell to enumerate the internal Active Directory environment. They executed a series of commands to gather information about the network, users, and groups:

- **Network information:**

- 1 ping -n
- 2 netstat -ano
- 3 arp -a
- 4 net share

- **User and group information:**

- 1 query user
- 2 nslookup
- 3 quser
- 4 net group /domain

- **Specific group membership:**

- 1 powershell "Get-ADGroupMember -Identity "" -Recursive | Select-Object Name,
 - 2 ObjectClass"
 - 3
 - 4 dsquery group -name "" | dsget group -members -expand | dsget user -samid -
 - 5 display -email"
 - 6 powershell "Get-ADGroupMember -Identity "" -Recursive | Where-Object {
 - 7 \$_.ObjectClass -eq "computer" } | Select-Object Name, SamAccountName"
 - 8
- powershell -exec bypass -c "Get-ADUser -Filter * -Prop * | select sAMAccountName
- net group "Domain Controllers" /domain
- net group "Enterprise Admins" /domain
- net group "Organization Management" /domain
- net group "domain admins" /domain

- **Process enumeration:**

```
1 tasklist /SVC | findstr $selfname.exe
```

- **Directory listing:**

```
1 dir \\c$
2 dir \\c$\inetpub
3 dir \\c$\inetpub\custerr
4 dir \\c$\inetpub\wwwroot\
```

Credential dumping: The attacker also attempted to dump credentials from the compromised machine by targeting both the LSASS process and the NTDS database file. The following commands were observed:

```
1 ntdsutil "ac i ntds" "ifm" "create full $temp" q q
2 Procdump64.exe -accepteula -ma lsass.exe $temp\lsass.dmp
```

Dumping the LSASS process allows the attacker to extract in-memory credentials, while accessing the NTDS database enables retrieval of Active Directory account password hashes. This combination of techniques allows the attacker to obtain privileged credentials for lateral movement, privilege escalation, and deeper compromise.

Victimology

The victimology observed in this campaign shows a combination of strategic and opportunistic characteristics. Confirmed victims include government-related entities, such as the ministry in Taiwan and the diplomatic organization in Indonesia, as well as software development companies in Taiwan, Lebanon, and Syria. Additional affected organizations were identified in Hong Kong, Colombia, Macedonia, Nepal, and Serbia.

Targeting of government and software development organizations may indicate a cyber-espionage objective, although our confidence remains low due to the limited post-compromise activity observed, which primarily consisted of credential access, system reconnaissance, and

lateral movement. The compromise of government and software development organizations could indicate an interest in gathering political intelligence or intellectual property.

At the same time, the use of SharkLoader and Cobalt Strike, alongside the exploitation of public-facing applications and malicious installers and droppers, suggests the attacker may also be opportunistically targeting vulnerable systems. The absence of clear evidence of data exfiltration thus far does not exclude this possibility, as Cobalt Strike's file operation and data exfiltration modules could be employed at a later stage.

Although the full scope of the campaign is not yet known, the combination of targeted and opportunistic activity suggests it should continue to be closely monitored.

Attribution

Our investigation reveals no code or infrastructure overlap linking SharkLoader to any existing threat actor at this time. The TTPs employed during the operation also do not align with those of known actors.

However, analysis of the post-exploitation open-source tools used during the campaign revealed that several reconnaissance tools, including FScan, Searchall, and Pillager, were developed by individuals identified as Chinese speaking developers on GitHub.

We assess StrikeShark to be a Chinese-speaking threat actor with low confidence. This assessment is based on limited indicators and should be considered preliminary. Further investigation is required to characterize this cluster more fully, and the possibility remains that other actors may also be utilizing these tools.

Conclusion

Our investigation discovered a previously undocumented intrusion cluster that we are tracking as StrikeShark. The StrikeShark campaign represents a sophisticated malware threat to entities worldwide. The use of SharkLoader to deploy Cobalt Strike, coupled with API hook installation to evade detection, demonstrates a significant level of technical expertise. The campaign's broad targeting across sectors and geographic regions suggests a potential focus on espionage or information gathering. While the precise objectives remain under investigation, the combination of targeting government entities and software developers warrants heightened vigilance.

Given that our visibility is limited to incidents observed through Kaspersky telemetry, we suspect the actual number of compromises may be significantly higher and extend beyond

these victims as the threat actor actively used several exploitations of public facing application.

Indicators of compromise

Additional information about this activity, including indicators of compromise, is available to customers of the [Kaspersky Intelligence Reporting Service](#). If you are interested, please contact intelreports@kaspersky.com.

C559CC68986933200FD5D9E4388E2F58	Installer
B3352B42432DEDC4A519F011DC8B5D5A	Dropper
24FCEBDEECBA65004FDB0923763D74FD	Dropper
9C872A0D5D5A38950E8B9AC9B488BE3F	SharkLoader DLL
AA3086BE652C8B20B0B29B2730D57119	SharkLoader DLL
A514D1BB62D7916475946FE7C07AC0AA	Encrypted file
9CBD560F820C95D7C38342CD558CB5C6	Encrypted file
connect-microsoft[.]com	
ms-record[.]com	
ms-record[.]top	
ms-tray[.]top	

- [Malware Technologies](#)
- [Targeted attacks](#)
- [Malware Descriptions](#)
- [Vulnerabilities](#)
- [Malware](#)
- [DLL hijacking](#)
- [CobaltStrike](#)
- [StrikeShark](#)
- [SharkLoader](#)

StrikeShark: investigating a new campaign delivering Cobalt Strike through SharkLoader

Your email address will not be published. Required fields are marked *

Name *

Email *

This site uses Akismet to reduce spam. [Learn how your comment data is processed.](#)