

Lazarus supply-chain attack in South Korea

[welivesecurity.com/2020/11/16/lazarus-supply-chain-attack-south-korea](https://www.welivesecurity.com/2020/11/16/lazarus-supply-chain-attack-south-korea)

ESET telemetry data recently led our researchers to discover attempts to deploy Lazarus malware via a supply-chain attack in South Korea. In order to deliver its malware, the attackers used an unusual supply-chain mechanism, abusing legitimate South Korean security software and digital certificates stolen from two different companies.

Lazarus toolset

The Lazarus group was first identified in Novetta's report Operation Blockbuster in February 2016; US-CERT and the FBI call this group HIDDEN COBRA. These cybercriminals rose to prominence with the infamous case of cybersabotage against Sony Pictures Entertainment.

Some of the past attacks attributed to the Lazarus group attracted the interest of security researchers who relied on Novetta et al.'s white papers with hundreds of pages describing the tools used in the attacks – the Polish and Mexican banks, the WannaCryptor outbreak, phishing campaigns against US defense contractors, Lazarus KillDisk attack against Central American casino, etc. – and provides grounds for the attribution of these attacks to the Lazarus group.

Note that the Lazarus toolset (i.e., the collection of all files that are considered by the security industry as fingerprints of the group's activity) is extremely broad, and we believe there are numerous subgroups. Unlike toolsets used by some other cybercriminal groups, none of the source code of any Lazarus tools has ever been disclosed in a public leak.

Latest Lazarus supply-chain attack

To understand this novel supply-chain attack, you should be aware that South Korean internet users are often asked to install additional security software when visiting government or internet banking websites.

WIZVERA VeraPort, referred to as an integration installation program, is a South Korean application that helps manage such additional security software. With WIZVERA VeraPort installed on their devices, users receive and install all necessarily software required by a specific website with VeraPort (e.g., browser plug-ins, security software, identity verification software, etc.). Minimal user interaction is required to start such a software installation from a website that supports WIZVERA VeraPort. Usually, this software is used by government and banking websites in South Korea. For some of these websites it is mandatory to have WIZVERA VeraPort installed for users to be able to access the sites' services.

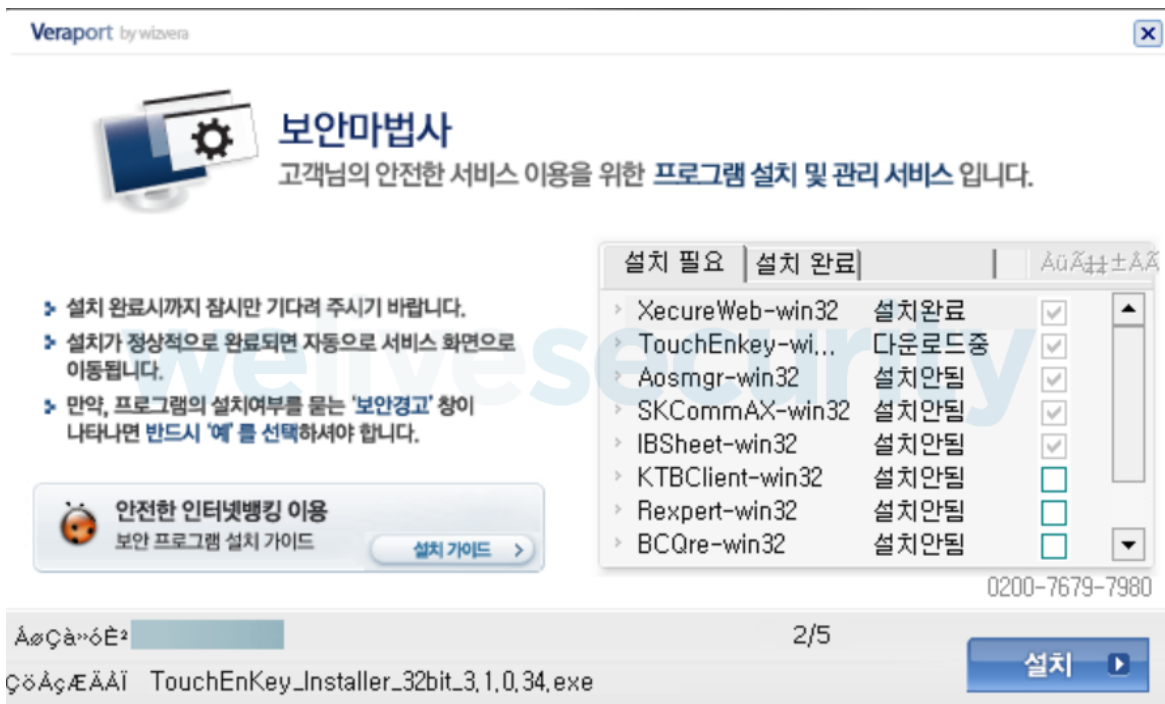


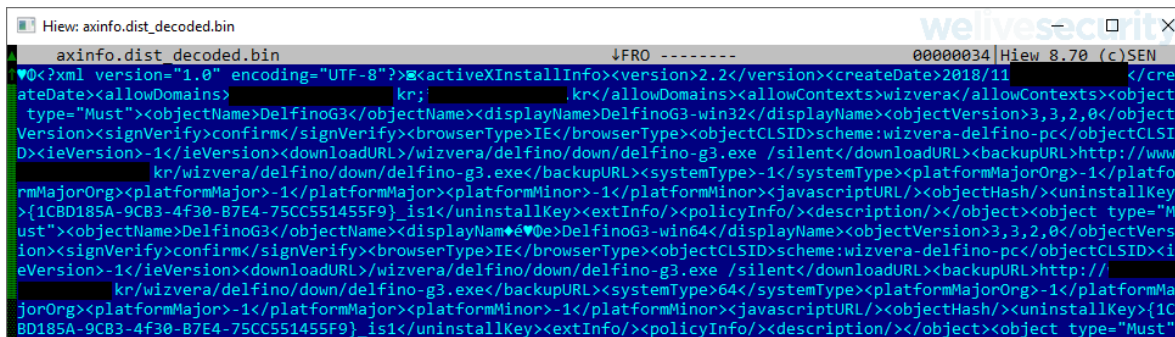
Figure 1. A WIZVERA VeraPort window displayed to the user when installing additional software

The Lazarus attackers abused the above-mentioned mechanism of installing security software in order to deliver Lazarus malware from a legitimate but compromised website. However, it should be noted that a successful malware deployment using this method requires a number of preconditions; that's why it was used in limited Lazarus campaigns. To make this attack possible:

- the victim must have WIZVERA VeraPort software installed
- the victim must visit a compromised website that already has support for WIZVERA VeraPort
- this website must have specific entries in its VeraPort configuration file that allows attackers to replace regular software in its VeraPort software bundle with their malware.

It is important to note that, based on our analysis, we believe that these supply-chain attacks happen at websites that use WIZVERA VeraPort, rather than at WIZVERA itself.

Websites that support WIZVERA VeraPort software contain a server-side component, specifically some JavaScripts and a WIZVERA configuration file. The configuration file is base64-encoded XML containing the website address, a list of software to install, download URLs, and other parameters.



```
axinfo.dist_decoded.bin
<?xml version="1.0" encoding="UTF-8"?><activeXInstallInfo><version>2.2</version><createDate>2018/11/11</createDate><allowDomains>kr; .kr</allowDomains><allowContexts>wizvera</allowContexts><object type="Must"><objectName>DelfinoG3</objectName><displayName>DelfinoG3-win32</displayName><objectVersion>3,3,2,0</objectVersion><signVerify>confirm</signVerify><browserType>IE</browserType><objectCLSID>scheme:wizvera-delfino-pc</objectCLSID><ieVersion>-1</ieVersion><downloadURL>/wizvera/delfino/down/delfino-g3.exe /silent</downloadURL><backupURL>http://www.kr/wizvera/delfino/down/delfino-g3.exe</backupURL><systemType>-1</systemType><platformMajorOrg>-1</platformMajorOrg><platformMajor>-1</platformMajor><platformMinor>-1</platformMinor><javascriptURL></javascriptURL><objectHash></objectHash><uninstallKey>{1CBD185A-9CB3-4F30-B7E4-75CC551455F9} is1</uninstallKey><extInfo><policyInfo><description/></object><object type="Must"><objectName>DelfinoG3</objectName><displayName>DelfinoG3-win64</displayName><objectVersion>3,3,2,0</objectVersion><signVerify>confirm</signVerify><browserType>IE</browserType><objectCLSID>scheme:wizvera-delfino-pc</objectCLSID><ieVersion>-1</ieVersion><downloadURL>/wizvera/delfino/down/delfino-g3.exe /silent</downloadURL><backupURL>http://www.kr/wizvera/delfino/down/delfino-g3.exe</backupURL><systemType>64</systemType><platformMajorOrg>-1</platformMajorOrg><platformMajor>-1</platformMajor><platformMinor>-1</platformMinor><javascriptURL></javascriptURL><objectHash></objectHash><uninstallKey>{1CBD185A-9CB3-4F30-B7E4-75CC551455F9} is1</uninstallKey><extInfo><policyInfo><description/></object></activeXInstallInfo>
```

Figure 2. An example of a WIZVERA VeraPort configuration (redacted by ESET)

These configuration files are digitally signed by WIZVERA. Once downloaded, they are verified using a strong cryptographic algorithm (RSA), which is why attackers can't easily modify the content of these configuration files or set up their own fake website. However, the attackers can replace the software to be delivered to WIZVERA VeraPort users from a legitimate but compromised website. We believe this is the scenario the Lazarus attackers used.

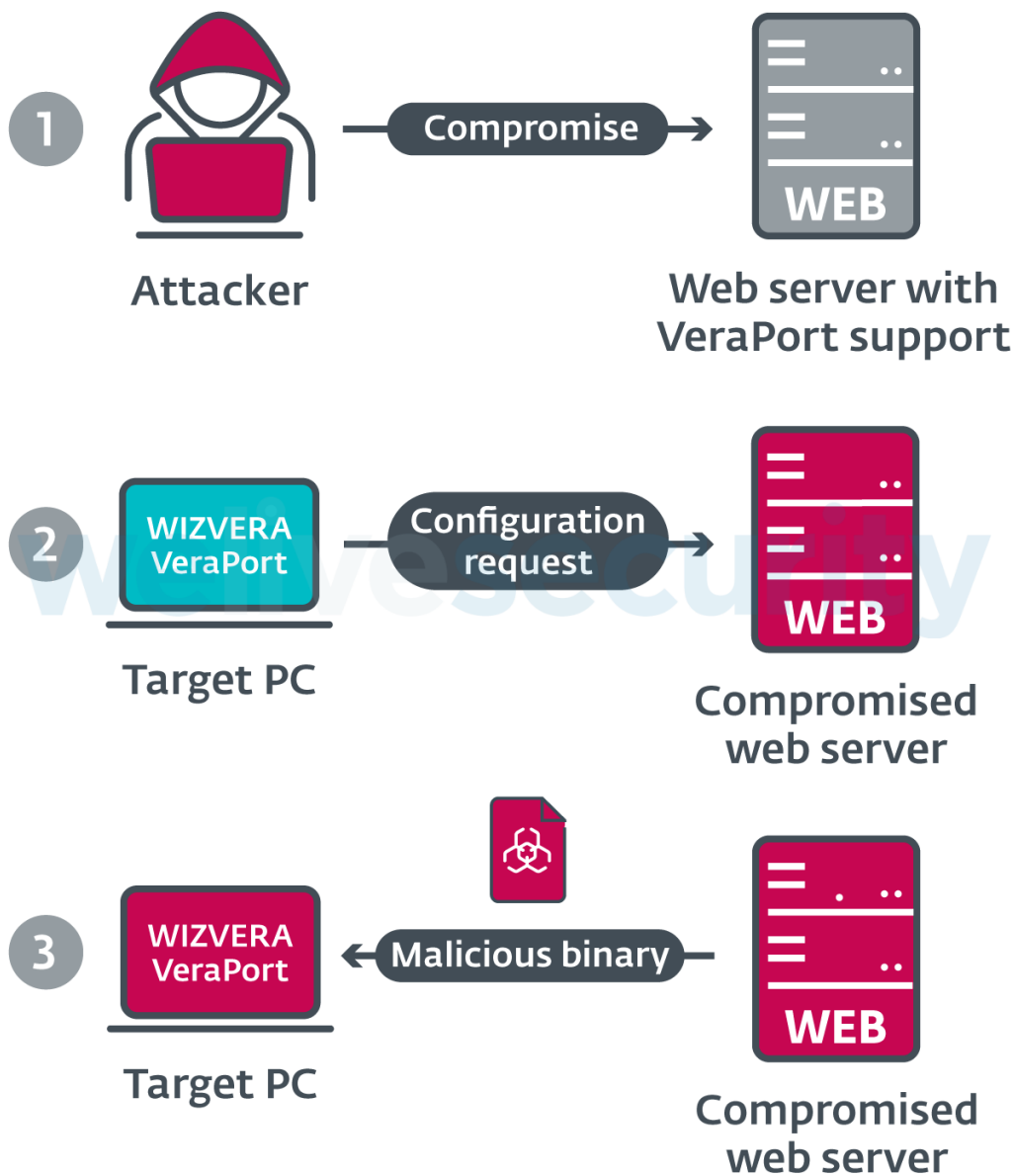


Figure 3. Simplified scheme of the WIZVERA supply-chain attack conducted by the Lazarus group

It should be noted that WIZVERA VeraPort configurations contain an option to verify the digital signature of downloaded binaries before they are executed, and in most cases this option is enabled by default. However, VeraPort only verifies that the digital signature is valid, without checking to whom it belongs. Thus, to abuse WIZVERA VeraPort, attackers must have any valid code-signing certificate in order to push their payload via this method or get lucky and find a VeraPort configuration that does not require code-signing verification.

So far, we have observed two malware samples that were delivered using this supply-chain attack and both were signed:

SHA-1	Filename	Digital signature
3D311117D09F4A6AD300E471C2FB2B3C63344B1D	Delfino.exe	ALEXIS SECURITY GROUP, LLC
3ABFEC6FC3445759730789D4322B0BE73DC695C7	MagicLineNPIZ.exe	DREAM SECURITY USA INC

The attackers used illegally obtained code-signing certificates in order to sign the malware samples. Interestingly, one of these certificates was issued to the US branch of a South Korean security company.

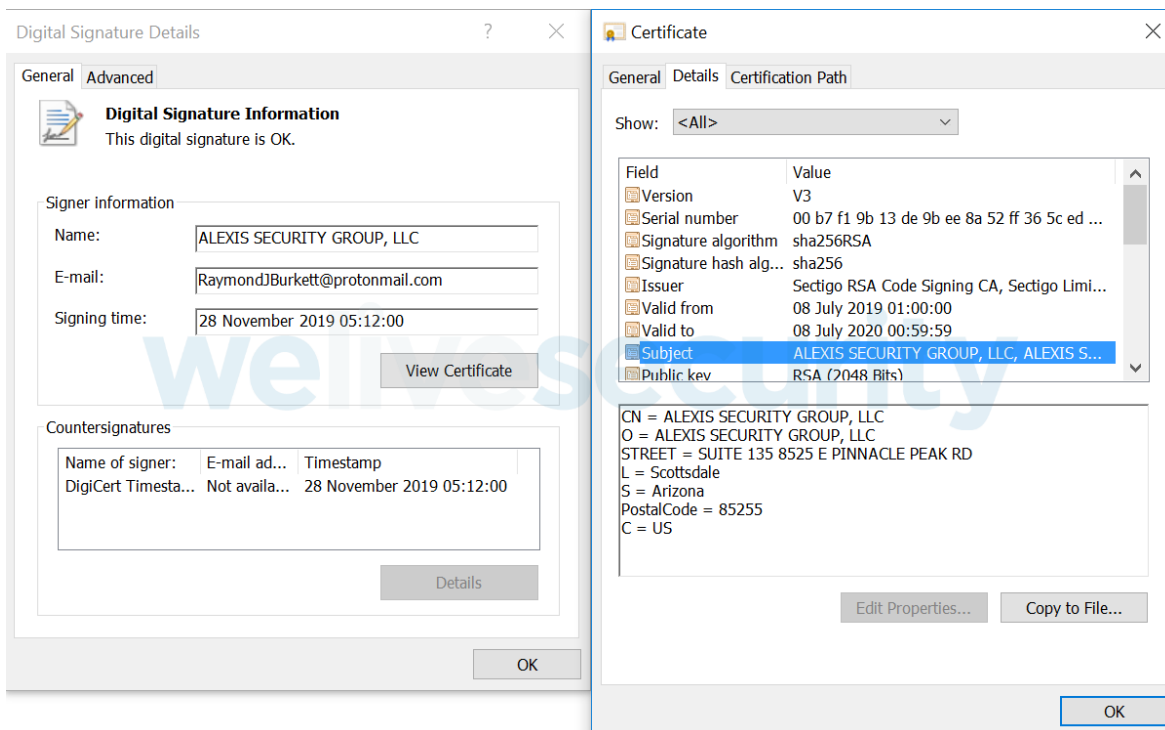


Figure 4. The ALEXIS SECURITY GROUP, LLC code-signing certificate used to sign Lazarus malware

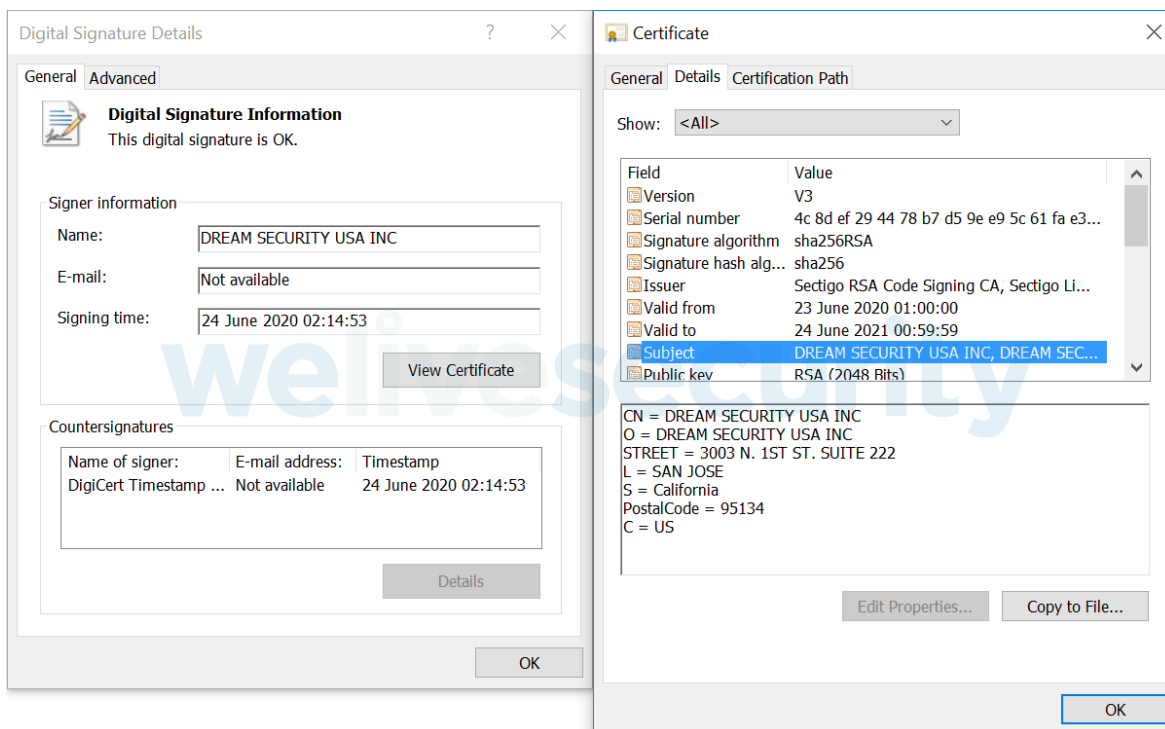


Figure 5. The DREAM SECURITY USA INC code-signing certificate used to sign Lazarus malware

The attackers camouflaged the Lazarus malware samples as legitimate software. These samples have similar filenames, icons and VERSIONINFO resources as legitimate South Korean software often delivered via WIZVERA VeraPort. Binaries that are downloaded and executed via the WIZVERA VeraPort mechanism are stored in %Temp%\[12_RANDOM_DIGITS]\.

It should be noted that WIZVERA VeraPort's configuration has an option not only to verify digital signatures, but also to verify the hash of downloaded binaries. If this option is enabled, then such an attack cannot be performed so easily, even if the website with WIZVERA VeraPort is compromised.

Attribution

We strongly attribute this supply-chain attack to the Lazarus group, based on the following aspects:

1. Community agreement: The current attack is a continuation of what KrCERT has called Operation Bookcodes. While KrCERT hasn't attributed that campaign to the Lazarus group, Kaspersky did in their report about [Q2 2020 APT trends](#).
2. Toolset characteristics and detection:
 1. The initial dropper is a console application that requires parameters, executing the next stages in a cascade and utilizes an encryption, cf. [the watering hole attacks against Polish and Mexican banks](#)
 2. The final payload is a RAT module, with TCP communications and its commands indexed by 32-bit integers, cf. [KillDisk in Central America](#)
 3. Many tools delivered via this chain are already flagged as NukeSped by ESET software. For example, the signed Downloader in the *Analysis* section is based on a project called WinHttpClient and it leads to the similar tool with hash 1EA7481878F0D9053CCD81B4589CECAEFC306CF2, which we link with with a sample from Operation Blockbuster (CB818BE1FCE5393A83FBFCB3B6F4AC5A3B5B8A4B). The connection between the latter two is the dynamic resolution of Windows APIs where the names are XOR-encrypted by 0x23, e.g., dFWwLHFMjMELQNBWJLM is the encoding of GetTokenInformation.
3. Victimology: the Lazarus group has a long history of attacks against victims in South Korea like [Operation Troy](#), including DDoS attacks [Ten Days of Rain](#) in 2011, [South Korean Cyberattacks](#) in 2013, or [South Korean cryptocurrency exchanges](#) targeted in 2017.
4. Network infrastructure: the server-side techniques of webshells and the organization of C&Cs are covered very precisely in KrCERT's white paper #2. The current campaign uses a very similar setup as well.
5. Eccentric approach:
 1. In intrusion methods: The unusual method of infiltration is a clue that could be attributed to a sophisticated and professionally organized actor like Lazarus. In the past, we saw how a vulnerability in software existing only in specific networks was leveraged by this group, and not visible with any other APT actor. For example, [the case of "A Strange Coinminer"](#) delivered through the ManageEngine Desktop Central software.
 2. In encryption methods: We saw a Spritz variant of RC4 in the watering hole attacks against Polish and Mexican banks; later Lazarus used a modified RC4 in [Operation In\(ter\)ception](#). In this campaign, it is a modified A5/1 stream cipher that degrades to a single-byte XOR in many cases.

Malware analysis

It is a common characteristic of many APT groups, especially Lazarus, that they unleash their arsenal within several stages that execute as a cascade: from the dropper to intermediate products (the Loader, serving as an injector) up to the final payloads (the Downloader, the Module). The same is true for this campaign.

During our analysis we found similarities in code and architecture between Lazarus malware delivered via this WIZVERA supply-chain attack and the malware described in the Operation BookCodes report ([part one](#), [part two](#)) published by Korea Internet & Security Agency this year.

Comparison with Operation BookCodes

Table 1. Common characteristics between two Lazarus operations

Parameter/ Campaign	Operation BookCodes	Via WIZVERA Vera Port
Location of targets	South Korea	South Korea
Time	Q1-Q2 2020	Q2-Q3 2020
Methods of compromise	Korean spearphishing email (link to download or HWP attachment) Watering hole website	Supply-chain attack
Filename of the dropper	C:\Windows\SoftwareDistribution\Download\BIT[4-5digits].tmp	C:\Windows\SoftwareDistribution\Download\BIT388293.tmp

Parameter/ Campaign	Operation BookCodes	Via WIZVERA Vera Port
Binary configuration file	perf91nc.inf (12000 bytes)	assocnet.inf (8348 bytes)
Loader name	nwsapagentmonsvc.dll	Btserv.dll iasregmonsvc.dll
RC4 key	1qaz2wsx3edc4rfv5tgb\$%^&*!@#&\$	1q2w3e4r!@#&\$%^&*
Log file	%Temp%\services_dll.log	%Temp%\server_dll.log

Signed initial downloader

This is the Lazarus component delivered via the VeraPort hijack described earlier. The signed initial downloaders are Themida-protected binaries, which download, decrypt and execute other payloads in memory, without dropping them to the disk. This downloader sends an HTTP POST request to a hardcoded C&C server, decrypts the server's answer using the RC4 algorithm, and executes it in memory using its own loader for PE files.

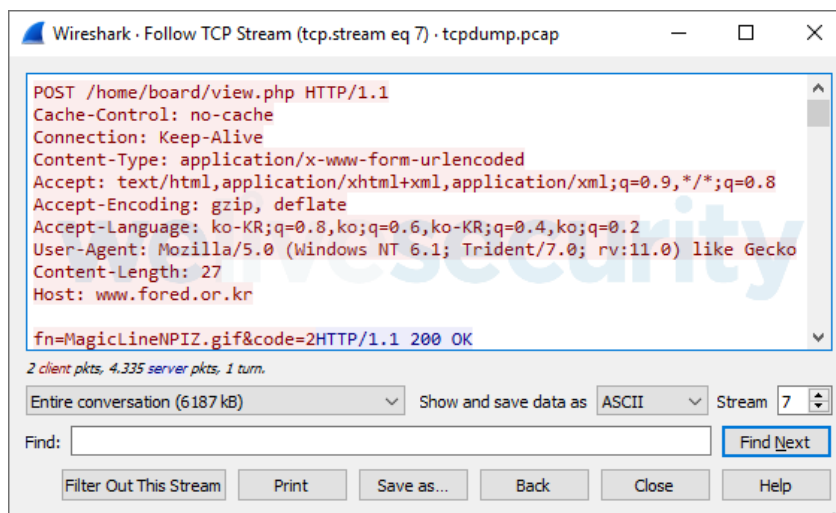
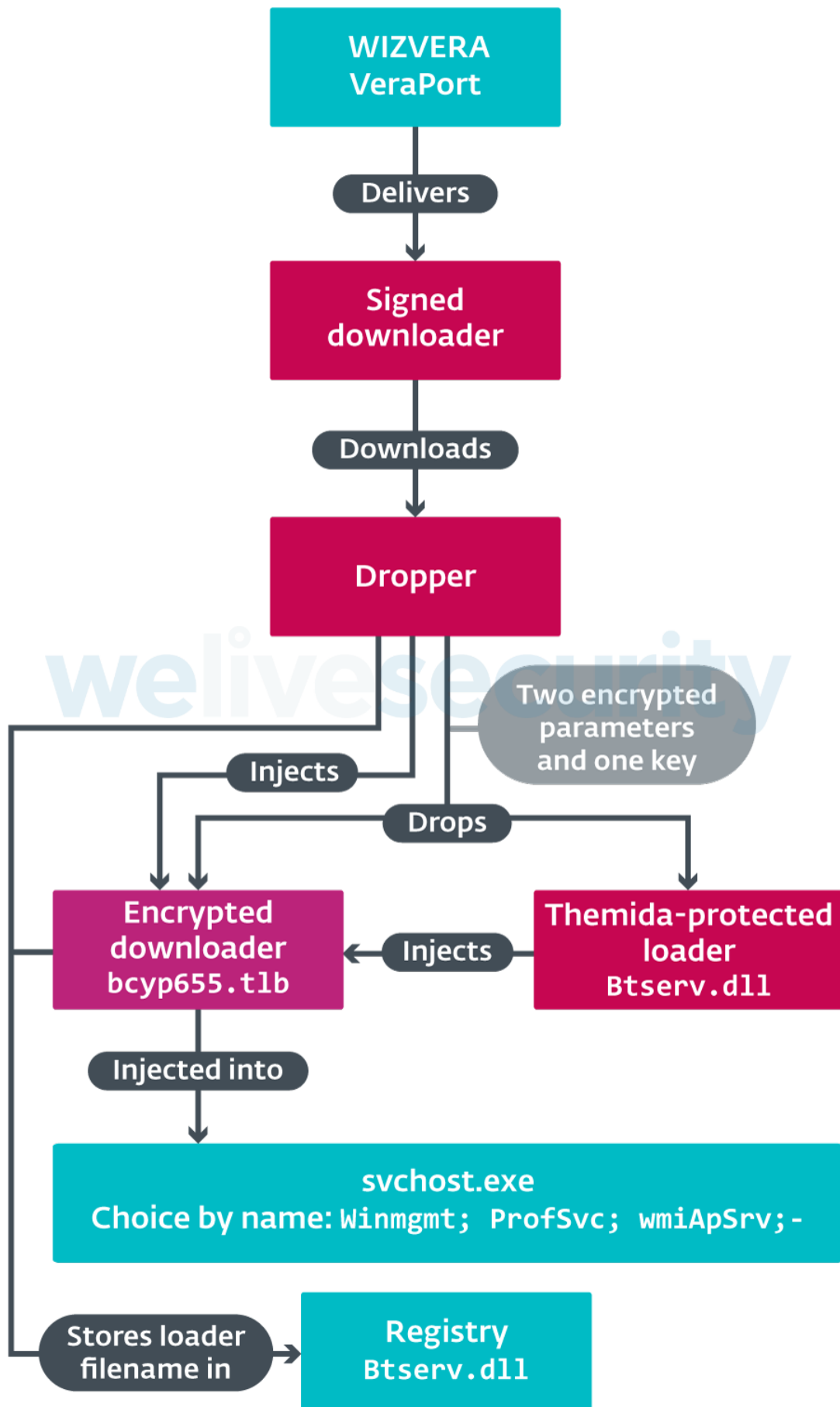


Figure 6. The POST request made by the initial downloader

Interestingly, both discovered samples send a small, hardcoded ID in the body of the POST request: MagicLineNPIZ.gif or delfino.gif.



Dropper

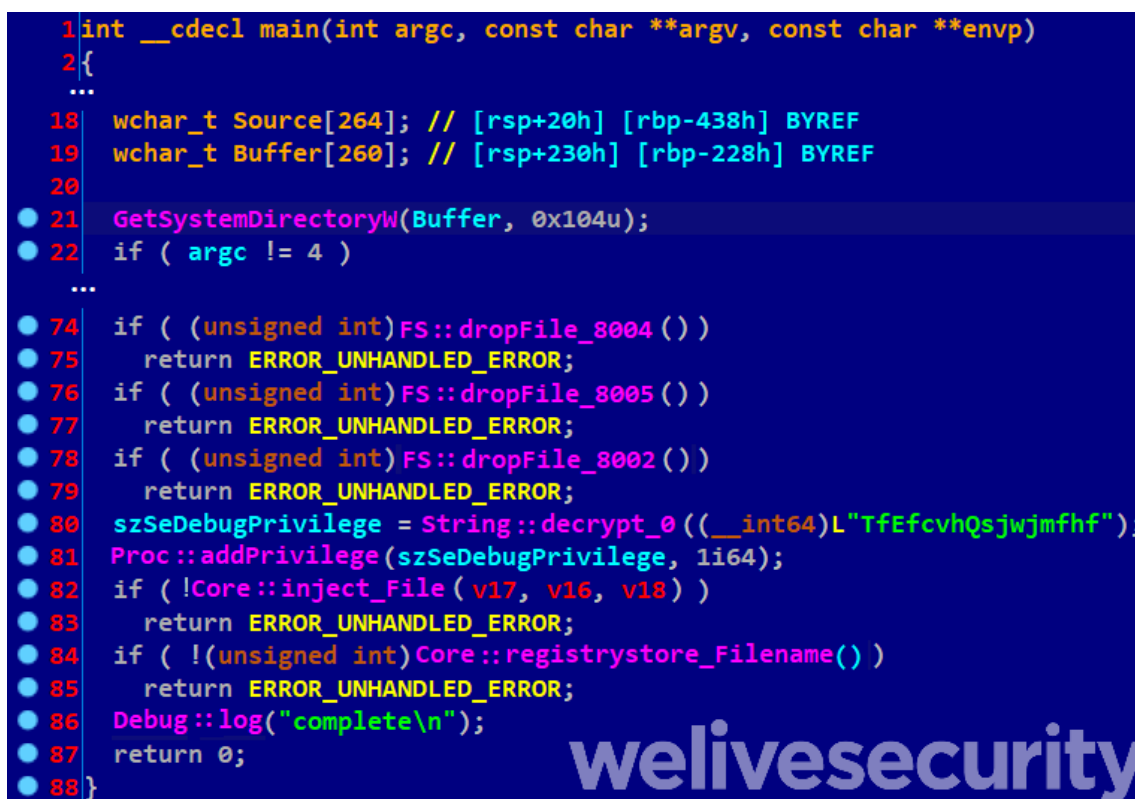
This is the initial stage of the cascade. While one can't see any polymorphism or obfuscation in the code, it encapsulates three encrypted files in its resources. Moreover, it's a console application expecting three parameters in an encrypted state: the name of the first file (the Loader, Btserv.dll), the name of the second file (the Downloader, bcyp655.tlb), and the necessary decryption key for the previous values (542).

BIT388293.tmp oJaRh5CUzlaOjg== aGlzejw/PyR+Zmg= 542

The extraction of resources is one of two main roles of the dropper; it does so in the %WINDOWS%\SYSTEM32 folder, decrypting the Loader and preserving the encrypted state of the Downloader that will be decrypted just before being injected into another process. It also drops the configuration file assocnet.inf that will later be leveraged by the final payloads, namely the Downloader and the Module. Then it chooses a service by checking the following list of three legitimate service names Winmgmt;ProfSvc;wmiApSrv; and injects the Downloader into the matched service using reflective DLL injection.

The file name of the Loader is stored in the following Windows registry value:

HKLM\SYSTEM\CurrentControlSet\Control\Lsa\Security Packages



```

1 int __cdecl main(int argc, const char **argv, const char **envp)
2 {
3     ...
18  wchar_t Source[264]; // [rsp+20h] [rbp-438h] BYREF
19  wchar_t Buffer[260]; // [rsp+230h] [rbp-228h] BYREF
20
21  GetSystemDirectoryW(Buffer, 0x104u);
22  if ( argc != 4 )
23      ...
74  if ( (unsigned int)FS::dropFile_8004() )
75      return ERROR_UNHANDLED_ERROR;
76  if ( (unsigned int)FS::dropFile_8005() )
77      return ERROR_UNHANDLED_ERROR;
78  if ( (unsigned int)FS::dropFile_8002() )
79      return ERROR_UNHANDLED_ERROR;
80  szSeDebugPrivilege = String::decrypt_0((__int64)L"TfEfcvhQsjwjmfhf");
81  Proc::addPrivilege(szSeDebugPrivilege, 1i64);
82  if ( !Core::inject_File(v17, v16, v18) )
83      return ERROR_UNHANDLED_ERROR;
84  if ( !(unsigned int)Core::registrystore_Filename() )
85      return ERROR_UNHANDLED_ERROR;
86  Debug::log("complete\n");
87  return 0;
88 }

```

Figure 8. The decompiled code of the dropper

Loader

This component is a Themida-protected file. We estimate the version of Themida to be 2.0-2.5, which agrees with KrCERT's report (page 20). The Loader serves as a simple injector that is looking for its injection parameters in the resources: the name of the encrypted file and the decryption key, which is the string "542". The instance delivered by the dropper looks for the file bcyp655.tlb (the Downloader). It creates a mutex Global\RRfreshRA_Mutex_Object. The choice of the targeted service and the injection method are the same as in the dropper.

Let us talk for a while about the encryption method used by the dropper and by this loader. The common key is the string "542", which is initially provided as a command-line parameter to the Dropper and subsequently as a 3-byte encrypted resource for the Loader. To expand a short master key to a larger expanded key (so-called key scheduling), the MD5 hash of the string is computed, which is 7DCD340D84F762EBA80AA538B0C527F7. Then it takes first three double words, let's

denote them $A := 0x7DCD340D$, $B := 0x84f762EB$, $C := 0xA80aa538$. The length of an encrypted buffer is divided by 3, and this is the number of iterations that transforms the initial sequence (A,B,C) into the proper key. In every iteration (X,Y,Z) becomes (X^Y, Y^Z, X^Y^Z) . Because the XOR operation (denoted \wedge) is commutative and transitive, and its square is zero, which leaves everything unchanged, we can compute that after 8 iterations we get the identity, so the key could reach just 7 pairwise different states and is equal to the first 12 characters of the MD5 hash of "542" if the length is a multiple of 24.

What is interesting is how the remainder of the length division by 3 is treated. If the number of iterations was increased by this remainder, then we would reach just another of the 7 states of the key. However, the twist is in the change of operation: \wedge is replaced with the OR operation in the code for the remainder. For example, the key with the remainder 1 becomes {FE F7 3A F9 F7 D7 FF FD FF F7 FF FD} for one of the states (of (C, A^B , B^C) to be precise), so we get new possible transformations of the key that tend to be more likely to be ones than zeroes.

That was the part preparing the key. The encryption algorithm itself looks like [A5/1](#) at first glance. It was a secret technology developed in 1987 and used in over-the-air communication privacy in the GSM cellular telephone standard until reverse-engineered in 1999. The crucial part of the algorithm is *three* linear feedback shift registers (LFSRs). However, only the lengths of LFSRs in the malware code coincide with the official implementation, not the constants.

Table 2. Comparison of crypto algorithms between malware and the official implementation

LFSR	Malware code	Official A5/1
1	Length: 19	Length: 19
	Constants: 13, 16, 17, 18	Constants: 13, 16, 17, 18
2	Length: 22	Length: 22
	Constants: 12, 16, 20, 21	Constants: 20, 21
3	Length: 23	Length: 23
	Constants: 17,18,21,22	Constants: 7, 20, 21, 22

The decryption loop in each iteration basically derives a 1-byte XOR key for the corresponding byte of the encrypted buffer. The purpose of LFSRs is that they could transform the key, so the whole process is much more complicated. But due to the mentioned change of the operation, LFSRs would not affect it and the 1-byte XOR key remains the same for all iterations.

Downloader, aka WinHttpClient

The main downloader is dropped by the Dropper component under the bcyp655.tlb name and injected into one of the services by the Loader. Its main purpose is to deliver additional stages onto the victim's computers. The network protocol is based on HTTP but requires several stages to establish a trusted connection.

The malware fingerprints the victim's system: see Figure 9.

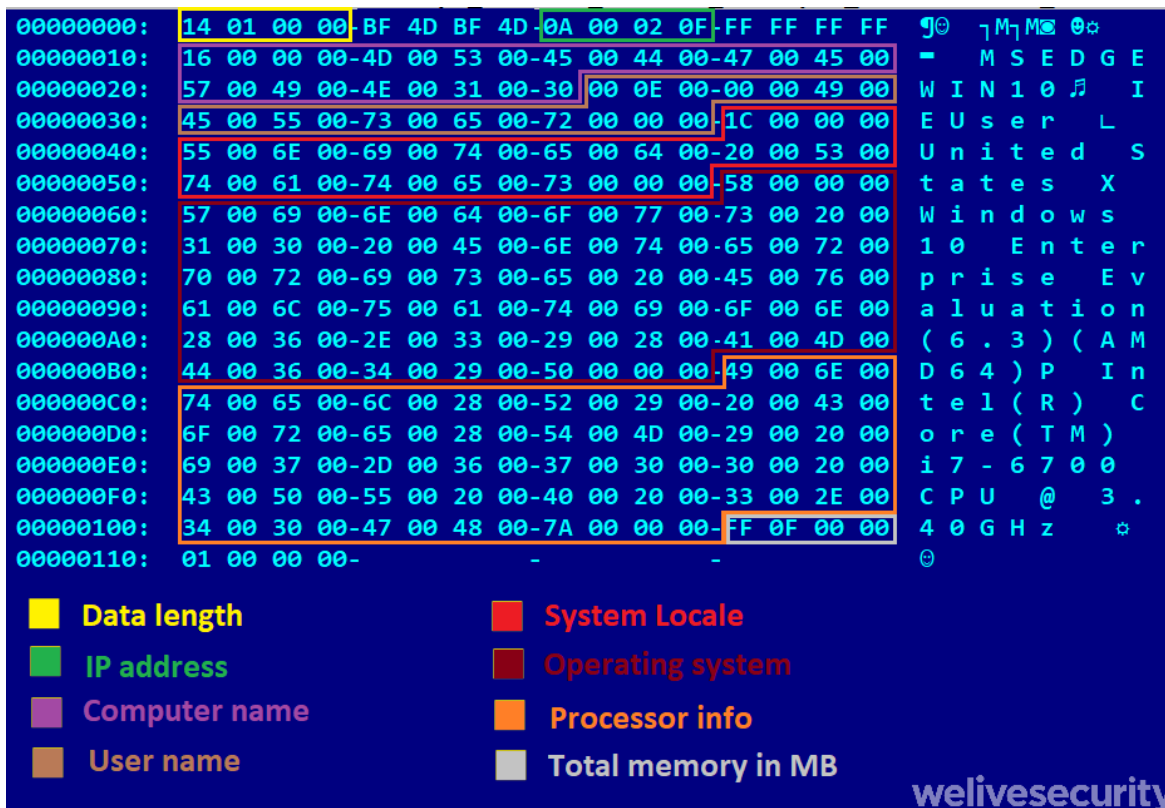


Figure 9. The length of the buffer is 0x114 and contains campaign ID, local IP address, Windows version, processor version (cf. KrCERT page 59, Figure [4-17])

The first step is authorization. After sending randomly generated, generic parameters code and id, the expected response starts with <!DOCTYPE HTML PUBLIC Authentication En> followed by additional data delimited by a semicolon. However, in the next POST request the parameters are already based on the victim's IP. Because we didn't know which victims were targeted, during our investigation, we always received a "Not Found" reply, not the successful "OK".

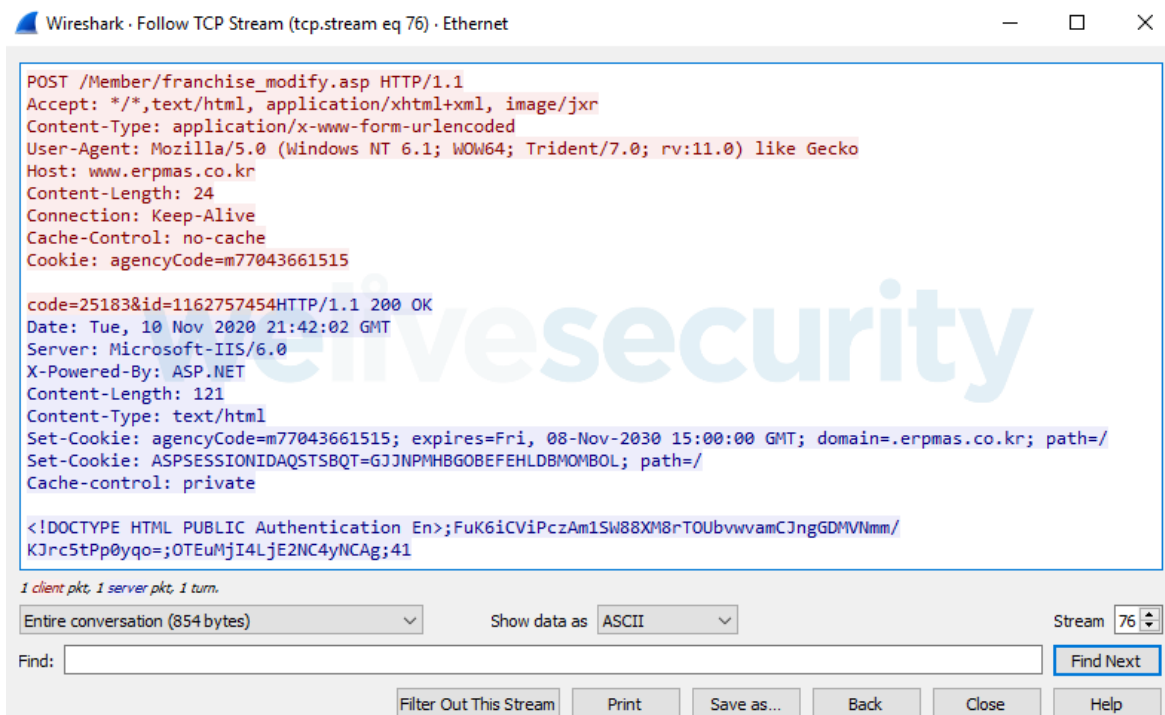


Figure 10. Primary message exchange with C&C having generic parameters code and id

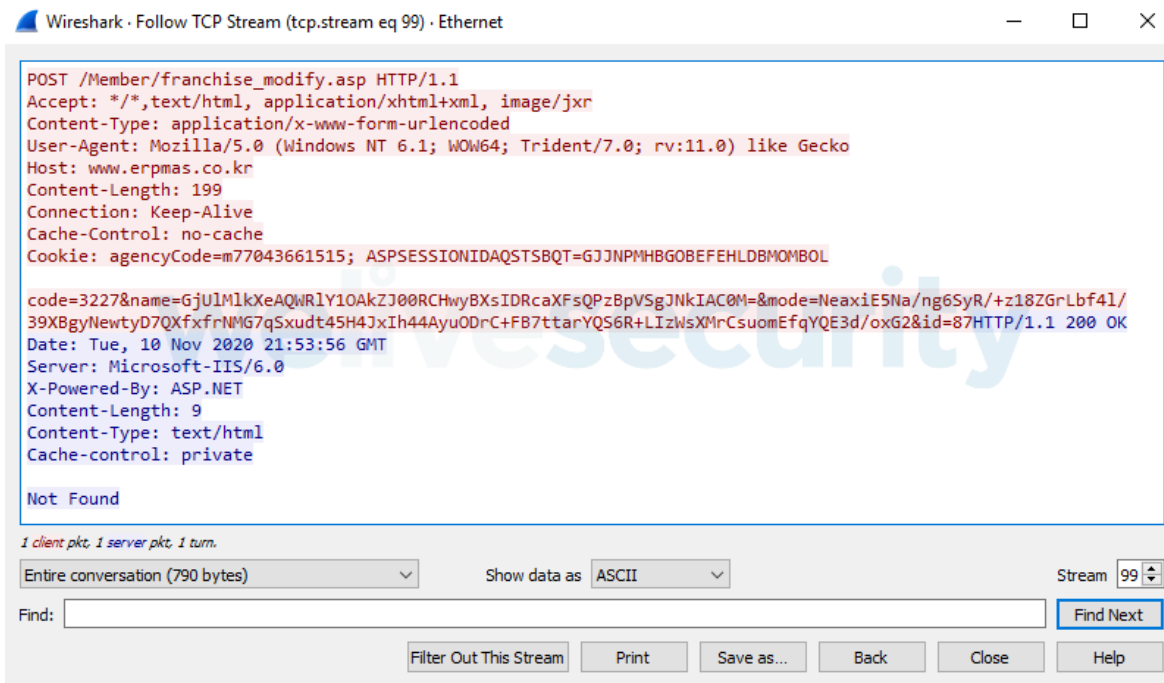


Figure 11. Secondary message exchange with C&C having a specific parameter name

If the victim passes these introductory messages and the connection is acknowledged, then the decrypted response starts with an interesting artifact: a keyword ohayogonbangwa!! . As a whole, we haven't found that word on the internet, but the closest meaning could be "Ohayo, Konbangwa" (おはようこんばんあ), which is "Good morning, good evening" in Japanese. From this point, there are more messages that are exchanged, with the final exchange asking for an executable to load into memory.



Figure 12. Japanese artifact in the code

Module, the final RAT payload

This is a RAT with a set of typical features used by the Lazarus group. The commands include operations on the victim's filesystem and the download and execution of additional tools from the attacker's arsenal. They are indexed by 32-bit integers and coincide with those reported by KrCERT on page 61.

```
67     switch ( dwCommand )
68     {
69         case CMD_0x97853646:
70             Command::listDrives(v1, 0x97853646);
71             goto _next;
72         case CMD_0x97853647:
73             Command::listDirectories(v1, 0x97853647, (__int64)v2);
74             goto _next;
75         case CMD_0x97853648:
76             Command::uploadFiles(v1, 0x97853648i64, v2);
77             goto _next;
78         case CMD_0x97853649:
79             ...
86             Command::removeFiles((__DWORD*)(v1 + 13104), &v3, 4, 1);
87             goto _next;
```

Figure 13. Some of the commands supported by Module

Conclusion

Attackers are constantly trying to find new ways to deliver malware to target computers. Attackers are particularly interested in supply-chain attacks, because they allow them to covertly deploy malware on many computers at the same time. In recent years ESET researchers analyzed such cases as [M.E.Doc](#), [Elmedia Player](#), [VestaCP](#), [Statcounter](#), and the [gaming industry](#). We can safely predict that the number of supply-chain attacks will increase in the future, especially against companies whose services are popular in specific regions or in specific industry verticals.

This time we analyzed how the Lazarus group used a very interesting approach to target South Korean users of WIZVERA VeraPort software. As mentioned in our analysis, it's the combination of compromised websites with WIZVERA VeraPort support and specific VeraPort configuration options that allow attackers to perform this attack. Owners of such websites could decrease the possibility of such attacks, even if their sites are compromised, by enabling specific options (e.g. by specifying hashes of binaries in the VeraPort configuration).

Special thanks to Dávid Gábriš and Peter Košinár.

For any inquiries, or to make sample submissions related to the subject, contact us at threatintel@eset.com

Indicators of Compromise (IoCs)

ESET detection names

Win32/NukeSped.HW
Win32/NukeSped.FO
Win32/NukeSped.HG
Win32/NukeSped.HI
Win64/NukeSped.CV
Win64/NukeSped.DH
Win64/NukeSped.DI
Win64/NukeSped.DK
Win64/NukeSped.EP

SHA-1 of signed samples

3D311117D09F4A6AD300E471C2FB2B3C63344B1D
3ABFEC6FC3445759730789D4322B0BE73DC695C7

SHA-1 of samples

5CE3CDFB61F3097E5974F5A07CF0BD2186585776
FAC3FB1C20F2A56887BDBA892E470700C76C81BA
AA374FA424CC31D2E5EC8ECE2BA745C28CB4E1E8
E50AD1A7A30A385A9D0A2C0A483D85D906EF4A9C
DC72D464289102CAAF47EC318B6110ED6AF7E5E4
9F7B4004018229FAD8489B17F60AADB3281D6177
2A2839F69EC1BA74853B11F8A8505F7086F1C07A
8EDB488B5F280490102241B56F1A8A71EBEEF8E3

Code signing certificate serial numbers

00B7F19B13DE9BEE8A52FF365CED6F67FA
4C8DEF294478B7D59EE95C61FAE3D965

C&C

http://www.ikrea.or.kr/main/main_board.asp
<http://www.fored.or.kr/home/board/view.php>
<https://www.zndance.com/shop/post.asp>
<http://www.cowp.or.kr/html/board/main.asp>
<http://www.style1.co.kr/main/view.asp>
http://www.erpmas.co.kr/Member/franchise_modify.asp
https://www.wowpress.co.kr/customer/refuse_05.asp
https://www.quecue.kr/okproj/ex_join.asp
http://www.pcdesk.co.kr/Freeboard/mn_board.asp
http://www.gongsinet.kr/comm/comm_gongsi.asp
<http://www.goojoo.net/board/banner01.asp>
<http://www.pgak.net/service/engine/release.asp>
https://www.gncaf.kr/cafe/cafe_board.asp
https://www.hsbutton.co.kr/bbs/bbs_write.asp
<https://www.hstudymall.co.kr/easypay/web/bottom.asp>

Mutexes

Global\RRfreshRA_Mutex_Object

References

KrCERT/CC, “[Operation BookCodes TTPs#1 Controlling local network through vulnerable websites](#)”, English Translation, 1st April 2020
KrCERT/CC, “[Operation BookCodes TTPs#2 스피어 피싱으로 정보를 수집하는 공격망 구성 방식 분석](#)”, Korean, 29th June 2020
P. Kálnai, M. Poslušný: “[Lazarus Group: a mahjong game played in different sets of tiles](#)”, Virus Bulletin 2018 (Montreal)
P. Kálnai: “[Demystifying targeted malware used against Polish banks](#)”, WeLiveSecurity, February 2017
P. Kálnai, A. Cherepanov “[Lazarus KillDisks Central American casino](#)”, WeLiveSecurity, April 2018
D. Breitenbacher, K. Osis: “[Operation In\(ter\)ception: Aerospace and military companies in the crosshairs of cyberspies](#)”, June 2020
Novetta et al, “[Operation Blockbuster](#)”, February 2016, <https://www.operationblockbuster.com/resources>
Marcus Hutchins, “[How to accidentally stop a global cyber-attack](#)”, May 2015
Kaspersky GReAT: “[APT trends report Q2 2020](#)”, July 2020
A. Kasza: “[The Blockbuster Saga Continues](#)”, Palo Alto Networks, August 2017
US-CERT CISA, <https://us-cert.cisa.gov/northkorea>
WeLiveSecurity: “[Sony Pictures hacking traced to Thai hotel as North Korea denies involvement](#)”, December 2014
R. Sherstobitoff, I. Liba. J. Walter: “[Dissecting Operation Troy: Cyberespionage in South Korea](#)”, McAfee® Labs, May 2018
McAfee Labs: “[Ten Days of Rain](#)”, July 2011

Fireye/Mandiant: "[Why Is North Korea So Interested in Bitcoin?](#)", September 2017

Choe Sang-Hun: "[Computer Networks in South Korea Are Paralyzed in Cyberattacks](#)", March 2013

A5/1 stream cipher, [Wikipedia](#)

MITRE ATT&CK techniques

Note: This table was built using [version 8](#) of the MITRE ATT&CK framework.

Tactic	ID	Name	Description
Resource Development	T1584.004	Compromise Infrastructure: Server	The Lazarus group uses compromised servers as infrastructure.
	T1587.001	Develop Capabilities: Malware	The Lazarus group developed custom malware and malware components.
	T1588.003	Obtain Capabilities: Code Signing Certificates	The Lazarus group obtained code-signing certificates.
Initial Access	T1195.002	Supply Chain Compromise: Compromise Software Supply Chain	The Lazarus group pushed this malware using a supply-chain attack via WIZVERA VeraPort.
Execution	T1106	Native API	The Lazarus payload is executed using native API calls.
Persistence	T1547.005	Boot or Logon Autostart Execution: Security Support Provider	The Lazarus malware maintains persistence by installing an SSP DLL.
Defense Evasion	T1036	Masquerading	The Lazarus malware masqueraded as a South Korean security software
	T1027.002	Obfuscated Files or Information: Software Packing	The Lazarus group uses Themida-protected malware.
	T1055	Process Injection	The Lazarus malware injects itself in svchost.exe.
	T1553.002	Subvert Trust Controls: Code Signing	The Lazarus group used illegally obtained code-signing certificates to sign the initial downloader used in this supply-chain attack.
Command and Control	T1071.001	Application Layer Protocol: Web Protocols	The Lazarus malware uses HTTP for C&C.
	T1573.001	Encrypted Channel: Symmetric Cryptography	The Lazarus malware uses the RC4 algorithm to encrypt its C&C communications.
Exfiltration	T1041	Exfiltration Over C2 Channel	The Lazarus malware exfiltrates data over the C&C channel.