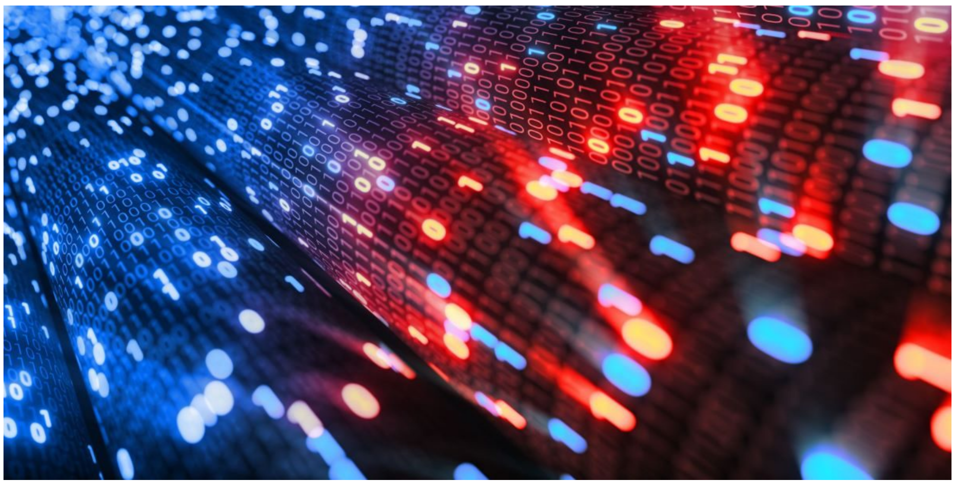
A cascade of compromise: unveiling Lazarus’ new campaign

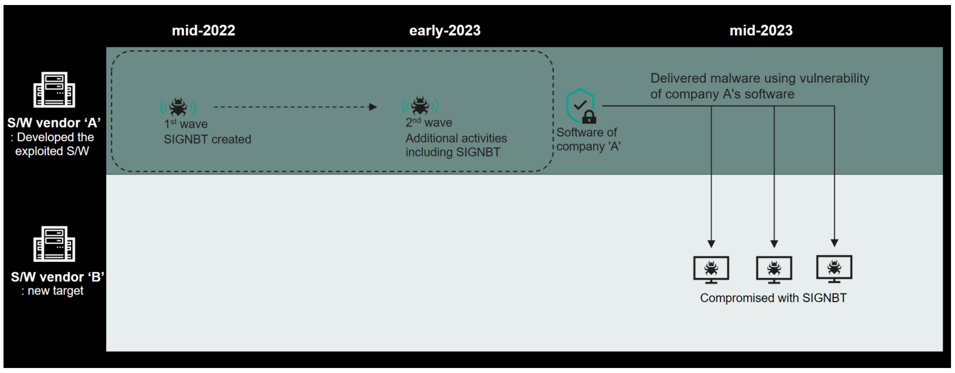


Authors

- Seongsu Park

Earlier this year, a software vendor was compromised by the Lazarus malware delivered through unpatched legitimate software. What’s remarkable is that these software vulnerabilities were not new, and despite warnings and patches from the vendor, many of the vendor’s systems continued to use the flawed software, allowing the threat actor to exploit them. Fortunately, a proactive response by us detected an attack on another vendor and effectively thwarted the attacker’s efforts.

Upon further investigation, we discovered that the software vendor that developed the exploited software had previously fallen victim to Lazarus several times. This recurring breach suggested a persistent and determined threat actor with the likely objective of stealing valuable source code or tampering with the software supply chain, and they continued to exploit vulnerabilities in the company’s software while targeting other software makers.


mid-2022
early-2023
mid-2023
S/W vendor ‘A’ : Developed the exploited S/W
S/W vendor ‘B’ : new target
1st wave SIGNBT created
2nd wave Additional activities including SIGNBT
Software of company ‘A’
Delivered malware using vulnerability of company A’s software
Compromised with SIGNBT

Infection timeline

The adversary demonstrated a high level of sophistication, employing advanced evasion techniques and introducing SIGNBT malware for victim control. In addition, other malware found in memory included Lazarus’ prominent LPEClient, a tool known for victim profiling and payload delivery that has previously been observed in attacks on defense contractors and the cryptocurrency industry.

Executive summary:

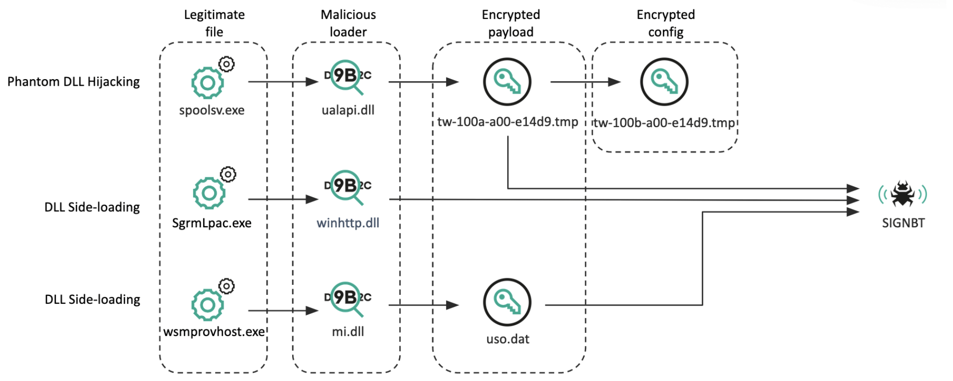
- A software vendor was compromised through the exploitation of another high-profile software.
- The SIGNBT malware used in this attack employed a diverse infection chain and sophisticated techniques.
- LPEClient used in this attack was observed executing a range of targeted attacks associated with the Lazarus group.

For more information, please contact: intelreports@kaspersky.com

SIGNBT loader

In mid-July 2023, we detected a series of attacks on several victims who had been targeted through legitimate security software designed to encrypt web communications using digital certificates. The exact method by which this software was exploited to deliver the malware remains elusive. However, we identified post-exploitation activity within the processes of the legitimate software. In one instance, while examining the memory of the compromised security software from a victim's system, we discovered the presence of the SIGNBT malware accompanied by a shellcode. This shellcode was responsible for launching a Windows executable file directly in memory.

The actor uses various tactics to establish and maintain persistence on compromised systems. These include the creation of a file called ualapi.dll in the system folder, which is automatically loaded by the spoolsv.exe process at each system boot. Additionally, in several instances, registry entries were recorded to execute legitimate files for the purpose of malicious side-loading, further ensuring a resilient persistence mechanism.



Methods for loading the final payload

Leveraging the spoolsv.exe process for hijacking purposes is a long-standing strategy for Lazarus. Automatically loading the ualapi.dll file after each reboot is not a new technique for this actor. We have seen similar tactics used by the Gopuram malware in the past.

The malicious ualapi.dll file was developed using a public source code known as Shareaza Torrent Wizard. It follows a typical Lazarus group approach of utilizing public source code as a foundation and injecting specific malicious functions into it. This loader malware has a routine to verify the victim. It retrieves the victim's MachineGuid by reading it from the Windows registry and then compares it with an embedded MachineGuid value. To access this embedded MachineGuid value, the malware locates the sequence "43 EB 8C BD 1D 98 3D 14" and reads the DWORD immediately following it. Only if the victim's MachineGuid matches the expected one does the malware proceed to the next step. The malware then reads the payload from a hard-coded file path and continues its malicious activities.

- Payload path: C:\Windows\system32\config\systemprofile\appdata\Local\tw-100a-a00-e14d9.tmp

The loader process retrieves the first 32 bytes from tw-100a-a00-e14d9.tmp and uses this data as an AES decryption key to decrypt the remaining contents. Once decrypted, the payload, a Windows executable identified as SIGNBT, is loaded directly into memory. In this case, the loaded payload also reads the configuration file from the same path, but with a slightly different file name.

- Config file: C:\Windows\system32\config\systemprofile\appdata\Local\tw-100b-a00-e14d9.tmp

Inside this file is a base64-encoded string, mirroring the approach used in the previous SIGNBT malware method. The first 32 characters of this string serve as the AES decryption key, while the subsequent data contains configuration information used by the malware. This decrypted configuration data includes details such as three C2 addresses, which are referred to as proxies, sleep intervals, version information, monitored targets, and various other parameters critical to the malware's operation.

SIGNBT

The majority of SIGNBT malware instances are launched through the malware loader, which operates exclusively in memory. Upon execution, the malware begins communicating with the C2 server by sending a beacon after initialization of its configuration data. In its C2 communication, the malware uses distinctive strings that start with SIGNBT. This unique characteristic has earned it the designation of SIGNBT. In addition, the malware uses different prefixes at each stage of its C2 operation to verify and maintain its activities.

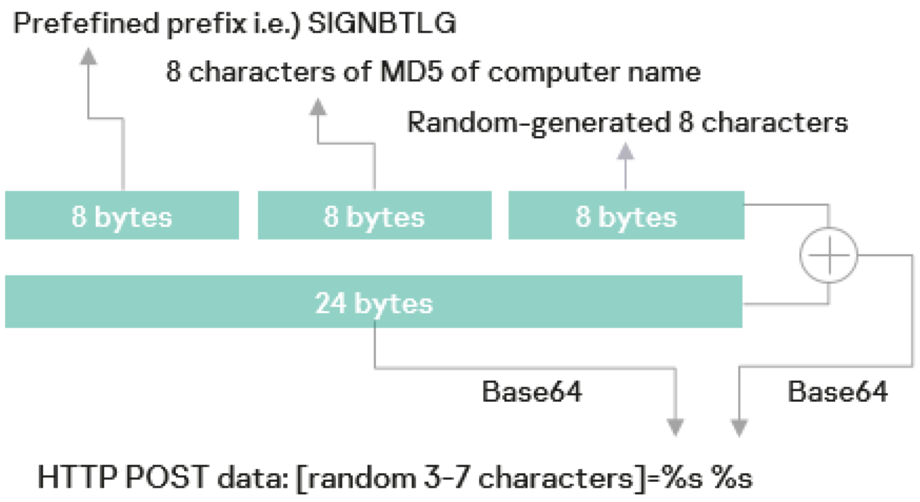
Prefix name	**Description**
SIGNBTLG	Initial connection.
SIGNBTKE	Success – update the key and ask for a profiling process.
SIGNBTGC	Ask for commands.

SIGNBTFI	Operation failed.
SIGNBTSR	Operation success.

The malware employs a multi-step process to create a 24-byte value for various purposes. First, it generates this value with the following components:

1. **8 bytes of hard-coded value (SIGNBTLG):** this is a fixed part of the value and serves to validate the legitimacy of the client’s connection.
2. **8 bytes from the MD5 hash of the hostname:** the first 8 bytes of the MD5 hash of the victim’s computer name are included, helping to distinguishing each victim.
3. **8 bytes of randomly generated identifier:** another 8 bytes are randomly generated, probably used for session identifiers.

After creating this 24-byte value, the malware generates an additional 24 bytes of random data. These two sets of 24 bytes are then XORed together using another randomly generated 24-byte key. Subsequently, both the resulting value and the 24-byte key are encoded with base64. Finally, these encoded values are combined with either three or seven randomly generated HTTP parameter names. In all future C2 communications, the malware uses a similar structure, making it more challenging to detect and analyze its communications.



Prefefined prefix i.e.) SIGNBTLG
8 characters of MD5 of computer name
Random-generated 8 characters
8 bytes
8 bytes
8 bytes
24 bytes
Base64
Base64
HTTP POST data: [random 3-7 characters]=%s %s

Structure of HTTP POST data

The malware uses a mechanism to validate the response data received from the C2 server. Specifically, it checks to see if the response data contains a hard-coded HTML script.

```
<!DOCTYPE html><html><head></head><body marginwidth="0" marginheight="0"
style="background-color:transparent"><script>
[delivered data]
</script></body></html>
```

During the validation process, the malware decodes the first 12 bytes from the C2 server using base64, replacing the spaces with plus signs to create a seven-character string. This process is then repeated with the next 12 bytes. The first seven characters from each set are then XORed and compared to the “success” string. This repetitive procedure is applied to every HTTP communication sequence to verify that the response aligns with the expected “success” criterion.

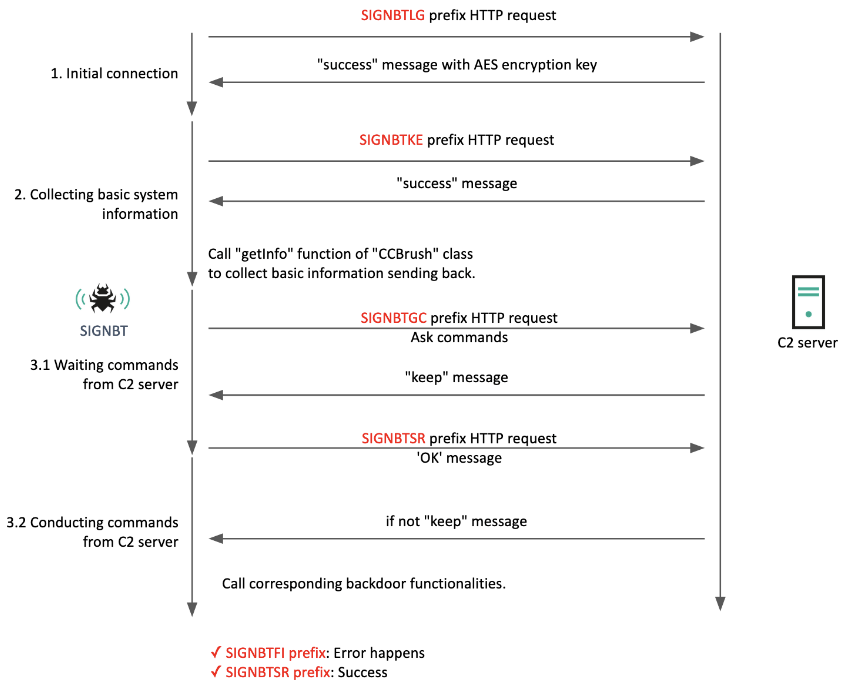
Next, the malware sends HTTP requests with the SIGNBTKE header, and if it receives a “success” message from the C2 server, it activates the getInfo function within the CCBrush class. This function gathers various information about the victim’s computer, such as computer name, product name, OS details, system uptime, CPU information, system locale, time zone, network status, and malware configuration data. After sending this system-specific information, the malware sends another HTTP request with the SIGNBTGC prefix, this time using a randomly chosen embedded HTTP parameter from a list of 100 possible parameter names.

```
client, output, h, slotname, adk, adf, pi, w, format, url, ea, flash, tt_state, dt, bpp,
bdt, idt, shv, ptt, saldr, frm, ife, pv, ga_vid, ga_sid, ga_hid, ga_fc, nhd, u_tz, u_his,
u_java, u_h, u_w, u_ah, u_aw, u_cd, u_nplug, u_nmime, adx, ady, biw, bih, isw, ish, ifk,
scr_x, scr_y, eid, oid, pvsid, pem, loc, eae, brdim, vis, rsz, abl, pfx, fu, bc, ifi, uci,
fsb, dtd, atyp, ei, s, t, bl, imn, ima, imad, aftp, adh, conn, ime, imex, imeh, imea,
```

6 imeb, wh, scp, net, mem, sto, sys, rt, zx, su, tb, calp, rui, u, XU, TREX, UID, SID, dr,

7 XDR, dt

The data received from the C2 server is decrypted using AES with a decryption key obtained from a SIGNBTLG HTTP request. If the decrypted data is “keep”, the malware responds with an “OK” message using the SIGNBTSR prefix, indicating a successful communication. If there are problems, the malware uses the SIGNBTFI prefix to convey the nature of the problem or failure in communication. To summarize, the C2 communication process can be described as follows:


SIGNBTLG prefix HTTP request
"success" message with AES encryption key
1. Initial connection
SIGNBTKE prefix HTTP request
"success" message
2. Collecting basic system information
Call "getInfo" function of "CCBrush" class to collect basic information sending back.
SIGNBT
SIGNBTGC prefix HTTP request
Ask commands
3.1 Waiting commands from C2 server
"keep" message
SIGNBTSR prefix HTTP request
'OK' message
C2 server
3.2 Conducting commands from C2 server
if not "keep" message
Call corresponding backdoor functionalities.
✓ SIGNBTFI prefix: Error happens
✓ SIGNBTSR prefix: Success

C2 communication process

If the delivered data does not equal “keep”, indicating that specific instructions or actions are required, the malware proceeds to invoke the corresponding class and function for backdoor behavior. The SIGNBT malware is equipped with an extensive set of functionalities designed to exert control over the victim’s system. To perform these functions, the malware receives instructions from the C2 server in the form of a class name, function name, and any necessary parameters. It then executes the relevant function embedded in the malware’s codebase.

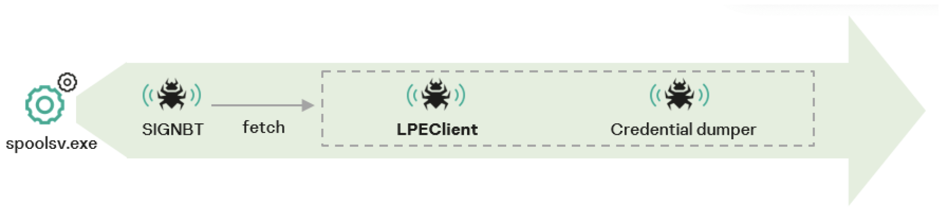
Class name	**Function name**
CCBrush	getInfo, testConnect, setSleep, setHibernate, sendConfig, setConfig
CCList	getProcessList, processKill, runFile, runAsUser, injectDll, freeDll
CCComboBox	getDriveList, getFileDir, changeFileTime, secDelete, folderProperty, changeFileName, makeNewFolder
CCButton	startDownload, upFile, selfMemload, scrCapture
CCBitmap	ping, netshAdvfirewall, netstat, reg, sc, whoami, arp, nslookup, systeminfo, ipconfig, net, ver, wmic, deploy, copy

The name of each backdoor command is straightforward, implementing commonly used Windows commands such as ping, netstat, and systeminfo. It’s important to note that the backdoor is capable of implanting an additional payload for auto execution, internally named “deploy”. This backdoor function receives file paths via command-line arguments decrypted with AES. Using this command, SIGNBT has been observed to implant the phantom DLL we already described in the SIGNBT loader section above.

Based on the analysis, it is evident that the actor’s initial compromise of the victim involved exploiting vulnerabilities within the software exploit. They then proceeded to deploy the SIGNBT malware using a DLL side-loading technique. Furthermore, the actor used the backdoor capability “deploy” to implant an additional payload for automated execution. This multifaceted attack demonstrates a high level of sophistication and a deliberate effort to infiltrate and maintain control over the victim’s system.

LPEClient

Using the comprehensive backdoor as described above, the actor deploys additional malware in the victim’s memory. Notably, these newly delivered malware variants predominantly execute in the system’s memory only, without touching the disk. Based on our telemetry, the actor has been observed to deliver such tools as LPEClient and credential dumping utilities to the victim machines.

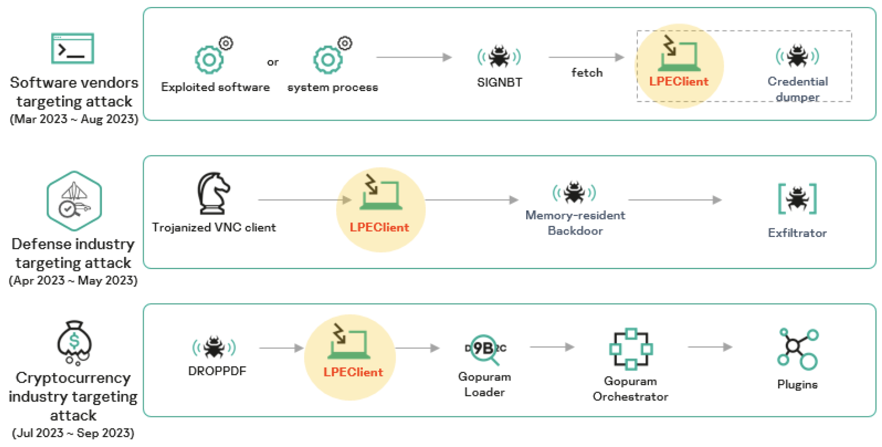


Additional payload delivered by SIGNBT

The LPEClient malware is not new and was first discovered during an investigation of a defense contractor attack in 2020. It is designed to collect victim information and download additional payloads from a remote server to run in memory. Although it has been previously noted in our threat intelligence reports to our customers, recent discoveries indicate that LPEClient has undergone significant evolution. It now employs advanced techniques to improve its stealth and avoid detection, such as disabling user-mode syscall hooking and restoring system library memory sections. This indicates a continued effort by the threat actors to increase the sophistication and effectiveness of their malware.

Connections with other campaigns

One of the malware strains employed in this attack, known as LPEClient, has featured prominently in recent activity attributed to the Lazarus group. This particular malware consistently serves as the initial infection vector, enabling victim profiling and facilitating the delivery of additional payloads. Over an extended period of time, one of these campaigns specifically targeted defense contractors and nuclear engineers. In a recent incident, the threat actor compromised a victim by delivering LPEClient via a Trojanized VNC or Putty client for an intermediate infection. Another campaign targeting the cryptocurrency industry was discovered in July 2023. In this financially motivated campaign, the actor leveraged the Gopuram malware, associated with the 3CX supply chain attack. Interestingly, the actor also used LPEClient malware in this case. Prior to the introduction of the Gopuram cluster, LPEClient was used to deliver the subsequent malware. These three campaigns attributed to Lazarus in 2023 illustrate different initial infection vectors and infection chains, but they consistently relied on LPEClient malware to deliver the final payload.



The infection chains of the three campaigns attributed to Lazarus in 2023

Conclusions

The Lazarus group remains a highly active and versatile threat actor in today's cybersecurity landscape. The threat actor has demonstrated a profound understanding of IT environments, refining their tactics to include exploiting vulnerabilities in high-profile software. This approach allows them to efficiently spread their malware once initial infections are achieved. Moreover, the activities of this notorious actor transcend geographic boundaries and industry sectors. They have targeted various industries, each with distinct objectives and using different tools, tactics and techniques. This underscores their recent and ongoing activity characterized by sophisticated methods and unwavering motivations.

Indicators of Compromise

SIGNBT loader

9cd90dff2d9d56654dbecdcd409e1ef3	%system%\ualapi.dll

88a96f8730b35c7406d57f23bbba734d	%system%\ualapi.dll
54df2984e833ba2854de670cce43b823	%system%\ualapi.dll
Ae00b0f490b122ebab614d98bb2361f7	%system%\ualapi.dll
e6fa116ef2705ecf9677021e5e2f691e	
31af3e7fff79bc48a99b8679ea74b589	C:\GoogleD\Coding\JS\Node\winhttp.dll

SIGNBT
9b62352851c9f82157d1d7fcafeb49d3

LPEClient

3a77b5054c36e6812f07366fb70b007d	%systme%\wbem\wbemcomn.dll
E89fa6345d06da32f9c8786b65111928	%ProgramData%\Microsoft\Windows\ServiceSetting\ESENT.dll

File path
C:\GoogleD\Coding\JS\Node\SgrmLpac.exe
C:\GoogleD\Coding\JS\Node\winhttp.dll
C:\Windows\system32\config\systemprofile\appdata\Local\tw-100a-a00-e14d9.tmp
C:\Windows\system32\config\systemprofile\appdata\Local\tw-100b-a00-e14d9.tmp
C:\ProgramData\ntuser.008.dat
C:\ProgramData\ntuser.009.dat
C:\ProgramData\ntuser.001.dat
C:\ProgramData\ntuser.002.dat
C:\ProgramData\Microsoft\Windows\ServiceSetting\ESENT.dll

C2 servers
hxxp://ictm[.]or[.]kr/UPLOAD_file/board/free/edit/index[.]php
hxxp://samwoosystem[.]co[.]kr/board/list/write[.]asp
hxxp://theorigin[.]co[.]kr:443/admin/management/index[.]php
hxxp://ucware[.]net/skins/PHPMailer-master/index[.]php
hxxp://www[.]friendmc[.]com/upload/board/asp20062107[.]asp
hxxp://www[.]hankooktop[.]com/ko/company/info[.]asp
hxxp://www[.]khmcpharm[.]com/Lib/Modules/HtmlEditor/Util/read[.]cer
hxxp://www[.]vietjetairkorea[.]com/INFO/info[.]asp
hxxp://yoohannet[.]kr/min/tmp/process/proc[.]php
hxxps://admin[.]esangedu[.]kr/XPaySample/submit[.]php
hxxps://api[.]shw[.]kr/login_admin/member/login_fail[.]php
hxxps://hicar[.]kalo[.]kr/data/rental/Coupon/include/inc[.]asp
hxxps://hspje[.]com:80/menu6/teacher_qna[.]asp
hxxps://kscmfs[.]or[.]kr/member/handle/log_proc[.]php
hxxps://kstr[.]radiology[.]or[.]kr/upload/schedule/29431_1687715624[.]inc
hxxps://little-pet[.]com/web/board/skin/default/read[.]php
hxxps://mainbiz[.]or[.]kr/SmartEditor2/photo_uploader/popup/edit[.]asp
hxxps://mainbiz[.]or[.]kr/include/common[.]asp
hxxps://new-q-cells[.]com/upload/newsletter/cn/frame[.]php
hxxps://pediatrics[.]or[.]kr/PubReader/build_css[.]php
hxxps://pms[.]nninc[.]co[.]kr/app/content/board/inc_list[.]asp
hxxps://safemotors[.]co[.]kr/daumeditor/pages/template/template[.]asp
hxxps://swt-keystonevalve[.]com/data/editor/index[.]php
hxxps://vnfmal2022[.]com/niabbs5/upload/gongji/index[.]php
hxxps://warevalley[.]com/en/common/include/page_tab[.]asp
hxxps://www[.]blastedlevels[.]com/levels4SqR8/measure[.]asp
hxxps://www[.]droof[.]kr/Board/htmlEdit/PopupWin/Editor[.]asp
hxxps://www[.]friendmc[.]com:80/upload/board/asp20062107[.]asp
hxxps://www[.]hanlasangjo[.]com/editor/pages/page[.]asp
hxxps://www[.]happinesscc[.]com/mobile/include/func[.]asp
hxxps://www[.]healthpro[.]or[.]kr/upload/naver_editor/subview/view[.]inc
hxxps://www[.]medric[.]or[.]kr/Controls/Board/certificate[.]cer
hxxps://www[.]muijae[.]com/daumeditor/pages/template/simple[.]asp
hxxps://www[.]muijae[.]com/daumeditor/pages/template/template[.]asp
hxxps://www[.]nonstopexpress[.]com/community/include/index[.]asp
hxxps://www[.]seoulanesthesia[.]or[.]kr/mail/mail_211230[.]html
hxxps://www[.]seouldementia[.]or[.]kr/_manage/inc/bbs/jiyeuk1_ok[.]asp
hxxps://www[.]siriuskorea[.]co[.]kr/mall/community/bbs_read[.]asp
hxxps://yoohannet[.]kr/min/tmp/process/proc[.]php

MITRE ATT&CK Mapping

Tactic	Techniques
Initial Access	T1189
Execution	T1203
Persistence	T1547.012, T1574.002
Privilege Escalation	T1547.012
Defense Evasion	T1140, T1574.002, T1027.001, T1027.002, T1620
Credential Access	T1003.001
Discovery	T1057, T1082, T1083
Collection	T1113
Command and Control	T1071.001, T1132.002, T1573.001
Exfiltration	T1041