Snake and XLoader (Mac Version) – Malware Book Reports

By muzi View all posts

Archived: 2026-04-05 14:31:51 UTC

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:

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.

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.

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.

Figure 4: GetOS Function

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

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
- fI4sWHk: PE32 executable (GUI) Intel 80386 Mono/.Net assembly, for MS Windows
- kIbwf02ld: 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`.

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

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.

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

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.

Figure 5: dnSpy Error

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

Figure 7: 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.

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.

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().

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.

Figure 11: otool -L outputs only dylib

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.

Figure 13: Example Stack String Within XLoader

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.

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.

Figure 16: Hidden App's Info.plist

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 [R BX + 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
                        (48|49|4c|4d) 8? ?? [0-16]                      // MOV RDI, RBX
                        (E8|FF) ?? ?? ?? ??                             // Call func
        }

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

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