JASPER THE UNFRIENDLY LOADER

The cyber threat intelligence team at Solis continues to track new and emerging malware families and ransomware that often target various industries. We recently collaborated with SentinelOne and Stairwell on a newly discovered loader, JasPer Loader, that has been observed loading IcedID and a second-stage shellcode downloader. The presence of this malware running in an environment could lead to significant damage and could allow an adversary to load additional malware into a target network.

- JasPer Loader is a lightweight Dynamic Link Library (DLL) file that mimics legitimate software based on JasPer, a collection of software (i.e., a library and programs) for the coding and manipulation of images[1]. The DLL has been trojanized to contain an encrypted payload that is executed upon calling an added export to the DLL.
- The DLL is password protected and will not function without a user-provided key. Additionally, hard-coded keys are utilized to further protect staged payloads and the embedded C2 address from signature-based detection.
- The loader is designed to be configurable, as it is able to execute an arbitrary DLL payload hosted at an arbitrary staging URL. In this case, the third-party file sharing site qaz[.]im was utilized to host the staged payload.

Solis worked with Stairwell and SentinelOne to carry out a collaborative analysis to learn more about JasPer Loader's capabilities. A SentinelOne Active Response (STAR) Rule was created by Solis and validated by SentinelOne to detect and mitigate this malicious loader from running. In addition, a YARA rule was created by Stairwell to detect and identify the presence of this malware. Further details regarding how this malware works can be found below.

STATIC ANALYSIS

A static analysis was conducted on a sample of JasPer Loader with a filename of `JasPer.dll` and a SHA256 hash of `78bb0fd18def2602188ca0004ac5428ed039b8abef4926c7e9e9b908a1efa5b8`. SentinelOne observed that the DLL does contain legitimate code from JasPer and is a trojanized version of a DLL distributed with ImageMagick, a widely used open-source project for editing and manipulating digital images[2].

The legitimate executable has a SHA256 hash of `1a9c8a4f300af28e12ff33d992deb4c8e203881b555c4b3e79d0c2e1605f3d7a`.

[1] https://jasper-software.github.io/jasper-manual/releases/version-2.0.33/html/index.html
[2] https://imagemagick.org/index.php

The malicious executable contains an additional export, named `Push`, that contains the malicious functionality, and contains the following PE metadata:

Field	Value
ProductName	JasPer JPEG v2 compression library
FileDescription	ImageMagick library and utility programs
OriginalFilename	JasPer
InternalName	ImageMagick
FileVersion	1.701.0 (8 Feb 2004)
ProductVersion	1.701.0 (8 Feb 2004)
CompanyName	Michael David Adams
LegalCopyright	Copyright © 2001-2003 Michael David Adams
Comments	http://www.ece.uvic.ca/~mdadams/jasper/
LangID	040904B0
Charset	Unicode
Language	English (United States)
PDB Path	E:\repo\ImageMagick\ImageMagick-6.9.3\vc14\x64\bin\CORE_RL_jp2_.pdb
Compile Time	2016-03-27 16:02:52

Table 1: PE Metadata

The trojanized DLL successfully evaded detection by all AV vendors on VirusTotal when it was initially uploaded on March 29, 2023.

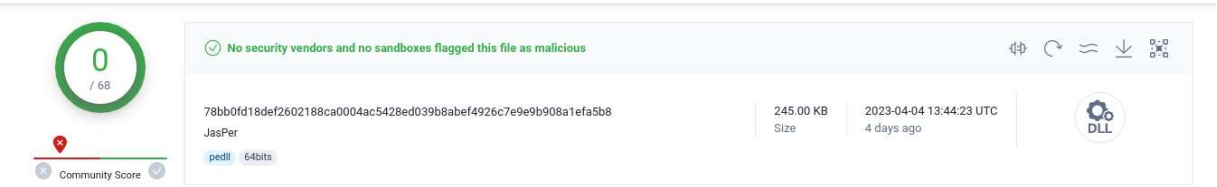

Figure 1: Initial Detections

FIRST-STAGE: DLL

Upon execution, the DLL attempts to perform XOR decryption of second-stage shellcode stored in the `.rsrc` directory. The DLL does so by expecting a key to be passed as an argument to the exported function `Push` with the command line argument `/k`. Note that this means that the DLL is most likely meant to be run using a command line tool such as `Rundll32.exe`.

Using the provided command line key, the loader decrypts the second-stage shellcode in a buffer in memory and calls the shellcode to begin execution. In the case of this sample, the password that must be provided to properly decrypt the second-stage shellcode is the string `Pro9lom`.The decryption of the second-stage payload in this sample can be replicated with the following one-liner in Binary Refinery[3]:

```
emit 78bb0fd18def2602188ca0004ac5428ed039b8abef4926c7e9e9b908a1efa5b8 |
       vsect .rsrc        |
       snip 0x624:0x2593 |
       xor Pro9lom        |
       peek
```

```
→  JasPer emit 78bb0fd18def2602188ca0004ac5428ed039b8abef4926c7e9e9b908a1efa5b8 |
               vsect .rsrc        |
               snip 0x624:0x2593 |
               xor Pro9lom        |
               peek

--------------------------------------------------------------------------------------------------------------------------------
08.047 kB; 80.54% entropy; zlib compressed data
--------------------------------------------------------------------------------------------------------------------------------
0000: 48 89 5C 24 08 48 89 74 24 10 44 89 44 24 18 57 48 83 EC 30 48 8B F9 48 8B DA B9 1F 1E 5A  H.\$.H.t$.D.D$.WH..0H..H.....Z
001E: D4 E8 7C 19 00 00 65 48 8B 14 25 30 00 00 00 48 8B F0 4C 8B 42 60 4D 8B 48 18 49 83 C1 10  ..|...eH..%0...H..L.B`M.H.I...
003C: 49 8B 11 49 3B D1 0F 84 C1 00 00 00 48 8B 4A 30 48 3B F9 72 0B 8B 42 40 48 03 C1 48 3B F8  I..I;.......H.J0H;.r..B@H..H;.
005A: 72 05 48 8B 12 EB DE 48 85 C9 0F 84 9F 00 00 00 4C 8D 44 24 20 48 8B D3 E8 19 1C 00 00 85  r.H....H........L.D$.H........
0078: C0 0F 84 8A 00 00 00 4C 63 44 24 28 45 33 C9 48 8B 5C 24 20 4D 85 C0 7E 28 48 8B D3 41 8B  .......LcD$(E3.H.\$.M..~(H..A.
0096: C1 25 03 00 00 80 7D 07 FF C8 83 C8 FC FF C0 48 98 41 FF C1 8A 4C 04 50 30 0A 48 FF C2 49  .%....}........H.A...L.P0.H..I
00B4: 83 E8 01 75 DB 33 C0 38 03 74 30 48 FF C0 80 3C 18 00 75 F7 48 85 C0 74 22 48 8B D3 48 8B  ...u.3.8.t0H...<..u.H..t"H..H.
00D2: CF E8 44 00 00 00 33 C0 38 03 74 09 48 FF C0 80 3C 18 00 75 F7 48 FF C3 48 03 D8 EB CA 65  ..D...3.8.t.H...<..u.H..H....e
00F0: 48 8B 04 25 30 00 00 00 33 D2 4C 8B 44 24 20 48 8B 48 60 48 8B 49 30 FF D6 48 8B 5C 24 40  H..%0...3.L.D$.H.H`H.I0..H.\$@
010E: 33 C0 48 8B 74 24 48 48 83 C4 30 5F C3 CC 48 89 5C 24 08 55 56 57 41 54 41 55 41 56 41 57  3.H.t$HH..0_..H.\$.UVWATAUAVAW
--------------------------------------------------------------------------------------------------------------------------------
```

Figure 2: Shellcode Decryption in Binary Refinery

SECOND-STAGE: SHELLCODE

Following the above execution chain, the DLL executes the decrypted shellcode, which acts as a downloader that utilizes a staging URL hosted on at `qaz[.]im`. The website `qaz[.]im` is an anonymous email, paste, and file sharing provider, allowing users to host files for 24-hour periods before deletion.

[3] https://github.com/binref/refinery

Notably, the threat actor utilized google translate's website feature[4], which generates a translated view of the webpage and generates a link to the translated view with the domain `translate[.]goog`. This is likely to bypass domain reputation checks, as a request to Google's infrastructure is less anomalous than an anonymous paste site. The staging URL is also embedded within the `.rsrc` section of JasPer Loader, and is decrypted with a hard-coded, 4-byte XOR key: `4D 95 0A 00`. The full, decrypted URL has been included below:

```
hxxps[:]//qaz-im[.]translate[.]goog/load/3Thihz/ce371842-bcf0-4cd8-b22e-
6f82b4e9107f?_x_tr_sl =auto&_ x_tr_tl =en&_x_tr_hl=ru&_x_tr_pto=wapp
```

The decryption of the embedded staging URL can be replicated with the following one-liner from Binary Refinery:

```
emit 78bb0fd18def2602188ca0004ac5428ed039b8abef4926c7e9e9b908a1efa5b8 |
       vsect .rsrc            |
       snip 0x2593:0x2616     |
       xor h:4D950A00         |
       peek
```

```
→  JasPer emit 78bb0fd18def2602188ca0004ac5428ed039b8abef4926c7e9e9b908a1efa5b8 |
               vsect .rsrc            |
               snip 0x2593:0x2616     |
               xor h:4D950A00         |
               peek

------------------------------------------------------------------------------------------------------------------------
00.131 kB; 62.59% entropy; ASCII text, with no line terminators
------------------------------------------------------------------------------------------------------------------------
0000: 68 74 74 70 73 3A 2F 2F 71 61 7A 2D 69 6D 2E 74 72 61 6E 73 6C 61 74 65 2E 67 6F 6F 67 2F  https://qaz-im.translate.goog/
001E: 6C 6F 61 64 2F 33 54 68 69 68 7A 2F 63 65 33 37 31 38 34 32 2D 62 63 66 30 2D 34 63 64 38  load/3Thihz/ce371842-bcf0-4cd8
003C: 2D 62 32 32 65 2D 36 66 38 32 62 34 65 39 31 30 37 66 3F 5F 78 5F 74 72 5F 73 6C 3D 61 75  -b22e-6f82b4e9107f?_x_tr_sl=au
005A: 74 6F 26 5F 78 5F 74 72 5F 74 6C 3D 65 6E 26 5F 78 5F 74 72 5F 68 6C 3D 72 75 26 5F 78 5F  to&_x_tr_tl=en&_x_tr_hl=ru&_x_
0078: 74 72 5F 70 74 6F 3D 77 61 70 70                                                            tr_pto=wapp
------------------------------------------------------------------------------------------------------------------------
```

Figure 3: Staging URL Decryption in Binary Refinery

At the time of analysis, the 24-hour expiration for the staged payload had already expired.

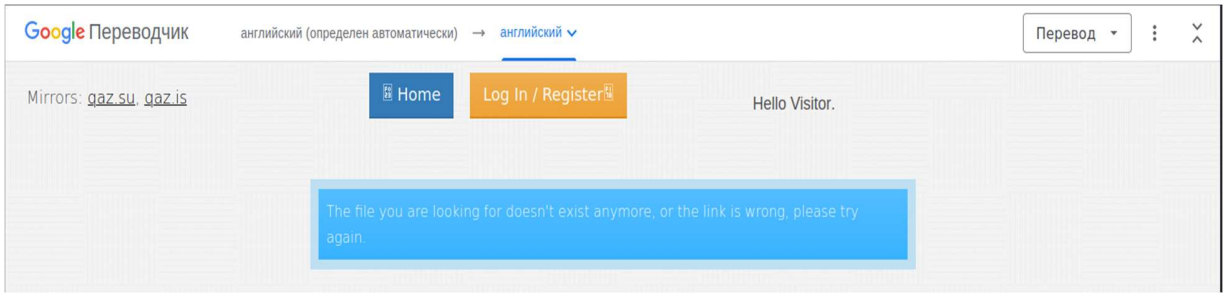

Figure 4: Staging Website

[4] https://translate.google.com/?op=websites

While no staged payload was observed, the shellcode's functionality reveals features about the staged payload. The code expects the staged payload to be XORed with the key `91 17 A8 04` and contain structured data containing the following strings:

- `"/object/"`
- `"<content>"`
- `"<export>"`
- `"<params>"`
- `"<name>"`
- `"<drop_disk>"`
- `"<frd_dll>"`

The shellcode will parse the structured data and execute the payload according to options specified. The shellcode expects the payload to be a PE file and can either load the file into memory and execute it, including a specific exported function from a DLL, or it can write the file to the %TEMP% directory.

CONCLUSION AND RELATED SAMPLE

This loader is relatively simple but robust enough to execute arbitrary payloads so long as they are in the correct format and appears to be under ongoing development. Due to the small number of samples at this time, it is unclear whether this is a single group's proprietary loader or if this is a loader incorporated into the larger eCrime economy. SentinelOne identified a related sample of JasPer Loader that contains an embedded IcedID payload[5] instead of a downloader shellcode, but at this time there is not enough information to assess whether this loader is developed by the same actor as IcedID. The developer of this loader is capable enough to exhibit a deep understanding of common static analysis techniques as well as to evade detection by all major AV vendors on VirusTotal at the time of upload.

The details of the related file that loads IcedID are as follows:

Field	Value
SHA256	f41ea8e983c0e9e63eb3b0066eab277c45841f0c38f741e7486e846313b8c042
C2 Domains	afrakonla[.]com pinchersoftqum[.]com

[5] https://tria.ge/230310-an7gyacf5x

Campaign ID	607958445

Table 2: IcedID JasPer Loader Sample Details

ACCOLADES

Solis would like to give a special thank you to RedSense for their assistance in this analysis. Stairwell was also instrumental in being able to create a YARA detection rule based on the malware analysis that they provided on this file. The SentinelOne Rule was created by Solis, and it was validated by SentinelOne for proper detection and mitigation for environments that run SentinelOne Endpoint Security. Due to this collaborative effort, we were able to alert the wider security community in detecting and protecting against malware like this.

Att&CK Tactic	Technique
Defense Evasion	T1027 Obfuscated Files or Information
Execution	T1129 Shared Modules
Staged Capabilities	T1608 Resource Development (sub-technique)
Command and Control	T1105 Ingress Tool Transfer
Defense Evasion	T1620 Reflective Code Loading
Defense Evasion	T1036 Masquerading

Table 3: MITRE Att&CK Tactics

INDICATORS OF COMPROMISE

Att&CK Tactic	Technique
SHA256:	f41ea8e983c0e9e63eb3b0066eab277c45841f0c38f741e7486e846313b8c042
SHA256:	78bb0fd18def2602188ca0004ac5428ed039b8abef4926c7e9e9b908a1efa5b8
URL	hxxps[:]qaz-im[.]translate[.]goog/load/3Thihz/ce371842-bcf0-4cd8-b22e-6f82b4e9107f?_x_tr_sl =auto&_ x_tr_tl =en&_x_tr_hl=ru&_x_tr_pto=wapp
Domain	afrakonla[.]com
Domain	pinchersoftqum[.]com

SOLIS DETECTION RULE

The listed detections were created to detect this malware running in SentinelOne EDR environments.

Query:

(Name Contains Anycase "rundll32" AND CmdLine In Contains Anycase ("Push /k")) OR (TgtFileInternalName Contains "ImageMagick" AND TgtFileDescription Contains "ImageMagick library and utility programs")

Note: The above query is not suitable for usage as a STAR Rule, given that some false positive alerting can trigger.

STAR Rule:

Name Contains Anycase "rundll32" AND CmdLine In Contains Anycase ("Push /k")

STAIRWELL YARA RULE

The posted YARA rule was created by Daniel Mayer at Stairwell whose analysis contributed to building a YARA detection rule for this malware.

YARA Rule

```
rule JasPer_Downloader
{
     meta:
        author = "Daniel Mayer (daniel@stairwell.com)"
        description = "A rule for detecting JasPer loader"
        version = "1.0"
        date = "2023-03-30"
        sha256 = "78bb0fd18def2602188ca0004ac5428ed039b8abef4926c7e9e9b908a1efa5b8"

    strings:
        // shellcode payload
        $decrypt1 = {
            FF C0                                   // inc     eax
            48 98                                   // cdqe
            41 FF C1                                // inc     r9d
            8A 4C ?? ??                             // mov     cl, ??
            30 0A                                   // xor     [rdx], cl
            48 FF C2                                // inc     rdx
            49 83 E8 01                             // sub     r8, 1
         }

        $decrypt2 = {
            FF C0                                   // inc     eax
            48 98                                   // cdqe
            41 FF C1                                // inc     r9d
            8A 8C ?? ?? ?? ?? ??                    // mov     cl, ??
            30 0A                                   // xor     [rdx], cl
            48 FF C2                                // inc     rdx
            49 83 E8 01                             // sub     r8, 1
         }

        // file on disk
        $decrypt_enc_buffer = {
            4C 8B C1                                // mov     r8, rcx
            33 D2                                   // xor     edx, edx
            8B C6                                   // mov     eax, esi
            41 F7 F6                                // div     r14d
            FF C6                                   // inc     esi
            8A 44 ?? ??                             // mov     al, ??
            41 30 00                                // xor     [r8], al
            49 FF C0                                // inc     r8
         }


    condition:
        any of them
}
```