Unveiling JsOutProx: A New Enterprise Grade Implant

blog.yoroi.company/research/unveiling-jsoutprox-a-new-enterprise-grade-implant/

December 20, 2019

12/20/2019

Introduction

During our threat intelligence source monitoring operations, we spotted a new sophisticated malware implant that seems to be unrelated to mainstream cyber weapons. In fact, the recovered sample raised many interrogatives into the malware research community due to the extensive usage of obfuscation anti-reverse techniques, hardening the investigative efforts.

For this reason, we decided to dig into this piece of malware and figure out its inner secrets, uncovering a modular architecture with advanced offensive capabilities, such as the presence of functionalities able to deal with multi factor authentication (MFA).

Technical Analysis

Hash	6bf0d9a7ca91f27a708c793832b0c7b6e3bc4c3b511e8b30e3d1ca2e3e2b90a7
Threat	JsOutProx
Brief Description	Malicious JS file
Ssdeep	12288:9jAtRUr07Jo0W9vrd6ye8hKaVimlc+/eHFca7 +mJO1Za6D4aYQZdV81u34YYbga0:RAO07JbAvrsype6lZTv

Table 1. Sample information

The starting point is a Javascript file containing more than ten thousand lines of heavily obfuscated code.

The first line of this file embeds a huge array of Base64 encoded elements, but its raw decoding led only to other incomprehensible data, evidencing the presence of a more complex layer of protection.

Figure 1. Array with Base64 encoded elements.
Navigating the code, we identified a series of instructions resembling a sort of initialization that grabbed our attention. The function “t_ey” is used as deobfuscation function for some of the string chucks preconfigured into the “t_ep” array, enabling us to recover some cleartext.

This initial code cleanup revealed interesting information such as some of the static configuration initialized during the initial malware execution stages. Among these info, we recovered also its remote C2 address 91.189.180.199, operated by “ServeTheWorld”, an Norwegian provider renting his servers in Oslo, and a particular tag reporting the name *“JsOutProx”*. Extremely characteristic.

Figure 2. Initialization of basic malware information.
Continuing to analyze the code, we reconstructed the approach used by the attacker to obfuscate the payload: all the necessary information has been encrypted, splitted, and then encoded in Base64 chunks stored into different structures named as *“ta”*, *“t_ep”*, *“t_eq”*.

Figure 3. Other structures containing Base64 data.
As anticipated before, thanks to the decoding routine *“t_ey”* it is possible to retrieve at runtime the cleartext code to reify the structure of the malware, stored in the *“t_fT” object* visible in the next figure.

Figure 4. Core structure of the malware.
The structure contains objects and functions used by the malware to pursue its malicious actions. In many cases we noticed a naming correspondence between couples of objects, for example between *“Outlook”* and *“OutlookPlugin”*, or *“Proxy”* and *“ProxyPlugin” objects*. This indicates the malware has a modular structure containing specific plugins able to perform a wide range of actions, such as exfiltrate data by populating the associated object. For example, the "OutlookPlugin" is able to steal information about emails and contacts and it does that by filling the Outlook object shown in the previous figure.

Each plugin embeds an obfuscated function named *“receive”*, which has the purpose to perform the specific action. This function name is constant and represent a sort of common interface between malware modules.

Figure 5. “receive” function into “InfoPlugin”.

Check-in and Command List

Once created the main structure, the first function ran by the malware is “init”. It is designed to create an identification string for the victim machine, gathering from system information and storing them into the *“t_fT[“ID”]” variable*.

As visible the next picture, the identification string is composed using the computer name, username, OS version. Then, an additional suffix will be appended to it containing the current action the malware is performing. In this specific case it is *“ping”*, a sort of heartbeat to make sure the command and control services is up and running.

Figure 6. Creation of victim ID.
Then, the malware moves in an endless loop in which it invokes the *“receive”* function every 5 seconds. Substantially, this function is an interface that allows the attacker to interact with the implant.

The malware, in fact, connects to the C2 and retrieves a string indicating the next command to execute into the “*np[0x0]”* variable. It has many capabilities and it able to handle a complete infection life-cycle. It can update itself, restart itself, execute another JavaScript code, other VB scripts and even remove its traces.

Figure 7. Switch structure with supported commands.
Summing up, the analyzed version of the implant supports the following actions:

Command	**Action**
upd	Update the implant
rst	Restart the implant
l32	Start another process with the same script
dcn	Kill the implant
rbt	Reboot the machine
shd	Shutdown the machine
lgf	??
ejs	Evaluate Javascript code
epg	??
evb	Execute VisualBasic code
idn	??
sdn	Load a .NET dll
uis	Uninstall the implant

ins	Install the implant
int.g	Send the sleep time to C2
int.s	Update the sleep time

Table 2. Supported commands.

Additionally, the default case of the main loop structure includes a series of IF-Else evaluating the received string in order to check for specific prefixes. Each prefix is associated to a plugin module.

Figure 8. Invocation of plugins.
Analyzing this selection chain was possible to reconstruct the full list of supported plugin for this version of the implant.

Command	**Plugin**
pr	ProcessPlugin
cl	ClipboardPlugin
fi	FilePlugin
lb	LibraryPlugin
do	DownloadPlugin
sc	ScreenPlugin
ou	OutlookPlugin
px	ProxyPlugin
cn	ShellPlugin
tk	TokensPlugin
in	InfoPlugin
ds	DnsPlugin
pm	PromptPlugin

Table 3. Malware's plugins.

At this point, all the attention moves on understand what the functions and the plugins do and how they work. So, we provide, in the next paragraphs, a brief view of the most interesting functions and plugins.

Persistency

By the way, to ensure its survival after reboot, the malware writes a copy of the initial JS script into the “*%appdata%”* and “*%temp%”* folder, setting the persistence on the system through the popular *“HKCU\Software\Microsoft\Windows\CurrentVersion\Run”* registry key.

Figure 9. Setting of persistence.

Plugin Modules

The number of supported plugin indicates the malware is pretty mature and extremely dangerous. Some of these supported plugins encode standard functionalities of many RAT and recon malware, but some of them hides interesting and even uncommon features.

The “Process” Plugin

The ProcessPlugin is able to manipulate other processes running in the system. It can kill them by PID and by name, create new processes through WSH or WMI and also **collect a memory dump** of a specific process. This is an uncommon feature for a common malware, it might indicate operators behind its command and controls could have analytical skills in order to investigate the surrounding environment, or also exfiltrate data through memory scraping.

Figure 10. Part of ProcessPlugin code.

The “Dns” Plugin

The DnsPlugin handles the machine’s DNS configuration. It can send to the C2 the current configuration and also **set a new one**, just like visible in the following screen.

Figure 11. Part of DnsPlugin code.

The “Token” Plugin and the Object SymantecVIP

This plugin is particularly interesting. The name refers to the exfiltration of some type of token. However, the plugin is specifically designed for the **theft of SymantecVIP One Time Password**, the multifactor authentication technology used in enterprise grade Single-Sign-On services adopted in many corporate environments.

Figure 12. Part of TokenPlugin code.

The “Outlook” Plugin

Instead, the OutlookPlugin weaponize the implant with common information stealing capabilities enabling the attackers to gather account information and contact list.

The interesting part of this plugin is that it is the only one info-stealing plugin embedded in the malware.

Commodity malware typically include support to multiple email clients, web browsers and ftp clients, but this one just handle Outlook: the de facto standard email client installed in the majority of enterprise environment.

Figure 13. Part of OutlookPlugin code.

The “Prompt” Plugin

Another interesting functionality is provided by the PromptPlugin. It empowers the attacker to present his victim a custom message prompt provided by the command and control server.

Figure 14. Part of PromptPlugin code.

The Bridge Between JS and .NET

All the JavaScript plugins seem to be only an high-level interface used by the attacker to communicate with his implant. Digging into the code to search for the implementation of the plugins, we come across another mysterious component named *“DotUtil”* which seems to be the link between the JavaScript interface and the implementation of the actual operations.

Also, digging the JS code, a method called *“hasDotnet”* made us wondering about the existence of additional .NET artifacts which may hide the implementation of some of the above-mentioned functionalities. Moreover, this method contains references to the download and the memory loading of an instance of a *“DotUtil”* module.

Figure 15. Part of DotUtil module.
Dynamically analyzing the malware, we were able to intercept the download of a .NET DLL called *“libDotJs.dll”*. Analyzing it, we discover a complete mapping between the functions implemented in it and the ones declared into *“DotUtil”* component.

Figure 16. Mapping between DotUtil and .NET DLL.
From a wider perspective, we can assert with a good confidence that the implant is composed of at least two different layers: the JavaScript interface, delivered to the target machine as first stage and subject to continuous changes to lower its detectability, providing the core mechanism to ensure a flexible access to the target machine, and an advanced functional layer leveraging .NET dependencies.

Figure 17. Macro perspective of the malware composition.

Conclusion

The complexity and the engineering grade of this threat, dubbed *JsOutProx,* are an element of uniqueness in the current panorama. It is a toolkit with peculiar remote access capabilities. From an architectural point of view, *JsOutProx* contains all the function prototypes inside the core engine and could be remotely extended at run time.

The implementation of its functionalities has been decoupled from the JavaScript core using shared interfaces realized through the “dotUtil” class, the loader of its NET plugins. These classes are provided remotely through serialization, this decoupling provides a malleable modular implementation enabling the implant operators to a versatile code management.

Another relevant aspect of *JsOutProxy* is the capability to deal with SymantecVIP technology. This led us to think that, this implant, has been designed to hit High-Value targets. Also, this new threat appears emerging during these days, it has never been publicly seen before this December and it is probably still under development.

Cybaze-Yoroi Zlab will continue to actively hunt for it.

Indicator of Compromise

Hashes:

- 6bf0d9a7ca91f27a708c793832b0c7b6e3bc4c3b511e8b30e3d1ca2e3e2b90a7
- af10e6d1e3a3b4ed1d5524da25b782a4deddbd14d04e259f13dd1502d43b3045

C2:

91.189.180.199:9989

Yara Rules

```
rule JsOutProx_Dec_2019{
        meta:
        description = "Yara Rule for JsOutProx"
        author = "Cybaze Zlab_Yoroi"
        last_updated = "2019-12-19"
        tlp = "white"
        category = "informational"

        strings:
         $re1 = /,'([0-9a-zA-Z])+=',/
         $re2 = /[a-zA-Z]{1}_[a-zA-Z]{2}\[0x/
         $a1 = "WScript"
         $a2 = "0x"
    condition:
         $re1 and $a1 and #a2>20000 and #re2>1000
}
```

This blog post was authored by Antonio Farina, Luca Mella and Antonio Pirozzi of Cybaze-Yoroi Z-LAB