Deobfuscating Ostap: TrickBot's 34,000 Line JavaScript Downloader

by Alex Holland on September 3, 2019

Introduction

For a malicious actor to compromise a system, they need to avoid being detected at the point of entry into the target's network. Commonly, phishing emails delivering malicious attachments (T1193) serve as the initial access vector.[1]

Adversaries also need a way to execute code on target computers without tipping off automated tools and the monitoring efforts of security teams. One of the most common code execution techniques is to use interpreted scripting languages (T1064) that can run on an operating system without additional dependencies.[2] On Windows, popular interpreted languages that are abused by attackers include PowerShell, VBScript, JScript, VBA (Visual Basic for Applications), and commands interpreted by Command shell (cmd.exe).

Network attackers and defenders are in a constant state of competition to out-do the other to gain an advantage that could determine the outcome of an intrusion attempt. Against this background, we regularly see malicious actors change their tooling to increase the chances of a successful intrusion, particularly the downloaders used to initially compromise systems.

In early August 2019, we noticed that high-volume malicious spam campaigns delivering TrickBot started using Ostap, a commodity JavaScript (or more specifically, JScript) downloader. Previously, TrickBot campaigns relied on downloaders that used obfuscated Command shell and later PowerShell commands that were triggered by VBA AutoOpen macros to download their payloads.

In this post, I explain how to deobfuscate Ostap and describe a Python script I wrote (deobfuscate_ostap.py) that automates the deobfuscation of this JScript malware. The tool is available to download on GitHub.[3]

TrickBot, also known as The Trick, is a modular banking Trojan and dropper thought to be operated by at least three threat actors, tracked in the security community as TA505, Grim Spider and Wizard Spider.[4][5][6][7] While JavaScript-based downloaders aren't new, TrickBot's latest downloader is notable for its size, virtual machine detection and anti-analysis measures. For example, the Ostap samples analysed in this post generated incomplete traces in two different public sandboxes and neither downloaded their respective TrickBot payloads.[8][9] Moreover, a sample that was uploaded to VirusTotal had a low detection rate of 6/55 (11%) when it was first uploaded, suggesting that Ostap is effective at evading most anti-virus engines.

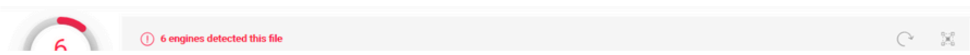

Figure 1 – VirusTotal detection summary for one of the Ostap samples.

Ostap, TrickBot's JScript Downloader

Downloaders are a type of malware designed to retrieve and run secondary payloads from one or more remote servers. Their simple function means that downloaders are rarely more than several hundred lines of code, even when obfuscated. Ostap counters this trend in that it is very large, containing nearly 35,000 lines of obfuscated code once beautified. Historical TrickBot campaigns suggest that their operators prefer code obfuscation that is lengthier than most other e-crime actors to bypass detection, as seen, for example in campaigns in August 2018.[10]

Figure 2 – Line, word and byte count of a sample of Ostap used to deliver TrickBot after being beautified. The downloader is 34,757 lines long.

Macro Analysis

The downloader is delivered as a Microsoft Word 2007 macro-enabled document (.DOCM) that contains the two components of the downloader: a VBA macro and the JScript (figure 3). The emails and samples analysed were themed as purchase orders, suggesting that the campaigns were likely intended to target businesses rather than individuals.

Figure 3 – Lure document of the downloader.

The JScript component of the downloader is stored in the body of the document as white text, resulting in a high word and page count.

Figure 4 – JScript in lure document.

The VBA macro is saved in a project called "Sorry". When the document is opened, it first copies the JScript to files named 2angola.dot and 2angola.dotu in the user's default Word template directory (%AppData%\Microsoft\Templates). The procedure is triggered by a Document.Open event.[11]

Figure 5 – Annotated VBA code that runs when the document is opened.

The rest of the macro only runs if the document is closed, which is achieved by monitoring for a Document.Close event (figure 6).[12] This is an anti-sandbox measure used to defeat behavioural analysis by sandboxes that don't imitate user activity such as closing documents.

Figure 6 – Annotated VBA code that runs when the document is closed.

If the document is closed, the macro renames 2angola.dot to 2angola.Jse and then runs it:

1. The macro calls the Create method from the Win32_Process WMI class to run a new Explorer.exe process with 2angola.Jse as its command line argument (figure 7).[13]
2. When a new Explorer.exe process is created where one is already running, the new process is created with the */factory,{75DFF2B7-6936-4C06-A8BB-676A7B00B24B} -Embedding* command-line arguments (figure 8). The CLSID corresponds to the ProgID called "CLSID_SeparateMultipleProcessExplorerHost".[14]
3. Explorer runs 2angola.Jse using Windows Script Host (WScript.exe), the default file handler for JScript Encoded Files (.JSE), as shown in figure 9. The file extension of 2angola.dot is renamed to *.Jse* ensure that the JScript is opened using WScript.exe. Relying on default file associations means that the macro can evade detection by indirectly referencing WScript, a program commonly used for malicious purposes in the context of macros.

Figure 7 – Sysmon event showing an Explorer.exe process running the JScript file after being launched by WMI Provider Host (WmiPrvSE.exe).

Figure 8 – Sysmon event showing the new Explorer.exe process being created with the arguments */factory,CLSID {75DFF2B7-6936-4C06-A8BB-676A7B00B24B} -Embedding*.

Figure 9 – Sysmon event showing WScript.exe running the JScript file.

Anti-Analysis Measures

Interestingly, the Ostap includes a fake Windows Script Host runtime error that occurs shortly after the script is run. It's likely that the fake error was included to discourage manual examination of the downloader.

Figure 10 – Fake error message displayed early during the runtime of the downloader.

Figure 11 – Variable storing the fake error message in TrickBot's downloader.

Some samples of the downloader contain the characters **/ at the beginning on the JSE file. This is another anti-analysis measure that is used to trip up automated JavaScript analysis tools which may interpret the rest of the script as being part of a comment block, rather than executable code.

Once deobfuscated, several other anti-analysis measures are revealed. For example, Ostap queries WMI to check if it is running in a virtual machine by looking for a blacklist of running processes:

- AgentSimulator.exe
- anti-virus.EXE
- BehaviorDumper

- BennyDB.exe
- ctfmon.exe
- fakepos_bin
- FrzState2k
- gemu-ga.exe (Possible misspelling of Qemu hypervisor's guest agent, qemu-ga.exe)
- ImmunityDebugger.exe
- KMS Server Service.exe
- ProcessHacker
- procexp
- Proxifier.exe
- python
- tcpdump
- VBoxService
- VBoxTray.exe
- VmRemoteGuest
- vmtoolsd
- VMware2B.exe
- VzService.exe
- winace
- Wireshark

Many sandboxes run these processes in their guest images, such as Cuckoo Sandbox and its derivatives which use a Python agent. The script also checks for a blacklist of host and user names.

- Emily
- HANSPETER-PC
- HAPUBWS
- Hong Lee
- IT-ADMIN
- JOHN-PC
- Johnson
- Miller
- MUELLER-PC
- Peter Wilson
- SystemIT | admin
- Timmy
- WIN7-TRAPS

Beautifying the JScript

The JScript that is written to disk is one line, making it difficult to analyse manually. To make it more readable, you can reformat and add indentations to the code using Einar Lielmanis's JS Beautifier tool, which also works for JScript because they share a similar syntax.[15]

js-beautify 2angola.Jse > 2angola.Jse.beautified

Identifying Code Structure, Key Variables and Functions

Now that the code is readable, we can begin analysing the script's structure, variables and functions. Our aim here is to identify the functions responsible for deobfuscating the downloader.

The script includes many junk variables that aren't used anywhere else in the script. We can simply remove these variables. It is often possible to distinguish the variables that have been automatically generated by an obfuscator from meaningful ones because their naming convention will differ.

For example, in figure 12 you can see some of the variable assignments in the script. All of them are junk code, except the variable called *gunsder*, which looks interesting because it contains the string "from". It's also referenced 2,515 times, which is promising.

Figure 12 – Some of the variables in the script.

In figure 13, you can see at line 15 a function called *xxqneol*. The variable that we identified as interesting, *gunsder*, is concatenated with other strings. After concatenation, you can see that the returned string is a reference to the fromCharCode() method which converts a Unicode character code into a character.[16] This function is supplied a parameter called *etsfhis*. Before calling fromCharCode, the function checks that the second parameter, *vqjpvi*, is the character *h*. This function is also referenced 7,540 times, so it's likely that this function is used in the deobfuscation of the script.

Now that we understand what the function does, we can give it, its variables and parameters meaningful names (figure 14).

Figure 13 – Function *xxqneol* before deobfuscation.

Figure 14 – Renamed *xxqneol* function.

Analysis of Character Code Calculation Functions

Next, we can look at the functions where fromCharCode is referenced to understand how it is used. After cleaning up the code in figure 15, you can see that the function uses arithmetic operators to calculate a Unicode character code from the values stored in an array called *pkkwrit4*. The Unicode character code and the character *h* are then supplied to the fromCharCode function, which returns a Unicode character. In this case, the character returned is *f*. Each character in the downloader has its own function to calculate its character code. This particular sample has 7,540 functions that are used to calculate all the characters codes.

Figure 15 – One of the many functions used to calculate Unicode character codes.

Figure 16 – Cleaned up function.

Writing a Python Script (deobfuscate_ostap.py) to Automate Deobfuscation

Since we don't want to have to manually calculate and decode 7,540 Unicode character codes, let's write a Python script to do this for us.

By looking for code similarities we can work out what actions we need the script to perform. In the functions that calculate the Unicode character codes, the final character code value is always calculated using the elements at index 0 and 1 of an array. Some arithmetic is performed on these elements before they are supplied to the fromCharCode function. So far we've seen addition and subtraction used in Ostap samples in the wild.

We can use Python's re module to write regular expressions that match the elements in each array at index 0 and 1 and store them in lists.[17] Next, we'll clean up the matches using the re.sub() function and then convert them into integers. We can then use Python's zip() function to perform the arithmetic on the values in the index 0 and 1 lists.[18] The script tries subtraction and addition operations to deobfuscate the downloader. Finally, the script converts the character codes into Unicode characters, removes line breaks and prints the result.

The script is available on GitHub to download and can be modified to support automated analysis pipelines.[3] To test the script, a YARA rule was written to detect Ostap and then run against 100 samples from August 2019. The extracted and deduplicated URLs are at the end of the report.

Analysis of the Deobfuscated Downloader

After running the script, we can examine the deobfuscated strings from the downloader, including the URL where the TrickBot payload is hosted:

- hxxps://185.180.199[.]102/angola/mabutu.php?min=14b

Figure 17 – Deobfuscated strings of Ostap sample using deobfuscate_ostap.py.

The strings are very similar to older Ostap samples from 2018 onwards, enabling us to make a high confidence assessment that the downloaders used to deliver TrickBot in August 2019 belong to this family of malware. Public reporting shows that this malware has been used in campaigns unrelated to TrickBot since 2016, delivering various financial malware families.[19][20] The variety of malware delivered by Ostap suggests that it is commodity malware that is popular among different threat actors, including now TrickBot's operators.

Ostap’s aggressive anti-analysis features and low detection rate compared to downloaders that use other interpreted scripting languages make it an attractive choice for malware operators seeking a downloader.

YARA Rule

```
rule win_ostap_jse {
      meta:
            author = "Alex Holland @cryptogramfan
(Bromium Labs)"
            date = "2019-08-29"
            sample_1 =
"F3E03E40F00EA10592F20D83E3C5E922A1CE6EA36FC326511C38F45
B9C9B6586"
            sample_2 =
"38E2B6F06C2375A955BEA0337F087625B4E6E49F6E4246B50ECB567
158B3717B"

      strings:
            $comment = { 2A 2A 2F 3B } // Matches on
**/;
            $array_0 = /\w{5,9}\[0\]=\d{1,3};/
            $array_1 = /\w{5,9}\[1\]=\d{1,3};/

      condition:
            (filesize > 1100KB and filesize < 400KB) and
(($comment at 0) and (#array_0 > 100) and (#array_1 >
100)) or ((#array_0 > 100) and (#array_1 > 100))
}
```

Hashes (SHA-256)

- F3E03E40F00EA10592F20D83E3C5E922A1CE6EA36FC326511C38F45B9C9B6586 – Last_order_specification_1217492.docm
- 38E2B6F06C2375A955BEA0337F087625B4E6E49F6E4246B50ECB567158B3717B – Heiress_Documents_id18598.docm

Extracted URLs

- hxxps://185.130.104[.]149/odr/updateme.php?oxx=p
- hxxps://185.130.104[.]149/odr/updateme.php?oxx=up
- hxxps://185.130.104[.]149/odr/updateme.php?oxx=z
- hxxps://185.130.104[.]236/deerhunter/inputok.php?min=29h

- hxxps://185.130.104[.]236/deerhunter/inputok.php?min=up3
- hxxps://185.130.104[.]236/deerhunter2/inputok.php?min=6h
- hxxps://185.130.104[.]236/deerhunter2/inputok.php?min=8h
- hxxps://185.130.104[.]236/deerhunter2/inputok.php?min=9a
- hxxps://185.130.104[.]236/deerhunter2/inputok.php?min=9h
- hxxps://185.130.104[.]236/targ/inputok.php?min=13s
- hxxps://185.130.107[.]236/deerhunter3/inputok.php?min=12a
- hxxps://185.159.82[.]15/hollyhole/c644.php?min=up
- hxxps://185.159.82[.]15/hollyhole/c644.php?min=17ha
- hxxps://185.159.82[.]15/hollyhole/c644.php?min=18h
- hxxps://185.159.82[.]15/hollyhole/c644.php?min=19a
- hxxps://185.159.82[.]15/hollyhole/c644.php?min=19h
- hxxps://185.159.82[.]15/hollyhole/c644.php?min=a
- hxxps://185.159.82[.]15/hollyhole/c644.php?min=m
- hxxps://185.159.82[.]15/hollyhole/c644.php?min=m2
- hxxps://185.159.82[.]15/hollyhole/c644.php?min=t2
- hxxps://185.159.82[.]15/hollyhole/c644.php?min=tu
- hxxps://185.159.82[.]15/hollyhole/c644.php?min=w
- hxxps://185.159.82[.]15/hollyhole2/c644.php?min=19h
- hxxps://185.159.82[.]15/hollyhole2/c644.php?min=79
- hxxps://185.159.82[.]20/t-30/x644.php?min=m
- hxxps://185.159.82[.]20/t-34/x644.php?min=24
- hxxps://185.159.82[.]20/t-34/x644.php?min=f
- hxxps://185.159.82[.]20/t-34/x66744.php?min=u2
- hxxps://185.180.199[.]102/angola/mabutu.php?min=14b
- hxxps://189.130.104[.]236/deerhunter3/inputok.php?min=13h

References

[1] MITRE ATT&CK technique T1193 "Spearphishing Attachment", https://attack.mitre.org/techniques/T1193/

[2] MITRE ATT&CK technique T1064 "Scripting", https://attack.mitre.org/techniques/T1064/

[3] https://github.com/cryptogramfan/Malware-Analysis-Scripts/blob/master/deobfuscate_ostap.py

[4] "Security Primer: TrickBot", Multi-State Information Sharing and Analysis Center, March 2019, https://www.cisecurity.org/wp-content/uploads/2019/03/MS-ISAC-Security-Primer-Trickbot-11March2019-mtw.pdf

[5] "Threat Group Cards: A Threat Actor Encyclopedia", ThaiCERT, p. 226, https://www.thaicert.or.th/downloads/files/A_Threat_Actor_Encyclopedia.pdf

[6] "Threat Group Cards: A Threat Actor Encyclopedia", ThaiCERT, p. 259

[7] "Threat Group Cards: A Threat Actor Encyclopedia", ThaiCERT, p. 272

[8] https://app.any.run/tasks/dc86fb23-b8ac-49db-8c22-a53b88236676/

[9] https://www.hybrid-analysis.com/sample/38e2b6f06c2375a955bea0337f087625b4e6e49f6e4246b50ecb567158b3717b?environmentId=120

[10] https://www.virustotal.com/gui/file/1512b7e34006ff7b69c76601fcf554668a3378d31c77b44507960d46e3a7c02c/details

[11] https://docs.microsoft.com/en-us/office/vba/api/word.document.open

[12] https://docs.microsoft.com/en-us/office/vba/api/word.document.close(even)

[13] https://docs.microsoft.com/en-us/windows/win32/cimwin32prov/create-method-in-class-win32-process

[14] https://en.wikipedia.org/wiki/ProgID

[15] https://github.com/beautify-web/js-beautify

[16] https://www.w3schools.com/jsref/jsref_fromcharcode.asp

[17] https://docs.python.org/3/library/re.html

[18] https://docs.python.org/3.3/library/functions.html#zip

[19] https://www.cert.pl/en/news/single/ostap-malware-analysis-backswap-dropper/

[20] https://www.carbonblack.com/2017/06/12/carbon-black-threat-research-dissects-emerging-mouseover-malware/

The post Deobfuscating Ostap: TrickBot's 34,000 Line JavaScript Downloader appeared first on Bromium.

Recent Articles By Author

- Buran Ransomware Targets German Organisations through Malicious Spam Campaign
- Changes to Emotet in September 2019
- Decrypting L0rdix RAT's C2

More from Alex Holland

*** This is a Security Bloggers Network syndicated blog from Bromium authored by Alex Holland. Read the original post at: https://www.bromium.com/deobfuscating-ostap-trickbots-javascript-downloader/

anti-analysis, deobfuscation, downloader, Javascript, JScript, ostap, Threat Research, threats, TrickBot

← Should you take the CCSP/SSCP before the CISSP?

CySA+: Maintaining employee skill level →

Join the Community

- Add your blog to Security Creators Network
- Write for Security Boulevard
- Bloggers Meetup and Awards
- Ask a Question
- Email: info@securityboulevard.com

Useful Links

- About
- Media Kit
- Sponsor Info
- Copyright
- TOS
- DMCA Compliance Statement
- Privacy Policy

Related Sites

- Techstrong Group
- Cloud Native Now
- DevOps.com
- Digital CxO
- Techstrong Research
- Techstrong TV
- Techstrong.tv Podcast
- DevOps Chat
- DevOps Dozen
- DevOps TV