Decoding a DanaBot Downloader

security-soup.net/decoding-a-danabot-downloader/

admin March 15, 2022

Summary

I came across a fairly interesting VBS-based DanaBot downloader the other day, and I figured it was worth doing a quick write-up on the obfuscation scheme and a few of the other TPPs I observed. The social engineering pretext used in this campaign was interesting as it leveraged an “unclaimed property” themed lure and required user interaction to deliver the first stage payload. A VBS file then fetches the DanaBot downloader. The VBS file contains an embedded URL that is not obfuscated, but the actual execution mechanism is encoded in a very long string.

In this blog, we will take a quick look at the social engineering pretext, then review the obfuscation scheme itself. Finally we will wrap up with coverage of three different methods to analyze and decode the VBS (each in order of complexity and potential to make you hate yourself). The first method we will review is the usage of a VBS debugger for quick a win. We will then review an alternative where we can debug the VBS file itself without any special tools, but just by editing a few lines of code in the file. Finally, we will conclude with (debatably) the ultimate exercise in futility, which is writing a Python decoder from scratch. This method doesn't do anything for us beyond instilling a sense of satisfaction and provide an opportunity to understand the underlying obfuscation scheme a little bit better, and learn a little Python to boot. Let's go!

Figure 1. This Analyst's thought process in meme format.

DanaBot Overview and Delivery

I don't typically do much analysis on DanaBot as I simply don't see it as often as the other eCrime variants that are delivered in massive volumes and in widespread campaigns. In this case, I found it interesting based on the social engineering scheme that required user interaction and the website landing page that had several elements that attempted to reassure victims and instill a sense of security. In addition, DanaBot caught my eye as it has been covered in the news and via OSINT reports from Zscaler that have linked DanaBot to recent DDoS campaigns against Ukrainian organizations, possibly in support of strategic objectives related to Russia's war efforts. The authors make a point to stress that "It is unclear whether this is an act of individual hacktivism, state-sponsored, or possibly a false flag operation." To be clear, there is no known link here observed between the campaign covered in this blog and DDoS events in Ukraine — they are simply both linked to DanaBot.

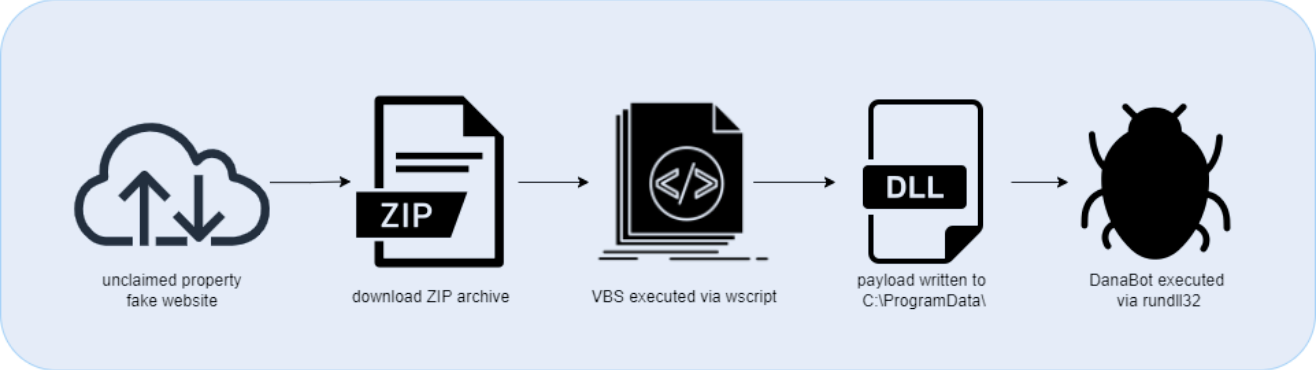


Figure 2. Execution chain for delivery of DanaBot

The DanaBot malware is a banker/infostealer originally discovered by Proofpoint researchers in 2018. It is operated by a financially motivated criminal group tracked as "SCULLY SPIDER" by CrowdStrike in a Malware as a Service (MaaS) model with multiple affiliate partners. Although DanaBot's core functionality has focused on stealing banking credentials, it has been known to be used in DDoS operations before.

DanaBot has been delivered via a variety methods in the past, including cracked games, sabotaged code packages, and phishing emails. In this case the VBS files was hosted on a fake "unclaimed property" website: www[.]moneyunclaimed[.]net.

- Filename: rpetitto-s980361ad.vbs
- SHA256: a4f1ea5dd434deee93bdf312f658a7a26f767c7683601fa8b23ef096392eef17

I have observed other domains in recent campaigns with a similar theme such as www[.]unclaimed2[.]com and/or www[.]unclaimedhq[.]com. All of these sites lure a potential victim to performing a "search" for unclaimed property. The threat actors use a questionable selection of thumbnail portraits, but I'm assuming most visitors won't notice or care. The site is slick enough to likely fool most potential victims.

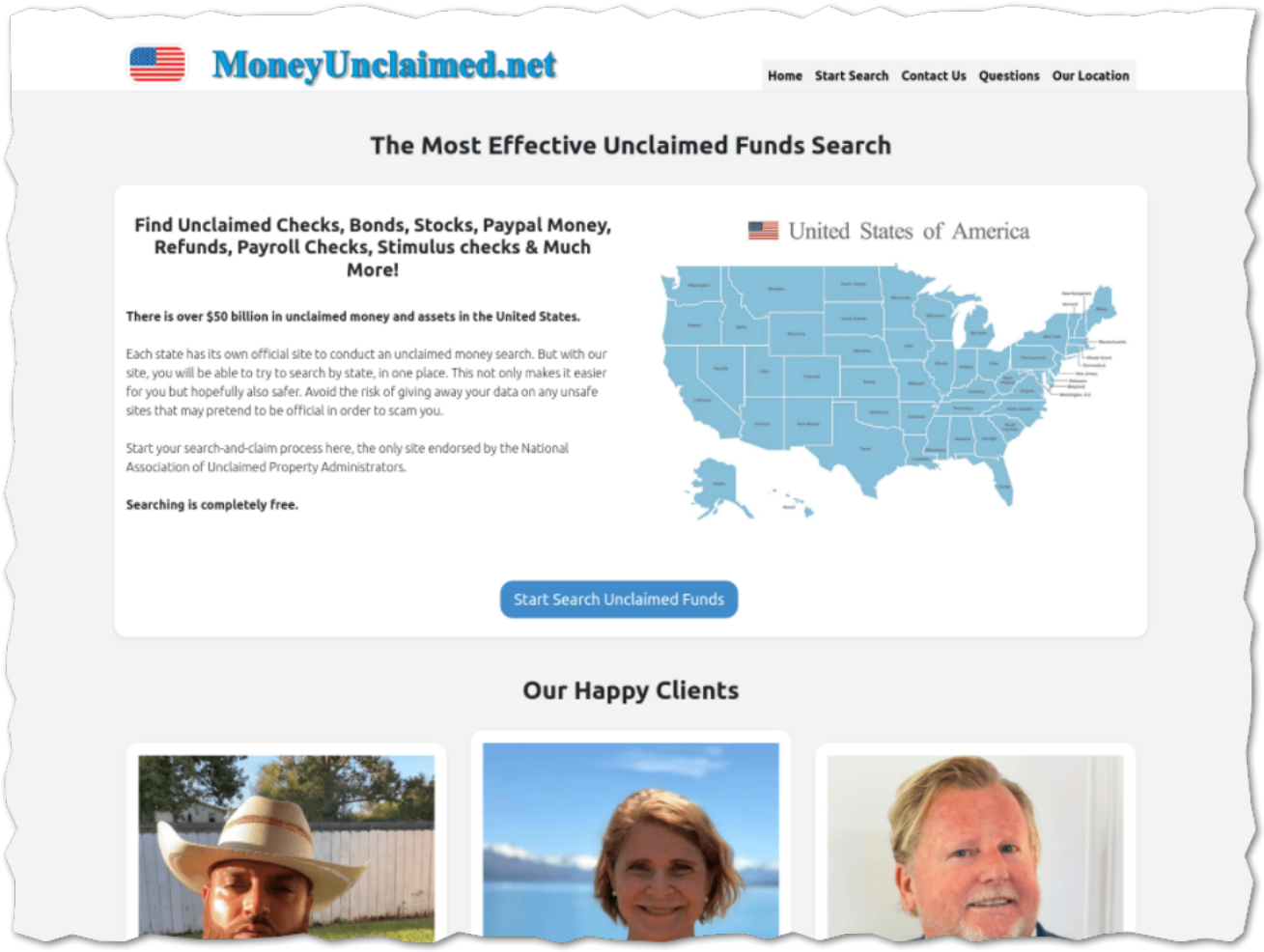


Figure 3. The landing page

And this is where things get interesting. The website is interactive, and in fact even requires user interaction to complete the initial malware delivery. If a user clicks on “Search”, they will be taken to a fake search page and prompted to input their First and Last Names and their state of residence. It even has a captcha in an attempt to appear more legitimate! There are other themes leveraged in the page such as making reference to the McAfee and GoDaddy brands in the sites’ footers to further cultivate a sense of trust and security with the victim.

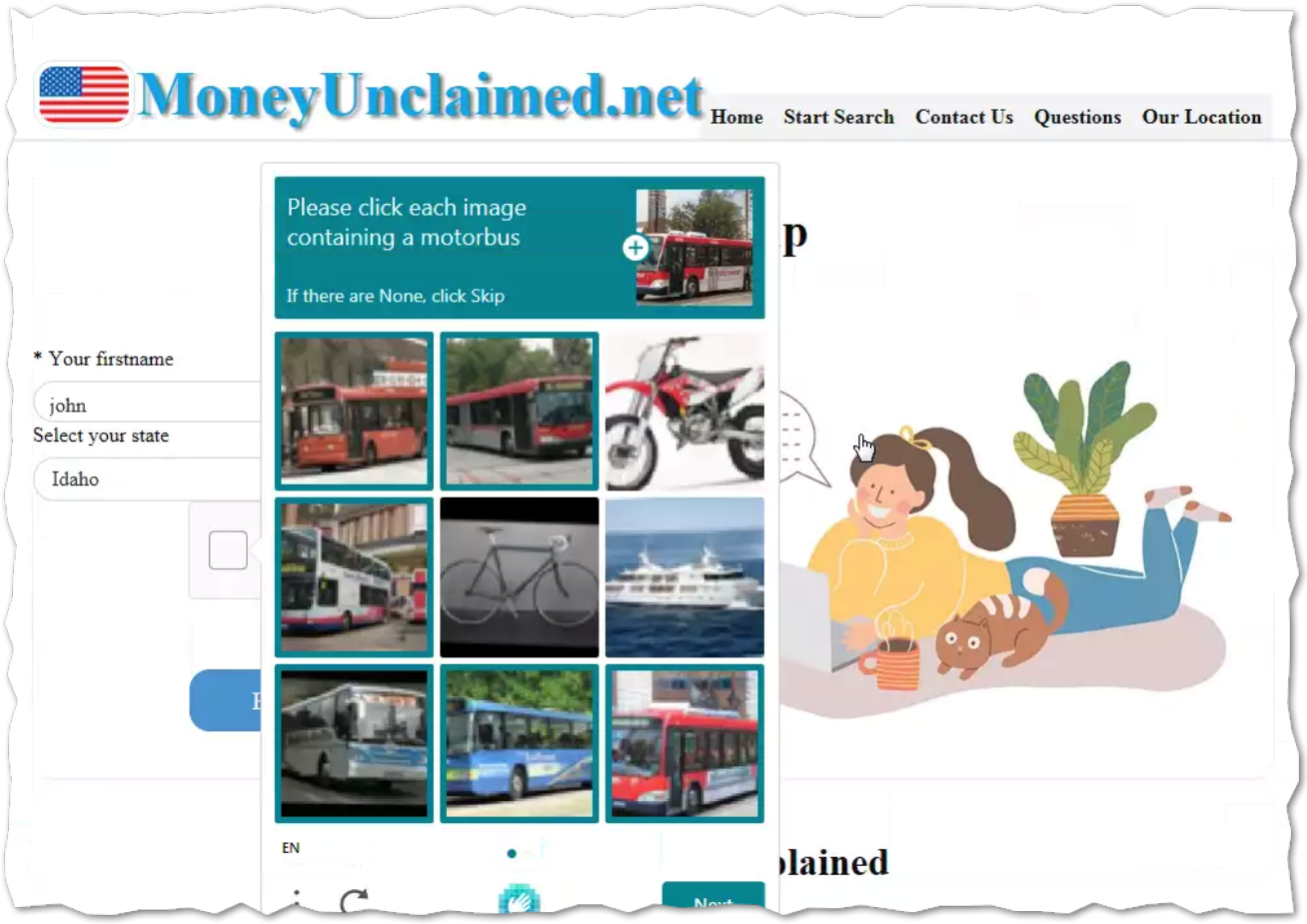


Figure 4. Running the property “report”

If the potential victim is “fortunate” enough to get results, they are prompted to download a “report” that allegedly contains their unclaimed property findings. However, in reality this ZIP archive that is downloaded contains a copy of winRAR and also the initial VBS downloader.

The Obfuscation Scheme

The URL is plainly available, but execution is not. If all you care about is IOCs, you can just stop here. But we don’t just care about IOCs. Because IOCs without contextual behavior aren’t that helpful. We can do better. Unfortunately, simply accessing the VBS code turned out to be just the first stage of the battle. This particular file has two loops that encode the script that is later executed as a function. In most analysis scenarios, speed is of the essence, so it is often better to rely on these tools to dump the code via dynamic analysis sandbox or debugger, but I often prefer to take a static analysis approach and manually decode the scripts in order to teas out the underlying subtleties of their operation. It is also just a fun exercise, akin to putting together a puzzle or deciphering a riddle. This type of approach may also provide some insight into adversary tactics, techniques, and procedures (TTPs) that would otherwise be lost (or at least glossed over) when employing dynamic analysis.

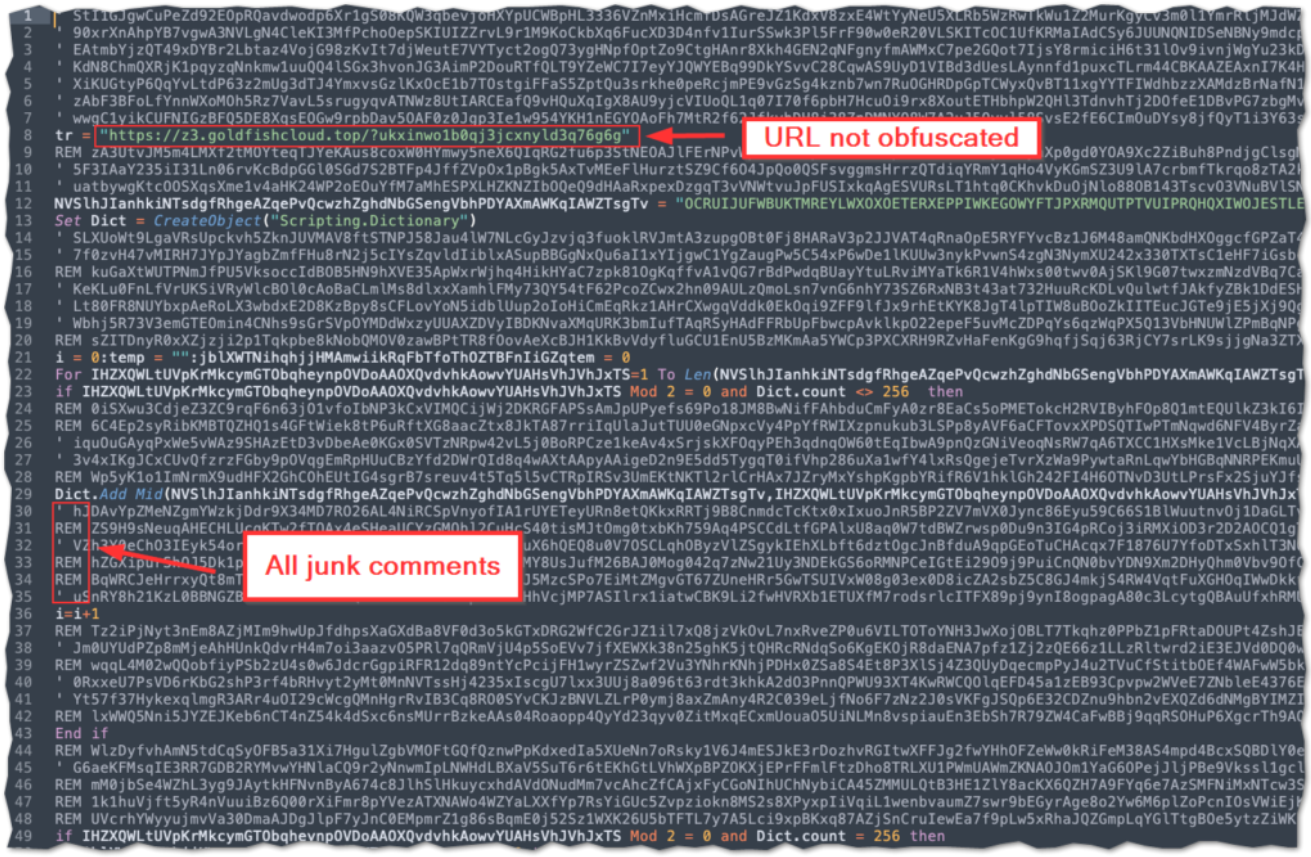


Figure 5 . The entire VBS content

The primary obfuscation technique utilized in this script is a string operations. The URL that hosts the final DanaBot payload is in clear text, but the function containing the execution is encoded in a very long string (over 2,000 characters). There are two loops that iterate through this string. All of the text in comments or quotes is meaningless and can be ignored for the purpose of analysis.

Figure 6. Prettified VBS

The first loops takes slices of two characters each and adds them to a dictionary object. The second loop iterates through the string and takes slices of two characters and then looks up those values in the dictionary it just created. It then accesses the key that corresponds to the lookup value. Since the dictionary is 256 keys long, each key correlates to a character in the extended ascii set. Finally, these keys are converted to their ascii values, stored in a final variable and then executed as s function.

Decoding the Downloader 3 Ways

So at this point we have a basic understanding of how the code works and we have the network IOC for the next stage payload. The final piece of information an analyst would typically investigate is the manner in which the next stage is executed. This is important for many reasons — but perhaps most importantly — the understanding of the specific tactics, techniques, and procedures (TTPs) can provide helpful contextual enrichment for developing detection content and identifying potential residual disk artifacts. In this next section, we will take a look at 3 options at how one could go about fully decoding the script to isolate these execution details.

Method 1: Using Vbsedit to debug

First, and perhaps most viable, is simply using a debugger to execute the code in a controlled manner and capturing the result as output. I have found the tool that is easiest and best for this purpose to be VbsEdit. This tool is not free, however, the lifetime license is very reasonably priced. There is also an evaluation license that allows usage of the tool, but

implements some guardrails with an additional delay and some obstacles in the form of message prompts in the evaluation mode. Either way, the tool works great and you can set break points and use the debugger console to step through the code. You can use this to populate variables and/or jump straight to the full output as shown below.

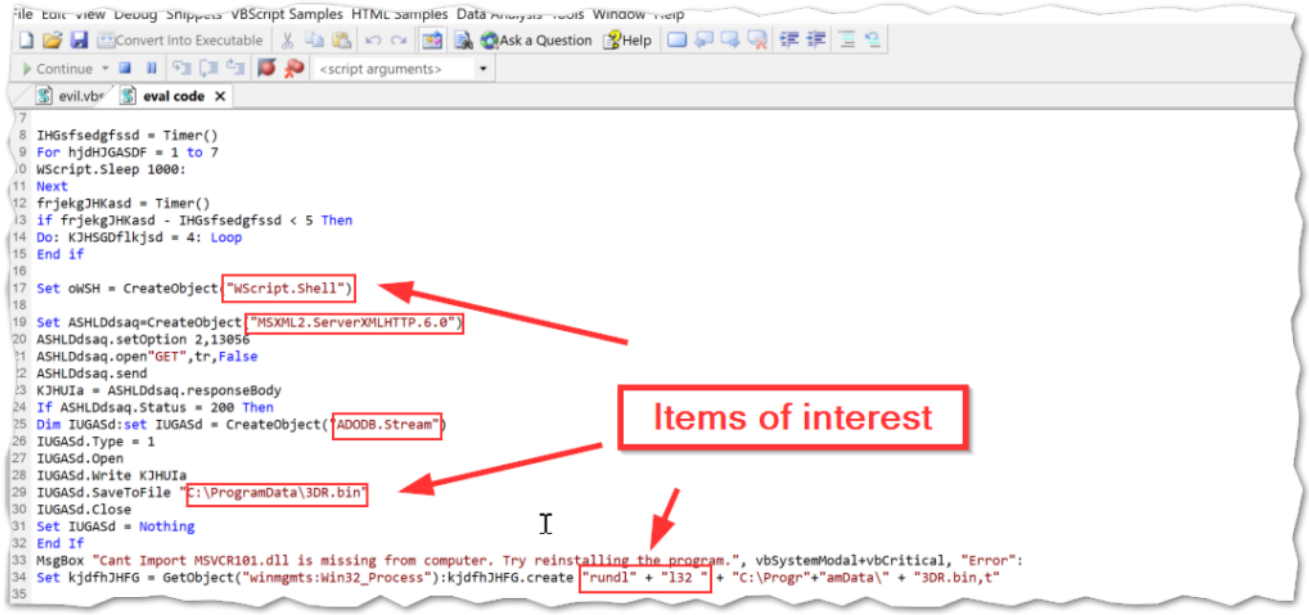


Figure 7. Debugger output

This is is a powerful tool whose capability to debug the code for quick output is hard to beat. Automatically de-obfuscating the code saves a ton of time, which can then be spent on other analysis or using the gathered information to pivot further on an investigation. In this sample, we uncovered multiple tool marks of interest: including the creation of a shell object and usage of wscript to kick off the script, the full path for the next stage payload written to disk, and the usage of rundll32 to execute that payload. This is a common technique that most EDR platforms should detect, but if you are following along at home, there are ATT&CK tagings below if you need to check for coverage.

Method 2: Modifying the VBS code to print to file

The second method we will look at is somewhat slower, but still provides quick output. The advantage to this method is that it does not require any additional tooling beyond a text editor and the ability to run VBscript. The idea here with this approach is that instead of executing the script within a shell object, we will simply re-direct the script's content as output to a file of our choosing. This requires the addition of just 4 extra lines of code. It should be noted that using this method is safest to also comment out the "Execute(temp)" function by prepending the line with "REM".

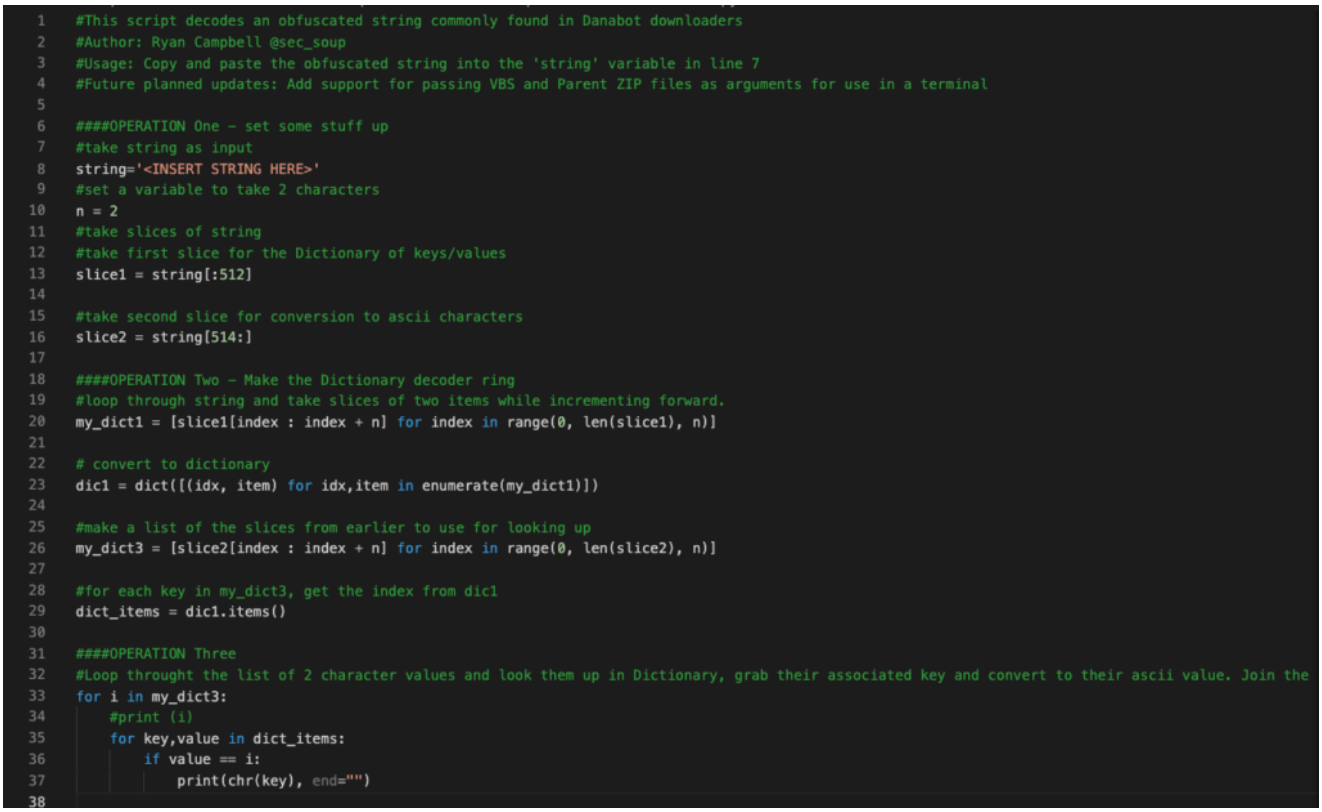
```
Set objFSO = CreateObject("Scripting.FileSystemObject")
outfile = "<INSERT PATH TO FILE>"
sET objFile = objFSO.CreateTextFile(outFile,True)
objFile.Write <INSERT THE FUNCTION'S ARGUMENT HERE> & vbCrLf
```

The variable names in the code don't matter. You can change them to whatever you wish. I just use what I copied off the guy that showed me how to do this. One additional note: the 4 line in the above can be moved around to wherever you like, sort of like using it as a break point, as that will output the script to file at that point in its execution. The fourth line also needs to be modified with the specific argument that is called by the function being debugged.

Figure 8. File modification to write to file

Method 3: Writing a Python Script (slow and pointless?)

Alright, so I've spent plenty of my time here writing a blog post that probably three people will ever read, and if you think that was a waste of time, well have I got a surprise for you… Because I wasn't content to just extract my IOCs and TTPs and write my little Yara rule (see below). No. I decided I wanted to commune with this code for a more deeper understanding of how it worked, and how it fit in with the greater mysteries of the Universe. Also, I hadn't written any code in a while, and I thought it could be a fun exercise to crack the ole knuckles, scrape off the rust, and slap together some Python. Ultimately, I didn't unlock any mysteries of the Universe, but I did write some dodgy Python and learned a few things — so I will count that as a win in my book.



```python
#This script decodes an obfuscated string commonly found in Danabot downloaders
#Author: Ryan Campbell @sec_soup
#Usage: Copy and paste the obfuscated string into the 'string' variable in line 7
#Future planned updates: Add support for passing VBS and Parent ZIP files as arguments for use in a terminal

####OPERATION One - set some stuff up
#take string as input
string='<INSERT STRING HERE>'
#set a variable to take 2 characters
n = 2
#take slices of string
#take first slice for the Dictionary of keys/values
slice1 = string[:512]

#take second slice for conversion to ascii characters
slice2 = string[514:]

####OPERATION Two - Make the Dictionary decoder ring
#loop through string and take slices of two items while incrementing forward.
my_dict1 = [slice1[index : index + n] for index in range(0, len(slice1), n)]

# convert to dictionary
dic1 = dict([(idx, item) for idx,item in enumerate(my_dict1)])

#make a list of the slices from earlier to use for looking up
my_dict3 = [slice2[index : index + n] for index in range(0, len(slice2), n)]

#for each key in my_dict3, get the index from dic1
dict_items = dic1.items()

####OPERATION Three
#Loop throught the list of 2 character values and look them up in Dictionary, grab their associated key and convert to their ascii value. Join the
for i in my_dict3:
    #print (i)
    for key,value in dict_items:
        if value == i:
            print(chr(key), end="")

```

Figure 9. If a Python falls in the forest with nobody around, does it still make a sound?

The script here is very simple as you can see below. We don't need any fancy imports or dependencies, just plain ole Python does the trick. Basically, the Python script works exactly like the VBScript. It first takes a string variable, then sets another variable to ensure we grab a pair of characters from the string. We then take two slices of the string, the first being for the creation of the dictionary, and the second used to build the command. After the first slice is converted into a dictionary, the second slice is simply indexed.

We then need to access the items in the dictionary and loop through the second slice's index. For each character pain in that loop, we look up the key for that value in the dictionary and convert the key's decimal value it into its ascii character. Then it is just a matter of joining the characters and printing them out to the console. Voilà!

Figure 10 . The fruits of our labors.

Conclusion

So that's it, my take on conducting analysis on a recent DanaBot downloader sample. This campaign caught my eye as the social engineering tactic was fairly convincing. The VBScript itself used an interesting string obfuscation method as well, although nothing novel. I've shared a few resources and tools that can hopefully enable analysts to improve the velocity their data gathering for triage and investigations. ATT&CK tagging is provided below, and I've included some IOCs from similar campaigns, and include a quick Yara signature that could help detect this particular downloader. I hope some of the analysis techniques will be helpful to those learning the ropes and/or looking for new methods to add to their current toolkit. Thanks for reading!

Detection

```
rule VBS_Downloader
//Downloaders were observed delivering Danabot in February/March of 2022
{
  meta:
    description = "Simple rule to detect VBS downloaders"
    created = "2022-03-13"
    author = "Ryan Campbell @sec_soup"

strings:
  $a = "CreateObject(\"Scripting.Dictionary\")"
  $b = "Dict.Add Mid("
  $c = "-1,2), i"
  $d = "Mod 2 = 0 and Dict.count <> 256 then"
  $e = "Mod 2 = 0 and Dict.count = 256 then"
  $f = "Chrw(Dict.Item(Mid("
  $g = "Execute(temp)"


condition:
  6 of ($a,$b,$c,$d,$e,$f,$g)
  filesize < 50KB
}
```

IOCs

www[.]moneyunclaimed[.]net
www[.]unclaimed2[.]com
www[.]unclaimedhq[.]com
z3[.]goldfishcloud[.]top
2186495019ee3d4838df3482eaa3c6b37f08d68b8ef0675342cb761ccf04c4fc

ATT&CK Tagging

- **Execution**
  - **User Execution (ATT&CK ID: T1204)**
  - **Command and Scripting Interpreter: Visual Basic (ATT&ACK ID: T1059.005)**
  - **Signed Binary Proxy Execution: Rundll32 (ATT&CK ID: T1218.011)**
- **Defense Evasion**
  - **Deobfuscate/Decode Files or Information (ATT&CK ID: T1140)**
  - **Masquerading (ATT&CK ID: T1036)**
- **Command and Control**
  Remote File Copy (ATT&CK ID: T1105)

References

[1] https://malpedia.caad.fkie.fraunhofer.de/details/win.danabot

[2] https://www.proofpoint.com/us/threat-insight/post/danabot-new-banking-trojan-surfaces-down-under-0

[3] https://www.zscaler.com/blogs/security-research/danabot-launches-ddos-attack-against-ukrainian-ministry-defense

[4] https://go.crowdstrike.com/rs/281-OBQ-266/images/Report2020CrowdStrikeGlobalThreatReport.pdf

[5] https://www.zscaler.com/blogs/security-research/spike-danabot-malware-activity#:~:text=DanaBot%20is%20a%20malware%2Das,affiliates%20to%20the%20threat%20landscape.

[6] https://www.virustotal.com/gui/file/2186495019ee3d4838df3482eaa3c6b37f08d68b8ef0675342cb761ccf04c4fc

[7] https://www.vbsedit.com/

[8] https://github.com/Sec-Soup

[9] https://github.com/Sec-Soup/Python-ToolBox/blob/master/vbs-decode-dana/vbs-decode-dana.py