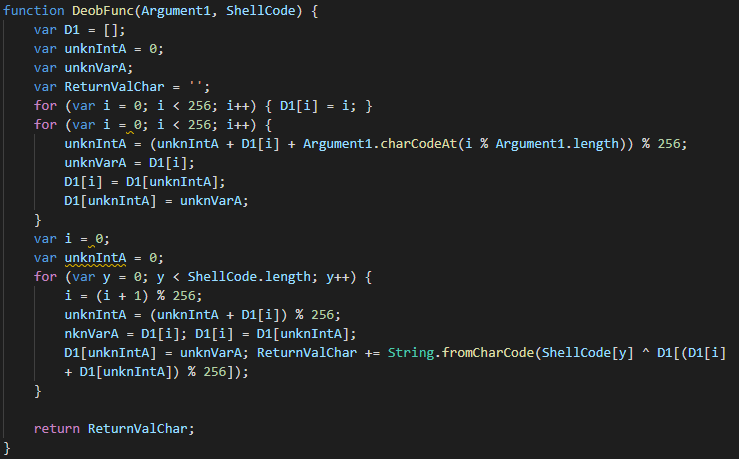
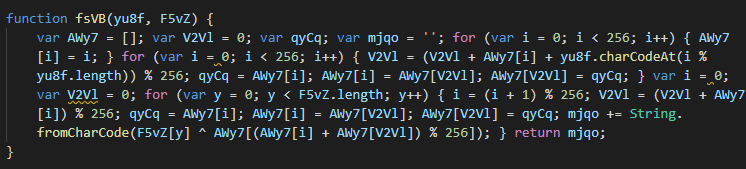
Lab - JavaScript Attachments

The goal of this lab is to deobfuscate the first stage of the provided JavaScript malware; you do **not** need to analyze any of the code past the first stage. Complete the following tasks:

This is a malicious sample; use all safety precautions when analyzing this sample. To deobfuscate this code you will likely want to modify and execute the script. Ensure you use a VM in an environment dedicated to malware analysis. In addition, take appropriate snapshots of your VM before running the script and disable your network adapter.

Perform analysis on this sample is from the lab archive. The password of the archive is **infected**.

**I have renamed the variables in a copy of the code file, to make it easier for me to follow along, and I have also reformatted it for readability resulting in additional lines. I assume this is acceptable since the code functionality and control flow remains the same – though it may cause confusion / difficulty explaining. Hopefully this is acceptable:**

**versus**

1. Analyze the JavaScript file in a text editor. Describe what area of the script is responsible for deobfuscating the first stage. Explain why you feel this to be the case.

the main function is pretty simple, it calls three functions and then executing a shellcode block. The first call (to brd() or GetDataBlock()) pulls the shellcode from the function that stores it and assigns it to a variable.

The second function (called LXv5() or DataBlockSplicer()) takes that shellcode, performs some uperations on it which appear to walk through the data, reordering and cutting chunks out of it eventually returning a string derived from the shellcode it was provided as an argument

The third function walks (fsVB() or DeobFunc) through the shellcode data again performing several modulus operations involving an argument whose value ultimately originates from “Wscript.Arguments” which as far as I can tell is used as an interface for command-line-arguments.

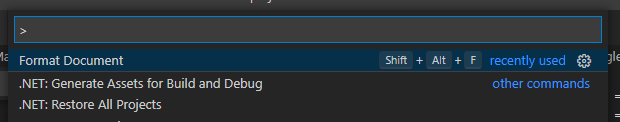
1. Something is missing from the malware to be able to deobfuscate it successfully. The value you need is EzZETcSXyKAdF\_e5I2i1

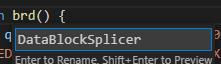
Where and how do you use it?

I presume this would be the value provided over the command line which is passed to the third function, and used in the modulus operations to finalize deobfuscation of the shellcode. You would make use of it by executing Wscript.exe on the JS file, and then tacking on the parameters – in this case “EzZETcSXyKAdF\_e5I2i1”

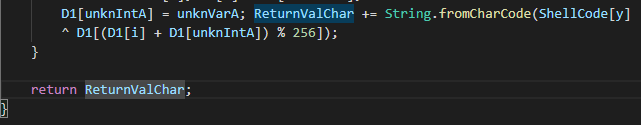
1. Deobfuscate the first stage. Provide **detailed** instructions on how you accomplished this along with screenshots of the deobfsucated code. If you do not explain how you did this in detail, it will be assumed you were unable to deobfuscate it. You may use manual and dynamic techniques; the latter would be useful.

The first thing I did was format the document – I have a number of extensions for VSCode and I cannot recall which parts of my UI ship by default, but I hit ctrl+shift+P and typed in “format document” in the box which pops up.

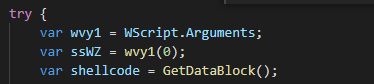
 This applied some basic formatting for readability. The next thing I did was ctrl+f search for semicolons, to break up any contiguous lines that were missed by the formatter, and broke them into individual lines.

 Next Up I immediately noticed that the last function in the file is nothing but a single had no arguments, and performed no operations… it just held and returns a massive block of unreadable data… Immediately I renamed the function and variable to Datablock and getDataBlock(). You can quickly rename a function without worrying about scope boundaries in Vscode by right clicking on the name, and selecting rename symbol. It will automatically search the file for any other calls to the same object and reflect the changes.

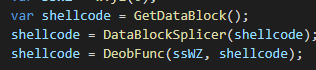
I then went through each function starting from the bottom. My strategy to avoid distracting elements of the code was to start by looking at the return value, and all interactions with it (bottom up first); and then to look at arguments and interactions with arguments (top down second)… First step was to rename those values (i.e. returnval, and argA,, argB, argC – updating the names as context reveals more information about them). When you select or hover a symbol in vscode, it identifies other locations where that symbol is used, which makes it easy to trace the interactions and important lines backwards from the return value.

I.e. 

After I had finished renaming the variables and functions and making sense of what I could from the bottom up, I went to look at the main function, and found that much of the functionality was clear from the changes I had made so far. There were 3 variables, one was set to the Wscript.Arguments object, the next was assigned a value from that object (presumably the first argument), and the third was drawing assigned a value from the function named getDataBlock().

 From context I knew that this variable held a large indecipherable block of data, probably shellcode. So I renamed it to shellcode.

Due to the simplicity of the main function I didn’t feel it was necessary to rename the two other variables.

I then renamed the remaining functions according to what information I had currently gleaned – the order in which they are called (not reflected in this screencap, because I have since renamed them again). At this point the shellcode parameters were also Identified so I went into the functions and renamed that argument to ‘shellcode.’

From this point on it was just a process of breaking down the code and trying to read through and understand it – analyzing I guess. I kept the same approach style, where I would try to trace the return value to Identify key points, and then try to Identify the roles of variables and functions from those lines… and back and back and back. And whenever I got stuck / confused I would look at it again from a top-down angle rather than bottom-up, and see if I can make sense of the function from what I had gleaned… Rinse and repeat.

1. Try to understand stage two of the malware, or beyond**!** What happens next? What can you figure out and discern? Go as far and deep as you can, and provide explanations and screenshots. Focus on discovering malicious functionality and IOC’s – show and explain how you did so.

I replaced the evaluate() call with Wscript.Echo() to print the deobfuscated payload.

**Deliverables**

Turn in your lab report as a Word document or PDF. All labs can be completed in under 10 pages, but I do allow students to go beyond that, if they have **quality** material to share. Generally, if you have less than 3 or 4 pages or minimal text and only all screenshots, it is likely inadequate. Labs are graded on quality, not length.