Defeating Macro Document Static Analysis with Pictures of My Cat

B **billdemirkapi.me**/defeating-macro-document-static-analysis-with-pictures-of-my-cat

Bill Demirkapi September 16, 2020

Over the past few weeks I've spent some time learning Visual Basic for Applications (VBA), specifically for creating malicious Word documents to act as an initial stager. When taking operational security into consideration and brainstorming ways of evading macro detection, I had the question, *how* does anti-virus detect a malicious macro?

The hypothesis I came up with was that anti-virus would parse out macro content from the word document and scan the macro code for a variety of malicious techniques, nothing crazy. A common pattern I've seen attackers counter this sort-of detection is through the use of macro obfuscation, which is effectively scrambling macro content in an attempt to evade the malicious patterns anti-virus looks for.

The questions I wanted answered were:

1. How does anti-virus even retrieve the macro content?
2. What differences are there for the retrieval of macro content between the implementation in Microsoft Word and anti-virus?

Discovery

According to Wikipedia,Open Office XML (OOX) "is a *zipped*, XML-based file format developed by Microsoft for representing spreadsheets, charts, presentations and word processing documents". This is the file format used for the common Microsoft Word extensions `docx` and `docm`. The fact that Microsoft Office documents were essentially a zip file of XML files certainly piqued my interest.

Since the OOX format is just a zip file, I found that parsing macro content from a Microsoft Word document was simpler than you might expect. All an anti-virus would need to do is:

1. Extract the Microsoft Office document as a ZIP and look for the file `word\vbaProject.bin`.
2. Parse the OLE binary and extract the macro content.

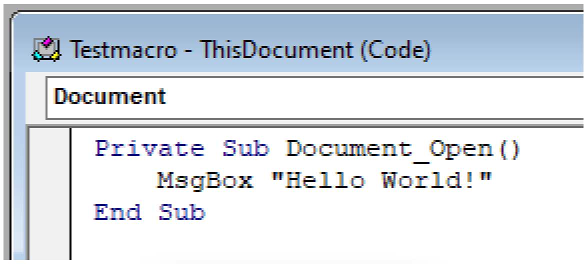
The differences I was interested in was how the methods would handle errors and corruption. For example, common implementations of ZIP extraction will often have error checking such as:

1. Does the local file header begin with the signature `0x04034b50`?

2. Is the minimum version bytes greater than what is supported?

What I was really after was finding ways to break the ZIP parser in anti-virus *without* breaking the ZIP parser used by Microsoft Office.

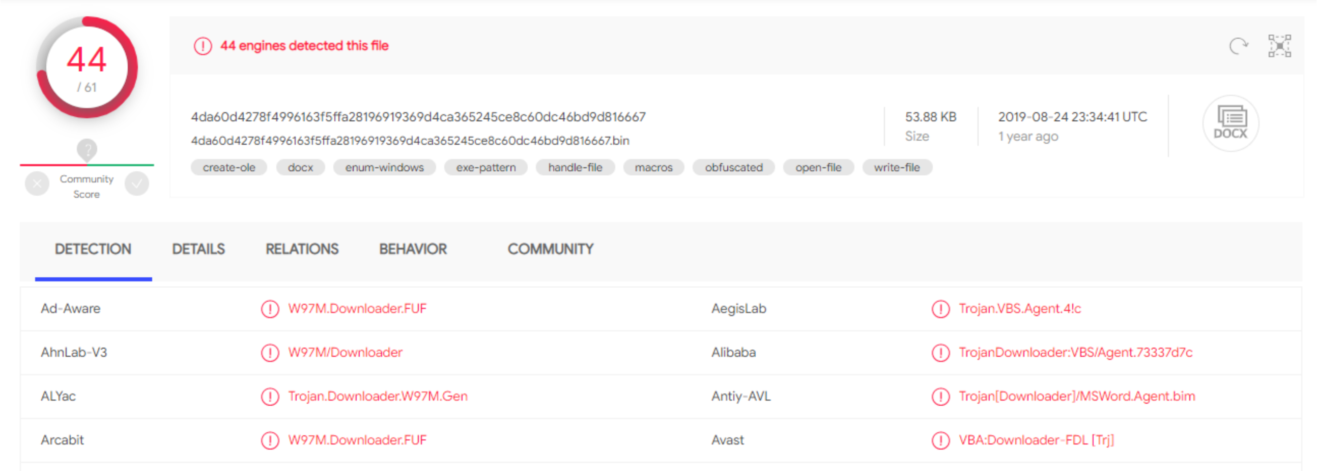
Before we get into corrupting anything, we need a base sample first. As an example, I simply wrote a basic macro "Hello World!" that would appear when the document was opened.



```
Private Sub Document_Open()
    MsgBox "Hello World!"
End Sub
```

Hello World!

For the purposes of testing detection of macros, I needed another sample document that was heavily detected by anti-virus. After a quick google search, I found a few samples shared by @malware_traffic here. The sample named `HSOTN2JI.docm` had the highest detection rate, coming in at 44/61 engines marking the document as malicious.



44 engines detected this file

4da60d4278f4996163f5ffa28196919369d4ca365245ce8c60dc46bd9d816667
4da60d4278f4996163f5ffa28196919369d4ca365245ce8c60dc46bd9d816667.bin

53.88 KB
Size

2019-08-24 23:34:41 UTC
1 year ago

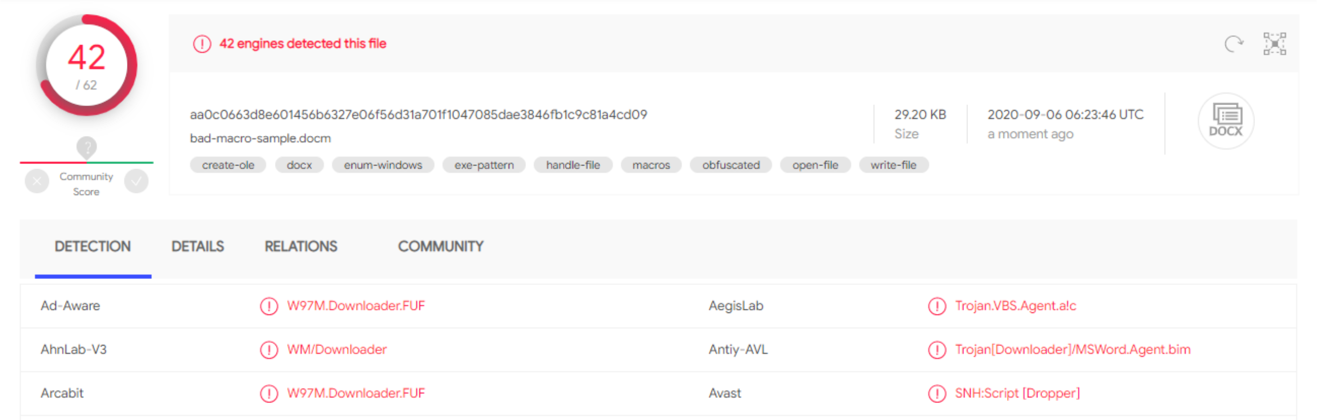
create-ole docx enum-windows exe-pattern handle-file macros obfuscated open-file write-file

Ad-Aware	W97M.Downloader.FUF	AegisLab	Trojan.VBS.Agent.4!c
AhnLab-V3	W97M/Downloader	Alibaba	TrojanDownloader:VBS/Agent.73337d7c
ALYac	Trojan.Downloader.W97M.Gen	Antiy-AVL	Trojan[Downloader]/MSWord.Agent.bim
Arcabit	W97M.Downloader.FUF	Avast	VBA:Downloader-FDL [Trj]

To ensure that detections were specifically based on the malicious macro inside the document's `vbaProject.bin` OLE file, I...

1. Opened both my "Hello World" and the `HSOTN2JI` macro documents as ZIP files.
2. Replaced the `vbaProject.bin` OLE file in my "Hello World" macro document with the `vbaProject.bin` from the malicious `HSOTN2JI` macro document.

Running the scan again resulted in the following detection rate:



42 engines detected this file

aa0c0663d8e601456b6327e06f56d31a701f1047085dae3846fb1c9c81a4cd09
bad-macro-sample.docm
create-ole docx enum-windows exe-pattern handle-file macros obfuscated open-file write-file

29.20 KB
Size

2020-09-06 06:23:46 UTC
a moment ago

Ad-Aware	W97M.Downloader.FUF	AegisLab	Trojan.VBS.Agent.a!c
AhnLab-V3	WM/Downloader	Antiy-AVL	Trojan[Downloader]/MSWord.Agent.bim
Arcabit	W97M.Downloader.FUF	Avast	SNH:Script [Dropper]

Fortunately, these anti-virus products were detecting the actual macro and not solely relying on conventional methods such as blacklisting the hash of the document. Now with a base malicious sample, we can begin tampering with the document.

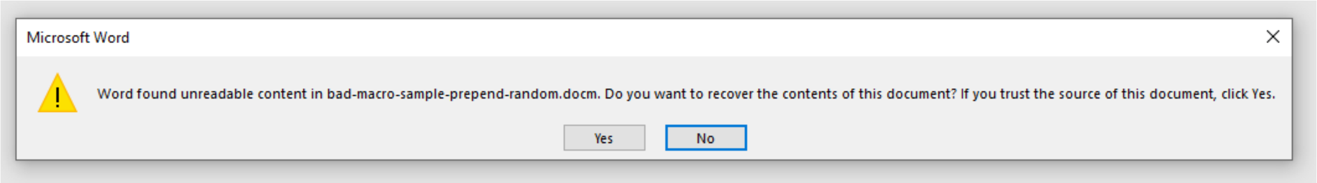
Exploitation

The methodology I used for the methods of corruption is:

1. Modify the original base sample file with the corruption method.
2. Verify that the document still opens in Microsoft Word.
3. Upload the new document to VirusTotal.
4. If good results, retry the method on my original "Hello World" macro document and verify that the macro still works.

Before continuing, it's important to note that the methods discussed in this blog post does come with drawbacks, specifically:

1. Whenever a victim opens a corrupted document, they will receive a prompt asking whether or not they'd like to recover the document:



Word found unreadable content in bad-macro-sample-prepend-random.docm. Do you want to recover the contents of this document? If you trust the source of this document, click Yes.

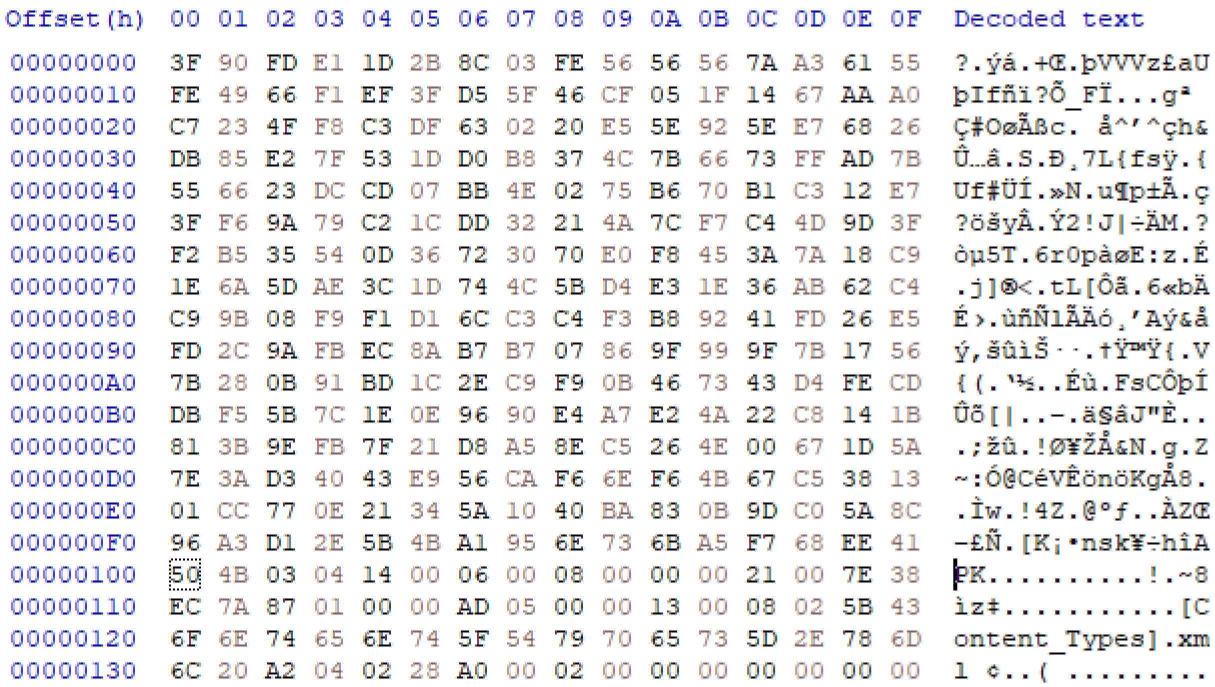
1. Before the macro is executed, the victim will be prompted to save the recovered document. Once the victim has saved the recovered document, the macro will execute.

Although adding any user interaction certainly increases the complexity of the attack, if a victim was going to enable macros anyway, they'd probably also be willing to recover the document.

General Corruption

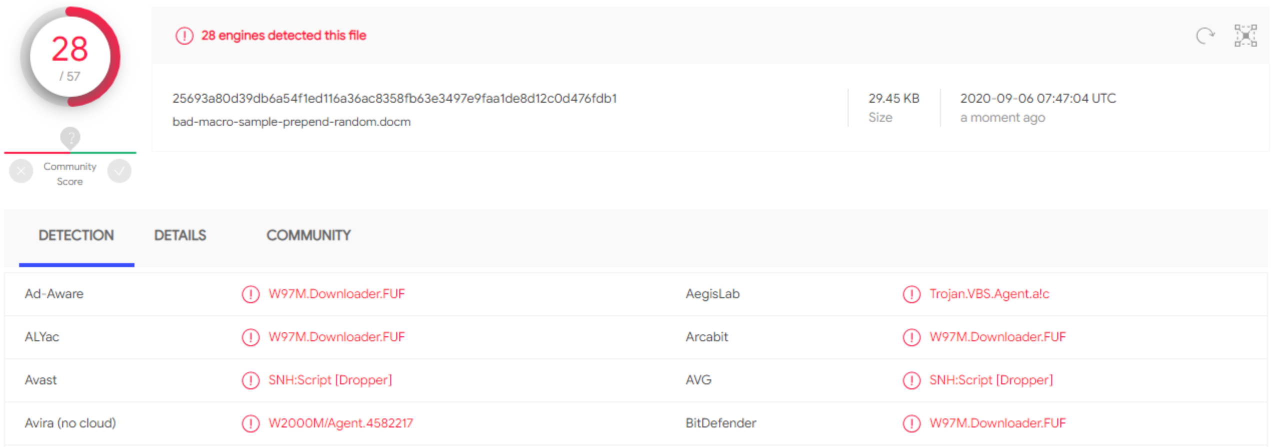
We'll first start with the effects of general corruption on a Microsoft Word document. What I mean by this is I'll be corrupting the file using methods that are non-specific to the ZIP file format.

First, let's observe the impact of adding random bytes to the beginning of the file.



```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text

00000000  3F 90 FD E1 1D 2B 8C 03 FE 56 56 56 7A A3 61 55  ?.ýá.+Œ.þVVVz£aU
00000010  FE 49 66 F1 EF 3F D5 5F 46 CF 05 1F 14 67 AA A0  þIfñï?Õ_FÏ...gª 
00000020  C7 23 4F F8 C3 DF 63 02 20 E5 5E 92 5E E7 68 26  Ç#OøÃßc. å^'^çh&
00000030  DB 85 E2 7F 53 1D D0 B8 37 4C 7B 66 73 FF AD 7B  Û…â.S.Ð¸7L{fsÿ.{
00000040  55 66 23 DC CD 07 BB 4E 02 75 B6 70 B1 C3 12 E7  Uf#ÜÍ.»N.u¶p±Ã.ç
00000050  3F F6 9A 79 C2 1C DD 32 21 4A 7C F7 C4 4D 9D 3F  ?öšyÂ.Ý2!J|÷ÄM.?
00000060  F2 B5 35 54 0D 36 72 30 70 E0 F8 45 3A 7A 18 C9  òµ5T.6r0pàøE:z.É
00000070  1E 6A 5D AE 3C 1D 74 4C 5B D4 E3 1E 36 AB 62 C4  .j]®<.tL[Ôã.6«bÄ
00000080  C9 9B 08 F9 F1 D1 6C C3 C4 F3 B8 92 41 FD 26 E5  É›.ùñÑlÃÄó¸'Aý&å
00000090  FD 2C 9A FB EC 8A B7 B7 07 86 9F 99 9F 7B 17 56  ý,šûìŠ··.†Ÿ™Ÿ{.V
000000A0  7B 28 0B 91 BD 1C 2E C9 F9 0B 46 73 43 D4 FE CD  {(.'½..Éù.FsCÔþÍ
000000B0  DB F5 5B 7C 1E 0E 96 90 E4 A7 E2 4A 22 C8 14 1B  Ûõ[|..–.ä§âJ"È..
000000C0  81 3B 9E FB 7F 21 D8 A5 8E C5 26 4E 00 67 1D 5A  .;žû.!Ø¥ŽÅ&N.g.Z
000000D0  7E 3A D3 40 43 E9 56 CA F6 6E F6 4B 67 C5 38 13  ~:Ó@CéVÊönöKgÅ8.
000000E0  01 CC 77 0E 21 34 5A 10 40 BA 83 0B 9D C0 5A 8C  .Ìw.!4Z.@ºƒ..ÀZŒ
000000F0  96 A3 D1 2E 5B 4B A1 95 6E 73 6B A5 F7 68 EE 41  –£Ñ.[K¡•nsk¥÷hîA
00000100  50 4B 03 04 14 00 06 00 08 00 00 00 21 00 7E 38  PK..........!.~8
00000110  EC 7A 87 01 00 00 AD 05 00 00 13 00 08 02 5B 43  ìz‡...........[C
00000120  6F 6E 74 65 6E 74 5F 54 79 70 65 73 5D 2E 78 6D  ontent_Types].xm
00000130  6C 20 A2 04 02 28 A0 00 02 00 00 00 00 00 00 00  l ¢..( .........
```



28
/ 57

28 engines detected this file

25693a80d39db6a54f1ed116a36ac8358fb63e3497e9faa1de8d12c0d476fdb1
bad-macro-sample-prepend-random.docm

29.45 KB
Size

2020-09-06 07:47:04 UTC
a moment ago

Ad-Aware	W97M.Downloader.FUF	AegisLab	Trojan.VBS.Agent.a!c
ALYac	W97M.Downloader.FUF	Arcabit	W97M.Downloader.FUF
Avast	SNH:Script [Dropper]	AVG	SNH:Script [Dropper]
Avira (no cloud)	W2000M/Agent.4582217	BitDefender	W97M.Downloader.FUF

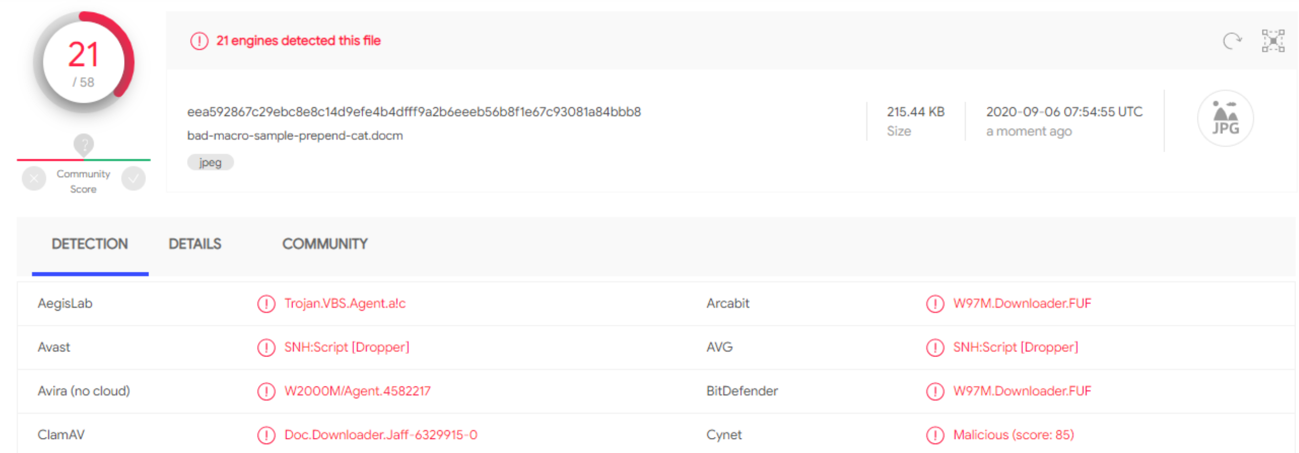
With a few bytes at the beginning of the document, we were able to decrease detection by about 33%. This made me confident that future attempts could reduce this even further.

Result: 33% decrease in detection

Prepending My Cat

This time, let's do the same thing except prepend a JPG file, in this case, a photo of my cat!

You might think that prepending some random data should result in the same detection rate as an image, but some anti-virus marked the file as clean as soon as they saw an image.



21 engines detected this file

eea592867c29ebc8e8c14d9efe4b4dfff9a2b6eeeb56b8f1e67c93081a84bbb8
bad-macro-sample-prepend-cat.docm
jpeg

215.44 KB
Size

2020-09-06 07:54:55 UTC
a moment ago

AegisLab	Trojan.VBS.Agent.a!c	Arcabit	W97M.Downloader.FUF
Avast	SNH:Script [Dropper]	AVG	SNH:Script [Dropper]
Avira (no cloud)	W2000M/Agent.4582217	BitDefender	W97M.Downloader.FUF
ClamAV	Doc.Downloader.Jaff-6329915-0	Cynet	Malicious (score: 85)

To aid in future research, the anti-virus engines that marked the random data document as malicious but did *not* mark the cat document as malicious were:

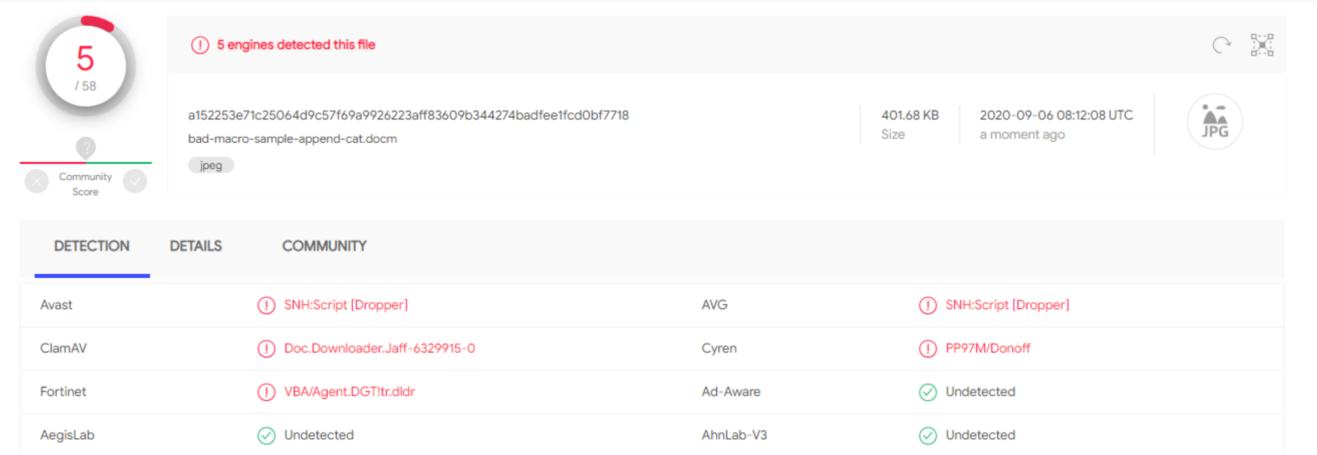
```
Ad-Aware
ALYac
DrWeb
eScan
McAfee
Microsoft
Panda
Qihoo-360
Sophos ML
Tencent
VBA32
```

The reason this list is larger than the actual difference in detection is because some engines strangely detected this cat document, but did *not* detect the random data document.

Result: 50% decrease in detection

Prepending + Appending My Cat

Purely appending data to the end of a macro document barely impacts the detection rate, instead we'll be combining appending data with other methods starting with my cat.



5 engines detected this file

a152253e71c25064d9c57f69a9926223aff83609b344274badfee1fcd0bf7718
bad-macro-sample-append-cat.docm
jpeg

401.68 KB
Size

2020-09-06 08:12:08 UTC
a moment ago

Avast	SNH:Script [Dropper]	AVG	SNH:Script [Dropper]
ClamAV	Doc.Downloader.Jaff-6329915-0	Cyren	PP97M/Donoff
Fortinet	VBA/Agent.DGT!tr.dldr	Ad-Aware	Undetected
AegisLab	Undetected	AhnLab-V3	Undetected

What was shocking about all of this was even when the ZIP file was in the middle of two images, Microsoft's parser was able to reliably recover the document and macro! With only extremely basic modification to the document, we were able to essentially prevent most detection of the macro.

Result: 88% decrease in detection

Zip Corruption

Microsoft's fantastic document recovery is not just exclusive to general methods of file corruption. Let's take a look at how it handles corruption specific to the ZIP file format.

Corrupting the ZIP Local File Header

The only file we care about preventing access to is the `vbaProject.bin` file, which contains the malicious macro. Without corrupting the data, could we corrupt the file header for the `vbaProject.bin` file and still have Microsoft Word recognize the macro document?

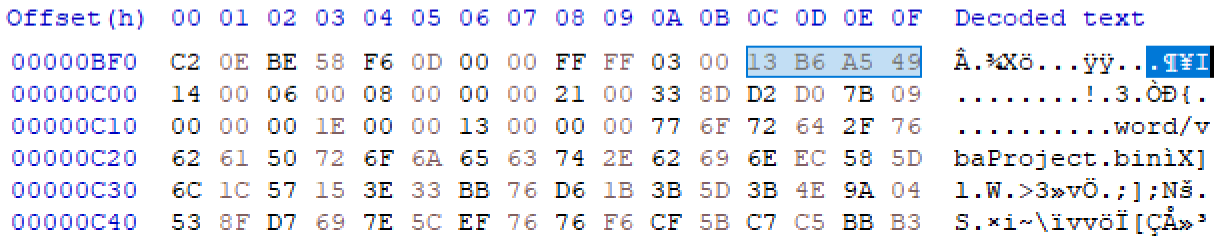
Let's take a look at the structure of a local file header from Wikipedia:

Local file header

Offset	Bytes	Description[30]
0	4	Local file header signature = 0x04034b50 (read as a little-endian number)
4	2	Version needed to extract (minimum)
6	2	General purpose bit flag
8	2	Compression method
10	2	File last modification time
12	2	File last modification date
14	4	CRC-32 of uncompressed data
18	4	Compressed size
22	4	Uncompressed size
26	2	File name length (*n*)
28	2	Extra field length (*m*)
30	*n*	File name
30+*n*	*m*	Extra field

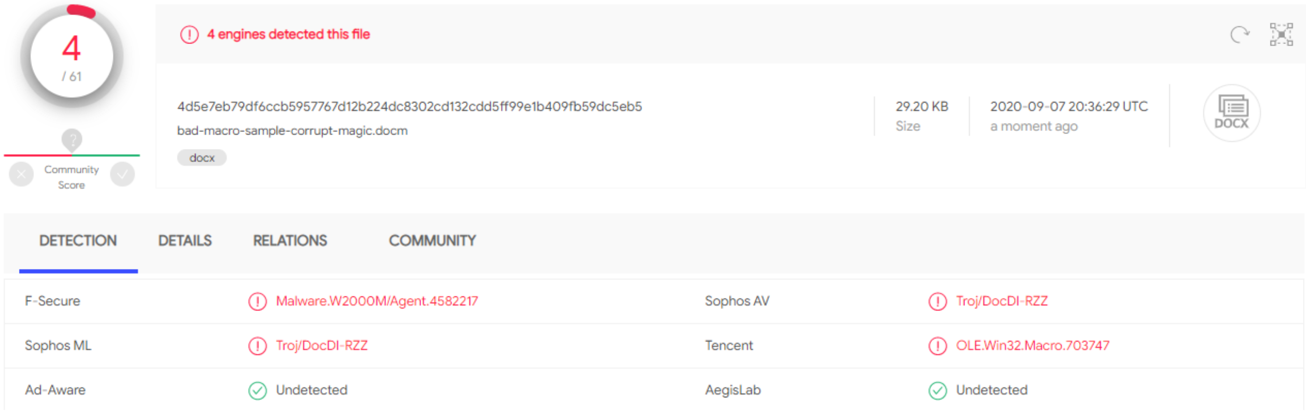
I decided that the local file header signature would be the least likely to break file parsing, hoping that Microsoft Word didn't care whether or not the file header had the correct magic value. If Microsoft Word didn't care about the magic, corrupting it had a high chance of interfering with ZIP parsers that have integrity checks such as verifying the value of the magic.

After corrupting only the file header signature of the `vbaProject.bin` file entry, we get the following result:



```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text

00000BF0  C2 0E BE 58 F6 0D 00 00 FF FF 03 00 13 B6 A5 49  Â.¾Xö...ÿÿ....¶¥I
00000C00  14 00 06 00 08 00 00 00 21 00 33 8D D2 D0 7B 09  ........!.3.ÒÐ{.
00000C10  00 00 00 1E 00 00 13 00 00 00 77 6F 72 64 2F 76  ..........word/v
00000C20  62 61 50 72 6F 6A 65 63 74 2E 62 69 6E EC 58 5D  baProject.binìX]
00000C30  6C 1C 57 15 3E 33 BB 76 D6 1B 3B 5D 3B 4E 9A 04  l.W.>3»vÖ.;];Nš.
00000C40  53 8F D7 69 7E 5C EF 76 76 F6 CF 5B C7 C5 BB B3  S.×i~\ïvvöÏ[ÇÅ»³
```


4 / 61

Community Score

4 engines detected this file

4d5e7eb79df6ccb5957767d12b224dc8302cd132cdd5ff99e1b409fb59dc5eb5
bad-macro-sample-corrupt-magic.docm
docx

29.20 KB
Size

2020-09-07 20:36:29 UTC
a moment ago

DOCX

F-Secure	Malware.W2000M/Agent.4582217	Sophos AV	Troj/DocDl-RZZ
Sophos ML	Troj/DocDl-RZZ	Tencent	OLE.Win32.Macro.703747
Ad-Aware	Undetected	AegisLab	Undetected

With a ZIP specific corruption method, we almost completely eliminated detection.

Result: 90% decrease in detection

Combining Methods

With all of these methods, we've been able to reduce static detection of malicious macro documents quite a bit, but it's still not 100%. Could these methods be combined to achieve even lower rates of detection? Fortunately, yes!

Method	Detection Rate Decrease
Prepending Random Bytes	33%
Prepending an Image	50%
Prepending and Appending an Image	88%
Corrupting ZIP File Header	90%
Prepending/Appending Image *and* Corrupting ZIP File Header	100%

Interested in trying out the last corruption method that reduced detection by 100%? I made a script to do just that! To use it, simply execute the script providing document filename as the first argument and a picture filename for the second parameter. The script will spit out the patched document to your current directory.

As stated before, even though these methods can bring down the detection of a macro document to 0%, it comes with high costs to attack complexity. A victim will not only need to click to recover the document, but will also need to save the recovered document before the malicious macro executes. Whether or not that added complexity is worth it for your team will widely depend on the environment you're against.

Regardless, one must heavily applaud the team working on Microsoft Office, especially those who designed the fantastic document recovery functionality. Even when compared to tools that are specifically designed to recover ZIP files, the recovery capability in Microsoft Office exceeds all expectations.