VBA Stomping — Advanced Maldoc Techniques

By Carrie Roberts

Published: 2020-04-11 · Archived: 2026-04-05 17:15:57 UTC

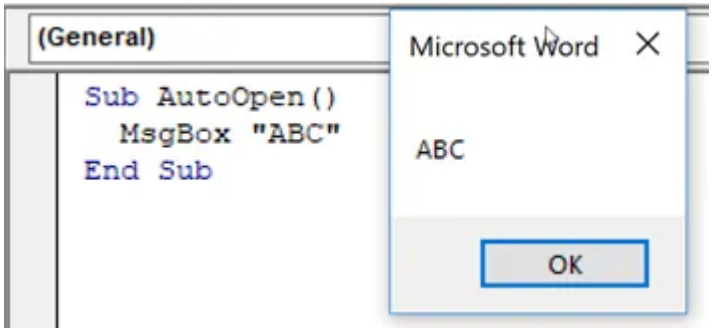
Authors: Kirk Sayre (@bigmacjpg), Harold Ogden (@haroldogden) and Carrie Roberts (@OrOneEqualsOne)

Introduction

There are powerful malicious document (maldoc) generation techniques that are effective at bypassing anti-virus detection. A technique which we refer to as *VBA stomping* was originally brought to our attention by Dr. Vesselin Bontchev (see here). VBA stomping refers to destroying the VBA source code in a Microsoft Office document, leaving only a compiled version of the macro code known as p-code in the document file. Maldoc detection based only on the VBA source code fails in this scenario. In this blog post we will demonstrate detailed examples of VBA stomping as well as introduce some additional techniques.

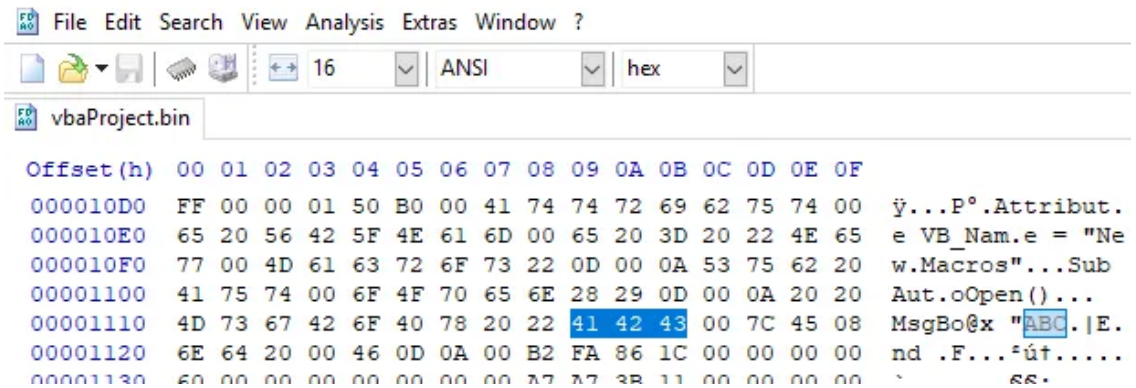
VBA Stomping

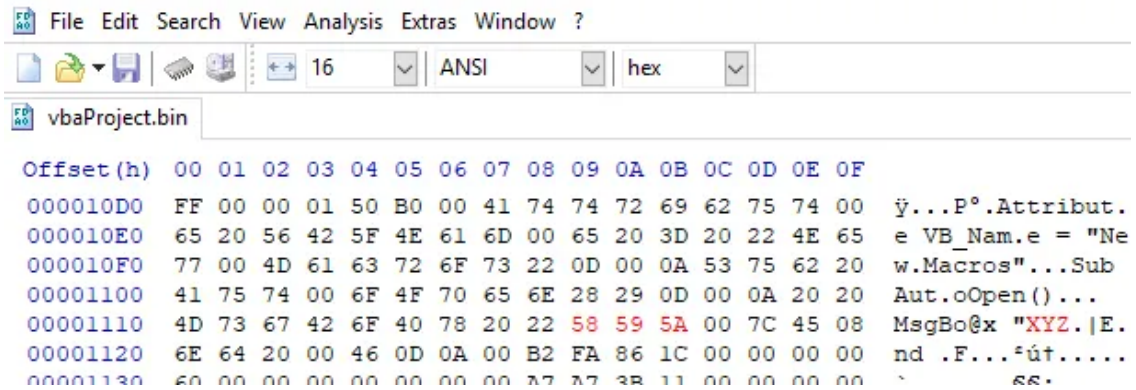
First, we will demonstrate VBA stomping on a simple, non-malicious macro. This document simply displays a message box with the text “ABC” when the document is opened. The VBA source code and resulting message box is shown below.



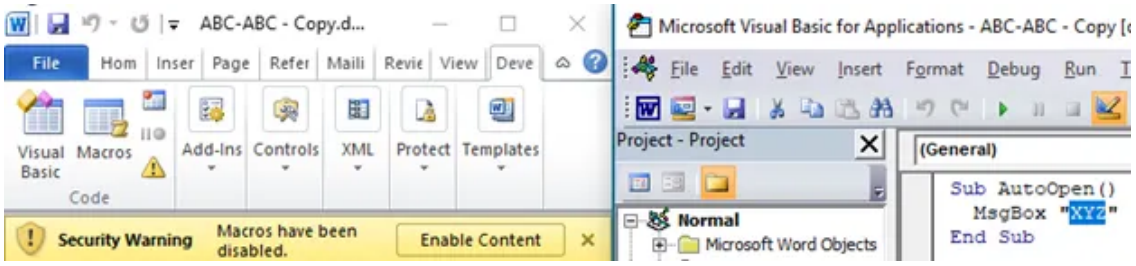
Now we are interested in modifying the VBA source code shown above, while leaving the intermediate p-code unchanged. For the purposes of this blog post we will concentrate on the current .docx/.xlsx/.docm/.xlsm (Office 2007+) format. However, the techniques discussed here can easily be applied to the older .doc/.xls format. In Office 2007+ files the VBA source code and p-code is typically located in a file called vbaProject.bin. Note that this is the default filename but it can be renamed. To manually modify this file, we need to first unzip the compressed .docm/.xlsm file and then open vbaProject.bin file in a hex editor. In this example, we will change “ABC” to “XYZ”, but only within the VBA source code storage location, not the p-code section. The VBA source code is stored in a compressed form which explains the unprintable or odd characters you see in the following image.

Press enter or click to view image in full size

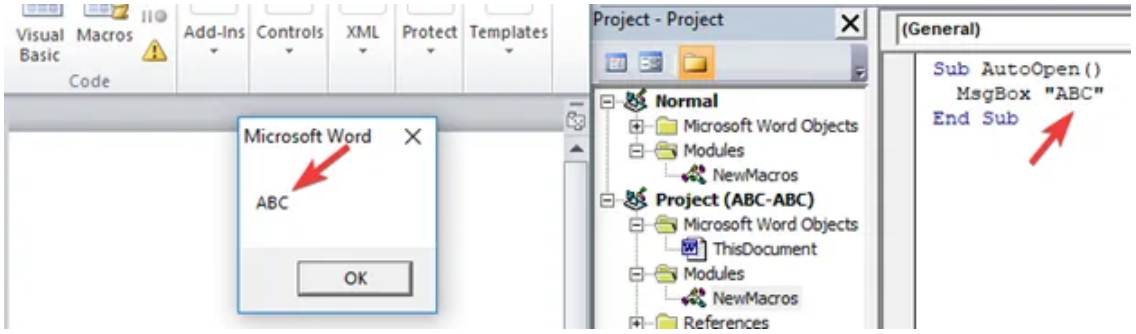




Now that we have manually edited the VBA source to change “ABC” to “XYZ”, we will open the document and inspect the VBA source code ***BEFORE*** the “Enable Content” button is clicked.



The document has been opened but macros are not enabled. Inspecting the macro in the code editor indicates that a message box with the text “XYZ” will be displayed but this is not the case. In fact, as soon as the content is enabled, a message box showing “ABC” is displayed and the source code in the code editor is updated to match!



Well, that surely was misleading. The source code said that “XYZ” was going to be displayed in the message box but instead “ABC” was displayed. What is going on?

As Dr. Bontchev explains, the p-code stored in the document is what actually executes, as long as it is compatible with the current VBA version on the system. In addition, what gets displayed in the macro editor (once content is enabled) is not the decompressed VBA source, but instead it is the decompiled p-code. Interesting!

If we open the document in a different version of Word (which uses a different VBA version) the p-code will not be reusable. This will force the VBA source code to be decompressed and recompiled to p-code, resulting in the display of “XYZ” in the message box. So now we have a single document, which on one version of Office displays “ABC”, but on another version displays “XYZ”.

Note that this is an important caveat. **A VBA stomped maldoc can only be executed using the same VBA version used to create the document.** This limitation can be addressed by doing reconnaissance on a target prior to maldoc generation to determine the appropriate Office version to use or by generating maldocs with multiple Office versions and spraying them against the target.

What is the effect of VBA stomping from a defensive standpoint? Maldoc detection is often based solely on the VBA source. Even many of the tools available for analyzing documents fail to identify the discrepancy between the VBA source and the p-code. The analysis of the tampered document by the olevba script, by Philippe Lagadec, shows the decompressed source only and lacks the p-code detail.

```
c:\Maldocs>olevba ABC-ABC.docm
olevba 0.51 - http://decalage.info/python/oletools
Flags        Filename
-----------  -----------------------------------------------------
OpX:MA------ ABC-ABC.docm
===============================================================================
FILE: ABC-ABC.docm
Type: OpenXML
-------------------------------------------------------------------------------
VBA MACRO ThisDocument.cls
in file: word/vbaProject.bin - OLE stream: u'VBA/ThisDocument'
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(empty macro)
-------------------------------------------------------------------------------
VBA MACRO NewMacros.bas
in file: word/vbaProject.bin - OLE stream: u'VBA/NewMacros'
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Sub AutoOpen()
  MsgBox "XYZ"
End Sub
+----------+----------+--------------------------------------+
| Type     | Keyword  | Description                          |
+----------+----------+--------------------------------------+
| AutoExec | AutoOpen | Runs when the Word document is opened |
+----------+----------+--------------------------------------+
```

The Python script oledump.py, by Didier Stevens, gives similar results unless run with the p-code dumper plugin. The output in this case gives some indication that the p-code will result in a message box display of “ABC” as shown in the image below.

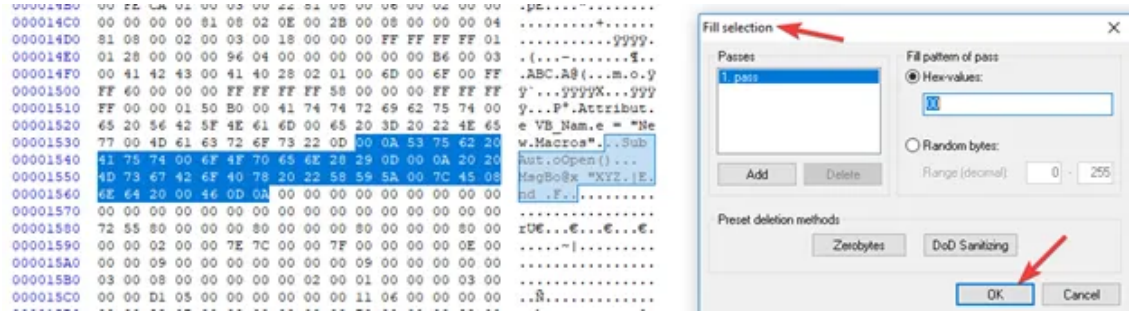
```
c:\Maldocs>oledump.py -s A3 -v ABC-ABC.docm
Attribute VB_Name = "NewMacros"
Sub AutoOpen()
  MsgBox "XYZ"
End Sub

c:\Maldocs>oledump.py -p plugin_pcode_dumper.py ABC-ABC.docm
A: word/vbaProject.bin
 A1:       410 'PROJECT'
 A2:        71 'PROJECTwm'
 A3: M    1383 'VBA/NewMacros'
               Plugin: P-code dumper
                 0000: 96 04 00 00 00 00          [......    ]
                 0001: B6 00 03 00 41 42 43 00 41 40 [....ABC.A@]
                       28 02 01 00                [(...      ]
                 0002: 6F 00                      [o.        ]
 A4: m    1097 'VBA/ThisDocument'
               Plugin: P-code dumper
```

Dr. Bontchev released a Python script called pcodedmp.py for displaying p-code as shown in the image below. The output has the added benefit of showing operation names instead of op-codes as above.

```
c:\Maldocs>pcodedmp.py -d ABC-ABC.docm
Processing file: ABC-ABC.docm
=================================================================
Module streams:
VBA/ThisDocument - 1097 bytes
VBA/NewMacros - 1383 bytes
Line #0:
        FuncDefn (Sub AutoOpen())
Line #1:
        LitStr 0x0003 "ABC"
        ArgsCall MsgBox 0x0001
Line #2:
        EndSub
```

Instead of just modifying the VBA source code, we could completely wipe it out (“stomp” it) by overwriting it with zero’s or random bytes. The following screenshot shows how to do this manually in a hex editor.



```
00001500  FF 60 00 00 00 FF FF FF FF 58 00 00 00 FF FF FF  ÿ`...ÿÿÿÿX...ÿÿÿ
00001510  FF 00 00 01 50 B0 00 41 74 74 72 69 62 75 74 00  ÿ...P°.Attribut.
00001520  65 20 56 42 5F 4E 61 6D 00 65 20 3D 20 22 4E 65  e VB_Nam.e = "Ne
00001530  77 00 4D 61 63 72 6F 73 22 0D 00 00 00 00 00 00  w.Macros".......
00001540  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00001550  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00001560  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00001570  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00001580  72 55 80 00 00 00 80 00 00 00 80 00 00 00 80 00  rU€...€...€...€.
```

The result in this case is a VBA editor that shows no macro code at all, yet the message box with the text “ABC” still appears after enabling content. Now we re-run olevba on this file and no VBA source code is displayed and the statement “No suspicious keyword or IOC found” is displayed instead of the table that previously highlighted our use of a suspicious keyword (AutoOpen).

```
c:\Maldocs>olevba Stomped.docm
olevba 0.51 - http://decalage.info/python/oletools
Flags        Filename
-----------  -----------------------------------------------------------------
OpX:M------- Stomped.docm
===============================================================================
FILE: Stomped.docm
Type: OpenXML
-------------------------------------------------------------------------------
VBA MACRO ThisDocument.cls
in file: word/vbaProject.bin - OLE stream: u'VBA/ThisDocument'
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(empty macro)
-------------------------------------------------------------------------------
VBA MACRO NewMacros.bas
in file: word/vbaProject.bin - OLE stream: u'VBA/NewMacros'
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

No suspicious keyword or IOC found.
```

How Big of a Problem Is This?

We will now dive a little deeper into the how Office uses the compressed VBA source code and p-code. This will come into play later when we go over how to abuse Office functionality to easily modify maldocs to fool many AV scanners.

Get Carrie Roberts’s stories in your inbox

Join Medium for free to get updates from this writer.

Remember me for faster sign in

Office documents contain two places from which to pull the VBA, the compressed VBA source code and the p-code. The following table details which VBA data source is used when:

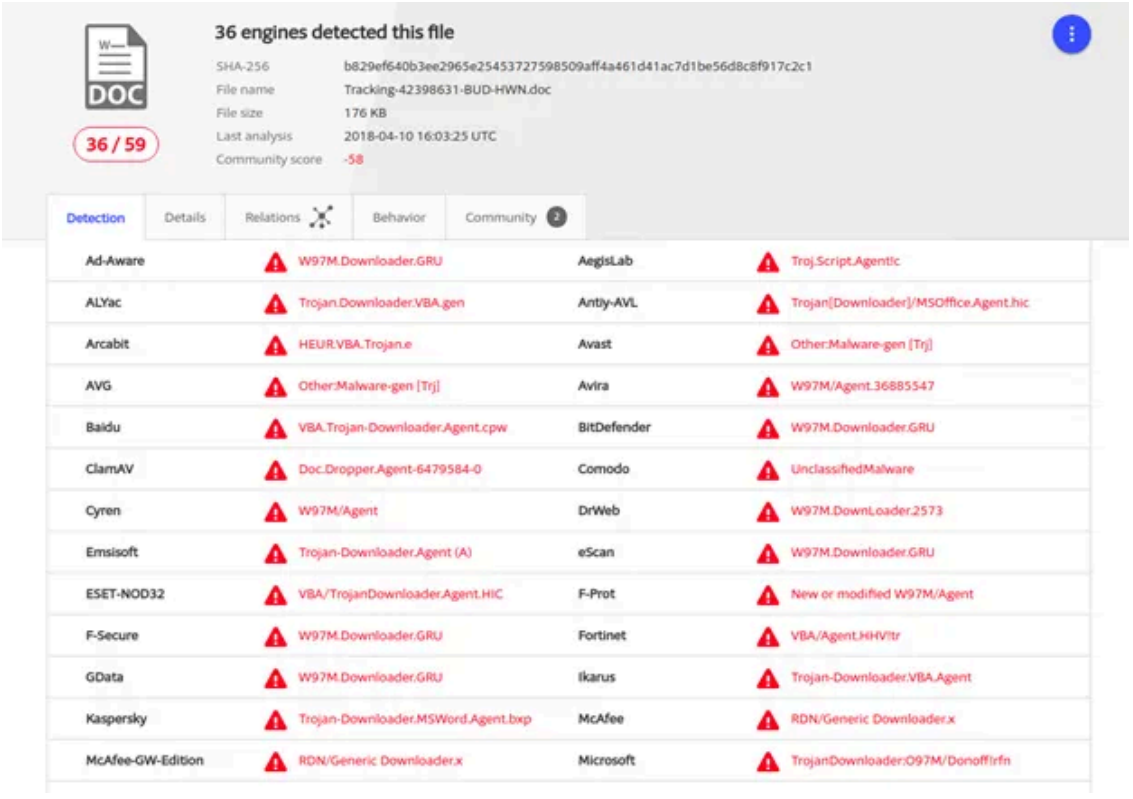
Press enter or click to view image in full size

Macros Enabled?	Viewing in IDE?	Valid VBA Source?	Valid P-Code?	Data Source Used
No	Yes	Yes	-	Compressed VBA Source
No	Yes	No	-	Blank or Error
Yes	-	-	***Yes***	P-Code
Yes	-	Yes	No	Compressed VBA Source
Yes	-	No	No	Error

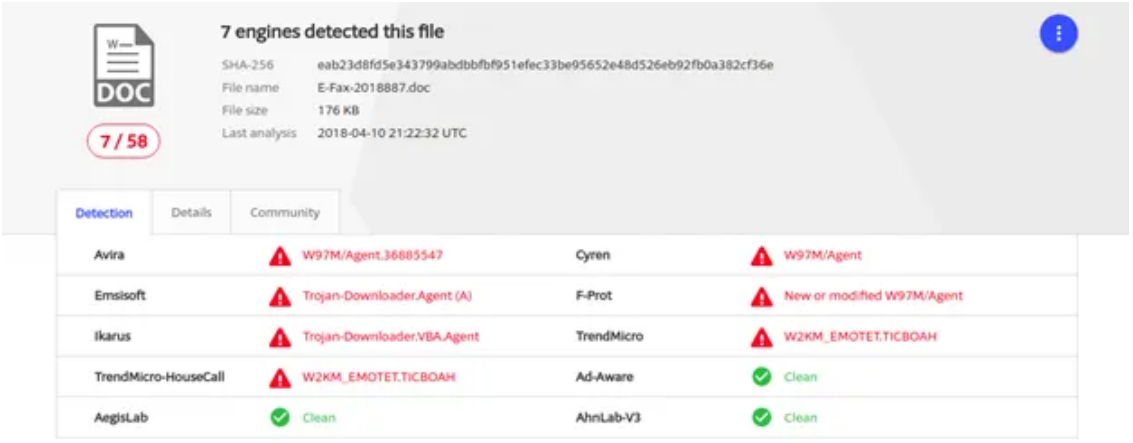
From this table we see that if we have valid p-code, the compressed VBA source code is ignored when macros are enabled. All macro functionality is performed by running the p-code. As an attacker this tells us that we can completely ruin or modify the compressed VBA source code and still have our maldoc perform its desired task.

Real World Example

Showing that this works on a toy example is all well and good, but is this really a legitimate threat? Let's take this to the next level by working with a known malicious document. In this case, we will start with a recent Emotet maldoc that is currently detected as malicious by 36/59 vendors on Virus total.



If we take this document and simply stomp the VBA source code as described above, the detection rate goes down to 7 out of 58 anti-virus solutions!

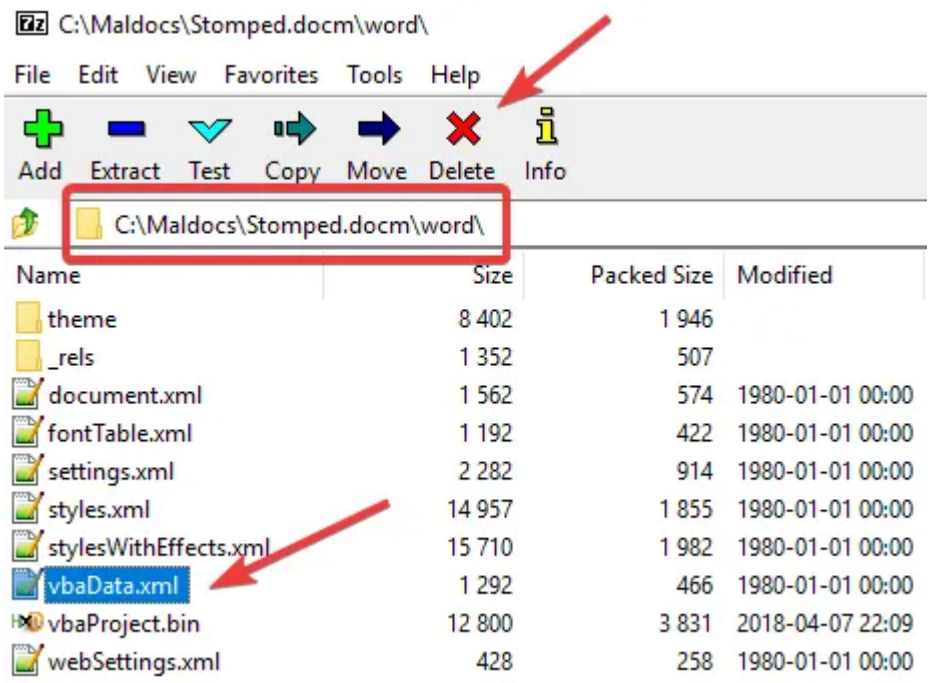


As you can see, this creates a serious problem for network defenders. Even manual analysis of such documents could be problematic. The obvious question at this point is "*Can VBA stomping be performed automatically by a tool or do we always have to manually edit the Office document?*" The answer to this question is ***yes, it certainly is possible to stomp VBA automatically*** with a tool. We have developed a POC utility to automatically stomp the compressed VBA source code in any Office document (this was used to VBA stomp the example Emotet document and no, we will not be releasing this utility). Given how easy it is to automate VBA stomping this is a real threat.

Making It Worse

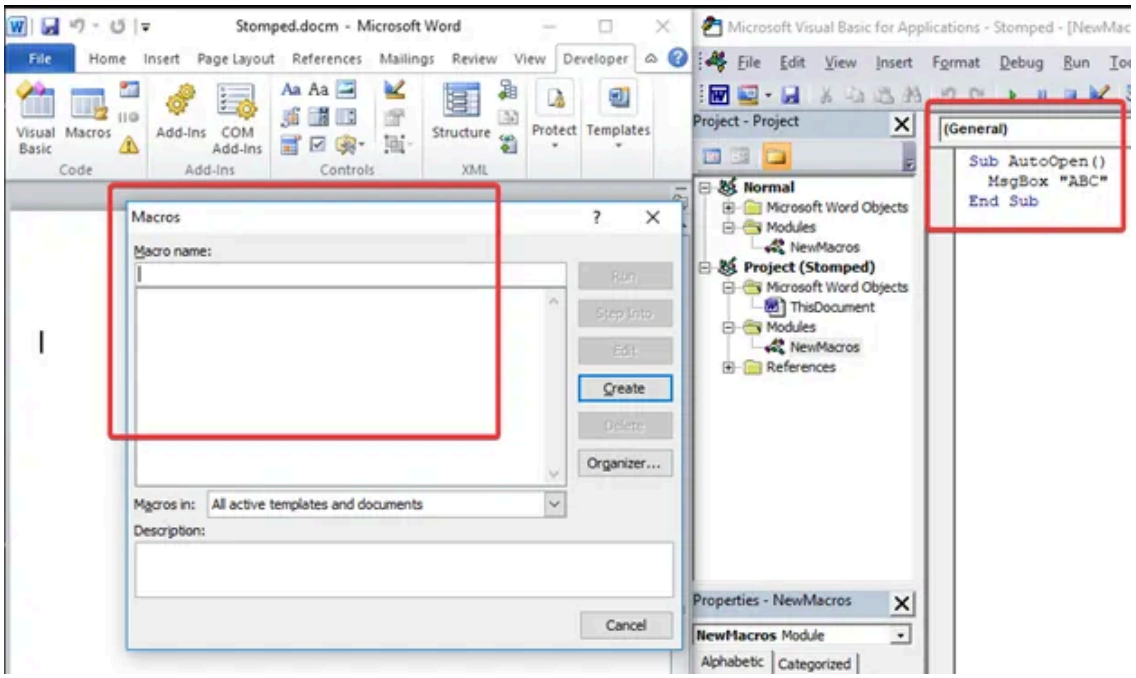
Consider the case where the maldoc is designed to close MS Word immediately after executing the malicious payload. In combination with VBA stomping, no offline VBA source extraction tools will display the VBA source and the VBA source will not even be displayed in the Office macro editor window until the macro is enabled. But enabling the macro causes Word to immediately exit without providing an opportunity to view the decompiled source. While we have shown that the "assembly-like" p-code is extractable, the p-code is difficult to interpret and cannot be analyzed in the VBA debugger. In addition, the pcode only runs on a specific VBA version. For these reasons, access to the VBA source code would be a great benefit to the analyst. But how can this be done for such a document?

After some experimentation, we determined a simple solution to stop MS Word from executing any method as soon as macros are enabled. Note that this solution works for Office 2007+ documents, additional research is being performed to address older document formats. The 2007+ solution is to delete the vbaData.xml file from the "word" directory of the .docm file as shown below (this is easily done with the 7-Zip program without having to manually unzip and re-zip the document).



The document can now be opened and the content can be enabled, resulting in the display of the decompiled p-code in the VBA code editor but no code execution. This could be a life saver in the event that the malware author has taken steps to prevent the analyst from having access to the VBA source code.

An interesting side effect of erasing vbaData.xml is that it causes MS Word to erroneously list no Macros in the Macros dialog (see image below).



Detection

We have written an open source tool for detecting VBA stomping in Office documents called “VBA Seismograph”. This tool has been tested under Ubuntu 16.04 and detects differences between the declared

function/variable names, string literals, and comment lines that appear in the compiled p-code and the VBA source code of an Office document. The tool is available at https://github.com/kirk-sayre-work/VBASeismograph.

Conclusion

In this blog post we have demonstrated how the compressed VBA source code in an Office document can be modified or destroyed to defeat AV scanning solutions and make manual analysis of maldocs more difficult. Defense against this technique involves detecting and flagging documents with valid p-code but invalid or missing VBA source code, or more generally, checking for discrepancies between the compressed VBA source code and the decompiled p-code. At this point in time we are not aware of any commercial AV solutions that do this check, so this is a potential area where AV scanning solutions can improve.

See vbastomp.com for the most up to date information on VBA stomping.
