Evil Clippy: MS Office maldoc assistant | Outflank

By Stan

Published: 2019-05-05 · Archived: 2026-04-05 15:05:58 UTC

At BlackHat Asia we released Evil Clippy, a tool which assists red teamers and security testers in creating malicious MS Office documents. Amongst others, Evil Clippy can hide VBA macros, stomp VBA code (via p-code) and confuse popular macro analysis tools. It runs on Linux, OSX and Windows.

In this blog post we will explore the features of Evil Clippy and the technology behind it. The latest source code of the tool can be found here:

https://github.com/outflanknl/EvilClippy

Latest binary releases are available at:

https://github.com/outflanknl/EvilClippy/releases

Use cases

At the time of writing, this tool is capable of getting malicious macros to bypass all major antivirus products and most maldoc analysis tools. It achieves this by manipulating MS Office files on a file format level.

The following screenshot shows detection rates of a default Cobalt Strike VBA macro *before* Evil Clippy is applied. This is a basic process injection macro which is obviously malicious and is detected by practically all major antivirus vendors.

After applying Evil Clippy to this document (EvilClippy.exe -s fake.vbs -g -r cobaltstrike.doc), all major antivirus engines fail to detect this macro.

Note that Evil Clippy only focuses on evasion of static analysis. A combination which can be particularly effective for initial compromise during red teaming operations is to deliver malicious documents generated with Evil

Clippy via HTML smuggling (which helps you bypass perimeter defenses). If you need to evade dynamic analysis of macros, then read our blog on bypassing AMSI for VBA.

Evil Clippy is written in C#. The code compiles perfectly fine with the Mono C# compiler and has been tested on Linux, OSX and Windows. Compilation and usage instructions can be found in the readme.

Technology

In order to understand how Evil Clippy works, we need to dive into the Compound File Binary Format (CFBF). This file format is used abundantly in MS Office. Evil Clippy supports the following two MS Office file types:

- *97 – 2003 format* (.doc and .xls), these files are entirely in the CFBF file format.
- *2007+ format* (.docm and .xlsm), these files are actually ZIP containers. If you unzip the file, you will find a file named "vbaProject.bin" in the "word" (.docm) or "xl" (.xlsm) directories. This file is a CFBF file which contains the macro information for the document.

Evil Clippy uses the OpenMCDF library to manipulate CFBF files. If you want to manipulate CFBF files manually, then FlexHEX is one of the best editors for this.

The pane on the bottom left shows the structure of this particular CFBF file. A CFBF file is like a file system in a single file: there are streams (think of them as "files") which can be stored in containers (think of them as "directories").

The "macros" container contains all subcontainers and streams related to VBA macros in a MS Word document. In MS Excel files this container is called "_VBA_PROJECT_CUR". The structure and contents of this container are explained by Microsoft in a lengthy and complex specification called MS-OVBA.

The most important streams in the macros container are:

- PROJECT: a stream with text data specifying project information. It can be regarded as a configuration file with CRLF separated lines which instruct the VBA editor GUI (amongst others).
- _VBA_PROJECT: a stream with binary data which instructs the VBA engine. There is no official documentation on the contents of this stream, but it is crucial in how macros are interpreted (which we will see later in this post).
- Dir: a stream which contains the VBA project layout. This stream is compressed and its format is rather well documented.
- Module streams: in the screenshot above there are two module streams named “ThisDocument” and “Module1”. These contain the actual code that is going to run. Each module stream consist of two parts: a PerformanceCache, which is undocumented, and CompressedSourceCode, which contains the VBA macro source code in a compressed format.

Note that some streams contain compressed data (most notably the dir and module streams), using a custom compression algorithm. Evil Clippy reuses code from Kavod.VBA.Compression to implement the compression algorithm.

For more information on this file format, watch our TROOPERS19 presentation titled MS Office File Format Sorcery. If you dive deeper into the MS-OVBA specifications, you will soon note the following issues:

- There are many features which are **not documented** (for example, the PerformanceCache mentioned above). It turns out however that many of these functions are critical for how VBA macros are actually processed and interpreted. More on this in the “VBA stomping” section below.
- The specifications lead to **ambiguity**. For example, module names are stored in multiple places. The section “Confusing analysis tools” later in this blog post will deal with this aspect.
- The specifications are **very long and complex**. What happens when you slightly deviate from the specs? We found that MS Office is much more robust and forgiving to such deviations than most analysis tools (example).

Evil Clippy exploits these issues to evade static analysis by antivirus engines as well as manual VBA macro analysis tools.

VBA stomping (via p-code)

The most powerful technique of Evil Clippy is “VBA stomping”. VBA stomping abuses a feature which is not officially documented: the undocumented PerformanceCache part of each module stream contains compiled pseudo-code (p-code) for the VBA engine. If the MS Office version specified in the _VBA_PROJECT stream matches the MS Office version of the host program (Word or Excel) then the VBA source code in the module stream is ignored and the p-code is executed instead.

In summary: if we know the version of MS Office of a target system (e.g. Office 2016, 32 bit), we can replace our malicious VBA source code with fake code, while the malicious code will still get executed via p-code. In the meantime, any tool analyzing the VBA source code (such as antivirus) is completely fooled.

Vesselin Bontchev was the first to publicly document this technique (samples here). The security team from Walmart has gathered great resources on this topic here. We briefly discussed this technique in our talk "The MS Office Magic Show" at DerbyCon. Evil Clippy is the first tool to fully weaponize this technique for red teamers and security testers.

For example, in order to replace the VBA source code in your malicious document with fake code and to have the malicious p-code run on 32 bit installs of Office 2016, run Evil Clippy with the following command line:

```
EvilClippy.exe -s fakecode.vba -t 2016x86 macrofile.doc
```

After executing this command, a file named "macrofile_EvilClippy.doc" will be created. Any common VBA analysis tools such as OleVBA, OleDump or VirusTotal will tell you that this file contains the fake code. However, if the file is opened on a 32 bit install of Office 2016 (our target version), the malicious p-code is executed instead.

So in order to make this attack work, we need to know the MS Office version of our target. Achieving this is not as difficult as it sounds. A common trick is to send the victim an email that contains a tracking pixel. If the pixel is retrieved by MS Outlook (which is usually part of the same MS Office install), the headers of this HTTP request will disclose the MS Office version number and whether it is a 32 or 64 bit install.

Evil Clippy can also automate the target version identification for you using a templating trick. The process hereto is as follows:

- You create a MS Word template (.dot / .dotm) that includes a malicious macro.
- Run the Evil Clippy tool with the –webserver option and point it to the malicious template file. It will then spin up a web server that listens for incoming connections on the port you specified.
- You subsequently create another document that points to this template document via a URL by referencing it as a template from the developer toolbar in Word (you can use an arbitrary extension, .dot extension is not required!).
- If this document is opened, MS Office will reach out to the web server to fetch the template. Evil Clippy will identify the target MS Office version from the HTTP header and will patch the version bytes in the _VBA_PROJECT stream of the malicious template before serving it.

Make sure to read the caveats in the readme before you use this option! The following screenshots shows this feature in action:

Hiding macros

Note that *after* running a macro using p-code in MS Office, the VBA engine will reconstruct the VBA source code from p-code and display it in the VBA editor GUI when opened. Hence, we need an additional trick to hide our malicious code from prying eyes.

One technique to achieve this is to modify the PROJECT stream. As mentioned above, this stream can be regarded as a configuration file for the VBA editor GUI. By simply removing a line such as “Module=Module1” we can make the module named “Module1” disappear in the VBA editor. Note that this does *not* work with the default “ThisDocument” or “ThisWorkbook” modules, so you have to place your malicious code in a separate module in order to make this trick work.

Evil Clippy can automatically hide all modules with the -g flag:

```
EvilClippy.exe -g macrofile.doc
```

Another trick to make macro code unaccessible is to mark the project as locked and unviewable. This feature has been added to Evil Clippy by Carrie Robberts from Walmart and can be used with the -u flag. Carrie has written a detailed blog on this topic.

Confusing analysis tools

Although the -g and -u flags can hide our malicious code from human eyes, they do not protect against security tools which analyse our file on a file format level. For example, pcodedmp can be used by an analyst to extract p-code from a document. However, we can use the ambiguity and complexity of the MS-OVBA specification to confuse various analysis tools, including pcodedmp.

Evil Clippy can confuse pcodedmp and many other analysis tools with the -r flag.

```
EvilClippy.exe -r macrofile.doc
```

After running pcodedmp on this file, it crashes with a “file not found” error and empty list of module names:

How does it achieve this? With the -r flag Evil Clippy sets random ASCII module names in the Dir stream. Most analyst tools use the module names specified here, while MS Office parses module names from the undocumented _VBA_PROJECT stream. By setting a random ASCII module name most P-code and VBA analysis tools crash, while the actual P-code and VBA still runs fine in Word and Excel. More information about the root cause of this issue can be read in this Twitter thread by Vesselin Bontchev, the author of pcodedmp.

Detection and mitigation

Evil Clippy only scratches the surface of issues resulting from the gap between official Microsoft specifications on VBA macros (MS-OVBA) and its actual implementation in MS Office. Since malicious macros are one of the most common methods for initial compromise by threat actors, proper defense against such macros is crucial. We believe that the lack of adequate specifications of how macros actually work in MS Office severely hinders the work of antivirus vendors and security analysts. This blog post serves as a call to Microsoft to change this for the better.

In the meantime there are several things that you can do to make the life of a malicious macro much harder:

- For users and teams in your organisation that do not use macros, turn them off.
- For users that *do* require working with macros, consider disabling macros in documents that are downloaded from the internet. Note that this feature is available in MS Office 2013 and 2016, and can be controlled via GPO.
- Use Attack Surface Reduction rules to limit the impact of malicious macros. Note that the most important rules can already be turned on with an E3 license. If you fear the impact of these rules in blocking mode, then consider enabling them in audit mode for monitoring.
- Deploy an antivirus product that hooks into the AMSI for VBA engine. Although this implementation of AMSI is far from perfect, it will raise the bar for malicious macros.
- Monitor the execution of macros via our Sysmon trick, which can help you in hunting evil macros.

Authors and contributions

Evil Clippy is maintained by Stan Hegt (@StanHacked).

After its initial release, the project has received significant code contributions from the security team at Walmart, especially from Carrie Robberts (@OrOneEqualsOne). If you want to contribute to Evil Clippy, we are happy to receive your pull request.

Special thanks to Nick Landers from Silent Break Security (@monoxgas) for pointing us towards OpenMCDF.
