# BETWEEN AN RTF AND OLE2 PLACE: AN ANALYSIS OF CVE-2012-0158 SAMPLES

*Paul Baccas*
Independent researcher, UK

Email pobicus@gmail.com

## ABSTRACT

Over the last few years, *Microsoft Office* viruses have become a thing of the past and now we generally see *Office* files as a delivery method for targeted attacks via vulnerabilities. File format features make obfuscating these targeted attacks easy, and detecting them hard.

Targeted attacks like 'Red October' and 'FakeM' relied on three different kinds of *Microsoft* vulnerabilities. Two of them, CVE-2009-3129 and CVE-2010-3333, posed challenges for detection development and have been discussed previously. However, the third, CVE-2012-0158, is quite different. The exploit itself could be in either a *Word* or an *Excel* document, but the delivery method could be a *Word*, *Excel* or, more commonly, RTF file. Digging through this complexity requires a strong understanding of RTF and OLE2, as well as all points in between.

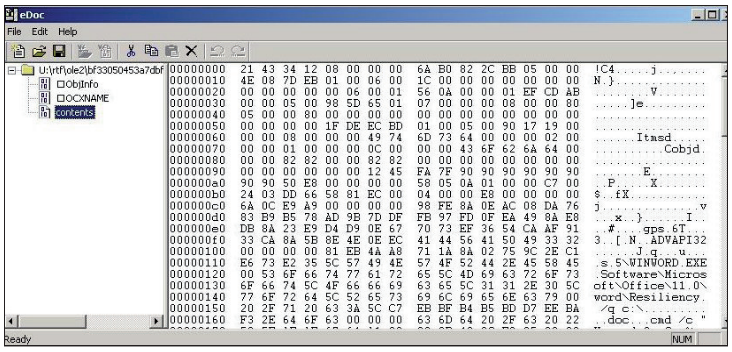
This paper will document some pitfalls of the file formats that make detection problematic, with particular attention placed on the small percentage (less than 1%) of CVE-2012-0158 associated with high-profile attacks.

## 1. INTRODUCTION

After April 2012's Patch Tuesday, *Microsoft* reached out to the author [1] looking for obfuscated/fuzzed files exploiting CVE-2012-0158 (MS12-027) because of work published on CVE-2012-3333 [2]. At the time, we had not looked at detection for the exploit, and when we saw that *Sophos* relied on detecting exploited files purely from within our OLE2 plug-in, we changed the methodology to one that would not rely on the OLE2 plug-in.

The first requirement for a detection modification was because the relevant OLE2 stream name had been changed (see Figure 1).

The second was in order to write detection for the raw RTF.



```
00000000  21 43 34 12 08 00 00 00  6A B0 82 2C BB 05 00 00  !C4.....j.......
00000010  4E 08 7D EB 01 00 06 00  1C 00 00 00 00 00 00 00  N.}.............
00000020  00 00 00 00 00 06 00 01  56 0A 00 00 01 EF CD AB  ........V.......
00000030  00 00 05 00 98 5D 65 01  07 00 00 00 08 00 00 80  .....]e.........
00000040  05 00 00 80 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000050  00 00 00 00 1F DE EC BD  01 00 05 00 90 17 19 00  ................
00000060  00 00 08 00 00 00 49 74  6D 73 64 00 00 00 02 00  ......Itmsd.....
00000070  00 00 01 00 00 00 0C 00  00 00 43 6F 62 6A 64 00  ..........Cobjd.
00000080  00 00 82 82 00 00 82 82  00 00 00 00 00 00 00 00  ................
00000090  00 00 00 00 00 00 12 45  FA 7F 90 90 90 90 90 90  .......E........
000000a0  90 90 50 E8 00 00 00 00  58 05 0A 01 00 00 C7 00  ..P.....X.......
000000b0  24 03 DD 66 58 81 EC 00  04 00 00 E8 00 00 00 00  $..fX...........
000000c0  6A 0C E9 A9 00 00 00 00  98 FE 8A 0E AC 08 DA 76  j..............v
000000d0  83 B9 B5 78 AD 9B 7D DF  FB 97 FD 0F EA 49 8A E8  ...x..}......I..
000000e0  DB 8A 23 E9 D4 D9 0E 67  70 73 EF 36 54 CA AF 91  ..#....gps.6T...
000000f0  33 CA 8A 5B 8E 4E 0E EC  41 44 56 41 50 49 33 32  3..[.N..ADVAPI32
00000100  00 00 00 00 81 EB 4A A8  71 1A 8A 02 75 9C 2E C1  ......J.q...u...
00000110  E6 73 E2 35 5C 57 49 4E  57 4F 52 44 2E 45 58 45  .s.5\WINWORD.EXE
00000120  00 53 6F 66 74 77 61 72  65 5C 4D 69 63 72 6F 73  .Software\Micros
00000130  6F 66 74 5C 4F 66 66 69  63 65 5C 31 31 2E 30 5C  oft\Office\11.0\
00000140  77 6F 72 64 5C 52 65 73  69 6C 69 65 6E 63 79 00  word\Resiliency.
00000150  20 2F 71 20 63 3A 5C C7  EB BF B4 B5 BD D7 EE BA   /q c:\.........
00000160  F3 2E 64 6F 63 00 00 00  63 6D 64 20 2F 63 20 22  ..doc...cmd /c "
```

*Figure 1: 'contents' with lower-case C.*

Future changes to detections were added as the bad actors modified their samples – either fuzzing the embedded OLE2 or the RTF, or other tricks such as password-protecting files [3]. Even with this dual approach, the detection still required almost weekly updates. Both file formats, RTF and OLE2, are ripe for fuzzing, partially because of the redundancies within the files. The author has previously talked about RTF manipulations [2] and OLE2 non-conformance [4] and so will not go into exhaustive details here.

## 2. METHODS AND TOOLS

### 2.1 Overview

A set of Python tools were developed to parse and classify the RTF and OLE2 files, and display the results. Use was made of the *oletools* Python package [5], which was useful for some things and not for others. The author would have liked to have had more time to help with this project but deadline pressures made it impossible. Some of the tools were based on the proprietary *Sophos* Virus Description Language (VDL) because modifying the internal tools was faster than using Python.

### 2.2 File formats

The OLE2 file format has been well documented and this paper will only cover part of that format. The RTF file format has not been so well documented and is described briefly below.

#### *2.2.1 RTF*

```
{\rtf1\ansi\deff0 {\fonttbl {\f0 Times New Roman;}}
\f0\fs60 Hello, World!}
```

*Table 1: My first RTF file (Hello.rtf).*

A simple RTF file, Hello.rtf (see Table 1) [6], is divided into:

```
<File>      '{' <header> <document> '}'
```

*Table 2: General RTF structure.*

Splitting the test according to the general RTF structure (see Table 2 [7]) you would have:

```
<header>     \rtf1\ansi\deff0
```

*Table 3: RTF <header>.*

This test RTF header is invalid according to the spec as it does not have a <*deflang*>(image) [7], and yet it opens in *Word 2010*[1].

*'**Note:** RTF readers can reject input if strongly illegal data is encountered that is most probably created maliciously … the RTF reader should probably reject the file.'* [7]

#### *2.2.2 Whitespace*

One of the ways in which an RTF can be manipulated is via the addition of whitespace. To make the Hello.rtf file more readable we could add

[1] If a null edit is made, the 75-byte file becomes 31KB.

some carriage returns (CR), 0x0d, and/or linefeeds (LF), 0x0a:

```
{\rtf1\ansi\deff0
{\fonttbl
{\f0 Times New Roman;}}
\f0\fs60
Hello, World!
}
```

*Table 4: My second RTF file (Hello2.rtf).*

Hello2.rtf is functionally the same as Hello.rtf, yet with more CRLFs you could add N more occurrences of 0x0d or 0x0a.

‘*Unlike most clear text files, an RTF file does not have to contain any carriage return/linefeed pairs (CRLFs) and CRLFs should be ignored by RTF readers …*’ [8]

Or

‘*You can get exactly the same document if you remove all the newlines in your RTF … Or you can insert many newlines, in certain places …*’ [9]

Also, you can insert spaces and still have no difference in the displayed RTF.

‘*… but insert a space after each \foo command …*’ [10]

Where \foo is an example of a generalized RTF control word.

```
{\rtf1 \ansi \deff0 {\fonttbl {\f0 Times New
Roman;}} \f0 \fs60 Hello, World!}
```

*Table 5: My third RTF (hello3.rtf).*

The tricks in Tables 4 and 5 can be combined as many times as you like – a fact that the bad actors are aware of.

### 2.2.3 Control word

A control word has the general form of:

/(\\[a-zA-Z]{1,32}(-?[0-9]{1,10})?)[^a-zA-Z0-9]/

more formally:

```
\<ASCII Letter Sequence><Delimiter>
```

*Table 6: Control word definition [7].*

Where <ASCII Letter Sequence> can be up to 32 upper- or lower-case characters. The <Delimiter> can be a space, a positive or negative number up to 10 digits, or any character other than a letter or digit.

This form differs slightly from that of Burke’s [11] /\\[a-z]+(-?[0-9]+)? ?/ with the addition of upper-case characters and the precision of the matching. It also differs from the regular expressions derived from analysing samples in Appendix I.

### 2.2.4 Special characters

There are a number of special characters, or escapes, that can follow the reverse solidus, or backslash (0x5c):

Ideally, these special characters (see Table 7) need to be handled within the parser, however, as these characters should not appear within the hex-encoded data within an \<object>, it may be considered sufficient to ignore them within objects.

| Special character | Meaning |
|---|---|
| \’ | Signifier that the next two characters should be treated as hex e.g. \’61 -> a. |
| \- | Hyphenation point. If a word could be wrapped, place the hyphen here. |
| * | If the next control word is not known, ignore the following group. |
| \: | Specifies a subentry in an index entry. |
| \\ | An escaped reverse solidus. |
| _ | Non-breaking hyphen. For hyphenated words, this specifies that the word cannot be line wrapped. |
| \{ | An escaped opening brace or curly bracket. |
| \| | Formula character. (Used by *Word 5.1* for the *Macintosh* as the beginning delimiter for a string of formula typesetting commands.) |
| \} | An escaped opening brace or curly bracket. |
| \~ | Non-breaking space. For phrases, this specifies that the phrase cannot be line wrapped. |

*Table 7: Special characters [12].*

### 2.2.5 Objects

‘*Microsoft OLE links, Microsoft OLE embedded objects, and Macintosh Edition Manager subscriber objects are represented in RTF as objects.*’ [13]

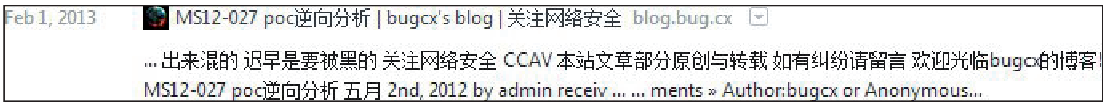
The objects consist of some data and a result as well as some preamble. The data is stored as a hex-encoded binary blob. The data should only contain hex characters ([0-9a-fA-F]), however, in almost all samples there are extraneous control words, special characters, or whitespace within the embedded object.

### 2.2.6 OLE2: encryption stream

The encryption stream within an OLE2 is described in various *Microsoft* documents [14, 15]. The actual algorithms are described in [MS-CRYPTO].pdf. *SophosLabs* have only encountered files encrypted with the RC4 CryptoAPI. The RC4 CryptoAPI encryption header [16] is in the Table and Workbook streams of *Word* and *Excel*, respectively.

| RC4 CryptoAPI encryption header |
|---|
| EncryptionVersionInfo (4 bytes) |
| EncryptionHeader.Flags (4 bytes) |
| EncryptionHeaderSize (4 bytes) |
| EncryptionHeader (variable) |
| EncryptionVerifier (variable) |

*Table 8: RC4 CryptoAPI encryption header.*


Feb 1, 2013 MS12-027 poc逆向分析 | bugcx's blog | 关注网络安全 blog.bug.cx

... 出来混的 迟早是要被黑的 关注网络安全 CCAV 本站文章部分原创与转载 如有纠纷请留言 欢迎光临bugcx的博客!
MS12-027 poc逆向分析 五月 2nd, 2012 by admin receiv ... ... ments » Author:bugcx or Anonymous...

*Figure 2: Internet history.*

The EncryptionHeader defines which encryption algorithm and hash are used. Also defined is the KeySize, which is important because:

> '*When used with small key lengths (such as 40-bit), brute-force attacks on the key without knowing the password are possible.*' [17]

Within *Excel*, documents can be encrypted using RC4 CryptoAPI, but as write-protected [18] documents. The password in this case is:

| | |
|---|---|
| Hex | `\x56\x65\x6C\x76\x65\x74\x53\x77\x65\x61\x74\x73\x68\x6F\x70".` |
| ASCII | `VelvetSweatshop` |

*Table 9: Excel write protection password [19].*

Upon encountering an RC4-encrypted document, *Excel* will try the write protection password first, and if that fails it will prompt for a password.

### *2.2.7 Exploit*

The information upon which the author's detections were initially based was under non-disclosure agreement (NDA) with Microsoft Active Protection Program (MAPP), however, the detection logic changed to be based on the samples. Others were able, without MAPP information, to create a Metasploit [20] module. *Mitre.org* describes the CVE as:

> '*The (1) ListView, (2) ListView2, (3) TreeView, and (4) TreeView2 ActiveX controls in MSCOMCTL.OCX in the Common Controls in Microsoft Office 2003 SP3, 2007 SP2 and SP3, and 2010 Gold and SP1; Office 2003 Web Components SP3; SQL Server 2000 SP4, 2005 SP4, and 2008 SP2, SP3, and R2; BizTalk Server 2002 SP1; Commerce Server 2002 SP4, 2007 SP2, and 2009 Gold and R2; Visual FoxPro 8.0 SP1 and 9.0 SP2; and Visual Basic 6.0 Runtime allow remote attackers to execute arbitrary code via a crafted (a) web site, (b) Office document, or (c) .rtf file that triggers "system state" corruption, as exploited in the wild in April 2012, aka "MSCOMCTL.OCX RCE Vulnerability."*' [21]

Various hacking websites have had reasonably complete data on the vulnerability. One of these, bug.cx [22], had a commented discussion of the exploit:

> 'cbsize < 8 *//应该是大于号*'

According to a colleague at *SophosLabs*, the Chinese characters translate as:

> 'is supposed to be greater than symbol'

The next comment:

> '*//栈溢出*'

translates as 'stack overflow'.

This site was up on 1 February 2013, but at the time of writing this paper the site is down.

The Metasploit module contains a stripped down OLE2 file (neither *Word* nor *Excel*) which has a Contents stream that contains the exploit and shellcode. We know from looking at Flash exploits [23], that *Word* documents can contain a Contents stream, and that the *Excel* equivalent is the Ctls Stream [24].

## 3. RESULTS

### 3.1 Sample set

After filtering the samples received from customers, there were 3,392 files left, submitted between 10 April 2012 and 4 April 2013 (359 days). Unfortunately, exact dates can only be confirmed for 2,939 of the files (Figure 3). There were an average of 8.2 files a day (an increase on CVE-2010-3333 [2]).

The sample set was scanned with a Python script, scan-tree.py (see Appendix II), to determine whether they were RTF, encrypted, *Word* or *Excel* files.

### 3.2 Types by date

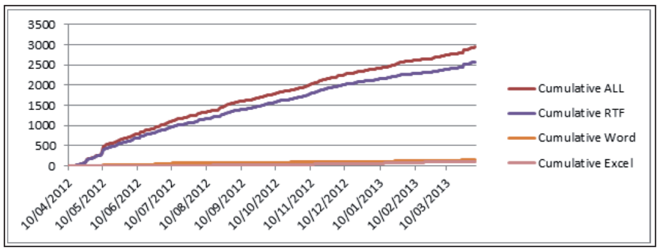


*Figure 3: File types by date.*

Of the 2,939 files, 2,571 (87.5%) were RTFs and the rest a mixture of *Excel* and *Word* OLE2 files, with seven being neither.

In Figure 3 we see that the distribution of RTF samples is similar to all samples. The *Word* samples are comprised of both plain *Word* files and encrypted *Word* files.

Of the 137 *Word* files, there were four encrypted files with known dates.

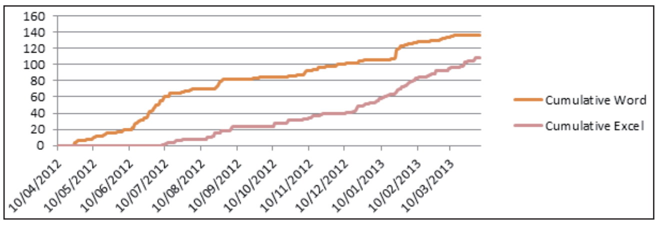


*Figure 4: Word and Excel.*

Of the 108 *Excel* files, the majority (71 files) were encrypted.

Finally, 17 files caused the script to raise an exception; again the author would have liked more time to investigate why the files caused the tools to error.

| Raw numbers | Percentage | First six characters | First six characters hex |
|---|---|---|---|
| 1673 | 59.03% | {\rtf1 | 7b5c72746631 |
| 220 | 7.76% | {\rt0{ | 7b5c7274307b |
| 181 | 6.39% | {\rta3 | 7b5c72746133 |
| 161 | 5.68% | {\rtX\ | 7b5c7274585c |
| 137 | 4.83% | {\rtA1 | 7b5c72744131 |
| 135 | 4.76% | {\rta1 | 7b5c72746131 |
| 46 | 1.62% | {\rtf{ | 7b5c7274667b |
| 43 | 1.52% | {\rts1 | 7b5c72747331 |
| 33 | 1.16% | {\rt | 7b5c72742020 |
| 32 | 1.13% | {\rtXÿ | 7b5c727458ff |
| 24 | 0.85% | {\rt## | 7b5c72742323 |
| 21 | 0.74% | {\rtt{ | 7b5c7274747b |
| 21 | 0.74% | {\rtt1 | 7b5c72747431 |
| 19 | 0.67% | {\rtxa | 7b5c72747861 |
| 15 | 0.53% | {\rt01 | 7b5c72743031 |
| 14 | 0.49% | {\rt.1 | 7b5c72740031 |
| 11 | 0.39% | {\rta\ | 7b5c7274615c |
| 8 | 0.28% | {\rtF{ | 7b5c7274467b |
| 7 | 0.25% | {\rtf2 | 7b5c72746632 |
| 5 | 0.18% | {\rtX1 | 7b5c72745831 |
| 5 | 0.18% | {\rt0 | 7b5c7274300d |
| 4 | 0.14% | {\rt\1 | 7b5c72745c31 |
| 3 | 0.11% | {\rt\ | 7b5c7274205c |
| 2 | 0.07% | {\rt?? | 7b5c72749090 |
| 2 | 0.07% | {\rtf\ | 7b5c7274665c |
| 2 | 0.07% | {\rtf9 | 7b5c72746639 |
| 2 | 0.07% | {\rta{ | 7b5c7274617b |
| 2 | 0.07% | {\rt91 | 7b5c72743931 |
| 1 | 0.04% | {\rtb3 | 7b5c72746233 |
| 1 | 0.04% | {\rt\a | 7b5c72745c61 |
| 1 | 0.04% | {\rtF1 | 7b5c72744631 |
| 1 | 0.04% | {\rt** | 7b5c72742a2a |
| 1 | 0.04% | {\rt♂i | 7b5c72740b69 |
| 1 | 0.04% | {\rTF{ | 7b5c7254467b |

Table 10: Magic header.

Figure 5, shows the distribution of the OLE2 files per day. The graph can be split up into time periods of: April 2012 to June 2012, *Word* only; July 2012 to April 2013, a mixture of *Word* and *Excel* (mainly *Excel* password).

## 3.3 RTF malarkey

In the larger sample set, there were 2,834 RTF files. On running the scan-rtf.py (Appendix I) several pieces of data were gathered.

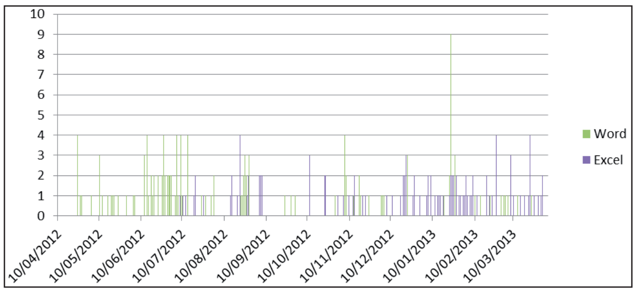


*Figure 5: Distribution of OLE2 files per day.*

### *3.3.1 Magic header*

The majority (59%) of the samples have what we would consider the correct magic header: '{\rtf1' or '0x7b5c72746631'.

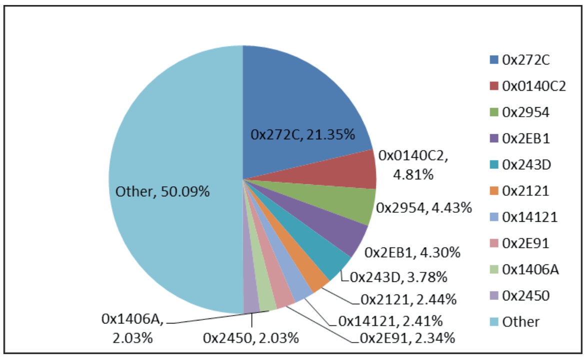


*Figure 6: Length of OLE2 object.*

As we can see from Figure 6, the majority of (mode) samples contain an OLE2 object of length 0x272c, which is also the median value. The largest object was 0x033C2F and the smallest 0x01e0.

The script attempts to parse the OLE2 file and looks for the first occurrence of 'Cobj' or '436f626a' and looks at the string (and represents it as a hex number) eight characters in. One in 10 of the files give no results for the script (for various reasons including encountering the end of the file). Of the remainder, approximately one in 43 give strange results (0x00000000 or 0xffffffff). This is either because there are multiple occurrences of 'Cobj' within the embedded OLE2 or because the OLE2 stream has been split or fragmented. Regardless of these outliers, the variation of this string is minimal, with the majority being 0x82820000.

## 3.4 OLE2 files

### *3.4.1 Non-password protected*

Looking at the remaining OLE2 files, the parsing was implemented in VDL for ease of coding.

With the smaller set of pure OLE2 files (without password protection) we find that the majority of files have the number 0x00300000, with 0x82820000 being relegated to second position. Due to the inbuilt parsing of OLE2 within the *Sophos* virus engine we do not have the issue with fragmentation. The outlier here is followed by eight nops (0x90).

The ActiveX type within the document should be one of four types (ListView, ListView2, TreeView and TreeView2).

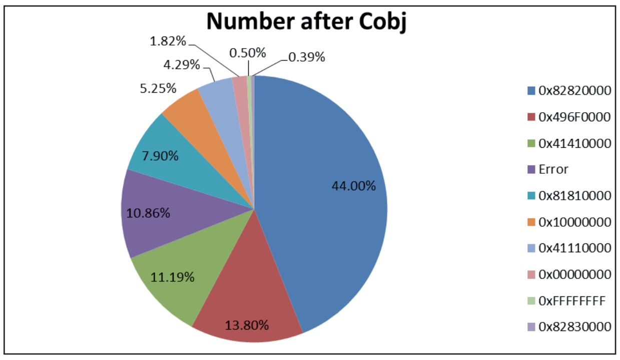


*Figure 7: Number after Cobj.*

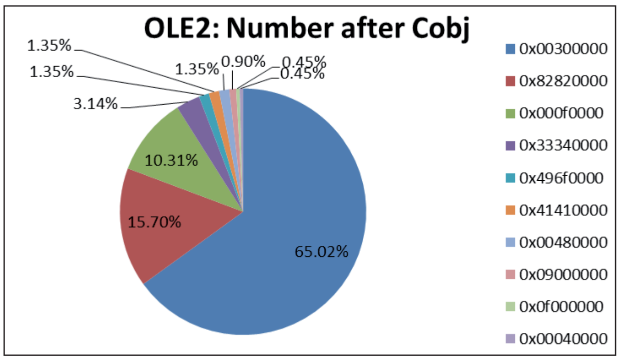


*Figure 8: OLE2: number after Cobj.*

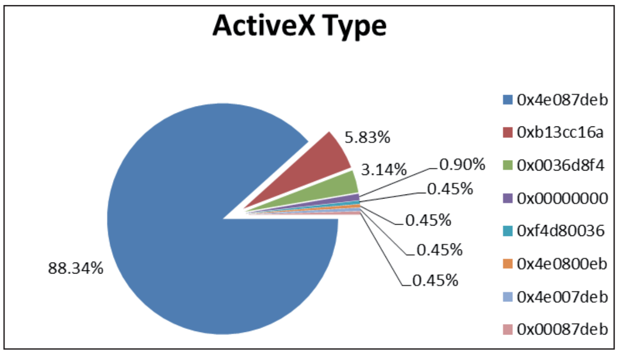


*Figure 9: ActiveX type.*

In 94.17% of the files the ActiveX type has not been fuzzed. Looking at Table 11, you can see some of the redundancy within the OLE2 file format, the files' ActiveX type should be the same in ASCII, Unicode and hex, i.e. 'TreeView' (in ASCII and Unicode) and 0xb13cc16a.

| | TreeView | ListView |
|---|---|---|
| **ASCII** | 13 | 203 |
| **ASCII percentage** | 5.83% | 91.03% |
| **Unicode** | 10 | 201 |
| **Unicode percentage** | 4.48% | 90.13% |

*Table 11: Strings within the OLE2 files.*

### 3.4.2 Password protected

Looking into the sample set, some of the files were being reported as 'Password protected file' by *Sophos* products, and as CVE-2012-0158-related by *Microsoft*. When the files were run they would either prompt for a password or run happily and malware would be installed. On querying the *Microsoft* AV team [25] on their detections, it became apparent that some of these documents had either the default write protect password of *Excel* or simple numeric passwords (see Table 12 and Figure 10).

| Password | Percentage | Keysize |
|---|---|---|
| 8861 | 8.41% | 80 |
| 3849 | 1.87% | 28 |
| 123 | 0.93% | 38 |
| 4155 | 0.93% | 28 |
| VelvetSweatshop | 87.85% | 28 |

*Table 12: Passwords vs keysize.*

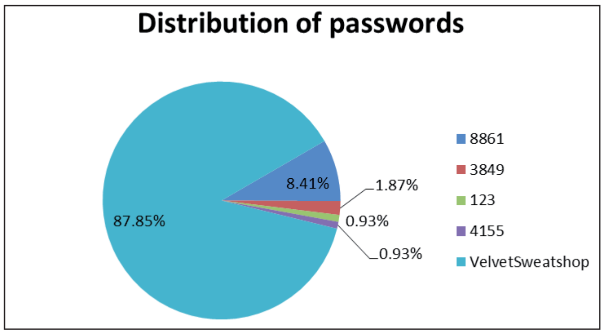


*Figure 10: Distribution of passwords.*

In fact, 91.26% of the *Excel* password-protected files were using the default password (see Figure 11).

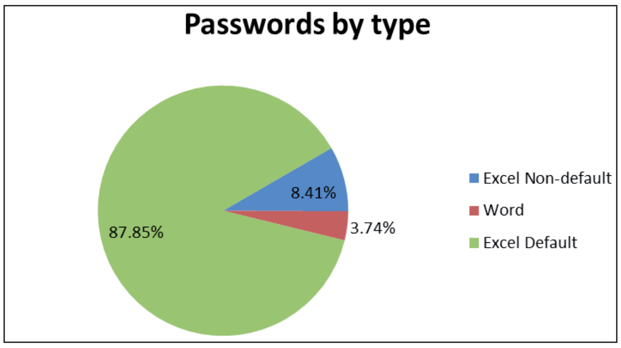


*Figure 11: Passwords by application.*

## 3.5 Case studies

### 3.5.1 Metasploit file

The module ms12_027_mscomctl_bof.rb [20] will by default create the file msf.doc[2].

| Header magic | Header magic (hex) | Length of OLE2 object | Number after Cobj |
|---|---|---|---|
| {\rtf1 | 7b5c72746631 | 0x272c | 0x82820000 |

*Table 13: Metasploit msf.doc.*

[2] bfc5437c9fe5276e4658676e02909949fc29c269.

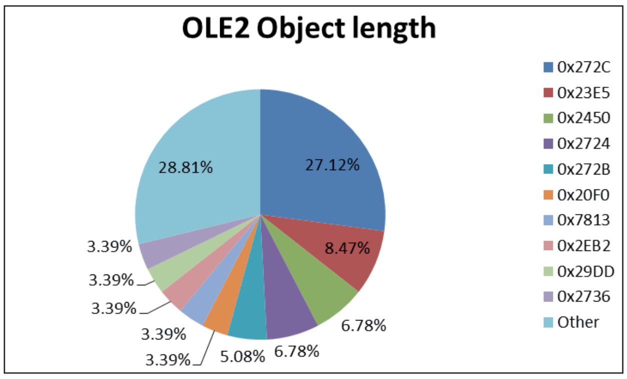


*Figure 12: Contagio object length.*

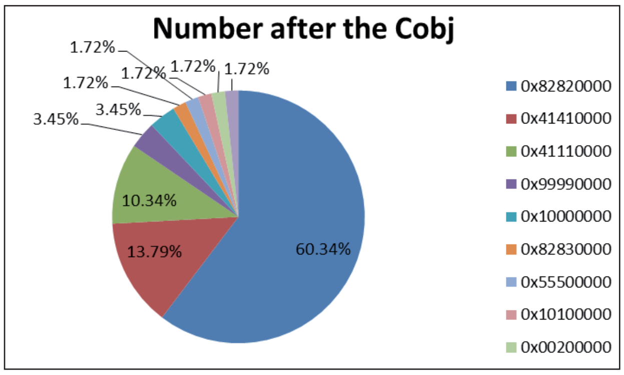


*Figure 13: Contagio number after the Cobj.*

### *3.5.2 Contagio files [26]*

Mila Parkour's *Contagio* website is a malware dump that is frequently useful for its collections of malware – often the samples hosted there are not shared elsewhere. One of the collections [26] claimed to be 90 files exploiting CVE-2012-0158, and while not all of them did exploit CVE-2012-0158, all of them were RTF files. The following results were obtained from their analysis:

| Number of samples | Percentage | ASCII header | Hex header |
|---|---|---|---|
| 38 | 64.41% | {\rtf1 | 0x7b5c72746631 |
| 5 | 8.47% | {\rtA1 | 0x7b5c72744131 |
| 3 | 5.08% | {\rtXÿ | 0x7b5c727458ff |
| 2 | 3.39% | {\rtt{ | 0x7b5c7274747b |
| 2 | 3.39% | {\rtt1 | 0x7b5c72747431 |
| 2 | 3.39% | {\rtX\ | 0x7b5c7274585c |
| 2 | 3.39% | {\rt.1 | 0x7b5c72740031 |
| 2 | 3.39% | {\rt | 0x7b5c72742020 |
| 1 | 1.69% | {\rtf{ | 0x7b5c7274667b |
| 1 | 1.69% | {\rt\1 | 0x7b5c72745c31 |
| 1 | 1.69% | {\rt## | 0x7b5c72742323 |

*Table 14: Header of RTFs in Contagio dump.*

### *3.5.3 Red October samples [27]*

The Red October attacks used several delivery methods and installed several payloads. *Kaspersky Lab* provided a list of known MD5s associated with Red October samples, exploiting CVE-2012-0158, and provided the files where they had them. Analysing them led to the following results:

| Number | Percentage | ASCII header | Hex header | OLE2 object length | Number after Cobj |
|---|---|---|---|---|---|
| 7 | 63.64% | {\rtf1 | 7b5c72746631 | 1411F | 0x496F0000 |
| 3 | 27.27% | {\rtX\ | 7b5c7274585c | 2EB1 | 0x82820000 |
| 1 | 9.09% | {\rt0{ | 7b5c7274307b | 2193 | 0x81810000 |

*Table 15: Red October sample analysis.*

## 4. CONCLUSION

Security researchers are already aware that the primary OLE2 file format readers (*Microsoft Word* and *Excel*) are tolerant of non-conformance to the technical specifications of the OLE2 format. Those same OLE2 readers appear equally tolerant of RTF non-conformance. This is worrying because even without there being vulnerabilities, all the Red October samples look to contain 'strongly illegal data' [7] – 36.36% illegal header and 72.71% non-hex data within the object.

Overall, of the RTF samples sent to *Sophos*, rejecting files based on header alone would have resulted in 40% of attacks being blocked. Most of the remaining files have non-hex characters within the object and/or other illegalities (not ending with a closing brace (0x7d) or characters outside [\x0a\x0d\x20-\x7e] [8, 10]).

The numbers after the Cobj results (Figure 7) suggest that most of the RTFs were based on the Metasploit file (or the files upon which Metasploit based their module).

Within the OLE2 sample set, the evidence (Figure 8) is that there was more knowledge of the exploit. With the addition of passwords, the OLE2 space is the more interesting, especially when you consider the question of how the bad guys generated password-protected files:

- They added a password to vulnerable files on a patched system.
- They used a third-party utility to add passwords.

The author is not currently aware of any embedded Flash malware (which uses the same streams) delivered in this way.

*Sophos* has looked for password-protected OLE2 files within RTFs and looked for the exploit embedded in *PowerPoint* but has yet to see them. This would be the next logical iteration of the exploit.

When the author asked 'Why were machines still vulnerable to CVE-2010-3333 [2]?' three reasons were postulated:

- Ignorance
- Laziness/busyness
- Non-licensed software.

To those we can add:

- Cost.

Cost: A consequence of Moore's Law is that a less-than-$500 computer becomes obsolete within 18 months, and if the computer runs without problems for 12 months it may be considered cheaper to buy a new computer than to install patches, anti-malware software etc. [28]. While intrinsically this feels wrong, for a small organization without a security-focused IT department, the total cost of maintaining software could be considered too high.

Ignorance: Security is not just a technical problem. Most APTs, (Assured Penetration Technologies) rely on human interaction. *Verizon*'s Data Breach Investigation Report contains a section entitled 'The inevitability of "the click"' [29]:

'*So how many e-mails would it take to get one click?*

…

*Running a campaign with just three e-mails gives the attacker a better than 50% chance of getting at least one click. Run that campaign twice and that probability goes up to 80%, and sending 10 phishing e-mails approaches the point where most attackers would be able to slap a "guaranteed" sticker on getting a click.*

…

*For example, a user needs to take action AND there needs to be a vulnerability on the system AND software has to be quietly installed AND there has to be a communication path back to the attacker, and, and, and …*'

For a dumb Bredo-like spam campaign, we believe that these numbers are higher by at least a factor of 10 (maybe 100). However, for a highly targeted attack, the number could easily be lower by a factor of two. So in an espionage attack, five emails may well be enough to gain access.

Laziness/busyness: When a patch becomes available it is best to apply it rapidly [30].

Non-licensed software: This is probably the biggest contributor to the longevity of this exploit. The areas where attacks are known to have worked correlate with where software licensing is an issue. On unlicensed software, patching is not a click-once-and-forget exercise, leaving the machines vulnerable. Plus, in situations where you have unlicensed OS and productivity suites, there is a high probability that the anti-malware and anti-spam software will be non-existent, out of date or of poor quality.

## REFERENCES

[1] Personal email correspondence with Microsoft employees in April 2010.

[2] Baccas, P. A time-based analysis of Rich Text Format manipulations: a deeper analysis of the RTF exploit CVE-2010-3333. http://www.sophos.com/en-us/why-sophos/our-people/technical-papers/a-time-based-analysis-of-rich-text-format-manipulations.aspx.

[3] Baccas, P. When is a password not a password? When Excel sees "VelvetSweatshop". http://nakedsecurity.sophos.com/2013/04/11/password-excel-velvet-sweatshop/.

[4] Wisniewski, C. How fast fingerprinting of OLE2 files can lead to efficient malware detection. http://nakedsecurity.sophos.com/2011/10/13/how-fast-fingerprinting-of-ole2-files-can-lead-to-efficient-malware-detection/.

[5] Python-oletools. http://www.decalage.info/python/oletools.

[6] Burke, S.M. RTF Pocket Guide, p.4. O'Reilly.

[7] Rich Text Format (RTF) Specification Version 1.9.1, p.12. Microsoft Corporation.

[8] Rich Text Format (RTF) Specification Version 1.9.1, p.7. Microsoft Corporation.

[9] Burke, S.M. RTF Pocket Guide, pp.4–5. O'Reilly.

[10] Burke, S.M. RTF Pocket Guide, p.6. O'Reilly.

[11] Burke, S.M. RTF Pocket Guide, p.9. O'Reilly.

[12] Rich Text Format (RTF) Specification Version 1.9.1, p.231. Microsoft Corporation.

[13] Rich Text Format (RTF) Specification Version 1.9.1, p.154. Microsoft Corporation.

[14] Word (.doc) Binary File Format ([MS-DOC].pdf, v20121003), p.26.

[15] Excel Binary File Format (.xls) Structure Specification ([MS-XLS].pdf, v20121003), p.47.

[16] Office Document Cryptography Structure Specification ([MS-OFFCRYPTO].pdf, v20121003), p.50.

[17] Office Document Cryptography Structure Specification ([MS-OFFCRYPTO].pdf, v20121003), p.96.

[18] Excel Binary File Format (.xls) Structure Specification ([MS-XLS].pdf, v20121003), p.64.

[19] Excel Binary File Format (.xls) Structure Specification ([MS-XLS].pdf, v20121003), p.105.

[20] MS12-027 MSCOMCTL ActiveX Buffer Overflow. http://www.metasploit.com/modules/exploit/windows/fileformat/ms12_027_mscomctl_bof.

[21] CVE-2012-0158. http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-0158.

[22] http://blog.bug.cx/2012/05/02/ms12-027-poc%E9%80%86%E5%90%91%E5%88%86%E6%9E%90/.

[23] Blasco, J. CVE-2012-1535: Adobe Flash being exploited in the wild. http://labs.alienvault.com/labs/index.php/2012/cve-2012-1535-adobe-flash-being-exploited-in-the-wild/.

[24] Flash in the Formula! http://nakedsecurity.sophos.com/2009/08/05/flash-formula/.

[25] Personal communication.

[26] CVE-2012-1535 - 7 samples and info. http://contagiodump.blogspot.co.uk/2012/08/cve-2012-1535-samples-and-info.html.

[27] "Red October". Detailed Malware Description 1. First Stage of Attack. http://www.securelist.com/en/analysis/204792265/Red_October_Detailed_Malware_Description_1_First_Stage_of_Attack.

[28] Cluley, G. German ministry replaced brand new PCs infected with Conficker worm, rather than disinfect them. http://nakedsecurity.sophos.com/2013/05/01/german-replaced-pcs-conficker.

[29] 2013 Data Breach Investigations Report. http://www.verizonenterprise.com/DBIR/2013/.

[30] Strategies to Mitigate Targeted Cyber Intrusions. http://www.dsd.gov.au/infosec/top-mitigations/top35mitigationstrategies-list.htm.

## APPENDIX I

```
# CVE-2012-0158 parsing
# Written by pob

import sys
from StringIO import StringIO
import re
import hashlib
import numpy
import binascii
def find_fpps(file,position,pattern,size):
    "doing a number of find's so create a function"
    file.seek(position)
    buff = file.read(size).upper()
        if buff.find(pattern) !=-1:
            return buff.find(pattern)
    else:
        return -1

def rex_fpps(file,position,pattern,size):
    "doing"
    file.seek(position)
    buff = file.read(size).upper()
    if re.search(pattern,buff) != None:
        return re.search(pattern,buff).start()
    else:
        return -1

def clean_buff(file,start_pos,size):
    "remove whitespace"
    file.seek(start_pos)
    buff = file.read(size).upper()
    new_buff = re.sub(r'\\OBJ[WH]\d{4,5}','',buff)
    new_buff = re.sub(r'\\{2}','',new_buff)
    new_buff = re.sub(r'\\[A-Z*]{0,10}[\d\x7d\x7e\
x5b\x7b]?','',new_buff)
    new_buff = re.sub(r'\{[^\}]*\}+','',new_buff)
    new_buff = re.sub(r'\}','',new_buff)
    new_buff = re.sub(r'[\s\x0d\x0a\x00]','',new_
buff)
    print new_buff.__len__(),
    print ",",
    return new_buff

def main():


    if len(sys.argv) < 2:
        print 'usage: rtf.py <bad.rtf>'
        return
    else:
        try:
            file = open(sys.argv[1], 'rb')
            hash =  hashlib.sha1()
            hash.update(file.read())
            filesha1 = hash.hexdigest()

        except Exception:
            print '[ERROR] CAN NOT OPEN FILE'
            return

        magic = find_fpps(file,0,'\x7b\x5c',16)

        if magic == None:
            print 'RTF Signature not found'
            return
        print 'RTF, ',
        print filesha1 + ",",
        buff = file.read()
        file.seek(0)
        header = file.read(6)
        print header + ",",
        print header.encode("hex") + ",",
        docfile = rex_fpps(file,0,r'(?:\b|[0-9])D\s*?
(?:\\[^0]*?)?0[^C]*?C[^F]*?F[^1]*?1[^1]*?1[^E]*?E[^
0]*?0',-1)
        if docfile == None:
            print "DOCFILE not found"
            return
        print 'DOCFILE,',

        boundcheck = rex_fpps(file,docfile,'\x7d\s*\
x7d\s*\x7d',-1)
        if boundcheck == -1:
            boundcheck = len(buff)
            print len(buff) + ",",
        print boundcheck -1,
        print ",",
        newbuff = clean_buff(file,docfile,boundcheck)

        docfile2 = newbuff.find(r'D0CF11E0')
        newbuff = newbuff[docfile2:]


        pos = newbuff.find(r'436F626A') + 16
        print 'yes,',
        print newbuff[pos:pos+8] + ",",
        newbuff = newbuff.rstrip()
        if len(newbuff) % 2 == 1:
            newbuff = newbuff + '0'
        hash.update(newbuff)
        print hash.hexdigest()
#       fname = '%40s.bin' % hash.hexdigest()
#       open(fname, 'wb').write(newbuff.lower().
decode('hex'))

if __name__ == '__main__':
    main()
```

## APPENDIX II

```
import os
import sys
from thirdparty.OleFileIO_PL import OleFileIO_PL

def walk(dir):
    for path, subdirs, files in os.walk(dir):
        for file in files:
          file = os.path.join(path, file)
          print "Checking ..." + file
          checktype(file)

def searchlistdir(file,pattern):
    ole = OleFileIO_PL.OleFileIO(file)
    index = ole.listdir()
    if ole.exists(pattern):
        return 1
    for ind in index:
        if pattern in ind:
          return 1

def checktype(path):
    file = open(path, 'rb')
    file.seek(0)
    buff = file.read(16)
    if (buff.find('\xd0\xcf') != -1):
        assert OleFileIO_PL.isOleFile(path)
        try:
         ole = OleFileIO_PL.OleFileIO(path)
         if ole.exists('Workbook'):
            if ole.exists('encryption'):
                    print "Excel pass: " + path
            elif searchlistdir(path,'Ctls') == 1:
                    print "Excel Ctls: " + path
            else:
                    print "Excel: " + path

         elif ole.exists('Worddocument'):
            if ole.exists('encryption'):
                    print "Word pass: " + path
            elif searchlistdir(path,'Contents') == 1:
                    print "Word Contents: " + path
            else:
                    print "Word: " + path

         else:
            print "OLE2: " + path
       except (IOError, RuntimeError):
         print "Exception OLE2: " + path
   elif (buff.find('\x7b\x5c') != -1):
       print "RTF: " + path
   else:
       print "Neither: " + path

if __name__=='__main__':
   if len(sys.argv) != 2:
       print "Usage: tree.py dir"
       sys.exit(100)
   walk(sys.argv[1])
```