DECEMBER 2011

virus BULLETIN

Fighting malware and spam

CONTENTS

IN THIS ISSUE

NEW AND IMPROVED INDUC

The Induc virus has been spreading successfully around the world since its first appearance in 2009, but back then it didn't contain a malicious payload. The latest variant contains a genuinely malicious payload and additional file-infecting and propagation capabilities. Robert Lipovský has the details.

page 4

KNOCKING AT HEAVEN'S GATE

'Heaven's Gate' is an undocumented feature used by the 32-bit Windows environment when running on 64-bit versions of Windows, which allows for the transition between 32-bit and 64-bit code. In August 2011, we saw the first virus to make use of it. Peter Ferrie takes a close look at W32/W64.Sobelow.

page 9

BOOK CLUB

In 'Worm: The First Digital World War', Mark Bowden writes about the team who worked together to combat the Conficker worm, focusing on some of the principal players in the Conficker Working Group and on their stories over the nine months of Conficker's activity. Paul Baccas reviews the book.

page 17

ISSN 1749-7027

vb

'Attribution is one of the things in the IT security industry that is dropped on the floor.'

Anon

ALL YOUR LULZ WILL BELONG TO US

Attackers read – we pay attention. The recent US DoD Cyberspace Policy Report scoped out a number of challenges in defending critical assets and infrastructure. In it were things we have known about for a long time that apply to the private realm, but they hold true even for the best-funded power structure in the world: attribution is a pain – hiding behind the veil of anonymity on the net is powerful, and attackers have the advantage.

We agree. The report not only admitted that attribution is a major difficulty even for the government's well-funded structure, but that addressing it properly will require years of R&D investment. It is a problem that is not going to go away any time soon. We already know that well-rehearsed attackers have an advantage over defenders. Looking at what pass as 'Advanced' attack tools nowadays, one would know that the advantage is generally not in the complexity of our technology. Instead, the advantage is in our coordination and craft – information collected from social networks, current events, conferences, meetings, travel, your friends and colleagues. This is a game of finding the weakest link and pounding it.

We thrive in the shadows. Attribution is one of the things in the IT security industry that is dropped on the floor. The data is accessible. The techniques to root us out are (for the most part) available, or could be. ISPs often choose not to cooperate with the security community, partly because it's easier to abide by particular sections of their contractual obligations, partly because they don't have the resources or understand the impact of the problems, and partly because some are making money on our side. Even legitimate ISPs and DNS registrars maintain odd boundaries. On the one hand, you've got ISPs testing 'deep packet manipulation' on unknowing users, and on the other, you've got researchers investigating contract breaches, clearly abused IP and domain resources, and the ISPs refusing to provide details until they are subpoenaed by under-resourced law enforcement contacts. We like that.

And then there are the myriad law enforcement problems across international boundaries. But now, the FBI, DHS, and other countries are cooperating further with researchers and local LE around the world – take, for example, the almost half-decade effort that culminated in Operation Ghost Click. But we'll see if the extraditions complete.

It takes years of evidence gathering to build an overwhelming case against cybercriminals and nation state actors, and only those cases that have certain, demonstrably concrete value can be taken on – this is good for us. We dread organization, cooperation and transparency on the part of the security industry and we dislike research efforts like the Kelihos botnet takeover, agreements like strong data breach laws, the Budapest convention and the recent ITU-Impact work. And the mistakes we make.

We continue to loot as we always have done: PII, CC, intellectual property, direct transfers of hard-earned cash, the results of research and investment and years of negotiations. For us it is catastrophic that these incidents are no longer hidden away under NDAs, because an informed public can be a powerful public. Damn you *Google* and your Aurora disclosure, *RSA* disclosure, and SEC disclosure guidance! Damn you, the possibility of federal breach notification law for private and public organizations! Our darkest corners are being lit.

Looking to the future, the possibilities for us to exploit big data stores are limitless. *Berico* recently highlighted architectural security concerns for Hadoop and big data implementations at federal data centres. It pleases us to know that data encryption carries with it many challenges, even today. And the possibilities to exploit mobile and 'smart' technologies are growing. While *Android* malware is on the increase, for the most part, the malware itself is immature – much like the adware markets of 2005. Our adware groups morphed into crimeware efforts, and even as *Windows*, Java, *Adobe Reader* and *Flash* are further hardened, we have continued to build our profits attacking these platforms with darker crimeware.

We are cybercrime and cyber espionage. And we make mistakes.

NEWS

HAPPY HOLIDAYS

The members of the *VB* team extend their warm wishes to all *Virus Bulletin* readers for a very happy holiday season and a healthy, peaceful, safe and prosperous new year.

Clockwise from top left: Helen Martin, John Hawes, Martijn Grooten, Simon Bates, Allison Sketchley, Paul Hettler.

CALL FOR PAPERS: VB2012 DALLAS

Virus Bulletin is seeking submissions from those wishing to present papers at VB2012, which will take place 26–28 September 2012 at the Fairmont Dallas hotel, Dallas, TX, USA.

The conference will include a programme of 30-minute presentations running in two concurrent streams: Technical and Corporate. Submissions are invited on all subjects relevant to anti-malware, anti-spam and related fields. In particular, VB welcomes the submission of papers that will provide delegates with ideas, advice and/or practical techniques, and encourages presentations that include practical demonstrations of techniques or new technologies.

The deadline for submission of proposals is Friday 9 March 2012. Abstracts should be submitted via the online abstract submission system at http://www.virusbtn.com/conference/abstracts/.

Full details of the call for papers, including a list of topics suggested by the attendees of VB2011, can be found at http://www.virusbtn.com/conference/vb2012/call/.

Any queries should be addressed to editor@virusbtn.com.

Prevalence Table – October 2011[1]

Malware	Type	%
Autorun	Worm	8.57%
Encrypted/Obfuscated	Misc	6.84%
LNK-Exploit	Exploit	5.01%
Sality	Virus	4.96%
Heuristic/generic	Virus/worm	4.65%
Adware-misc	Adware	4.58%
Zbot	Trojan	3.55%
Iframe-Exploit	Exploit	3.24%
Conficker/Downadup	Worm	3.04%
Crack/Keygen	PU	2.99%
Virut	Virus	2.97%
Agent	Trojan	2.50%
Cycbot	Trojan	2.35%
Delf	Trojan	2.31%
Downloader-misc	Trojan	2.21%
Heuristic/generic	Trojan	2.20%
Slugin	Virus	2.11%
AutoIt	Trojan	2.05%
VB	Worm	1.71%
Freeware-downloader	PU	1.70%
RogueSoftware-misc	Rogue	1.64%
Virtumonde/Vundo	Trojan	1.64%
Dorkbot	Worm	1.49%
Dropper-misc	Trojan	1.33%
Bifrose/Pakes	Trojan	1.32%
FakeAlert/Renos	Rogue	1.29%
BHO/Toolbar-misc	Adware	1.28%
Exploit-misc	Exploit	1.19%
Vobfus	Trojan	1.17%
Rogue-Registryfix	Rogue	1.08%
Kryptik	Trojan	1.06%
Crypt	Trojan	0.93%
Others[2]		15.11%
Total		100.00%

[1]Figures compiled from desktop-level detections.

[2]Readers are reminded that a complete listing is posted at http://www.virusbtn.com/Prevalence/.

MALWARE ANALYSIS 1

WIN32/INDUC.C: GETTING NOISIER IN THE LIBRARY

Robert Lipovský
ESET, Slovakia

The Delphi infector Win32/Induc is back, this time with a genuinely malicious payload and additional file-infecting and propagation capabilities.

Classic file-infecting viruses are not as common as they were 15 years ago, but occasionally we do come across a 'modern virus'. A good example of such a menace is the polymorphic file infector Win32/Sality [1].

In 2009, the virus Win32/Induc.A [2], which infected Delphi files at compile-time, caught our attention [3]. Apart from the infection process itself, this variant did not contain any other malicious payload and was considered a proof of concept, or code in development (although it did actually infect real systems). One of its interesting features [4] was that it appeared to have been influenced by a classic 1984 paper that describes an infection implemented by planting a 'bug' into a C compiler [5].

About two years later, in July 2011, a new version appeared. Win32/Induc.B featured some minor improvements, but was actually quite similar to the first variant. However, more dramatic changes appeared in August in the latest development of the virus, Win32/Induc.C. In this variant the infecting code had been modified and extended, and more malicious functionality had been added.

In this article, we will first analyse Win32/Induc.C – the most complex variant to date. Afterwards, we will briefly describe Induc's evolution by outlining the differences between this and the earlier versions of the virus.

ANALYSIS OF WIN32/INDUC.C

The fundamental characteristic of the Induc virus is that it infects a standard Delphi library, resulting in the infection of every application compiled in this modified Delphi development environment. In the case of Win32/Induc.C, the virus body (which the Delphi linker includes in all programs) is 52,736 bytes. (This does not include the malicious code inside the infected Delphi library, which drops and launches the virus body.)

Delphi library infection

Before analysing the virus body, let's take a look at how the Delphi infection works. Delphi applications link to the file SysInit.dcu. This is a component of the Delphi/Kylix Cross-Platform Runtime Library, specifically the System Initialization Unit. As the name suggests, code from this library is included in the initialization part of every Delphi application, and this is the target of the Induc infection. SysInit.dcu contains object code (.dcu is a Delphi Compiled Unit) compiled from SysInit.pas. Induc modifies this source file and compiles it into the resulting SysInit.dcu file. (The virus actually compiles SysInit.dcu 'indirectly' by compiling System.dcu.)

```
{$IFDEF MSWINDOWS}
{$I Defines.inc}

procedure _InitExe(InitTable: Pointer);
begin
  TlsIndex := 0;
  HInstance := GetModuleHandle(nil);
  Module.Instance := HInstance;
  Module.CodeInstance := 0;
  Module.DataInstance := 0;
  InitializeModule;
  CreateMyFile(@my_array,sizeof(my_array),'~.exe');
_StartExe(InitTable, @Module);
end;
{$ENDIF MSWINDOWS}
```

Figure 1: Malicious code is inserted into SysInit.pas.

Only two lines of code are added to SysInit.pas, as the rest is taken care of by the file Defines.inc, which is dropped by Induc. The malicious CreateMyFile function (implemented in Defines.inc) is called in _InitExe just before the _StartExe function call.

The method for launching the virus body is quite trivial. Defines.inc contains the virus body as a PE file as an array of 52,736 bytes and the CreateMyFile function simply drops the executable to the current directory with the name '~.exe' and executes it.

```
const my_array:array [0..52735] of byte=(77,90,80,0,2
,0,0,0,0,64,0,26,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
```

Figure 2: The virus body stored as an array of 52,736 bytes in Defines.inc.

Second file infection method

Win32/Induc.C features a method not present in the previous variants for infecting *any* .exe file, not just those compiled with Delphi. In this case, Induc acts as a prepender virus – upon infection it attaches the original executable under its body. The string '-=supernatural=-' is used as a delimiter to mark the beginning of the original file, as well as a marker to signify that a file has been infected.

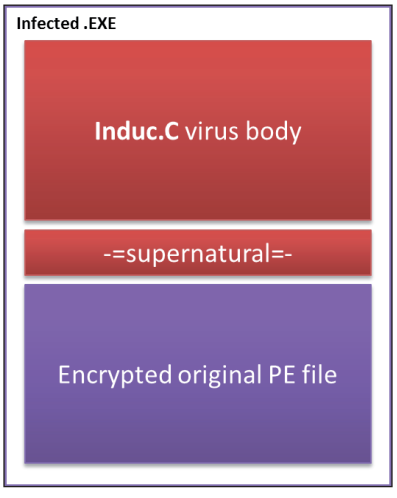


Figure 3: Induc's .exe file infection.

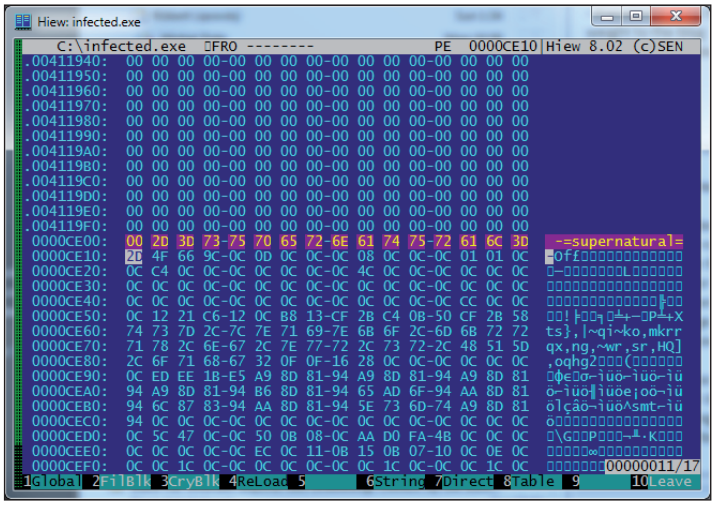


Figure 4: Executable infected with Win32/Induc.C.

Induc.C also uses a simple encryption algorithm for the original executable: xor 5, add 7. Our analysis (described in the next section) revealed that this second .exe infection vector is only used for infecting executables on removable drives. This fulfils the purpose of distributing the virus to other computers.

Virus body analysis

The malware code begins with a procedure containing some curious API function calls. There are a couple of multimedia functions (MCI Functions and PlaySoundA) about whose purpose we can only speculate. The virus executable tries to play a sound from its resources, called 'my_global_sound', and if it's successful, the virus terminates its execution. The execution doesn't take this path under normal circumstances, obviously, and the virus samples that we have analysed are fully functional. The first possible explanation that comes to mind is anti-emulation, or a way for the virus to defend itself from sandboxes. Another possibility is some residual debugging (or plain junk) code used by the author.

The main payload function is shown in Figure 5.

At the beginning of the function, there is a check as to whether the executable has been launched with the -autorun parameter, and one of two execution paths is chosen. The first execution of the virus is without the parameter. Two tasks are performed in this program branch:

- The virus 'schedules' itself to run with the -autorun parameter after the next system restart. It does so by copying itself to the Application Data\APMV\ directory with APMV.exe as its filename. Then a shortcut (.lnk file) is crafted, pointing to this file with the -autorun parameter, and placed in the Startup folder.
- Next, Induc.C checks whether there is another executable in the overlay (the original executable that has been infected, as explained in the previous section). The virus searches for the string '-=supernatural=-' and then decrypts (sub 7, xor 5) the original PE file if it is present, and drops it into the current directory as ~.exe. Afterwards, Induc creates a shortcut to this file (~.lnk), executes it, deletes the shortcut, and when the original executable terminates, the ~.exe file is deleted as well. (This is accomplished by calling the DeleteFile API in a cycle until it is successful.)

When executed with the -autorun parameter, the main payload is delivered: infecting Delphi, infecting executables on removable drives, and downloading other malware onto the system. There is even a very simple self-defence thread. Let's take a closer look at these functions:

- There is one shared function for going through the directory structure that is used when infecting both Delphi and .exe files. Which task to carry out is determined by the value in the AL register. At the beginning of the function, there is a call to the GetLogicalDriveStrings API to enumerate the drives on the system and, for drives that satisfy specific conditions, Induc.C searches the directory structure for files to infect. When infecting Delphi, the drive type DRIVE_CDROM is excluded, and the System

Volume Information folder must be present. For the .exe infections, the drive types must not be DRIVE_NO_ROOT_DIR, DRIVE_CDROM, or DRIVE_UNKNOWN and the System Volume Information folder must not be present. (USB sticks, for example, satisfy these conditions.) It is interesting that the author didn't use the DRIVE_REMOVABLE drive type for this purpose.

- The directory structure is traversed recursively using the standard method of FindFirstFile and FindNextFile.

In order to locate the Delphi installation folder, Induc.C searches for the following file paths:

bin\dcc32.exe

lib\sysinit.dcu

rtl\sys\system.pas

When they are found, the Defines.inc file is dropped, and the virus writes two lines to the rtl\sys\sysinit.pas source file:

```
{$I Defines.inc}
CreateMyFile(@my_array,sizeof(my_array),'~.exe');
```

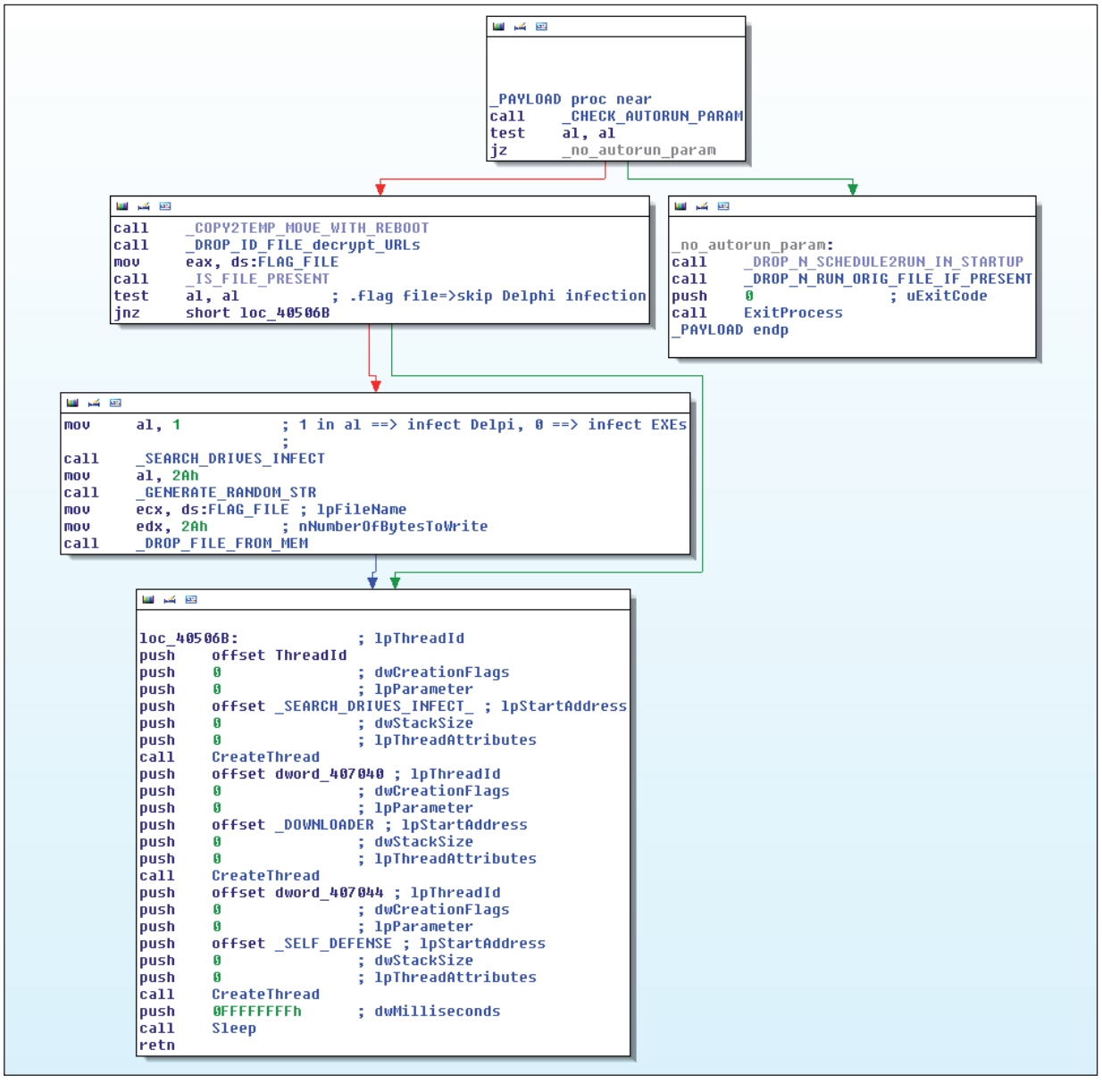


Figure 5: Main payload function of Induc.C.

This is explained in the 'Delphi library infection' section.

When the malicious modifications are made, Induc compiles the libraries using the following command:

```
%Delphi_path%\bin\dcc32.exe -Q "%Delphi_path%\
rtl\sys\system.pas" -M -Y -Z -$D- -0
```

Afterwards the compiled .dcu files are moved to the correct directory and the SysInit.pas source file reverts to its original form.

- The infection of .exe files is implemented in a separate thread and repeats in a cycle every five seconds – so that when a USB stick is inserted into the computer, it will become infected. When traversing the directory structure, files with the .exe extension are first checked to see whether they have been infected already (by searching for the '-=supernatural=-' string). The second condition for infection is that the file size must be between 100KB and 50MB.

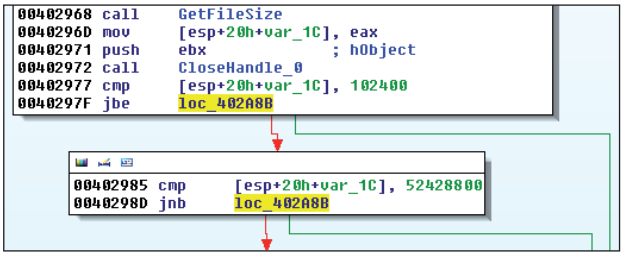


Figure 6: File size condition for infection.

As described earlier, the virus appends the original executable to its body after encrypting it with xor 5, add 7. The virus also takes the icon of the executable.

- Another thread is dedicated to the downloader feature – the main payload of Induc.C. We'll look at it more closely in the next section, as the techniques used are quite unusual.
- The last of the three CreateThread calls implements a very primitive self-defence mechanism. Every 0.5 seconds, the virus checks whether the Task Manager is running, and if it is, the virus terminates itself. This way, the user will not see the malicious process in the list of running processes. The method is very simple, and the author could instead have chosen rootkit techniques to hide the virus process from process listing. However, avoiding the use of rootkit techniques allows the malware to look a lot less suspicious and since this is a virus and will probably be executed again, this trick works (almost) as well. The API functions used for the process enumeration are the standard CreateToolhelp32Snapshot, Process32First, etc.

The downloader

The downloader thread contains a loop and a 15-minute sleep before it tries to download files from the Internet. The virus body contains three hard-coded URLs, which are also encrypted using the simple xor 5, add 7 'cipher'. The same encryption is also used for obfuscating the names of WinINet functions. Now things get *really* interesting. The URLs point to JPEG images – user avatars on three different discussion forums.

You might be wondering: why on earth is a virus downloading avatars? This is a trick that the virus author uses to dynamically store the URLs of other malware to download and execute. If we take a look inside the JPEG file, this is what we see:

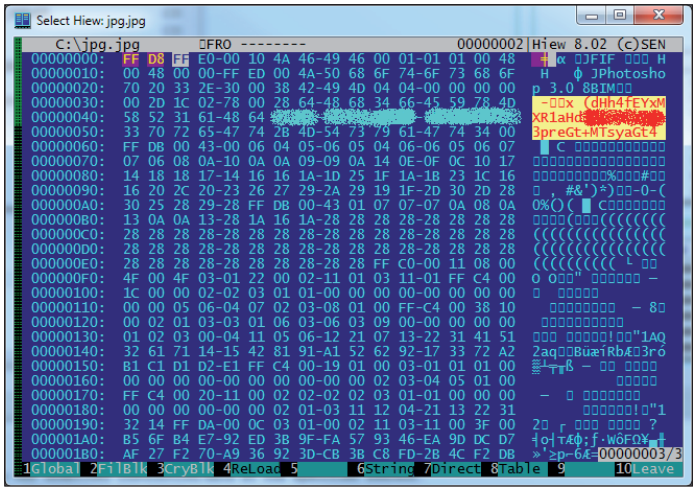


Figure 7: Encrypted URL hidden in downloaded image.

An encrypted URL is stored at the beginning of the JPEG file (where EXIF data is stored). The start of this field is indicated by the 'x' character. Following that is a WORD value which contains the length of the encrypted URL, and then the URL itself. The encryption algorithm is – you guessed it – xor 5, add 7, but the string is also encoded with Base64.

The virus then downloads the executable at that address to a randomly named file in the Temp directory and executes it, deleting it after it terminates. One particular piece of malware that Induc.C downloads is a password stealer that *ESET* detects as Win32/PSW.Delf.NQS. This has the capability to extract passwords from various applications, including FTP clients.

A couple of additional details on the downloader procedure:

- In the main cycle with the 15-minute delay, the virus attempts to download a file from the first URL and tries the other URLs if that one fails.

- Before downloading the JPEG file, Induc first checks its timestamp and compares it with the one it has recorded before. This way, the virus avoids downloading the same malware repeatedly.
- Induc verifies that the downloaded avatar is really a JPEG file:

```
CODE:00402121                cmp     byte ptr [eax], 0FFh
CODE:00402124                jnz     short loc_402154
CODE:00402126                cmp     byte ptr [eax+1], 0D8h
CODE:0040212A                jnz     short loc_402154
```

Figure 8: JPEG file type check.

- When downloading the avatar, the following URL parameter string is added to the HTTP request:

  ```
  ?uid=%id%&l=%random%
  ```

  This simple feature enables the virus operator to track the infected computers – effectively creating a botnet.

Induc.C uses three supporting files:

- %virus_filename%.id – stores a (random) ID of the infected computer
- %virus_filename%.dat – stores an encrypted timestamp of the avatar
- %virus_filename%.flag – marks that Delphi has already been infected.

DIFFERENCES IN INDUC.A

There are several major differences between the first variant of the Induc virus and its latest iteration:

- The Delphi infection didn't take place in the SysInit.pas library, but in SysConst.pas.
- Delphi versions 4 to 7 were affected.
- The Delphi installation directory was read from the *Windows* registry, not by searching the hard drive.
- All the virus code was in plain Delphi and was written to the infected SysConst.pas, and was clearly visible for analysis (after some beautification).
- There was no payload at all, apart from the Delphi infection itself. Induc.A didn't contain the .exe infection capability introduced by Induc.C.

Even though Induc.A received a lot of attention two years ago, it is now apparent that it was only an Alpha version of the virus.

DIFFERENCES IN INDUC.B

Induc.B appears to have been an improved version of Induc.A. The functionality was the same as before – there was no payload, only the infection of Delphi through SysConst.pas. However, a few anti-debugging tricks were added in this version, and the author made the code slightly harder to analyse by encrypting it. Some unused functions were also present – apparently the virus writer was experimenting.

CONCLUSION

The Induc virus has been spreading successfully around the world since its first appearance in 2009 [4], even though at that time it did not contain a malicious payload. Now, however, the author appears to have passed the Alpha testing stage and the virus poses a real threat to computer users – even though it is not polymorphic and its code is rather simple. Following the trend of modern malware, it acts as a vector to download and execute more malicious code on the infected system, and incorporates botnet capabilities.

I wonder what Induc.D will look like…

(*ESET*'s detection statistics for the Induc family can be found in [6] and a technical write-up of Induc.C is given in [7].)

REFERENCES

[1] ESET. Global Threat Report August 2011. http://go.eset.com/us/resources/threat-trends/Global_Threat_Trends_August_2011.pdf.

[2] ESET Threat Encyclopaedia: Win32/Induc.A. http://www.eset.eu/encyclopaedia/win32-induc-a-virus?lng=en.

[3] Abrams, R. The Retro Virus. ESET Threat Blog August 2009. http://blog.eset.com/2009/08/19/the-retro-virus.

[4] Bortnik, S.; Borghello, C.; Harley, D. W32/Induc.A FAQ. ESET Threat Blog. August 2009. http://blog.eset.com/2009/08/23/w32induc-a-faq.

[5] Thompson, K. Reflections on Trusting Trust (1984). Communications of the ACM Vol. 27, No. 8.

[6] Lipovský, R. The Induc Virus is back! ESET Threat Blog. September 2011. http://blog.eset.com/2011/09/14/the-induc-virus-is-back.

[7] ESET Threat Encyclopaedia: Win32/Induc.C. http://www.eset.eu/encyclopaedia/win32-induc-c-virus-lg-2?lng=en.

MALWARE ANALYSIS 2

AS ABOVE, SOBELOW

Peter Ferrie
Microsoft USA

In June 2009, an interesting article describing 'Heaven's Gate' appeared on a popular VX website. This is an undocumented feature used by the 32-bit *Windows* environment when running on 64-bit versions of *Windows*, which allows for the transition between 32-bit and 64-bit code. In August 2011, we saw the first virus to make use of it: W32/W64.Sobelow.

32-BIT PLATFORM

In infected 32-bit files, the virus begins by identifying the platform. It does this by checking the value of the GS selector. On 32-bit versions of *Windows*, the value of the GS selector is always zero and in this case the virus immediately passes control to the host. On 64-bit versions, the GS selector is always non-zero. If a 64-bit version of *Windows* is detected, the virus aligns the stack to a multiple of eight bytes, if necessary (because the 64-bit environment requires a 64-bit aligned stack, just as the 32-bit environment requires a 32-bit aligned stack). Then the virus uses a tricky method to jump to 64-bit mode.

A jump to 64-bit mode requires the 'magic' 64-bit gate selector ('Heaven's Gate') and the offset that serves as the location from which to resume execution. Since the size of the selector is only 16 bits, that leaves 16 bits unused on a 32-bit platform. The virus takes advantage of this with a single instruction that combines two required elements. The instruction pushes a value onto the stack that corresponds to the selector, and also carries the opcode for a far return instruction. The virus then performs a near call to that far return instruction (which implicitly saves the offset on the stack). The far return passes through the gate, causing the switch to 64-bit mode, and resumes execution at the first instruction after the call instruction – all in a completely position-independent manner. This is a very efficient way to do it.

CURRENT DIRECTORY

Now the virus is in 64-bit mode, but there is still one small change that must be made to the environment to make it usable: the current directory must be set. On 64-bit versions of *Windows*, the 32-bit environment exists as a sub-environment of the 64-bit environment, and a transition to the kernel often requires a traversal through the 64-bit environment to get there. As a result, many of the common data structures exist in two places: one used by the 32-bit mode, and the other used by the 64-bit mode. The current directory is one of these – but since the 32-bit environment can query and set the current directory without invoking the kernel, the 64-bit version is unused in normal circumstances.

Though it is unused in 32-bit mode, the 64-bit version of the current directory would be used in 64-bit mode if related 64-bit APIs were called, and it is not undefined – it is always set to '%windir%'. However, using this directory would pose a problem for the virus because '%windir%' is no longer a great location for finding files to infect. Furthermore, there is no 64-bit version of kernel32.dll in this mode (and it cannot be loaded), so there is no typical user-mode API to change it. In any case, the 32-bit version of the API alters the 32-bit data structure. The 64-bit version of the API is available only to native 64-bit applications. Even if the API were available, the virus would need to know its current location from the 32-bit mode in order to make it the same in the 64-bit mode, which would normally require calling an API from the 32-bit mode.

Instead of all of that, the virus digs straight into the RTL_USER_PROCESS_PARAMETERS structure for the 32-bit and 64-bit modes, thus avoiding the need for any APIs at all. This structure holds the pointer to the current directory. However, instead of simply copying from one to the other, as we might expect, the virus heads straight to an undocumented location nearby and copies the pointer there instead. It turns out that this undocumented location is used by the ntdll RtlDosPathNameToRelativeNtPathName_U() function while resolving the current directory, despite the existence of the official location. This can lead to interesting results if they somehow become unsynchronized – a query of the current directory might return a location other than the actual current directory, along with all of the mischief that implies.

At this point, we reach the start of the 'really' native 64-bit code, and the entrypoint for an infected 64-bit file.

64-BIT PLATFORM

In an infected 64-bit file, the first instruction constructs a pointer to the host entrypoint for execution later. In an infected 32-bit file, the first instruction at this location constructs a pointer to the same far return instruction that was used to enter 64-bit mode. When called from 64-bit mode with the proper parameters (the regular 32-bit code selector and the offset that serves as the location from which to resume execution, as before), it can be used to leave 64-bit mode.

The next instruction saves a value from an undocumented location in memory. In this case, the value corresponds to

the stack pointer in the 64-bit context structure that is active when an exception occurs. Whenever an exception occurs, the context is filled in to allow it to resume afterwards, and a mode switch occurs if necessary. However, in order to prevent recursive calls into the code that performs a mode transition, the value in the undocumented stack location is set to zero.

Since the virus uses exceptions intentionally and intercepts them when they occur, on return to the host, the value in that location would always be zero. If an exception occurred later in the host code (intentional or not), then the exception handler dispatcher in *Windows* would see that the saved stack pointer was invalid and terminate the application. The virus makes sure this value is restored before transferring control to the host.

IMPORT-ANT DETAILS

The virus resolves the address of ntdll.dll by walking some data structures in a way that is compatible with *Windows 7*. As noted above, the 64-bit version of kernel32.dll is not available in this mode, so the virus restricts itself to functions that are available from ntdll.dll, to the extent that no other DLLs are loaded. This includes the System File Checker DLL, so the virus is not able to avoid protected files that meet the infection criteria.

The virus uses the CRC32 method to avoid the need to store the strings, and stores the results onto the stack. The checksums are sorted alphabetically according to the strings they represent, allowing the virus to perform a single pass of the export table in order to resolve all of the APIs required. After resolving the APIs, the virus begins a search for files. The virus searches for files in the current directory and all subdirectories, using a linked list instead of a recursive function. It examines every file that is found, regardless of its extension.

The virus author likes to combine many push instructions for multiple API calls, perhaps as some kind of optimization, but perhaps just to make the code more difficult to analyse. Fortunately, due to some details related to the calling convention that *Microsoft* chose to implement on the 64-bit platform, this is mostly no longer possible. Despite that, we still have the following monstrosity:

```
push    rdi
push    rsi
push    rax
push    rcx
push    rsp
pop     rax
push    rbp
push    rbp
push    rsp
pop     rdi
push    rbp
push    rbp
push    rbp
push    rax
push    ...
push    OBJECT_ATTRIBUTES_size
push    rsp
pop     rsi
push    ...
push    rbp
sub     esp, 20h
mov     r9, rsp
mov     r8, rsi
mov     edx, FILE_WRITE_ATTRIBUTES | SYNCHRONIZE
push    rdi
pop     rcx
```

TOUCH AND GO

When a file is found that meets the infection criteria, it will be infected. Files are considered candidates for infection if they are *Windows* Portable Executable (PE) format, character mode or GUI applications for the *Intel 386+* CPU or the *AMD64*-compatible CPU, with a non-zero entrypoint if they are DLLs. The files must have no digital certificates, and they must have no bytes outside of the image. The latter condition is the infection marker. Interestingly, despite its reliance on exceptions during the infection process, the virus does not check that exceptions are allowed by the host – the NO_SEH (No Structured Exception Handling) flag is not cleared in the header. If the flag is not cleared, then *Windows* will terminate the application at the moment that an exception occurs.

INFECTION

When infecting a file, the virus removes the read-only attribute, if it is present. The virus resizes the file by a random amount in the range of 4–6KB in addition to the size of the virus. This additional data will exist outside of the image, and serve as the infection marker. The virus registers a Vectored Exception Handler to protect against problems, which also intercepts the end of infection exception.

If relocation data is present at the end of the file, the virus will move the data to a larger offset in the file, and place its code in the gap that has been created. If no relocation

data is present at the end of the file, the virus code will be placed here. The virus checks for the presence of relocation data by checking a flag in the PE header. However, this method is unreliable because *Windows* ignores the flag, and relies instead on the base relocation table data directory entry.

The virus increases the physical size of the last section of the file by the size of the virus code, then aligns the result. If the virtual size of the last section is smaller than its new physical size, then the virus sets the virtual size to be equal to the physical size, and increases and aligns the size of the image to compensate for the change.

It also changes the attributes of the last section to include the executable and writable bits. The executable bit is set in order to allow the program to run if DEP is enabled, and the writable bit is set because the virus needs to save the current stack pointer in its body in order to intercept exceptions.

The virus alters the host entrypoint to point to the last section, to the code that is appropriate to the file format. The virus saves the difference between the current entrypoint and the original entrypoint. This allows a transfer of control in a position-independent manner, and allows it to work on files that have Address Space Layout Randomization enabled.

When the infection routine has finished, the virus recalculates the checksum, if necessary, and then forces an exception to occur. This is an elegant way to reduce the code size, in addition to functioning as an effective anti-debugging method. Since the virus has protected itself against errors by installing a Vectored Exception Handler, the simulation of an error condition results in the execution of a common block of code to exit a routine. This avoids the need for separate handlers for successful and unsuccessful code completion. The common code restores the file date and time, and the read-only attribute if it was set previously. Then the virus searches for another file to infect.

When there are no more files to be found, the virus restores the stack pointer and undocumented value, and then returns to the saved entrypoint. For infected 32-bit files, the stack is further restored to account for any alignment that was performed originally.

THE GATE TO ...

Code that passes through 'Heaven's Gate' is currently immune to most, if not all, anti-malware emulators, but the act of using the gate in this way is suspicious in itself. Perhaps that will be enough to stop the technique before it becomes widespread.

FEATURE

MOBILE BOTNETS FOR SMARTPHONES: AN UNFOLDING CATASTROPHE?[1]

Hasan Ijaz, Muddassar Farooq
nexGIN RC, Pakistan

Syed Ali Khayam
NUST, Pakistan

The number of users subscribing to the voice, Internet and messaging services of cellular networks is increasing exponentially worldwide. The development of cellular botnets, therefore, poses a serious threat because of their potential to incapacitate and bring down cellular networks. These bots may launch SMS floods leading to DoS attacks, carry out identity theft, send SMS spam, download malicious executables and carry out illegitimate financial transactions. Since the core of a cellular network processes an enormous volume of traffic, the application of traditional security measures such as firewalls is not practical – thus posing unique challenges for detecting such mobile bots. In this paper we present a fully functional cellular botnet for *Symbian* smartphones with an efficient and effective command and control centre. Using a formal model, we show that with a zombie army of just 66 compromised cell phones, a botnet can incapacitate a cell site (BTS tower), resulting in complete denial of service to voice and SMS traffic. In a preliminary study, by using numbers in the phonebook of a mobile phone our bot sent an install service message to 150 users – 90 of whom downloaded the (fake, non-malicious) binary and installed it on their phones.

1. CELLULAR BOTNETS DEMYSTIFIED

In the last decade, cellular mobile networks have caused a paradigm shift in the world of computing. Contemporary 2.5/3/4G cellular networks have been deployed worldwide and now form the core for offering integrated services of voice, text and data at reasonably high data rates. To use these services effectively, mobile phones have evolved into full-blown computing platforms which offer data services and applications comparable to those on desktop computers[2]. As a result, they are becoming an attractive target for imposters and intruders. *McAfee* recently reported a more than 100% increase in malware, phishing and DoS attacks against mobile devices since 2006 [3]. On a similar

[1] This work is supported by the Pakistan National ICT R&D Fund.
[2] In 2008, the number of mobile phones in the world was estimated at 4.1 billion [1], while the estimated number of computers was just one billion [2].

note, a recent paper by Traynor *et al.* [4] used analytical and simulation results to highlight the significant damage that can be caused by a cellular botnet; i.e. a botnet comprising mobile phones as zombies.

In this paper, we report our experiences of developing SymBot[3], a fully functional botnet of mobile smartphones for the popular *Symbian* OS[4]. Our research effort is inspired by the Morris worm, one of the first publicly known computer worms [6]. While this worm opened the door for other malware which continues to plague the Internet to this day, many experts feel that the Morris worm did some good in that it exposed the inherently vulnerable designs of computer operating systems and the Internet and catalysed a serious security-centric rethinking of OS and networking design philosophies. Using our experiences of developing SymBot as a baseline, we argue that a similar reality check is required to enhance the security of mobile devices and networks.

2. BACKGROUND & RELATED WORK

In the last decade, botnets have emerged as one of the greatest threats to IP network availability and information confidentiality. As a result, detection of botnet activity has received significant attention in network security research [7, 8].

While botnets on the Internet are well studied, there is surprisingly little research literature on botnets for mobile devices and networks. There has been some research on how wireless links can be saturated to cause denial of service or how the open functionality of certain cellular services can help in a DoS attack [9]. However, to the best of our knowledge, there is only one paper on cellular botnets in which the zombies comprise mobile phones inside the cellular network. Traynor *et al.* [4] showed that botnets of as few as 11,750 phones can cause a reduction of throughput of more than 90% to area-code sized regions supported by most currently deployed systems.

In this paper, we provide a proof-of-concept implementation of a cellular bot – SymBot – for *Symbian* smartphones which is capable of launching prominent IP-based attacks.

The following section provides background on the *Symbian* OS and its development SDK, with details of the structures, facilities and API calls that were used to develop a cellular bot.

[3] The architecture of SymBot is generic and can be realized on other platforms (such as *Android*) and networks (3G).

[4] *Symbian* is reported to have had a 36.4% share of the mobile phone market in 2009, which is almost twice the market share of the second biggest phone vendor (*Samsung*) [5].

3. EXPLOITING SYMBIAN OS FOR SYMBOT DEVELOPMENT

In this section, we first provide an overview of the *Symbian* OS security framework. This is followed by details of how we implemented different attacks and communication modules in SymBot.

3.1 Symbian security framework

A number of security measures are adopted in the *Symbian* OS that deny installation of malicious applications. A developer has to explicitly state the desired capabilities of its application and hard-code them in the header of the executable image. The developer must also specify the system API calls that the application will use to achieve the desired functionality; this information is provided in a .mmp file. With this information, the developer submits the source code of his application and .mmp file to the *Symbian Signed* website[5]. This site issues a code signing certificate to the developer which he subsequently uses to create an install SIS package. The available signing options and capabilities are shown in Figure 1.

Using the SIS package, a developer can install his application on a *Symbian* smartphone with the help of an install server. The install server allows or disallows the installation by checking the capabilities required by the binaries, comparing them with the configuration of the device, and verifying the signature of the SIS package which contains all the binaries, resource files and the metadata required for installation. Once the application is started, the header of the executable image is loaded into memory and the loader associates the new process with the capabilities specified in the header. The capabilities are set only once and cannot be changed at runtime [10].

If an application makes a system call which is incompatible with its capabilities, *Symbian* OS does not allow the call to execute. This is the main security premise adopted by the *Symbian* OS to disallow installation of malware on-the-fly. However, the top left corner of Figure 1 shows capabilities (in shaded blocks) that can be given to an application through self-signing; i.e. there is no need to get a code signing certificate from the *Symbian* website. We now discuss the architecture of our SymBot, emphasizing how these capabilities can be misused to perform different exploits.

3.2 Architecture of SymBot

We have developed a GUI application – Currency Converter – which converts a given amount from one currency to another. Since the application needs to connect to the

[5] http://www.symbiansigned.com/.

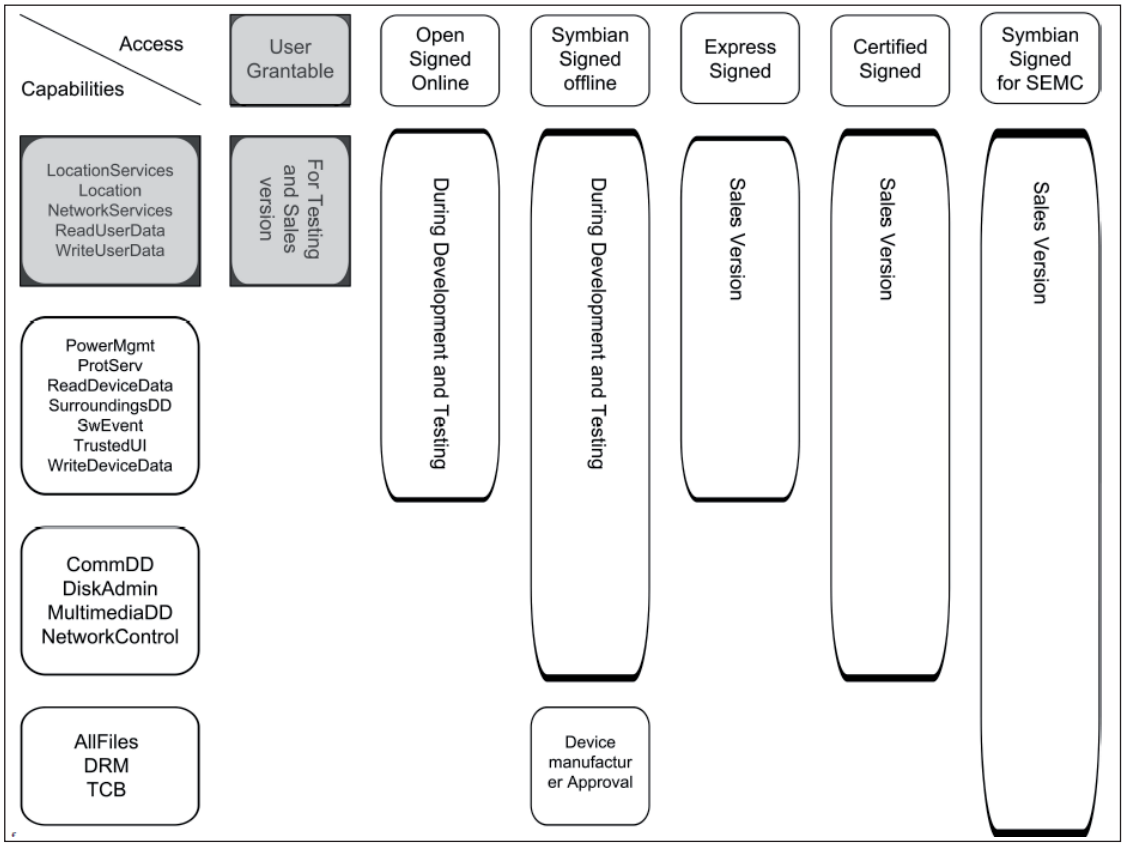


Figure 1: Symbian Signed grid [10].

Internet to receive daily foreign exchange rates, a user can easily be tricked into granting the application access to NetworkServices. The application is designed as a self-signed, multi-threaded process. Once the user launches the application, the bot starts working in the background. The main modules of SymBot are:

1. Central processing module
2. Watchdog module
3. Threat-invoking module
4. Encryption module
5. Communication module.

The remainder of this section describes each of these components.

Central processing module (CPM)

This module handles the flow of data and decision making for the bot. After infection, the CPM opens a TCP port and an SMS socket connection. Currently, the phone number of the C&C is hard-coded in the bot. The central processing module queries its location from the location update module. On receiving the location information, it is forwarded to the encryption module and an encrypted message is sent to the C&C. Depending on the commands, the CPM invokes the following sub-modules:

1. Location update. SymBot retrieves the GPS location of a phone using the CLocation API call defined in lbs.lib, which is enabled by the Location capability. Using this API call, SymBot periodically sends the location of the infected mobile phone to its C&C centre. The C&C aggregates the information to compute the number of zombies in a given site and if the number exceeds a threshold value, it has the capability to launch an effective DDos attack.
2. Financial data stealing. Most of the cellular operators in Pakistan allow users to share their credit via an SMS [11]: users send a template SMS containing a recipient phone number and amount of credit, and the credit is automatically transferred to the recipient. When the command for stealing financial data is received, this module checks the network operator to which the user is subscribed.

This is done by checking the default SMS message centre. Depending on the service provider, the template SMS is constructed and sent to the communication module. In order to evade detection, transfer could happen through a chain of zombies between the victim and the bot master.

3. Personal information stealing. SymBot accesses the phonebook using the CContactDataBase API call in the cntmodel.lib library. This API call is enabled by the ReadUserData and WriteUserData capabilities. Using this API call, an attacker can also access the additional information for contacts in the phonebook: name, email address, home phone numbers, etc. SymBot sends the complete set of information to the C&C, as this can be sold to telemarketing and advertisement companies. Moreover, the contact information could be exploited by scammers to launch phishing attacks on phonebook contacts by acting as bank employees, promotion managers for cellular operators, or other persons of trust. Interested readers can find information related to closure of a major online cellular phone directory at [12].

Watchdog module

The watchdog module stores the date and time of the attack, which is given to it by the CPM. It runs a thread which continuously matches the current system date and time with the attack date and time and accordingly commands the threat-invoking module to launch attacks.

Threat-invoking module

This module launches SMS spam or a DoS attack depending on the commands received from the watchdog module. The spam generator basically creates SMS messages containing a link to the website on which the bot SIS file is placed. (According to a survey of 2,150 UK mobile phone users [13], two thirds of the users received SMS spam, and 38% of the users received a text containing a link to another site.) Thus SMS spam can also use social engineering to trick a user into installing malicious applications. The DoS attack module launches an SMS flood on the network which is explained in detail in Section 4.

Encryption module

The C&C defines the mode of communication with the bot. If the C&C demands encrypted communication, the encryption module sends a binary SMS in Packet Data Unit (PDU) mode. The payload of the PDU contains binary information, e.g. an image, encrypted text, etc. A binary SMS can hold 140 bytes of data, allowing any block cipher – like AES or DES – to be applied to it. By using encrypted SMSs SymBot can hide its communication from a sniffing IDS which may be deployed to monitor the activities of potentially malicious applications. Using a 256-bit AES key, this module encrypts its data and sends it back to the main module in the following format: Encrypt(Length of Message | Message | Padding Bytes). The length of the message is used so that the receiver can extract only the original message without the padding that was used to enable encryption.

Communication module

The communication module handles the exchange of information to/from the networks outside the GSM. Most of the time, the communication activities are related to the exchange of information with the C&C.

Using the RSocketServ and RSocket API calls – defined in esock.lib and enabled by the NetworkSevices capability – the communication module opens a TCP or UDP port. This port can be utilized by the bot master to send malicious binaries containing platform security hacks – the latest platform security hacks are reported on www.symbianfreak.com. Once a mobile phone is compromised, the attacker can install malicious applications without the need for signature certificates. These applications can launch attacks on the critical core network entities (e.g. Home Location Register (HLR), Visited Location Register (VLR), SMS gateway, etc.) through meta commands [4].

An effective botnet must have the ability to exchange information among an army of zombies in a stealthy fashion. The communication module does this by sending SMSs through sockets, which are enabled by the RSocketServ, TSmsAddr, CSmsBuffer and RSmsSocketWriteStream API calls defined in esock.lib, smsu.lib, smu.lib and estor.lib, respectively. This approach has two inherent advantages: (1) communication takes place under the hood without alerting the user or raising any alarm, and (2) cheaper communication ensures that the user's credit depletes gradually, once again avoiding raising any alarm.

MMS is another option available to realize intra botnet communication. To send MMSs, we use the CMmsClientMtm API calls defined in mmscli.lib. Using this feature, a bot can download command files embedded in JPEG images and share them with other bots, making detection of malicious activity a significant challenge. However, MMS services are relatively expensive, so SymBot does not use them.

Table 1 shows a summary of SymBot's exploits, associated API calls and libraries.

Exploit	Library used	Capabilities required	Signing
Stealing personal info	cntmodel.lib	ReadUserData	SelfSigned
Opening proxies	esock.lib	NetworkServices	SelfSigned
Botnet communication	smu.lib	NetworkServices	SelfSigned
Financial data stealing	smcm.lib	NetworkServices	SelfSigned
Spamming	smcm.lib	NetworkServices	SelfSigned
Location tracking	lbs.lib	Location	SelfSigned

Table 1: SymBot exploits.

3.3 Command and control mechanism

An efficient command and control structure is fundamentally important for botnet operation. This requirement for efficiency is amplified in cellular environments where the data rates are typically much lower than on the Internet. Thus a cellular bot master has to devise intelligent methods to exchange information with the bots without being detected. Most of the methods used to disrupt IP botnets focus on detecting the command and control structure and disrupting the communication. The two well-known C&C mechanisms for IP botnets are P2P and centralized. For SymBot, we have developed a centralized C&C structure for three reasons:

1. In a P2P C&C mechanism, bots have to continuously search their neighbours for the search keys and command files. Moreover, the bots have to send keep-alive messages throughout the botnet. For cellular bots, such communication will lead to prohibitively high overheads. In particular, as the number of zombies in the botnet increases, the number of concurrent flows for the keep-alive messages and the searching mechanism will cause significant unwanted competition for scarce cellular resources which are not designed for concurrent flows [14].
2. Likewise, when the bots are submitting stolen information from the compromised mobile phones, the data flowing from all the neighbours would cause the smartphone memory to be depleted very quickly. This would not only alert the user but also lead to detection of the malware.
3. DDoS attacks cannot be successfully carried out with a P2P structure because estimation of the number of active bots within a cell using P2P communication has a very high overhead.

We have developed a generic C&C architecture which communicates with the bot through a GSM/CDMA/UMTS modem connected to a computer. In this way the C&C is placed in the cellular network. It has two advantages: (1) the C&C can utilize all the communication methods described earlier, and (2) the limitations faced by smartphones are eliminated by the use of a computer.

The C&C builds and maintains a vector space in which it saves the location of all bots connected to it. The columns of the vector space represent the sites in the city and the rows represent the number of bots in each site. Once the C&C finds that it has the necessary population in each site to launch an effective attack, it sends the attack commands to all bots.

4. DISTRIBUTED DENIAL OF SERVICE ATTACK

We have used existing models by Traynor [9] and enhanced them to calculate the number of zombies required to saturate the control channels of a GSM network, which results in denial of voice and messages services. Using our enhanced model, we found that an army of 66 zombies is required to launch DDoS at a site (BTS). Our analysis was based on the core network of *Telenor* in Islamabad that consists of 40 sites and 300,000 users. We discovered that we would need just 3,000 zombies (just 0.1% of the total subscribers) to launch a complete denial of service in the metropolitan area of Islamabad.

Using the model, we concluded (for the sake of brevity, we skip the calculations) that with approximately 22 bots in a single sector, and with 66 bots in a single cell, our GSM botnet could incapacitate the cellular core network for legitimate voice and text messaging services. Our botnet is location-aware and hence can trigger the DDoS attack once it finds 66 bots in a single sector. It is interesting to consider the fact that during the daytime, the universities and offices are crowded with people. In such a scenario, the botmaster could easily recruit the required number of zombies to launch a successful DDoS attack.

4.1 Preliminary user study of infection propagation

While designing our user study, we had to be mindful of the security and privacy of the subjects. To this end, we created a benign version of our currency conversion application and uploaded it on a website. The authors' mobile phones were then used to send SMS service messages, containing

the URL of the website, to the phone contacts, inviting them to download and install this useful application for free. This SMS message was sent to a total of 150 contacts. Interestingly, 90 of them downloaded and installed the benign application on their mobile phones. This preliminary study demonstrates that an alarming infection rate of 60% could be achieved even using this rudimentary infection/distribution strategy.

We then asked the users some questions about the choice they made. Based on their answers, we conclude that, although few of these users would download and install a piece of software referred to them through an Internet email, the same message on a mobile phone SMS can achieve a much higher infection rate. This is mainly because people still think that cellular networks and mobile phones are inherently 'safe' from malware; as a consequence, they are less wary about installing new applications on mobile phones. This prevalent mindset results in a very effective and easy distribution mechanism for malware which can use trivial SMS and MMS communication media to infect smartphones.

5. CONCLUSION AND FUTURE WORK

In this paper, we have presented our experiences of developing a fully functional botnet for the *Symbian* OS. We have shown that most of the malicious activities of a botnet can easily be implemented on the *Symbian* platform. The underlying communication media also provides the capacity to remotely control the botnet and to launch crippling attacks on the cellular infrastructure. Our analytical and experimental models have shown that 66 zombie smartphones can incapacitate a cell site through voice and SMS DoS attacks. We have also revealed some alarming infection rates that can be achieved in cellular networks using trivial distribution mechanisms like SMS and MMS. We believe that our findings will challenge the existing belief (or fallacy): mobile phones are inherently safe because of a reliable core of cellular networks. As a result, we hope that security countermeasures will receive their due attention.

REFERENCES

[1] ITU Corporate Annual Report 2008. http://www.itu.int/osg/csd/stratplan/AR2008 web.pdf.

[2] Gartner Forecast: PC Installed Base, Worldwide, 2004-2012. http://www.gartner.com/DisplayDocument?id=644708.

[3] McAfee Mobile Security Report 2009. http://www.mcafee.com/us/resources/reports/rp-mobile-security-2009.pdf.

[4] Traynor, P.; Lin, M.; Ongtang, M.; Rao, V.; Jaeger, T.; McDaniel, P.; La Porta, T. On cellular botnets: Measuring the impact of malicious devices on a cellular network core. Proceedings of the 16th ACM Conference on Computer and Communications security, 2009, pp. 223–234.

[5] De La Vergne, H.; Milanesi, C.; Zimmermann, A.; Cozza, R.; Huy Nguyen, T.; Gupta, A.; Lu, C.K. Competitive Landscape: Mobile Devices, Worldwide, 4Q09 and 2009.

[6] Spafford, E. The Internet worm program: an analysis. ACM SIGCOMM Computer Communication Review, vol. 19, no. 1, p. 57, 1989.

[7] Gu, G.; Zhang, J.; Lee, W. BotSnier: Detecting botnet command and control channels in network traffic. Proceedings of the 15th Annual Network and Distributed System Security Symposium (NDSS), 2008.

[8] Gu, G.; Porras, P.; Yegneswaran, V.; Fong, M.; Lee, W. Bothunter: Detecting malware infection through ids-driven dialog correlations. Proceedings of the 16th USENIX Security Symposium, 2007, pp. 167–182.

[9] Enck, W.; Traynor, P.; McDaniel, P.; La Porta, T. Exploiting open functionality in SMS-capable cellular networks. Proceedings of the 12th ACM Conference on Computer and Communications Security, 2005.

[10] Mueller, B. From 0 To 0Day On Symbian: Finding Low Level Vulnerabilities On Symbian Smartphones. 2009. https://www.sec-consult.com/files/SEC_Consult_Vulnerability_Lab_Pwning_Symbian_V1.03_PUBLIC.pdf.

[11] http://hulchul.urdupoint.com/HC/topic/151931-mobilink-telenor-ufone-warid-paktel-balance-share/.

[12] Working Cell Phone Number Directory Worries Privacy Advocate. http://www.pr-inside.com/working-cell-phone-number-directory-worries-r1550881.htm.

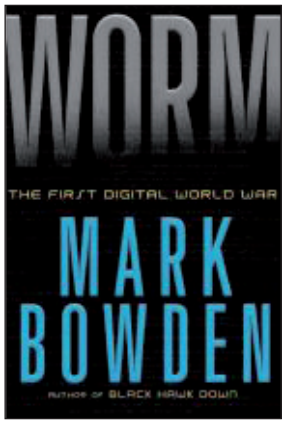
[13] Savvas, A. Two thirds of Britons say they have been victims of mobile spam. http://www.computerweekly.com/news/2240085822/Two-thirds-of-Britons-say-they-have-been-victims-of-mobile-spam.

[14] Traynor, P.; McDaniel, P.; La Porta, T.; et al. On attack causality in internet-connected cellular networks. Proceedings of the 16th USENIX Security Symposium, 2007, pp. 307–322.

BOOK REVIEW

BOOK WORM

Paul Baccas
Sophos, UK



Title: Worm: The First Digital World War

Author: Mark Bowden

Publisher: Atlantic Monthly Press

ISBN: 978-0-8021-1983-4

'*The bad guys are on the Conficker Working Group email lists.*' Gunter Ollmann (paraphrased).

I read this book on the flight from London to Barcelona for VB2011, and when I heard the above quote in the final panel discussion at the end of the conference I was left reeling. The book is about the team who worked together to combat the Conficker worm and focuses on some of the main players in the Conficker Working Group.

The author – who has a journalistic background and has written several other journalistic history books, most notably *Black Hawk Down* – treads lightly over the technical aspects of the worm and concentrates more on the history and the group dynamics of the multidisciplined group, or cabal, that countered Conficker.

According to the author the principal members of the Conficker cabal were: TJ Campana, John Crain, Andre DiMino, Rodney Joffe, Chris Lee, Andre Ludwig, Ramses Martinez, Phil Porras, Hassen Saidi, Paul Twomey, Paul Vixie and Rick Wesson. The book uncovers their stories over the nine months of Conficker's activity.

THE CHAPTERS

The 11 chapters are self-contained and can be read separately but really ought to be read in order. The first chapter, 'Zero', begins in November 2008 when Conficker first popped up on the radar of malware researchers and no anti-malware solution providers were able to detect it. This chapter introduces one of the main protagonists, Phil Porras, and it is here that the book is most technical, explaining in general ways about bots, IPs and some malware history.

Then we segue into the second chapter, 'MS08-067', in which we are introduced to TJ Campana, the PM of security at the MS Digital Crimes Unit. Here, the book details how *Microsoft* needed to release an out-of-band patch for the RPC vulnerability a month before Conficker appeared.

Next, in 'Remote Thread Injection', we encounter Hassen Saidi and the packing and encryption of the Conficker worm. A description is included here of the Domain Generating Algorithm which was used by the worm to connect to 250 pseudorandom websites a day. This chapter also analyses the name 'Conficker' – a mixture of the letters from 'trafficconverter.biz' (a website Conficker.A tried to contact) and a German expletive.

In 'An Ocean of Suckers', we are treated to a potted history of computer worms: from Brunner's *The Shockwave Rider* through the Morris or RTM worm to Code Red and Blaster. The author looks at how Conficker combined the techniques used by these worms with some botnet technology.

In 'The X-Men' we see the more formal beginnings of the Conficker Working Group, aka the Conficker cabal, where the group starts to coalesce and with the rest of the world waking up to the fact that something was lurking on the Internet in December 2008. The book is riddled with references to the *Marvel Comics* creation from which this chapter takes its title, with the cabal as the superheroes and the malware authors the agents of evil.

The book follows the threads of the story and the chapters overlap chronologically. In 'Digital Detectives' we are introduced to more of the 'X-Men' with some history of the evolution of Conficker from Gimmiv, one of its precursors. This chapter also explains how researchers in different locations and from different companies were already sinkholing Conficker domains.

In 'A Note from the Trenches' the arrival of the B variant is detailed, with a listing of some of the differences between the two versions. The cabal was sinkholing A variant domains, mainly via the use of *Amazon S3* and personal credit cards, but the B variant was a game changer, adding more TLDs. An estimate of the cost of pre-registering the URLs involved in both variants is given as $100,000 per month. The cabal began to contact registrars. All this happened against a backdrop of press awareness and the faltering interest of governmental agencies.

At the beginning of 2009, most of the members of the cabal met at a conference in Atlanta, Georgia. In 'Another Huge Win', the conference is discussed with the 'win' referring to the fact that ICANN agreed to help sinkhole the Conficker domains. It was in this meeting that contacting China, where a large proportion of the infection existed, was discussed. This rather formal reaching out was trumped by the fact that one of the cabal members was

already sharing the data from the mailing lists with the Chinese. This seeming betrayal of the group caused a split that permeates the rest of the book.

The split, which happened when the group was on a high and thought that the worm had been beaten, was compounded by the arrival of the next variant. Whether by chance or design the Conficker authors knew when to 'put the boot in'. The big difference was that instead of 250 domain names this variant could poll 50,000. Rumours abounded that the Conficker author was actually a member of the cabal:

'This is starting to stink of an inside job.'

'The people behind this are us.'

Suggestions were made that tackling this piece of malware was too much for a group of loosely affiliated researchers, and that they should get the government involved. In 'Mr. Joffe goes to Washington', the author describes how Rodney Joffe attempted to do just that (and in doing so trod on a few toes within US CERT), but despite presenting the problems to many government agencies he left disillusioned after a week.

The last two chapters, 'Cybarmageddon' and 'April Fools', work up to and beyond the malware's 1 April trigger date. This date turned out to be a damp squib thanks to the efforts of the cabal and other parts of the anti-malware industry in successfully combating it.

While we will never know why Conficker was created or what it originally meant to do, the sophistication of the code and the complexity of the effort needed to combat it was staggering. Was this a criminal gang? Or a governmental or quasi-governmental weapons test? Whatever it was, it highlighted the importance of working together and trusting one another.

VERDICT

The book is not a technical analysis of Conficker, though it may add to your knowledge. It is an analysis of the personalities and social interactions of some of the movers and shakers behind the Conficker Working Group.

The book is very readable; I was annoyed when my flight arrived 15 minutes earlier than scheduled because I was left with 10 pages to read! The individual chapters are each self-contained stories, which means that you do not have to read the book all in one go. The style is journalistic and the high quality writing is what one would expect from an author with Bowden's credentials. I would be more than happy to find this book under the Christmas tree this holiday season.

END NOTES & NEWS

Takedowncon 2 – Mobile and Wireless Security will be held 2–7 December 2011 in Las Vegas, NV, USA. EC-Council's new technical IT security conference series aims to bring industry professionals together to promote knowledge sharing, collaboration and social networking. See http://www.takedowncon.com/ for more details.

Black Hat Abu Dhabi takes place 12–15 December 2011 in Abu Dhabi. Registration for the event is now open. For full details see http://www.blackhat.com/.

FloCon 2012 will be held 9–12 January 2012 in Austin, TX, USA. For more information see http://www.flocon.org/.

RSA Conference 2012 will be held 27 February to 2 March 2012 in San Francisco, CA, USA. Registration is now open with an early bird rate available until 18 November. For full details see http://www.rsaconference.com/events/2012/usa/index.htm.

Black Hat Europe takes place 14–16 March 2012 in Amsterdam, The Netherlands. For details see http://www.blackhat.com/.

SOURCE Boston 2012 will be held 17–19 April 2012 in Boston, MA, USA. For further details see http://www.sourceconference.com/boston/.

The 3rd VB 'Securing Your Organization in the Age of Cybercrime' Seminar takes place 19 April 2012 in Milton Keynes, UK. Held in association with the MCT Faculty of The Open University, the seminar gives IT professionals an opportunity to learn from and interact with top security experts and take away invaluable advice and information on the latest threats, strategies and solutions for protecting their organizations. For details see http://www.virusbtn.com/seminar/.

Infosecurity Europe 2012 takes place 24–26 April 2012 in London, UK. See http://www.infosec.co.uk/.

The 21st EICAR Conference takes place 7–8 May 2012 in Lisbon, Portugal. The theme for this event will be '"Cyber attacks" – myths and reality in contemporary context'. For full details see http://www.eicar.org/17-0-General-Info.html.

NISC12 will be held 13–15 June 2012 in Cumbernauld, Scotland. The event will concentrate on 'The Diminishing Network Perimeter'. For more information see http://www.nisc.org.uk/.

The 24th annual FIRST Conference takes place 17–22 June 2012 in Malta. The theme of this year's event is 'Security is not an island'. For details see http://conference.first.org/.

Black Hat USA will take place 21–26 July 2012 in Las Vegas, NV, USA. For details see http://www.blackhat.com/.

The 21st USENIX Security Symposium will be held 8–10 August 2012 in Bellevue, WA, USA. See http://usenix.org/events/.

VB2012 will take place 26–28 September 2012 in Dallas, TX, USA. More details will be revealed in due course at http://www.virusbtn.com/conference/vb2012/. In the meantime, please address any queries to conference@virusbtn.com.

VB2013 will take place 2–4 October 2013 in Berlin, Germany. More details will be revealed in due course at http://www.virusbtn.com/conference/vb2013/. In the meantime, please address any queries to conference@virusbtn.com.

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/