

Richard A. Jacobsen (RJ5136)
ORRICK, HERRINGTON & SUTCLIFFE LLP
51 West 52nd Street
New York, New York 10019
Telephone: (212) 506-5000
Facsimile: (212) 506-5151

Gabriel M. Ramsey
(*pro hac vice application pending*)
ORRICK, HERRINGTON & SUTCLIFFE LLP
1000 Marsh Road
Menlo Park, California 94025
Telephone: (650) 614-7400
Facsimile: (650) 614-7401

Attorneys for Plaintiffs
MICROSOFT CORPORATION,
FS-ISAC, INC. and NATIONAL AUTOMATED
CLEARING HOUSE ASSOCIATION

**UNITED STATES DISTRICT COURT
EASTERN DISTRICT OF NEW YORK**

MICROSOFT CORP., FS-ISAC, INC., and
NATIONAL AUTOMATED CLEARING HOUSE
ASSOCIATION,

Plaintiffs

v.

JOHN DOES 1-39 D/B/A Slavik, Monstr, IOO,
Nu11, nvidiag, zebra7753, lexa_Mef, gss, iceIX,
Harderman, Gribodemon, Aqua, aquaSecond, it,
percent, cp01, hct, xman, Pepsi, miami, miamibc,
petr0vich, Mr. ICQ, Tank, tankist, Kusunagi,
Noname, Lucky, Bashorg, Indep, Mask, Enx,
Benny, Bentley, Denis Lubimov, MaDaGaSka,
Vkontake, rfcid, parik, reronic, Daniel, bx1, Daniel
Hamza, Danielbx1, jah, Jonni, jtk, Veggi Roma, D
frank, duo, Admin2010, h4x0rdz, Donsft,
mary.J555, susanneon, kainehave, virus_e_2003,
spaishp, sere.bro, muddem, mechan1zm,
vlad.dimitrov, jheto2002, sector.exploits AND
JabberZeus Crew CONTROLLING COMPUTER
BOTNETS THEREBY INJURING PLAINTIFFS,
AND THEIR CUSTOMERS AND MEMBERS,

Defendants.

FILED
CLERK
2012 MAR 19 AM 8:51
U.S. DISTRICT COURT
EASTERN DISTRICT
OF NEW YORK

CV 12-1335

Case No.

FILED UNDER SEAL

KORMAN, J.

MANN, M.J.

**DECLARATION OF JESSE D. KORNBLUM IN SUPPORT OF PLAINTIFFS'
APPLICATION FOR AN EMERGENCY TEMPORARY RESTRAINING ORDER,
SEIZURE ORDER AND ORDER TO SHOW CAUSE RE PRELIMINARY INJUNCTION**

I, Jesse D. Kornblum, declare as follows:

1. I am a Computer Forensics Researcher with Kyrus Technology. I make this declaration in support of Plaintiffs' Application For An Emergency Temporary Restraining Order, Seizure Order And Order To Show Cause Re Preliminary Injunction. I make this declaration of my own personal knowledge and, if called as a witness, I could and would testify competently to the truth of the matters herein.

2. I have over years of twelve years of experience in the field of computer and information security. I began my career as a Computer Crime Investigator with the U.S. Air Force Office of Special Investigations. Subsequently, I became Chief of Research and Development and ultimately Chief of the Computer Crime Investigations Division of the Air Force Office of Special Investigations. I have had roles as an instructor of computer science at the U.S. Naval Academy and Lead Information Technology Specialist with the U.S. Department of Justice, Computer Crimes and Intellectual Property Section. Most recently, I have had forensic research roles in the private sector. Currently, I am employed by Kyrus Technology, a technology company focused on reverse engineering, vulnerability research, computer forensics, and specialized software development related to computer security matters. A true and correct copy of my *curriculum vitae* is attached as **Exhibit A** to this declaration.

3. We were asked to conduct the underlying analysis to determine the similarity between copies of the Zeus Trojan botnet source code ("Zeus") and myriad binaries distributed by malicious actors. A "Trojan" program is a malicious program disguised as a legitimate application that is typically used to introduce viruses onto a computer or network. Our analysis is broken down into three main phases and is attached hereto as **Exhibit B**.

4. The first phase included an analysis of five portable executable binaries ("PE binaries") to determine a connection between these PE binaries and Zeus. A portable executable binary is a file which contains code and resources for executing on a computer running the Microsoft Windows operating system.

5. The second phase involved the analysis of three sets of binaries related to the

SpyEye, ICE-IX, and PCRE (aka “Zeus”) Trojans. These programs are recognized in the industry as being associated with “malware,” otherwise known as malicious software designed to disrupt or damage a computer, computer system, or network, or to gain sensitive information, or unauthorized access to computer systems.

6. In the third and final phase, we analyzed email messages sent by malicious actors that purported to be from the National Automated Clearing House Association, the trade organization for the ACH (direct deposit) system, to determine the functionality of links contained in the body of the emails. These three phases are described in more detail below.

7. Based on our analysis, we have concluded the following:

- a. It is highly probable that the PE binaries are copies of Zeus.
- b. The analyzed binaries related to SpyEye and ICE-IX are each highly similar to Zeus and support a finding that Zeus was developed with malicious intent.
- c. The email messages purportedly sent from the National Clearing House Association were designed to drive recipients to websites which would infect them with malware.

I. ANALYSIS

A. Phase I

8. We were provided 70 binary files, five of which were PE binaries. Of these five, four were packed using various means. A “packed” binary refers to an executable computer program which has been compressed and/or obfuscated. When executed, such programs use functionality added during the packing to return themselves to a functional equivalent of their original form.

9. The four packed binaries were unpacked to determine the functionality of their executable code. Ex. B at 5. Executable code is the set of sequential instructions executed by a computer and are generated from a programmer’s source code. The source code is the “blueprints” of the software, dictating what the program will do and how it will do it. The unpacked binary—2cc1076f3c6e65d0a59792b75370b04613258ffa—was used as a baseline for

functionality because no modifications to the binary were needed. *Id.* Every other packed executable was then compared against this baseline. Below are the PE binaries analyzed in

Phase I:

- 2cc1076f3c6e65d0a59792b75370b04613258ffa (baseline)
- 0cc6215d31e5e639a19b4ceb3d57ce64d62e9b2c (packed version 1)
- 9b259bc255fef873f1e41629fb67c30f0c40e5dc (packed version 2)
- 1bfdc4f2cfa48a1f063d1826992fbaf5e2924394 (packed version 3)
- bfcc02219321d1047cc0330454a61f6b276d06f6 (packed version 4)

10. We applied a number of analytic tools to the five PE binaries to determine the commonality between them and Zeus, using the unpacked PE binary as a baseline.

11. First, we conducted an analysis of each binary using “Virus Total.” Virus Total is a service that applies a number of Anti-Virus products to analyze suspicious files and URLs and detects the presence of malware, including Trojans. This analysis revealed a significant number of the Anti-Virus products applied by Virus Total identified these PE binaries as malicious. Indeed, for almost all of the PE binaries, a majority of the Anti-Virus products determined that the binary was malicious. *See* Ex. B at 6, 8, 10, 12 and 13. These findings are consistent with our conclusion that these binaries contain malicious software.

12. Next, we conducted an “Entry Point Analysis” to determine whether we had successfully unpacked a binary and to determine whether two binaries came from the same source code base. The “entry point” is the address of the first instruction to be executed in a binary. Because of the nature of the computer architecture, the first instruction is not necessarily at the beginning of the file. Here we compared the entry point code of the baseline PE binary to each of the four packed PE binary files. *See* Ex. B at 6-7, 9, 11, 13, and 14. In this case, the functionality of the entry point code was to disable any error message that may pop up during execution, and to attempt to get any command line arguments. *See* Ex. B at 7. Our comparisons determined that all five PE binaries are compiled from the same source base. Ex. B at 6-7, 9, 11, 13, and 14.

13. For the next analysis, we applied a Zynamics BinDiff program to the PE binaries. Zynamics BinDiff is a comparison tool that detects the similarities and differences between

binary files. We applied Zynamics BinDiff to compare the four unpacked PE binaries' executable code to that of the baseline. Ex. B at 9, 11, 13, and 15. An "executable file" determines the functionality of the binary. For all but one of the four packed PE binaries, our analysis determined that there was a significant overlap between each packed PE binary and the baseline binary. *Id.* For one unpacked PE binary, however, the Zynamics BinDiff program was unable to make a proper comparison against the baseline. *Id.* at 13. The significant similarities between the functions of the PE binaries (with the exception of one PE binary) indicate that the PE binaries have been compiled from the same source code.

14. Our entry point and Zynamics BinDiff analyses establish that all five PE binaries were compiled from the same code base. *Id.* at 15.

15. After determining that all five PE binaries were compiled from the same code, we compared the PE binary to that of Zeus to determine their similarities. We were able to obtain publicly available copies of the Zeus source code and compiled our own copy of Zeus to compare to each of the PE binary files described above. Ex. B at 15. Using Zynamics BinDiff, we ran a comparison of the executable baseline PE binary, with that of our compiled Zeus source code. *Id.* at 16. The comparison showed the baseline and our compiled versions are identical. *Id.* In other words, we determined that our samples are compiled versions of Zeus. *Id.*

16. Following this comparison, we searched for functions within our copy of Zeus that had a very low probability of being duplicated or copied by accident. We were able to determine that in every case, there was an exact or extremely high match between our copy of Zeus and the PE binaries that we analyzed. *Id.*

17. We also compared the PE binaries with our compiled version of Zeus using a program called The Interactive Disassembler ("IDA") to find and extract control flow graphs from both the binaries and Zeus. *Id.* at 17. Programs, like Zeus and binaries, are defined by a sequence of statements. *Id.* at 16. Each statement is an instruction to perform a discrete operation. These statements are linked together into a graph. *Id.* At every point where a value is tested, a statement can conditionally branch to a new node in the graph depending on the value.

Id. In this way, any logical instructions can be represented by computer code. *Id.* By using IDA, we were able to compare each of the PE binaries to the Zeus binary we compiled in graph form. *See* Ex. B at 16-22. These graphs are almost identical across each program. *Id.* at 21. We were also able to extract the specific functions within each program to compare to the other binaries. *Id.* at 24. Our results indicate that for the functions identified in the binaries, almost all of them are structurally identical to functions that are within Zeus. *Id.*

18. The similarities between Zeus and the PE binaries also show that it is highly likely that Microsoft compilers were used to build these versions of Zeus. A “Microsoft compiler” is a tool used to convert source code written by a programmer into a Window-based PE executable. When comparing the source code in Zeus to each PE binary, we were able to identify identical blocks of source code for identical functions in each. Ex. B at 22-23. This is significant given the fact that different compilers write different code to carry out the same function. *Id.* at 23-24. It is highly probable, then, that Zeus and the PE binaries were both developed using Microsoft compilers, providing additional support for our conclusion that the PE binaries are copies of Zeus.

19. Finally, we used the industry standard “fuzzy” hashing technique to compare the PE unpacked binaries and Zeus. *Id.* at 24-25. This technique allows for the comparison of files after converting the code into individual hashes, making it easily readable. We used this technique to compare files found in both the unpacked binaries and Zeus. *Id.* The files were found to be similar—with large stretches of identical patterns of bytes, consistent with our conclusion that these files are essentially the same. *Id.*

B. Phase II

20. For the second phase, we analyzed three sets of binaries and compared the capabilities of a sample from all three to the Zeus source code. These sets of binaries, which are regarded as malicious software in the industry, include: 1) PCRE, 2) SPYEYE, and ICE-IX.

21. We were unable to analyze the PCRE binary because this sample did not contain valid applications to analyze and were likely encoded with a password that was not provided.

Ex. B at 26.

22. We analyzed the SPYEYE sample set by reverse engineering a selected file, b33064449295083dbfec12634523d805. *Id.* After reverse engineering this file, we were able to determine that the capabilities of this binary are: 1) windows enumeration, 2) take screenshot of desktop, 3) retrieve clipboard data, 4) keyboard logging, 5) retrieve system information, 6) communicate with C&C server using HTTP, 7) enumerate user accounts, 8) file search, 9) remote process code injection, 10) manipulate windows registry, 11) process enumeration, 12) read arbitrary file contents, 13) standard TCP socket communication, and 14) download and execute payloads. *Id.*

23. We next analyzed the ICE-IX sample using the file 3c6839c4ce744c9c0ddf2ba06963c3f4. *Id.* After reverse engineering the binary we determined that its capabilities included: 1) take screenshot of desktop, 2) remote process code injection, 3) retrieve system information, 4) user account enumeration, 4) keyboard logging, 5) process enumeration, 6) file search capability, 7) get contents of arbitrary file, 8) encrypt/decrypt data using the Windows crypto API, 9) manipulate windows registry, 10) communicate with C&C via HTTP; 11) Standard TCP socket communication, and 12) download and execute payloads. *Id.* at 27.

24. We then compared the Zeus binaries and to SPYEYE and ICE-IX and determined that the functionality is very similar. Specifically, Zeus supports the following capabilities: 1) take screenshot of desktop, 2) remote process code injection, 3) retrieve system information, 4) keyboard logging, 5) VNC server, 6) HTTP injection, 7) communicate with C&C via HTTP; 8) download and execute payloads, 9) process enumeration, 10) self delete using bat file, 11) intercept Windows API functions, and 12) manipulate Windows Registry. This finding of similar capabilities supports our conclusion that the Zeus binaries were developed with malicious intent.

C. Phase III

25. In the final phase of our analysis, we examined e-mails purportedly sent by the

National Automated Clearing House Association (“NACHA”), but actually originating from malware authors. The subject and content of these emails contain references to ACH (direct deposit) payments being rejected. These emails directed the recipient to a URL that the e-mail states is a Microsoft Word document providing more information. In actuality, however, we found these URLs pointed to a website that hosts malicious software.

26. We analyzed the content hosted at the URL provided in these emails using Virus Total. The results of this analysis indicated both the content and domain were malicious. *See* Ex. B at 27-29. These findings are consistent with our conclusion that these emails were designed to drive recipients to infect themselves with malware.

I declare under penalty of perjury under the laws of the United States of America that the foregoing is true and correct to the best of my knowledge.

Executed this 18th day of March, 2012



Jesse D. Kornblum

EXHIBIT A

Jesse D. Kornblum

Kyrus Technology
Sterling, VA

jesse.kornblum@kyrus-tech.com
<http://jessekornblum.com/>

Education

M. Eng., Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 1999

B.S. Computer Science, Massachusetts Institute of Technology, 1999

Employment

Kyrus Technology Corporation 2010-Present
Computer Forensics Research Guru Sterling, VA

ManTech International Corporation 2005-2010
Senior Computer Forensic Scientist Falls Church, VA

United States Department of Justice 2004-2005
Lead Information Technology Specialist, Washington D.C.
Computer Crime and Intellectual Property
Section

United States Naval Academy 2003-2004
Instructor, Computer Science Department Annapolis, MD

Air Force Office of Special Investigations 2003
Chief, Computer Investigations and Operations Andrews AFB, MD

Air Force Office of Special Investigations 2001-2003
Chief of Research and Development, Computer Andrews AFB, MD
Investigations and Operations

Air Force Office of Special Investigations 1999-2001
Computer Crime Investigator Andrews AFB, MD

Service

Member of the Editorial Board for the journal *Digital Investigation*

Technical Program Committee Member for Digital Forensic Research Workshop 2005-2010

Technical Editor for *Windows Forensic Analysis* by Harlan Carvey

Member of the DFRWS Common Digital Evidence Storage Format Working Group

Awards and Honors

USNA Computer Science Department "Top Geek", Fall 2003

HQ AFOSI Company Grade Officer of the Quarter, 2nd Quarter 2002

Jesse D. Kornblum

Refereed Papers

- J. Kornblum, *Implementing BitLocker Drive Encryption for Forensic Analysis*, Digital Investigation, 5(3): 75-84, March 2009.
- J. Kornblum, *Auditing Hash Sets: Lessons Learned from Jurassic Park*, Digital Forensic Practice, 2(3):108-112, July 2008.
- E. Libster and J. Kornblum, *A Proposal for an Integrated Memory Acquisition Mechanism*, Operating Systems Review, 42(3):14-20, April 2008.
- J. Kornblum, *Using Every Part of the Buffalo in Windows Memory Analysis*, Digital Investigation, 4(1):24-29, March 2007.
- J. Kornblum, *Exploiting the Rootkit Paradox with Windows Memory Analysis*, International Journal of Digital Evidence, 5(1), Fall 2006.
- B. Carrier, E. Casey, S. Garfinkel, J. Kornblum, C. Hosmer, M. Rogers, and P. Turner, *Standardizing Digital Evidence Storage*, Communications of the ACM, February, 2006.
- J. Kornblum, *The Linux Kernel and the Forensic Acquisition of Hard Disks with an Odd Number of Sectors*, International Journal of Digital Evidence, Volume 3(2), Fall 2004.

Conference Papers

- J. Kornblum *Using JPEG Quantization Tables to Identify Imagery Processed by Software*, Digital Investigation, 5(S):21-25, Proceedings of the Digital Forensic Workshop, August 2008.
- J. Kornblum, *Identifying Almost Identical Files Using Context Triggered Piecewise Hashing*, Digital Investigation, 3(S):91-97, Proceedings of the Digital Forensic Workshop, August 2006.
- J. Kornblum, *Preservation of Fragile Digital Evidence by First Responders*, Digital Forensic Research Workshop, Syracuse, NY, August 2002.

Other Publications

- J. Kornblum, *When I'm Sixty Four (Bits)*, ManTech Tech Note 2009-01, August 2009.

Forensic Tools

- J. Kornblum [findaes](#), Finds AES key schedules
- J. Kornblum [hashdeep](#), Audits a set of known hashes against a given directory, 2008.
- J. Kornblum, [Miss Identify](#), Identifies PE executables that do not have an executable extension. Optionally identifies all executables in a set of input files, 2008.
- J. Kornblum, [dc3dd](#), a version of GNU dd patched for computer forensics, 2008.
- J. Kornblum, [ssdeep](#), Computes and matches context triggered piecewise hashes, also called fuzzy hashing. Matches similar but not identical files, 2006.
- J. Kornblum, [md5deep](#), A set of recursive programs for computing MD5, SHA-1, SHA-256, Tiger, and Whirlpool hashes. Capable of both positive and negative matching, 2002.

J. Kornblum, Investigator Controlled Evidence Extraction Engine (ICE³). Boot CD for automated disk imaging.

J. Kornblum, First Responder's Evidence Disk (FRED). Automated Windows incident response tool.

K. Kendall, J. Kornblum, N. Mikus, [foremost](#). A linux based file carving program. Recovers files from disk images based on their headers and footers, 2001.

EXHIBIT B



b71 Binary Analysis Report

Table of Contents

Executive Summary	4
Phase I	5
Binary: 2cc1076f3c6e65d0a59792b75370b04613258ffa	5
Virus Total Results	6
Entry Point Analysis	6
Binary: 0cc6215d31e5e639a19b4ceb3d57ce64d62e9b2c	7
Virus Total Results	8
Entry Point Execution Flow Comparison	9
BinDiff Analysis	9
Binary: 9b259bc255fef873f1e41629fb67c30f0c40e5dc	10
Virus Total Results	10
Entry Point Execution Flow Analysis	11
BinDiff Analysis	11
Binary: 1bfdc4f2cfa48a1f063d1826992fbaf5e2924394	12
VirusTotal Results	12
Entry Point Execution Flow Analysis	12
BinDiff Analysis	13
Binary: bfcc02219321d1047cc0330454a61f6b276d06f6	13
Virus Total Results	13
Entry Point Execution Flow Analysis	14
BinDiff Analysis	14
Initial Conclusion	15
Follow-Up Questions	15
Fuzzy Hashing	24
Phase II	26
PCRE	26
SPYEYE	26
ICE-IX	26
Zeus	27
Conclusion	27
E-mail Analysis	27
Appendix A	30
Appendix B	32

Appendix C..... 34
Appendix D..... 36
Appendix E..... 38
Appendix F..... 40

Executive Summary

Our analysis of over 70 binaries reveals a great deal of commonality between known copies of the Zeus Trojan and myriad binaries being distributed by malicious actors.

Our effort was broken down into three main phases. In the first phase we analyzed five PE binaries. Four of the five were packed using various means. We unpacked them and subjected them to a variety of analysis techniques in an attempt to connect them to the Zeus malware. In each case the results were highly probable that the binaries were in fact copies of Zeus.

In the second phase we were provided with several hundred binaries that were known or suspected to be related to the SpyEye, ICE-IX, and PCRE Trojans. Our analysis revealed that of the binaries we were able to analyze, each were highly similar to Zeus.

In the third and final phase we analyzed email messages sent by malicious actors that purported to be from the National Automated Clearing House Association, the trade organization for the ACH (direct deposit) system. These messages were designed to drive recipients to infect themselves with malware.

Phase I

We were provided 70 binaries, five of which were PE binaries. Of the five PE binaries, four were packed using various means. Those 4 were unpacked and the import tables were reconstructed for viewing in IDA Pro to determine the functionality of the executable. The unpacked binary:

2cc1076f3c6e65d0a59792b75370b04613258ffa

was used as a baseline for functionality because no modifications to the binary were needed. Every other packed executable was then compared against this baseline executable. Below are the binaries we are addressing in this paper:

- 2cc1076f3c6e65d0a59792b75370b04613258ffa (baseline)
- 0cc6215d31e5e639a19b4ceb3d57ce64d62e9b2c (packed version 1)
- 9b259bc255fef873f1e41629fb67c30f0c40e5dc (packed version 2)
- 1bfdc4f2cfa48a1f063d1826992fbaf5e2924394 (packed version 3)
- bfcc02219321d1047cc0330454a61f6b276d06f6 (packed version 4)

Binary: [2cc1076f3c6e65d0a59792b75370b04613258ffa](#)

This binary was not packed and we did not modify it before analyzing it. We are using it as our baseline for functional commonality. It contains the following functionality:

- HTTP communication capability
- Remote Process Injection. Uses WriteProcessMemory to inject executable code into a remote process. Generally this is either used by debuggers or malware. Since this binary has no debugger functionality, we assume the reason for its inclusion is malicious.
- Screenshot Capability. Allows this application to save and send back screenshots to the server. This allows an attacker to see what exactly is showing on the victim's screen.
- VNC-Type Server Functionality. Allows the attacker to control the mouse and keyboard of the victim's computer.
- Keyboard Logging Capabilities. Allows the attacker to send keystrokes to a server to get victim's passwords that are typed into the keyboard.

- Firefox Browser Logging. Hooks nspr4.dll to allow logging of all http and https activity to a file. This file is downloaded from the attacker to view all browsing activity.
- Windows mail download. Allows the attacker to view the victim's email if the user uses Windows Mail or Outlook Express.
- Self-Delete using a bat file.

Virus Total Results

Appendix A shows the results from Virus Total. When submitting the hash to virus total it is identified by most AVs as Zbot. 33 out of 43 engines detected this binary as malicious.

Entry Point Analysis

```
.text:0041A831      public start
.text:0041A831 start      proc near
.text:0041A831      = dword ptr -0Ch
.text:0041A831 var_C      = dword ptr -8
.text:0041A831 hObject    = dword ptr -4
.text:0041A831 pNumArgs

.text:0041A831      push     ebp
.text:0041A832      mov     ebp, esp
.text:0041A834      sub     esp, 0Ch
.text:0041A837      push     ebx
.text:0041A838      push     0
.text:0041A83A      xor     bl, bl
.text:0041A83C      call    sub_4199AE
.text:0041A841      test   al, al
.text:0041A843      jz     loc_41A8FE
.text:0041A849      push    8007h          ; uMode
.text:0041A84E      mov    byte ptr [ebp+var_C], bl
.text:0041A851      mov    byte ptr [ebp+hObject], 1
.text:0041A855      call   ds:SetErrorMode
.text:0041A85B      lea   eax, [ebp+pNumArgs]
.text:0041A85E      push  eax             ; pNumArgs
.text:0041A85F      call  ds:GetCommandLineW
.text:0041A865      push  eax             ; lpCmdLine
.text:0041A866      call  ds:CommandLineToArgvW
.text:0041A86C      test  eax, eax
.text:0041A86E      jz    short loc_41A8D9
.text:0041A870      xor   edx, edx
.text:0041A872      cmp   [ebp+pNumArgs], edx
.text:0041A875      jle   short loc_41A8AB
.text:0041A877      loc_41A877:          ; CODE XREF: start+78↓j
.text:0041A877      mov   ecx, [eax+edx*4]
.text:0041A87A      test  ecx, ecx
.text:0041A87C      jz    short loc_41A8A5
```

Figure 1 (2cc1076f3c6e65d0a59792b75370b04613258ffa Entry Point)

Figure 1 shows our baseline executable entry point. These are one of the metrics we used to determine if we had successfully unpacked a binary and to determine if two binaries came from the same code base. The code in Figure 1 essential just disables any error messages that may pop up during execution, and attempts to get any command line arguments.

Binary: 0cc6215d31e5e639a19b4ceb3d57ce64d62e9b2c

We unpacked this binary, and the others, using a manual combination of WinDbg, IDA Pro, and Imprec.

The first stage decoder is at 43E000, looks like it is copied to a virtual alloc'd buffer, in this case 0x9b0000. This buffer contains an MZ header and is stage2 of the decoder. We continued until we find another MZ header in a virtual alloc'd buffer, in this case we found that it does another iteration of decoding. Another virtual alloc'd buffer was found at a00000:

```
0:000> dc a00000
00a00000 6c385348 4b32686e 4f6f5a4e 50704364 HS8lnh2KNZoOdCpP
00a00010 45705864 3271775a 7058616c 55547043 dXpEZwq2laXpCpTU
00a00020 4c42674d 4549754c 6f68516f 6e445069 MgBLLuIEoQhoiPDn
00a00030 3234754f 59342f5a 2b30326c 31465636 Ou42Z/4Yl20+6VF1
00a00040 376f656d 7344524d 58564362 55477330 meo7MRDsbcVX0sGU
00a00050 686f7538 76423147 746a6163 36433841 8uohG1BvcajtA8C6
00a00060 71506461 78396f4f 4e4b4863 2b4c776b adPqOo9xcHKNkwL+
00a00070 33756f4f 53726642 74587773 63735a6a Oou3BfrSswXtjZsc
```

Another virtual alloc'd buffer:

```
0:000> db 00a50000
00a50000 58 50 58 41 58 43 58 4b-00 32 02 00 cc 33 01 00
XPXAXCXK.2...3..
00a50010 00 26 96 8e 70 00 17 f7-ec 05 bb ea f4 ff 94 01
.&..p.....
00a50020 2f 44 ef 7c e6 f5 d8 e8-08 04 cb d1 e8 7b d6 d9
/D.|.....{..
00a50030 98 f0 63 6c dd 0b 4b 4e-b9 fc a4 17 0c f0 54 53
..cl..KN.....TS
00a50040 3b b0 ae 1c 70 86 0f 1b-ae a2 22 07 9b b7 67 57
i...p....."...gW
00a50050 9a 97 04 02 e8 9b a9 7e-08 fc a7 7e 8a 9a 93 d3
.....~.....
00a50060 6f 46 7e 3b 8f 17 61 b1-62 4f 90 4f e8 48 8e 46
oF~i..a.bO.O.H.F
00a50070 48 76 78 70 fe 35 75 0c-d0 7a 82 c3 f3 17 9e e0
Hvxp.5u..z.....
```

...and another...

PROPRIETARY AND CONFIDENTIAL

```
0:000> db 00a10000
00a10000  4d 5a 90 00 03 00 00 00-04 00 00 00 ff ff 00 00
MZ.....
00a10010  b8 00 00 00 00 00 00 00-40 00 00 00 00 00 00 00
.....@.....
00a10020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
.....
00a10030  00 00 00 00 00 00 00 00-00 00 00 00 d8 00 00 00
.....
00a10040  0e 1f ba 0e 00 b4 09 cd-21 b8 01 4c cd 21 54 68
.....!..L.!Th
00a10050  69 73 20 70 72 6f 67 72-61 6d 20 63 61 6e 6e 6f  is program
canno
00a10060  74 20 62 65 20 72 75 6e-20 69 6e 20 44 4f 53 20  t be run in
DOS
00a10070  6d 6f 64 65 2e 0d 0d 0a-24 00 00 00 00 00 00 00
mode....$.
0:000> !vprot a10000
BaseAddress:      00a10000
AllocationBase:   00a10000
AllocationProtect: 00000004  PAGE_READWRITE
RegionSize:      00024000
State:           00001000  MEM_COMMIT
Protect:         00000004  PAGE_READWRITE
Type:            00020000  MEM_PRIVATE
0:000> .writemem C:\stage3.bin a10000 L24000
```

Stage3.bin is basically the same as the unpacked version. We finally got the unpacked version of this binary and were able to successful compare it with the baseline binary. We determined that it was compiled from the same source base as the baseline binary.

Virus Total Results

Appendix B shows the detailed VirusTotal results. A majority (28/43) of the AV engines in VirusTotal identified this binary as malicious.

Entry Point Execution Flow Comparison

```

.text:0041BEA6      public start
.text:0041BEA6      start
.text:0041BEA6      proc near
.text:0041BEA6      var_C              = dword ptr -0Ch
.text:0041BEA6      hObject           = dword ptr -8
.text:0041BEA6      pNumArgs          = dword ptr -4
.text:0041BEA6      push             ebp
.text:0041BEA7      mov              ebp, esp
.text:0041BEA9      sub              esp, 0Ch
.text:0041BEAC      push             ebx
.text:0041BEAD      xor              ecx, ecx
.text:0041BEAF      xor              bl, bl
.text:0041BEB1      call             sub_41B00B
.text:0041BEB6      test             al, al
.text:0041BEB8      jz               loc_41BF73
.text:0041BEBE      push             8007h ; uMode
.text:0041BEC3      mov              byte ptr [ebp+var_C], bl
.text:0041BEC6      mov              byte ptr [ebp+hObject], 1
.text:0041BECA      call             ds:SetErrorMode
.text:0041BED0      lea              eax, [ebp+pNumArgs]
.text:0041BED3      push             eax ; pNumArgs
.text:0041BED4      call             ds:GetCommandLineW
.text:0041BEDA      push             eax ; lpCmdLine
.text:0041BEDB      call             ds:CommandLineToArgvW
.text:0041BEE1      test             eax, eax
.text:0041BEE3      jz               short loc_41BF4E
.text:0041BEE5      xor              edx, edx
.text:0041BEE7      cmp              [ebp+pNumArgs], edx
.text:0041BEEA      jle              short loc_41BF20
.text:0041BEEC      loc_41BEEC: ; CODE XREF: start+78↓j
.text:0041BEEC      mov              ecx, [eax+edx*4]
.text:0041BEEF      test             ecx, ecx
.text:0041BEF1      jz               short loc_41BF1A

```

Figure 2 (0cc6215d31e5e639a19b4ceb3d57ce64d62e9b2c unpacked entry point)

Figure 2 is the entry point disassembled in IDA Pro. Notice how Figure 1 (baseline binary entry point) and Figure 2 are essentially identical even in the registers used. We determined from this analysis that we were on track to show the binaries were compiled from the same code base.

BinDiff Analysis

Zynamics BinDiff was used to do a full binary comparison between executables. It can quickly show functions that are identical using different methods like edge flowgraphs and call reference matching. Figure 3 shows a subset of the matched functions.

similarity	confidence	EA primary	name primary	EA secondary	name secondary	algorithm
1.00	0.99	00414EF4	sub_414EF4_416	00419FA6	sub_419FA6_1053	edges flowgraph MD index
1.00	0.99	00414F5C	sub_414F5C_417	0041A00E	sub_41A00E_1054	edges flowgraph MD index
1.00	0.99	00414FBB	sub_414FBB_418	0041A06D	sub_41A06D_1055	edges flowgraph MD index
1.00	0.99	0041507D	sub_41507D_419	00414E8A	sub_414E8A_955	edges flowgraph MD index
1.00	0.99	00415117	sub_415117_420	00414F24	sub_414F24_956	edges flowgraph MD index
1.00	0.99	0041519F	sub_41519F_421	00414FAC	sub_414FAC_957	edges flowgraph MD index
1.00	0.99	0041526C	sub_41526C_422	00415079	sub_415079_958	edges flowgraph MD index
1.00	0.99	0041550C	sub_41550C_423	0040E9B2	sub_40E9B2_856	call reference matching
1.00	0.99	00415558	sub_415558_424	0040E9FE	sub_40E9FE_857	call reference matching
1.00	0.99	004155DE	sub_4155DE_425	0040EA84	sub_40EA84_858	edges flowgraph MD index
1.00	0.99	00415603	sub_415603_426	0040EAA9	sub_40EAA9_859	prime signature matching
1.00	0.99	00415614	sub_415614_427	0040EABA	sub_40EABA_860	call reference matching
1.00	0.96	00415CFA	sub_415CFA_431	0040F1A0	sub_40F1A0_865	call reference matching
1.00	0.99	00415E16	sub_415E16_432	0040F2BC	sub_40F2BC_866	call reference matching
1.00	0.99	00416059	sub_416059_433	0040F4FF	sub_40F4FF_867	edges flowgraph MD index
1.00	0.99	0041625E	sub_41625E_434	0041DE05	sub_41DE05_1115	edges flowgraph MD index
1.00	0.99	004162FE	sub_4162FE_435	0041DEA5	sub_41DEA5_1116	edges flowgraph MD index
1.00	0.99	00416369	sub_416369_436	0041DF10	sub_41DF10_1117	edges callgraph MD index
1.00	0.99	00416886	sub_416886_437	00415A26	sub_415A26_964	edges flowgraph MD index
1.00	0.99	00416921	sub_416921_438	00415CA6	sub_415CA6_968	call reference matching
1.00	0.99	004169B3	sub_4169B3_439	00415D38	sub_415D38_969	call reference matching
1.00	0.99	00416A18	sub_416A18_440	00415D9D	sub_415D9D_970	call reference matching
1.00	0.99	00416AA9	sub_416AA9_441	00415E2E	sub_415E2E_971	call reference matching
1.00	0.99	00416CE0	sub_416CE0_442	00416065	sub_416065_972	edges flowgraph MD index
1.00	0.99	00416D47	sub_416D47_443	004160CC	sub_4160CC_973	call reference matching
1.00	0.99	00416DBA	sub_416DBA_444	00404E42	sub_404E42_564	edges flowgraph MD index
1.00	0.99	00416ED2	sub_416ED2_445	00404F5A	sub_404F5A_565	edges flowgraph MD index
1.00	0.99	004177A3	sub_4177A3_446	0040582B	sub_40582B_566	edges flowgraph MD index
1.00	0.99	00417819	sub_417819_447	004058A1	sub_4058A1_567	edges flowgraph MD index
1.00	0.99	00417A76	sub_417A76_448	00405AFE	sub_405AFE_568	edges flowgraph MD index

Figure 3 (0cc6215d31e5e639a19b4ceb3d57ce64d62e9b2c Bindiff against baseline)

In total, 905 functions were matched with BinDiff. 899 functions were matched with a similarity rating of 1.0 and confidence of greater than .9. To get this much similarity between these 2 binaries, they both must have been compiled from the same source code.

Binary: [9b259bc255fef873f1e41629fb67c30f0c40e5dc](#)

This binary needed to be unpacked to get its decoded contents. It was packed with the UPX packer.

Virus Total Results

22/43 anti-virus engines detected this binary as malicious. Appendix C has the detailed results.

Entry Point Execution Flow Analysis

```

UPX1:0041BEA6      public start
UPX1:0041BEA6 start      proc near
UPX1:0041BEA6      = dword ptr -0Ch
UPX1:0041BEA6      = dword ptr -8
UPX1:0041BEA6      = dword ptr -4
UPX1:0041BEA6      push     ebp
UPX1:0041BEA7      mov     ebp, esp
UPX1:0041BEA9      sub     esp, 0Ch
UPX1:0041BEAC      push     ebx
UPX1:0041BEAD      xor     ecx, ecx
UPX1:0041BEAF      xor     bl, bl
UPX1:0041BEB1      call    sub_41B00B
UPX1:0041BEB6      test   al, al
UPX1:0041BEB8      jz     loc_41BF73
UPX1:0041BEBE      push   8007h          ; uMode
UPX1:0041BEC3      mov    byte ptr [ebp+var_C], bl
UPX1:0041BEC6      mov    byte ptr [ebp+hObject], 1
UPX1:0041BECA      call   SetErrorMode
UPX1:0041BED0      lea   eax, [ebp+pNumArgs]
UPX1:0041BED3      push  eax           ; pNumArgs
UPX1:0041BED4      call  GetCommandLineW
UPX1:0041BEDA      push  eax           ; lpCmdLine
UPX1:0041BEDB      call  CommandLineToArgvW
UPX1:0041BEE1      test  eax, eax
UPX1:0041BEE3      jz    short loc_41BF4E
UPX1:0041BEE5      xor   edx, edx
UPX1:0041BEE7      cmp  [ebp+pNumArgs], edx
UPX1:0041BEEA      jle  short loc_41BF20
UPX1:0041BEEC      loc_41BEEC:          ; CODE XREF: start+78↓j
UPX1:0041BEEC      mov   ecx, [eax+edx*4]
UPX1:0041BEEF      test  ecx, ecx
UPX1:0041BEF1      jz    short loc_41BF1A
    
```

Figure 4 (9b259bc255fef873f1e41629fb67c30f0c40e5dc unpacked entry point)

After unpacking Figure 4 shows the same resemblance. Comparing Figure 1 and Figure 4 shows that the entry points are identical.

BinDiff Analysis

We used Zynamics BinDiff to compare this binary against our baseline.

similarity	confidence	EA primary	name primary	EA secondary	name secondary	algorithm
1.00	0.99	0041B61C	sub_41B61C_504	004148DD	sub_4148DD_942	edges flowgraph MD index
1.00	0.99	0041B670	sub_41B670_505	00414931	sub_414931_943	MD index matching (callGraph MD index, top down)
1.00	0.99	0041B70C	sub_41B70C_506	004149CD	sub_4149CD_944	MD index matching (callGraph MD index, top down)
1.00	0.99	0041B7A8	sub_41B7A8_507	00414A69	sub_414A69_945	edges flowgraph MD index
1.00	0.99	0041B7EB	sub_41B7EB_508	00414AAC	sub_414AAC_946	edges flowgraph MD index
1.00	0.99	0041B82A	sub_41B82A_509	00414AEB	sub_414AEB_947	edges flowgraph MD index
1.00	0.99	0041B874	sub_41B874_510	00414B35	sub_414B35_948	edges flowgraph MD index
1.00	0.99	0041B8A0	sub_41B8A0_511	00414B61	sub_414B61_949	edges flowgraph MD index
1.00	0.99	0041B8F4	sub_41B8F4_512	004170F7	sub_4170F7_1008	edges flowgraph MD index
1.00	0.96	0041B9BB	sub_41B9BB_513	004171BE	sub_4171BE_1009	call reference matching
1.00	0.96	0041B9F0	sub_41B9F0_514	004171F3	sub_4171F3_1010	call reference matching
1.00	0.99	0041BA26	sub_41BA26_515	00417229	sub_417229_1011	edges flowgraph MD index
1.00	0.99	0041BA75	sub_41BA75_516	00417278	sub_417278_1012	edges flowgraph MD index
1.00	0.99	0041BBA0	sub_41BBA0_517	004173A3	sub_4173A3_1013	edges flowgraph MD index
1.00	0.99	0041BC76	sub_41BC76_518	00417479	sub_417479_1014	edges flowgraph MD index
1.00	0.99	0041BCDE	sub_41BCDE_519	004174E1	sub_4174E1_1015	edges flowgraph MD index
1.00	0.99	0041BD68	sub_41BD68_520	0041756B	sub_41756B_1016	edges callgraph MD index
1.00	0.99	0041C130	sub_41C130_521	00417933	sub_417933_1017	edges flowgraph MD index
1.00	0.99	0041C31E	sub_41C31E_522	00417B21	sub_417B21_1018	edges flowgraph MD index
1.00	0.99	0041C7B6	sub_41C7B6_523	00417FB9	sub_417FB9_1019	edges flowgraph MD index
1.00	0.96	0041CC6F	sub_41CC6F_524	00418472	sub_418472_1020	call reference matching
1.00	0.96	0041CCB0	sub_41CCB0_525	004184B3	sub_4184B3_1021	call reference matching
1.00	0.99	0041CCD9	sub_41CCD9_526	004184DC	sub_4184DC_1022	edges flowgraph MD index
1.00	0.99	0041CCFC	sub_41CCFC_527	004184FF	sub_4184FF_1023	edges flowgraph MD index
1.00	0.99	0041CF13	sub_41CF13_528	00418716	sub_418716_1024	edges callgraph MD index
1.00	0.99	0041CF87	sub_41CF87_529	00414BB5	sub_414BB5_950	edges flowgraph MD index
1.00	0.99	0041CFE6	sub_41CFE6_530	00414C14	sub_414C14_951	edges flowgraph MD index
1.00	0.99	0041D0E4	sub_41D0E4_531	0040E874	sub_40E874_854	call reference matching
1.00	0.99	0041D11A	sub_41D11A_532	0040E8AA	sub_40E8AA_855	call reference matching
1.00	0.99	0041D164	sub_41D164_533	0041898D	sub_41898D_1027	edges flowgraph MD index

Figure 5 (9b259bc255fef873f1e41629fb67c30f0c40e5dc BinDiff against baseline)

898 function out of 907 functions matched had a similarity rating of 1.0 and confidence of greater than 0.92. This binary is virtually identical to the baseline and both come from the same code base.

Binary: [1bfdc4f2cfa48a1f063d1826992fbaf5e2924394](#)

VirusTotal Results

Appendix D contains the detailed results from VirusTotal. 20 out of 43 anti-virus engines in VirusTotal identified this binary as malicious.

Entry Point Execution Flow Analysis

```

.data:00413C42      public start
.data:00413C42 start      proc near
.data:00413C42      var_C                = dword ptr -0Ch
.data:00413C42      var_8                = dword ptr -8
.data:00413C42      pNumArgs             = dword ptr -4
.data:00413C42
.data:00413C42      push     ebp
.data:00413C43      mov     ebp, esp
.data:00413C45      sub     esp, 0Ch
.data:00413C48      push     ebx
.data:00413C49      push     0
.data:00413C4B      xor     bl, bl
.data:00413C4D      call    near ptr unk_412D44
.data:00413C52      test    al, al
.data:00413C54      jz     loc_413D0F
.data:00413C5A      push    8007h          ; uMode
.data:00413C5F      mov     byte ptr [ebp+var_C], bl
.data:00413C62      mov     byte ptr [ebp+var_8], 1
.data:00413C66      call    ds:SetErrorMode
.data:00413C6C      lea    eax, [ebp+pNumArgs]
.data:00413C6F      push    eax            ; pNumArgs
.data:00413C70      call    ds:GetCommandLineW
.data:00413C76      push    eax            ; lpCmdLine
.data:00413C77      call    ds:CommandLineToArgvW
.data:00413C7D      test   eax, eax
.data:00413C7F      jz     short loc_413CEA
.data:00413C81      xor     edx, edx
.data:00413C83      cmp    [ebp+pNumArgs], edx
.data:00413C86      jle    short loc_413CBC
.data:00413C88
.data:00413C88 loc_413C88:                ; CODE XREF: start+78↓j
.data:00413C88      mov     ecx, [eax+edx*4]
.data:00413C8B      test   ecx, ecx
.data:00413C8D      jz     short loc_413CB6

```

Figure 6 (1bfdc4f2cfa48a1f063d1826992fbaf5e2924394 entry point)

Comparing Figure 1 and Figure 6 we see the code at their entry points are identical.

BinDiff Analysis

Due to the packer for this binary, BinDiff could not properly compare this binary against the baseline.

Binary: [bfcc02219321d1047cc0330454a61f6b276d06f6](#)

Virus Total Results

Appendix E contains the detailed results from VirusTotal. 27 out of 43 anti-virus engines in VirusTotal identified this binary as malicious.

Entry Point Execution Flow Analysis

```

UPX1:0041BEA6      public start
UPX1:0041BEA6      start      proc near
UPX1:0041BEA6      var_C      = dword ptr -0Ch
UPX1:0041BEA6      hObject    = dword ptr -8
UPX1:0041BEA6      pNumArgs   = dword ptr -4
UPX1:0041BEA6      push      ebp
UPX1:0041BEA7      mov       ebp, esp
UPX1:0041BEA9      sub       esp, 0Ch
UPX1:0041BEAC      push      ebx
UPX1:0041BEAD      xor       ecx, ecx
UPX1:0041BEAF      xor       bl, bl
UPX1:0041BEB1      call     sub_41B00B
UPX1:0041BEB6      test     al, al
UPX1:0041BEB8      jz       loc_41BF73
UPX1:0041BEBE      push     8007h          ; uMode
UPX1:0041BEC3      mov     byte ptr [ebp+var_C], bl
UPX1:0041BEC6      mov     byte ptr [ebp+hObject], 1
UPX1:0041BECA      call     SetErrorMode
UPX1:0041BED0      lea     eax, [ebp+pNumArgs]
UPX1:0041BED3      push    eax            ; pNumArgs
UPX1:0041BED4      call     GetCommandLineW
UPX1:0041BEDA      push    eax            ; lpCmdLine
UPX1:0041BEDB      call     CommandLineToArgvW
UPX1:0041BEE1      test    eax, eax
UPX1:0041BEE3      jz      short loc_41BF4E
UPX1:0041BEE5      xor     edx, edx
UPX1:0041BEE7      cmp     [ebp+pNumArgs], edx
UPX1:0041BEEA      jle     short loc_41BF20
UPX1:0041BEEC      loc_41BEEC:          ; CODE XREF: start+78↓j
UPX1:0041BEEC      mov     ecx, [eax+edx*4]
UPX1:0041BEEF      test    ecx, ecx
UPX1:0041BEF1      jz      short loc_41BF1A

```

Figure 7 (bfcc02219321d1047cc0330454a61f6b276d06f6 entry point)

Comparing Figure 1 and Figure 7 we see the code at their entry points are identical.

BinDiff Analysis

similarity	confidence	EA primary	name primary	EA secondary	name secondary	algorithm
1.00	0.99	00414CD1	sub_414CD1_414	00419D83	sub_419D83_1051	edges callgraph MD index
1.00	0.99	00414DB2	sub_414DB2_415	00419E64	sub_419E64_1052	edges flowgraph MD index
1.00	0.99	00414EF4	sub_414EF4_416	00419FA6	sub_419FA6_1053	edges flowgraph MD index
1.00	0.99	00414F5C	sub_414F5C_417	0041A00E	sub_41A00E_1054	edges flowgraph MD index
1.00	0.99	00414FBB	sub_414FBB_418	0041A06D	sub_41A06D_1055	edges flowgraph MD index
1.00	0.99	0041507D	sub_41507D_419	00414E8A	sub_414E8A_955	edges flowgraph MD index
1.00	0.99	00415117	sub_415117_420	00414F24	sub_414F24_956	edges flowgraph MD index
1.00	0.99	0041519F	sub_41519F_421	00414FAC	sub_414FAC_957	edges flowgraph MD index
1.00	0.99	0041526C	sub_41526C_422	00415079	sub_415079_958	edges flowgraph MD index
1.00	0.99	0041550C	sub_41550C_423	0040E9B2	sub_40E9B2_856	call reference matching
1.00	0.99	00415558	sub_415558_424	0040E9FE	sub_40E9FE_857	call reference matching
1.00	0.99	004155DE	sub_4155DE_425	0040EA84	sub_40EA84_858	edges flowgraph MD index
1.00	0.99	00415603	sub_415603_426	0040EAA9	sub_40EAA9_859	prime signature matching
1.00	0.99	00415614	sub_415614_427	0040EABA	sub_40EABA_860	call reference matching
1.00	0.96	00415CFA	sub_415CFA_431	0040F1A0	sub_40F1A0_865	call reference matching
1.00	0.99	00415E16	sub_415E16_432	0040F2BC	sub_40F2BC_866	call reference matching
1.00	0.99	00416059	sub_416059_433	0040F4FF	sub_40F4FF_867	edges flowgraph MD index
1.00	0.99	0041625E	sub_41625E_434	0041DE05	sub_41DE05_1115	edges flowgraph MD index
1.00	0.99	004162FE	sub_4162FE_435	0041DEA5	sub_41DEA5_1116	edges flowgraph MD index
1.00	0.99	00416369	sub_416369_436	0041DF10	sub_41DF10_1117	edges callgraph MD index
1.00	0.99	00416886	sub_416886_437	00415A26	sub_415A26_964	edges flowgraph MD index
1.00	0.99	00416921	sub_416921_438	00415CA6	sub_415CA6_968	call reference matching
1.00	0.99	004169B3	sub_4169B3_439	00415D38	sub_415D38_969	call reference matching
1.00	0.99	00416A18	sub_416A18_440	00415D9D	sub_415D9D_970	call reference matching
1.00	0.99	00416AA9	sub_416AA9_441	00415E2E	sub_415E2E_971	call reference matching
1.00	0.99	00416CE0	sub_416CE0_442	00416065	sub_416065_972	edges flowgraph MD index
1.00	0.99	00416D47	sub_416D47_443	004160CC	sub_4160CC_973	call reference matching
1.00	0.99	00416DBA	sub_416DBA_444	00404E42	sub_404E42_564	edges flowgraph MD index
1.00	0.99	00416ED2	sub_416ED2_445	00404F5A	sub_404F5A_565	edges flowgraph MD index

Figure 8 (bfcc02219321d1047cc0330454a61f6b276d06f6 BinDiff against baseline)

899 out of 907 function had a similarity rating of 1.0 with confidence rating greater than 0.88. This binary is nearly identical to the baseline and must have been compiled from the same code base.

Initial Conclusion

After using entry point analysis and bindiff on the unpacked version of the binaries we are able to conclude that all 5 binaries were compiled from the same code base.

Follow-Up Questions

1. Are these binaries similar to Zeus, and if so, how similar?
2. Were these binaries compiled with a Microsoft toolchain, and what evidence supports this?

Fortunately, copies of the source code to Zeus have been made publicly available. Our manual analysis of the recovered applications revealed many structural similarities (see figure 9), but do these structural similarities originate from Zeus? To answer this question, we compiled our own copy of Zeus and compared our copy to each of the programs described so far.

We first compiled Zeus in the 'release' configuration with symbols and compared it to the unpacked version we were given with BinDiff.

similarity	confidence	EA primary	name primary	EA secondary	name secondary	algorithm
1.00	0.98	0040B33A	VncServer::hookerCallWindowProcA(long (*)(HWND_ *,uint,uint,...	00406DFF	sub_406DFF_257	address sequence
1.00	0.98	0040B2F1	VncServer::hookerCallWindowProcW(long (*)(HWND_ *,uint,uint,...	00406DB6	sub_406DB6_256	address sequence
1.00	0.98	0040B18D	VncServer::hookerDefDlgProcA(HWND_ *,uint,uint,long)	00406C52	sub_406C52_251	address sequence
1.00	0.98	0040B147	VncServer::hookerDefDlgProcW(HWND_ *,uint,uint,long)	00406C0C	sub_406C0C_250	address sequence
1.00	0.98	0040B21C	VncServer::hookerDefFrameProcA(HWND_ *,HWND_ *,uint,uint,...	00406CE1	sub_406CE1_253	address sequence
1.00	0.98	0040B1D3	VncServer::hookerDefFrameProcW(HWND_ *,HWND_ *,uint,uint,...	00406C98	sub_406C98_252	address sequence
1.00	0.98	0040B2AB	VncServer::hookerDefMDIChildProcA(HWND_ *,uint,uint,long)	00406D70	sub_406D70_255	address sequence
1.00	0.98	0040B265	VncServer::hookerDefMDIChildProcW(HWND_ *,uint,uint,long)	00406D2A	sub_406D2A_254	address sequence
1.00	0.98	0040B101	VncServer::hookerDefWindowProcA(HWND_ *,uint,uint,long)	00406BC6	sub_406BC6_249	address sequence
1.00	0.99	0040B08B	VncServer::hookerDefWindowProcW(HWND_ *,uint,uint,long)	00406B80	sub_406B80_248	MD index matching (callGraph MD index, top down)
1.00	0.99	0040D20C	VncServer::hookerEndPaint(HWND_ *,tagPAINTSTRUCT const *)	00412729	sub_412729_570	MD index matching (callGraph MD index, top down)
1.00	0.99	00409C81	VncServer::hookerGetCapture(void)	0041DAD1	sub_41DAD1_737	edges flowgraph MD index
1.00	0.99	00409B53	VncServer::hookerGetCursorPos(tagPOINT *)	0041D9A3	sub_41D9A3_733	MD index matching (flowgraph MD index, top down)
1.00	0.99	0040D2A7	VncServer::hookerGetDc(HWND_ *)	004127C4	sub_4127C4_572	MD index matching (callGraph MD index, top down)
1.00	0.99	0040D24C	VncServer::hookerGetDcEx(HWND_ *,HRGN_ *,ulong)	00412769	sub_412769_571	prime signature matching
1.00	0.97	00409D48	VncServer::hookerGetMessageA(tagMSG *,HWND_ *,uint,uint)	0041DB98	sub_41DB98_740	MD index matching (callGraph MD index, top down)
1.00	0.99	00409B21	VncServer::hookerGetMessagePos(void)	0041D971	sub_41D971_732	prime signature matching
1.00	0.97	00409D20	VncServer::hookerGetMessageW(tagMSG *,HWND_ *,uint,uint)	0041DB70	sub_41DB70_739	MD index matching (callGraph MD index, top down)
1.00	0.99	0040D365	VncServer::hookerGetUpdateRect(HWND_ *,tagRECT *,int)	00412882	sub_412882_575	MD index matching (flowgraph MD index, top down)
1.00	0.99	0040D3F8	VncServer::hookerGetUpdateRgn(HWND_ *,HRGN_ *,int)	00412915	sub_412915_576	prime signature matching
1.00	0.99	0040D2E6	VncServer::hookerGetWindowDc(HWND_ *)	00412803	sub_412803_573	MD index matching (callGraph MD index, top down)
1.00	0.99	0040B04D	VncServer::hookerOpenInputDesktop(ulong,int,ulong)	00406B12	sub_406B12_246	MD index matching (flowgraph MD index, top down)
1.00	0.97	00409D9B	VncServer::hookerPeekMessageA(tagMSG *,HWND_ *,uint,uint,u...	0041DBEB	sub_41DBEB_742	MD index matching (callGraph MD index, top down)
1.00	0.97	00409D70	VncServer::hookerPeekMessageW(tagMSG *,HWND_ *,uint,uint,...	0041DBC0	sub_41DBC0_741	MD index matching (callGraph MD index, top down)
1.00	0.98	0040B40C	VncServer::hookerRegisterClassA(tagWNDCLASSA *)	00406ED1	sub_406ED1_260	call sequence matching(sequence)
1.00	0.98	0040B4AB	VncServer::hookerRegisterClassExA(tagWNDCLASSEXA *)	00406F70	sub_406F70_262	call sequence matching(sequence)
1.00	0.98	0040B459	VncServer::hookerRegisterClassExW(tagWNDCLASSEXW *)	00406F1E	sub_406F1E_261	call sequence matching(sequence)
1.00	0.98	0040B3BF	VncServer::hookerRegisterClassW(tagWNDCLASSW *)	00406E84	sub_406E84_259	call sequence matching(sequence)
1.00	0.99	00409C31	VncServer::hookerReleaseCapture(void)	0041DA81	sub_41DA81_736	edges flowgraph MD index

Figure 9 (Compiled Zeus with Symbols against baseline)

BinDiff shows us the baseline and our compiled version is identical. 895 total functions were matched. 703 of those were functions had an associated symbol name. 698 out of the 895 matched functions had a similarity rating of 1.00 and confidence value of 0.92 or greater. In other words: our samples are compiled versions of Zeus.

Next we searched for functions within our copy of Zeus that had a very low probability of being duplicated or copied by accident. We chose the screenshot logic, the API interception logic, and VNC server implementation. In every case, there was an exact or extremely high match in the control flow graph between our copy of Zeus and the programs that we analyzed.

Programs are defined by a sequence of statements. Each statement is an instruction to perform a discrete operation. These statements are linked together into a graph. At every point where a program could do one thing or another, a statement can conditionally branch to a new node in the graph. In this way, any logical instructions can be represented by computer code.

We used the Interactive Disassembler (IDA) to find and extract control flow graphs from each of the applications we were given and also the copy of Zeus that we compiled. Below are these graphs displayed:

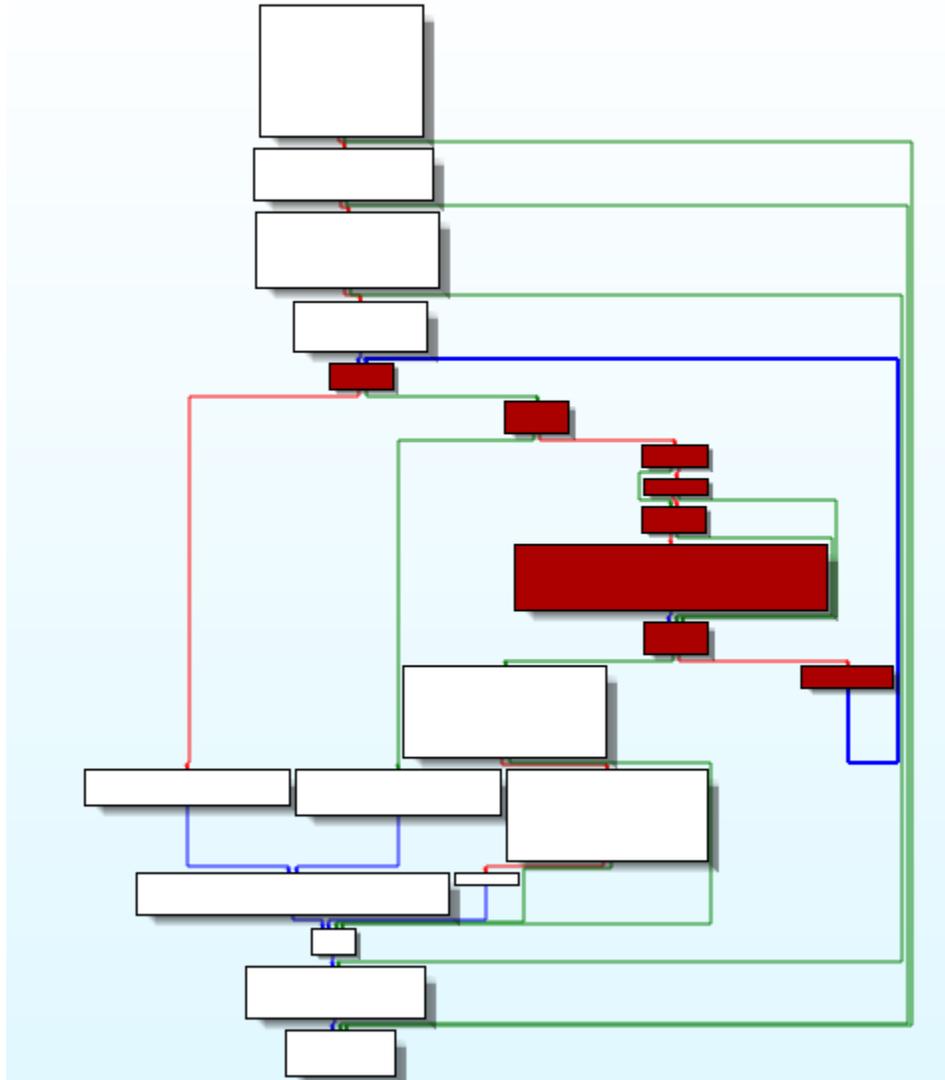


Figure 10 (Our Compiled Zeus)

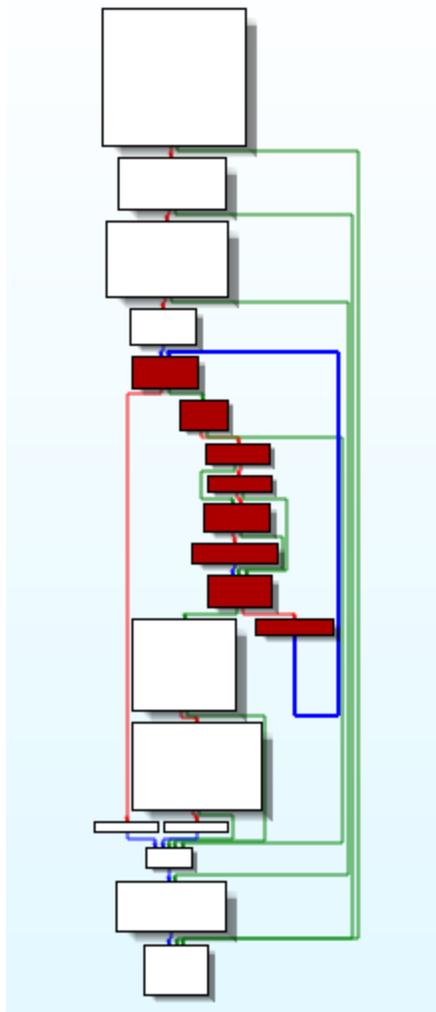


Figure 12 (9b259bc255fef873f1e41629fb67c30f0c40e5dc hooking function)

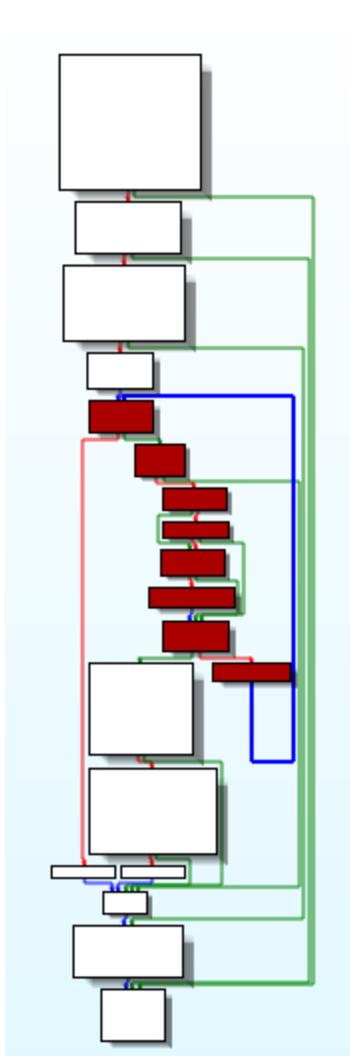


Figure 13 (0cc6215d31e5e639a19b4ceb3d57ce64d62e9b2c hooking function)

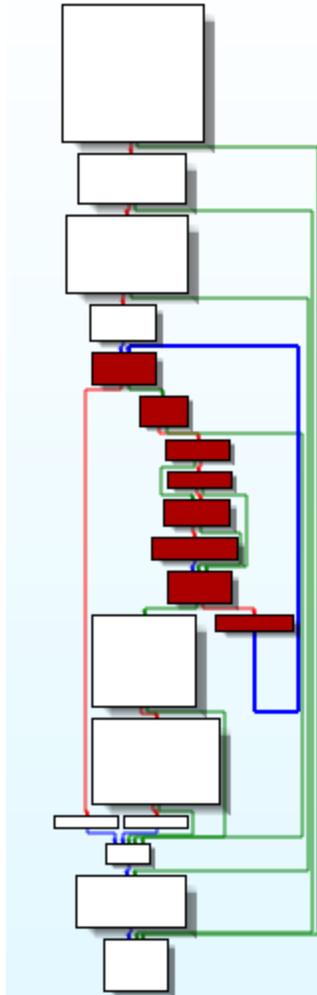


Figure 14 (1bfdc4f2cfa48a1f063d1826992fbaf5e2924394 hooking function)

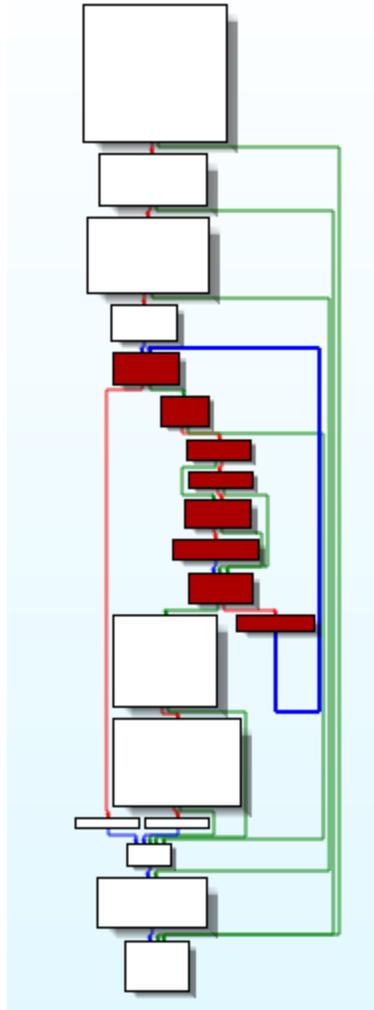


Figure 15 (bfcc02219321d1047cc0330454a61f6b276d06f6 hooking function)

We have highlighted in red all of the blocks that belong to a loop. Note that the structure of this function is identical across each program. Each instance has a single loop and the same sequence of tests. This function matches the function named `WaHook::_hook` (Appendix G).

This source code is responsible for detouring APIs to hook routines supplied by Zeus. These hook routines change the behavior of the operating system.

Another thing these similarities tell us is that it is highly likely that Microsoft compilers were used to build this version of Zeus. We built Zeus with a Microsoft compiler, and the following code was produced (from the above function):

```

lea    eax, [ebp+f10ldProtect]
push   eax           ; lpf10ldProtect
push   40h           ; flNewProtect
push   1Eh           ; dwSize
push   [ebp+lpBaseAddress] ; lpAddress
push   0FFFFFFFFh    ; hProcess
call   ds: __imp__VirtualProtectEx@20 ; VirtualProtectEx(x,x,x,x,x)
test   eax, eax
jz     loc_4170C1
    
```

The “push” statements are used to pass arguments to the call to the function “VirtualProtect. We can find this exact block in each of the other programs control flow graphs for this function:

```

lea    eax, [ebp+f10ldProtect]
push   eax           ; lpf10ldProtect
push   40h           ; flNewProtect
push   1Eh           ; dwSize
push   [ebp+lpBaseAddress] ; lpAddress
push   0FFFFFFFFh    ; hProcess
call   VirtualProtectEx
test   eax, eax
jz     loc_40C076
    
```

...and they are identical.

We compiled this function from source code using the gcc compiler. The Intel assembly language is very expressive and multiple statements are functionally equivalent to each other. Which statements are used is a choice that the compiler makes when it compiles the program. The choices that compilers make are generally quite different. Here is the resulting assembly code for the above snippet as produced by gcc:

```

mov    eax, [esp+7Ch+var_C]
mov    ecx, [esp+7Ch+lpAddress]
lea    esi, [esp+7Ch+f10ldProtect]
mov    [esp+7Ch+lpf10ldProtect], esi ; lpf10ldProtect
mov    [esp+7Ch+lpBaseAddress], ecx ; lpAddress
mov    [esp+7Ch+hProcess], eax ; hProcess
mov    [esp+7Ch+f1NewProtect], 40h ; '@' ; flNewProtect
mov    [esp+7Ch+lpBuffer], 1Eh ; dwSize
call   dword ptr ds: __imp__VirtualProtectEx@20 ; VirtualProtectEx(x,x,x,x,x)
sub    esp, 14h
test   eax, eax
jz     loc_2BB
    
```

The structure is radically different. Note that no “push” instructions are used. However, the resulting code is still functionally equivalent.

We also performed a mechanized comparison of the structure of the control flow graphs in each of the five programs, comparing the structure to that of the Zeus binary we built from source. We would have a program perform static control flow reconstructions from the program images, and then use a very simple algorithm to discover functions within the program. Once it discovered functions within the program, it extracts them into an intermediate form that can be analyzed with the NetworkX graph analysis library.

We asked NetworkX which graphs in each program were identical to other graphs. The results are below

Program	# of functions identified	Functions in Zeus matching functions in those programs
Compiled Zeus	154	-
0ccstage3.bin	139	125
1bfddump_.bin	100	103
9b25dump_.bin	139	125
bfccdump_.bin	139	125
2cc1076f3c6e65d0a59792b75370b04613258ffa	138	125

This shows that for the functions we identified in these binaries, almost all of them are structurally identical to functions that are within Zeus.

Fuzzy Hashing

We used the industry standard 'fuzzy' hashing technique via the ssdeep program to compare the unpacked binaries. The fuzzy hashing method works on byte-level similarity. It can be confused by function reordering and other simple obfuscation techniques.

Three of the files we analyzed, 9b25dump_.bin, 0ccstage3.bin, and bfccdump_.bin, were found to be similar to each other using fuzzy hashing. This result gives us a

high degree of confidence that these three files are essentially the same. They have large stretches of identical patterns of bytes.

Phase II

We were given another set of binaries and asked to analyze and compare the capabilities of a sample from all three sets of binaries.

PCRE

This sample set contained no valid win32 applications to analyze. These binaries are likely encoded with a password that was not provided.

SPYEYE

In this sample set we chose the file with the hash:

b33064449295083dbfec12634523d805

because the first layer of obfuscation was UPX which, due to time constraints, reduced the amount of time required to get the original binary. This file was a valid win32 application, but had two layers of obfuscation. The first layer was UPX. The second layer was not determined, but we were able to extract a binary that closely resembles the original. After some reverse engineering the capabilities of this binary are:

- Window enumeration
- Take screenshot of desktop
- Retrieve clipboard data
- keyboard logging
- Retrieve system information
- Communicate with C&C server using HTTP
- Enumerate user accounts
- File Search
- Remote process code injection
- Manipulate Windows registry
- Process enumeration
- Read arbitrary file contents
- Standard TCP socket communication
- Download and execute payloads

ICE-IX

In this sample we chose the file with the hash:

3c6839c4ce744c9c0ddf2ba06963c3f4

Because it was not obfuscated. After reverse engineering the binary the capabilities of this binary are:

- Take screenshot of desktop
- Remote process code injection
- Retrieve system information
- User account enumeration
- keyboard logging
- Process enumeration
- File search capability
- Get contents of arbitrary file
- Encrypt/Decrypt data using the Windows crypto API
- Manipulate Windows registry
- Communicate with C&C via HTTP
- Standard TCP socket communication
- Download and execute payloads

Zeus

Taking a closer look at our Zeus binaries and comparing them to Spyeye and ICE-IX functionality is very similar. Here is a list of the functionality Zeus supports

- Take screenshot of desktop
- Remote process code injection
- Retrieve system information
- keyboard logging
- VNC server
- HTTP injection
- Communicate with C&C via HTTP
- Download and execute payloads
- Process enumeration
- Self delete using bat file
- intercept Windows API functions
- Manipulate Windows registry

Conclusion

Based on the functionality of all the samples we analyzed, they all had a very similar set of capabilities that can be attributed to malicious intent.

E-mail Analysis

We were also given e-mails that had been sent purporting to be from NACHA but had actually originated from malware authors. The e-mails were sent with the intent

to compromise computers of the recipients. These e-mails are easy to find as their subject and body contain references to an ACH payment being rejected. The e-mails direct the recipient to a URL that the e-mail states is a Microsoft Word document providing more information. Actually, the URL in the e-mail is a link to a website that hosts malicious software.

For example, in an e-mail (file 11905A7A-00000B01.eml), an e-mail originally sent on Thu, 15 Feb 2012 with the subject "Your ACH transfer" purports to inform the recipient that an ACH transaction involving their account has failed.

It contains a URL to a "Transaction report":

```
<td><font face="Verdana"><a href="http://kurabiyeji.com/JXt8y6Au/index.html"
>report_7429595642193.doc</a> (Microsoft Word Document) </font></td>
```

The content hosted at this URL is known to be malicious by VirusTotal. The [clean-mx.de](http://support.clean-mx.de/clean-mx/viruses.php?domain=kurabiyeji.com&sort=first%20desc) database reports that the URL is known to be malicious:



<http://support.clean-mx.de/clean-mx/viruses.php?domain=kurabiyeji.com&sort=first%20desc>

VirusTotal reports the file served by that domain is malicious and detected as:

Antivirus	Result	Update
nProtect	Trojan.Agent.AUIJ	20120222
CAT-QuickHeal	-	20120222
McAfee	-	20120223
K7AntiVirus	-	20120222
TheHacker	-	20120222
VirusBuster	-	20120222
NOD32	JS/TrojanDownloader.HackLoad.AH	20120223
F-Prot	JS/Redir.IO	20120222
Symantec	-	20120223
Norman	-	20120222
ByteHero	-	20120225
TrendMicro-HouseCall	-	20120223
Avast	HTML:Script-inf	20120223
eSafe	-	20120221
ClamAV	-	20120223
Kaspersky	Trojan.HTML.Redirector.z	20120223
BitDefender	Trojan.Agent.AUIJ	20120223
ViRobot	-	20120222
Emsisoft	Trojan.HTML.Redirector!IK	20120223
Comodo	UnclassifiedMalware	20120223
F-Secure	Trojan.Agent.AUIJ	20120223

PROPRIETARY AND CONFIDENTIAL

DrWeb	-	20120223
VIPRE	-	20120222
AntiVir	-	20120222
TrendMicro	-	20120222
McAfee-GW-Edition	-	20120222
Sophos	Mal/JSRedir-H	20120223
eTrust-Vet	-	20120222
Jiangmin	-	20120222
Antiy-AVL	-	20120213
Microsoft	Trojan:JS/BlacoleRef.AA	20120222
SUPERAntiSpyware	-	20120206
Prevx	-	20120227
GData	Trojan.Agent.AUIJ	20120223
AhnLab-V3	JS/Blacoleref	20120222
VBA32	-	20120222
PCTools	-	20120221
Rising	-	20120223
Ikarus	Trojan.HTML.Redirector	20120223
Fortinet	-	20120223
AVG	-	20120223
Panda	-	20120222

Appendix A

Binary: 2cc1076f3c6e65d0a59792b75370b04613258ffa Virus Total Results

AhnLab-V3	Trojan/Win32.Zbot	20120107
AntiVir	TR/Hijacker.Gen	20120106
Antiy-AVL	Trojan/Win32.Zbot.gen	20120107
Avast	Win32:Zbot-NRC [Trj]	20120107
AVG	PSW.Generic9.AUZR	20120108
BitDefender	Gen:Variant.Kazy.1779	20120108
ByteHero	Trojan.Win32.Heur.Gen	20111231
CAT-QuickHeal	-	20120107
ClamAV	Trojan.Spy.Zbot-142	20120107
Commtouch	W32/Zbot.BR.gen!Eldorado	20120107
Comodo	UnclassifiedMalware	20120107
DrWeb	Trojan.PWS.Panda.1545	20120108
Emsisoft	Trojan-Spy.Win32.Zbot!IK	20120108
eSafe	-	20120103
eTrust-Vet	Win32/Zbot.CXZ	20120106
F-Prot	W32/Zbot.BR.gen!Eldorado	20120107
F-Secure	Gen:Variant.Kazy.1779	20120108
Fortinet	W32/Zbot.AT!tr	20120107
GData	Gen:Variant.Kazy.1779	20120108
Ikarus	Trojan-Spy.Win32.Zbot	20120107
Jiangmin	-	20120107
K7AntiVirus	Riskware	20120106

PROPRIETARY AND CONFIDENTIAL

Kaspersky	Trojan-Spy.Win32.Zbot.ctaq	20120108
McAfee	PWS-Zbot.gen.ds	20120108
McAfee-GW-Edition	PWS-Zbot.gen.ds	20120107
Microsoft	PWS:Win32/Zbot.gen!Y	20120107
NOD32	Win32/Spy.Zbot.YW	20120108
Norman	W32/Zbot.VAL	20120107
nProtect	Gen:Variant.Kazy.1779	20120107
Panda	Generic Trojan	20120107
PCTools	-	20120108
Prevx	-	20120108
Rising	-	20120106
Sophos	Troj/PWS-BSF	20120107
SUPERAntiSpyware	-	20120107
Symantec	-	20120108
TheHacker	-	20120106
TrendMicro	TROJ_GEN.FFFCBLU	20120107
TrendMicro-HouseCall	TROJ_GEN.FFFCBLU	20120108
VBA32	SScope.Trojan.FakeAV.01110	20120106
VIPRE	Trojan-Spy.Win32.Zbot.val (v)	20120108
ViRobot	-	20120107
VirusBuster	TrojanSpy.Zbot!/ky2LKcfC2c	20120107

Appendix B

Binary: 0cc6215d31e5e639a19b4ceb3d57ce64d62e9b2c Virus Total Results

AhnLab-V3	Trojan/Win32.FakeAV	20120102
AntiVir	TR/Kazy.48131.4	20120102
Antiy-AVL	Trojan/Win32.Injector.gen	20120102
Avast	Win32:MalOb-HP [Cryp]	20120102
AVG	Generic26.ZLQ	20120102
BitDefender	Gen:Variant.Kazy.48131	20120102
ByteHero	-	20111231
CAT-QuickHeal	-	20120102
ClamAV	-	20120102
Commtouch	-	20120102
Comodo	Heur.Suspicious	20120102
DrWeb	-	20120102
Emsisoft	Trojan-Spy.Win32.SpyEyes!IK	20120102
eSafe	Win32.TRKazy	20120101
eTrust-Vet	-	20120102
F-Prot	-	20120102
F-Secure	Gen:Variant.Kazy.48131	20120102
Fortinet	W32/Rorpian.C!tr	20120102
GData	Gen:Variant.Kazy.48131	20120102
Ikarus	Trojan-Spy.Win32.SpyEyes	20111231
Jiangmin	-	20120101
K7AntiVirus	Trojan	20120102

PROPRIETARY AND CONFIDENTIAL

Kaspersky	Trojan-Dropper.Win32.Injector.aiaz	20120102
McAfee	Artemis!98E1ECD8C6D7	20120102
McAfee-GW-Edition	Artemis!98E1ECD8C6D7	20120101
Microsoft	PWS:Win32/Zbot	20120102
NOD32	a variant of Win32/Kryptik.XDP	20120102
Norman	W32/Suspicious_Gen2.UDXVY	20120102
nProtect	-	20120102
Panda	Trj/CL.A	20120102
PCTools	-	20120102
Prevx	-	20120102
Rising	-	20111231
Sophos	Mal/Rorpian-D	20120102
SUPERAntiSpyware	-	20111230
Symantec	WS.Reputation.1	20120102
TheHacker	Trojan/Dropper.Injector.aiaz	20111231
TrendMicro	TROJ_FAKEAV.BMC	20120102
TrendMicro-HouseCall	TROJ_FAKEAV.BMC	20120102
VBA32	TrojanDropper.Injector.aiaz	20120102
VIPRE	Trojan.Win32.Generic!BT	20120102
ViRobot	-	20120102
VirusBuster	-	20120102

Appendix C

Binary: 9b259bc255fef873f1e41629fb67c30f0c40e5dc Virus Total Results

AhnLab-V3	Trojan/Win32.Zbot	20111218
AntiVir	-	20111216
Antiy-AVL	-	20111218
Avast	Win32:Malware-gen	20111218
AVG	PSW.Generic9.AVXE	20111218
BitDefender	Trojan.Generic.KDV.481715	20111218
ByteHero	Trojan.Win32.Heur.Gen	20111207
CAT-QuickHeal	-	20111218
ClamAV	-	20111218
Commtouch	W32/Zbot.DD7.gen!Eldorado	20111217
Comodo	TrojWare.Win32.Trojan.Agent.Gen	20111218
DrWeb	-	20111218
Emsisoft	Trojan-PWS.Win32.Zbot!IK	20111218
eSafe	-	20111215
eTrust-Vet	-	20111216
F-Prot	W32/Zbot.DD7.gen!Eldorado	20111217
F-Secure	Trojan.Generic.KDV.481715	20111218
Fortinet	W32/Zbot.EZ!tr.pws	20111218
GData	Trojan.Generic.KDV.481715	20111218
Ikarus	Trojan-PWS.Win32.Zbot	20111218
Jiangmin	-	20111218
K7AntiVirus	-	20111215

PROPRIETARY AND CONFIDENTIAL

Kaspersky	Trojan-Spy.Win32.Zbot.ctnl	20111218
McAfee	PWS-Zbot.gen.hb	20111218
McAfee-GW-Edition	PWS-Zbot.gen.hb	20111218
Microsoft	PWS:Win32/Zbot.gen!Y	20111218
NOD32	probably a variant of Win32/Spy.Agent.MOVGWFV	20111218
Norman	-	20111218
nProtect	-	20111218
Panda	Trj/CL.A	20111218
PCTools	-	20111218
Prevx	-	20111218
Rising	-	20111216
Sophos	Mal/Zbot-EZ	20111218
SUPERAntiSpyware	-	20111217
Symantec	-	20111218
TheHacker	-	20111218
TrendMicro	-	20111218
TrendMicro-HouseCall	TROJ_GEN.R3EC7LI	20111218
VBA32	-	20111214
VIPRE	Trojan.Win32.Generic!BT	20111218
ViRobot	-	20111218
VirusBuster	-	20111218

Appendix D

Binary: 1bfdc4f2cfa48a1f063d1826992fbaf5e2924394 Virus Total Results

AhnLab-V3	Spyware/Win32.Zbot	20120107
AntiVir	TR/Offend.7118272.1	20120106
Antiy-AVL	-	20120107
Avast	Win32:Spyware-gen [Spy]	20120107
AVG	PSW.Generic9.BAQF	20120107
BitDefender	-	20120107
ByteHero	-	20111231
CAT-QuickHeal	-	20120107
ClamAV	-	20120107
Commtouch	-	20120107
Comodo	-	20120107
DrWeb	Trojan.PWS.Panda.547	20120107
Emsisoft	Trojan-PWS.Win32.Zbot!IK	20120107
eSafe	-	20120103
eTrust-Vet	-	20120106
F-Prot	-	20120107
F-Secure	-	20120107
Fortinet	W32/Zbot.DDHL!tr	20120107
GData	Win32:Spyware-gen	20120107
Ikarus	Trojan-PWS.Win32.Zbot	20120107
Jiangmin	-	20120107
K7AntiVirus	Spyware	20120106

PROPRIETARY AND CONFIDENTIAL

Kaspersky	Trojan-Spy.Win32.Zbot.ddhl	20120107
McAfee	PWS-Zbot	20120107
McAfee-GW-Edition	PWS-Zbot	20120107
Microsoft	PWS:Win32/Zbot	20120107
NOD32	Win32/Spy.Zbot.YW	20120107
Norman	-	20120107
nProtect	-	20120107
Panda	Trj/CI.A	20120107
PCTools	-	20120107
Prevx	-	20120107
Rising	-	20120106
Sophos	-	20120107
SUPERAntiSpyware	-	20120107
Symantec	-	20120107
TheHacker	-	20120106
TrendMicro	TROJ_GEN.FFFCBA2	20120107
TrendMicro-HouseCall	TROJ_GEN.FFFCBA2	20120107
VBA32	-	20120106
VIPRE	Trojan.Win32.Generic!BT	20120107
ViRobot	-	20120107
VirusBuster	TrojanSpy.Zbot!Z8zuEWTrK2A	20120107

Appendix E

Binary: bfcc02219321d1047cc0330454a61f6b276d06f6 Virus Total Results

AhnLab-V3	Trojan/Win32.Agent	20111217
AntiVir	TR/PSW.Zbot.Y.2082	20111216
Antiy-AVL	Trojan/win32.agent.gen	20111217
Avast	Win32:Spyware-gen [Spy]	20111217
AVG	PSW.Generic9.AVOM	20111217
BitDefender	Gen:Variant.Kazy.48419	20111217
ByteHero	Trojan.Win32.Heur.Gen	20111207
CAT-QuickHeal	-	20111217
ClamAV	-	20111217
Commtouch	-	20111217
Comodo	TrojWare.Win32.Trojan.Agent.Gen	20111217
DrWeb	Trojan.PWS.Panda.1533	20111217
Emsisoft	Trojan-Spy.Win32.Zbot!IK	20111217
eSafe	-	20111215
eTrust-Vet	-	20111216
F-Prot	-	20111217
F-Secure	Gen:Variant.Kazy.48419	20111217
Fortinet	W32/Zbot.EZ!tr.pws	20111217
GData	Gen:Variant.Kazy.48419	20111217
Ikarus	Trojan-Spy.Win32.Zbot	20111217
Jiangmin	-	20111217
K7AntiVirus	Spyware	20111215

PROPRIETARY AND CONFIDENTIAL

Kaspersky	Trojan-Spy.Win32.Zbot.csyl	20111217
McAfee	PWS-Zbot.gen.hb	20111217
McAfee-GW-Edition	PWS-Zbot.gen.hb	20111216
Microsoft	PWS:Win32/Zbot.gen!Y	20111217
NOD32	a variant of Win32/Kryptik.XGG	20111217
Norman	-	20111217
nProtect	-	20111217
Panda	Trj/GIA	20111217
PCTools	Trojan.Gen	20111217
Prevx	-	20111217
Rising	-	20111216
Sophos	Mal/Zbot-EZ	20111217
SUPERAntiSpyware	-	20111217
Symantec	Trojan.Gen.2	20111217
TheHacker	-	20111216
TrendMicro	TROJ_GEN.FFFCZLF	20111217
TrendMicro-HouseCall	TROJ_GEN.FFFCZLF	20111217
VBA32	-	20111214
VIPRE	Trojan.Win32.Generic!BT	20111217
ViRobot	-	20111217
VirusBuster	-	20111216

PROPRIETARY AND CONFIDENTIAL

```

//Ñiöðáíyài iðëëíäëüíúâ iiêíäú â originalFunction.
{
//Äîiëñúääâî â êííäö áóóäðà, jump íà iðíäíëæáíëâ functionForHook.
LPBYTE pjmp = buf + opcodeOffset;
WRITE_JMP(pjmp, originalFunction/* + opcodeOffset*/, functionForHook/* +
opcodeOffset*/);
if(CWA(kernel32, WriteProcessMemory)(process, originalFunction, buf,
opcodeOffset + JMP_ADDR_SIZE, NULL) == 0)goto END;
}

//Ïèøèì èíææèò â óóíèòèþ.
{
WRITE_JMP(buf, functionForHook, hookerFunction);
hotPatchCallback(functionForHook, originalFunction);
if(CWA(kernel32, WriteProcessMemory)(process, functionForHook, buf,
INJECT_SIZE, NULL) == 0)goto END;
}

retVal = opcodeOffset + JMP_ADDR_SIZE; //Ðàçíäð âúðàçäííâí òðääíäíðà.

END:
//Äîññòäíäíëëääââî iðääâ.
CWA(kernel32, VirtualProtectEx)(process, functionForHook, OP_CODE_MAX_SIZE * 2,
oldProtect, &oldProtect);
}

return retVal;
}

```

Raw data on what 'file' says each file is:

```

munin@ubuntu-dev:~/sample_set_1$ find ./ -exec file '{}' \;
./: directory
./0231ced00c5e62deba427fa785e19e0481a21e5a: data
./fa3e447fcb80d73284c1ec082ecec8b5e8c69290: data
./9e3bc6596fe0ff57312ba7fe9144dfbb7321f5d5: data
./63552eb629f61e2c80f97f6b71394875ce18639d: data
./c014dafb8cd26a777e6abc94bb01a814e29c0dc9: DOS executable (COM)
./8e2adb39e651c50c9fd7cfeef66f27b4cded27f1: data
./9569c711275524c5c00547f0c90be3d2b36252d1: DOS executable (COM)
./efb2e69a4c2a74f1688166881a61477fc38cc486: data
./53cecd632d2fe0cd4416ce32d7767f0f39e24223: data
./cfbc6664715190458b3e5a83d22895507ff35f4f: data
./e2e44b8114f07cee665a21f0450a727326b3d341: data
./19174b7f1897b786e914ad6e7932d0d82f086c2e: data
./e8edff3539053ebfbf79fdaed6c3234a76de5b: data
./4c2ba64f8f975f752fc33f77733dd7df7b10064f: data
./3c12b15eb7b453d0230bc5c476b2acc2e69b14e5: data
./ae713567f6ebe908ebb9925d7ef65967b52571a4: data
./6b7dd7e579c9f6cc1276f183d0397800a9b5497b: data
./9e2a7be7d2f7a055ef8fe9a89325991158f3425c: Macintosh MFS data (locked)
created: Thu Dec 12 22:26:15 2052, last backup: Thu May 22 21:48:39
2003, block size: 390163271, number of blocks: 38443, volume name:
z\313\277\200{\224%=`9\274\375\275\273X\022\260\261_n
./cf90b2dcf802a44938cbe44774add891354bcb56: data
./8d4f841ffc243ce69c7a2ab2ebd45fc11623d14b: data
./ec8d707213a73c8978472d62d5578e5bf83e1f85: data
./ccdd65b99ded0f2c68d0b81525fab194f88d9052: data
./245dd76226340ba68e8e7c69ad558887e4cca708: data
./400ad5fb66574398e036ae817b653bcdabe7ca77: data
./ec9833c61f4547ba7c3f93b55eccc4b8aab516: data

```

PROPRIETARY AND CONFIDENTIAL

./2ebbc25ad676d9fddcf9483e184f0df193da275e: data
./fd441b7b7cf3c56e12ea8bdf5dcc712f5b51aee: data
./a8a422e21a040291cdb5cb676b3769fb5dfebb30: data
./8602882e53520155be5bf35e447b6a51d5c060a2: ASCII HTML document text
./71fc9a3c9332259716e8e60692cef4bbf8b46263: data
./b002711696f7b2dafc812e6b75a7bdefeb68848b: ASCII HTML document text,
with very long lines
./758ba418c1cffa97bc67b8f928095d6164cfbccb: data
./58fbfba34100d8252a35fb80a49220cbe742cddc: data
./7743d59c358a9830fa8a861e227f30f20395b0da: data
./89ccfe53c1fe40ad606ca75bb7bfa17aa470d7b4: data
./c6db00d3860ec87a80b9e681cce9bd360356a9fd: data
./1bfdc4f2cfa48a1f063d1826992fbaf5e2924394: PE32 executable for MS
Windows (GUI) Intel 80386 32-bit
./c0784b799676b1da42f7ddb0c260484aecc02b16: data
./09f11524999469ecbed82b80f6034bc2bc7df6e9: data
./748f7b05ecd7cf5d09902334fdcc04b255394379: HTML document text
./e86834d32fe96a51f6c1a0cfd62764522c4659ad: data
./adafa84402214d74744794c7bac6e886e5012ffd: data
./b858cb282617fb0956d960215c8e84dlccf909c6: very short file (no magic)
./697490076065b855c6f417a79ac9d69e7553008a: data
./b3da6a5e6ed5ef18d7c9fd9a570f01a850cc9867: data
./be080fcef59cd497eb9f686b90669f7413795187: data
./a2c35aa79379a3e72ad0607abbfe6095d5f4539d: SoftQuad troff Context
intermediate
./d2200e1a1587878a2c68ee66007226039ff23ec9: data
./2428aad59d5abb344f96273724147b9c24ffbc7d: ASCII text, with CRLF line
terminators
./ef99005f5ed1d8db4aa57e5c4fd1da040e370115: DBase 3 data file with
memo(s)
./d649b4d83a0d0a2c571187b79d9c815255c44feb: data
./14156629bf2f3c9bbd6a599dd64b6808bd0b28b6: data
./c19ae7572f1592d798e96d7a09b76e63c3b341b1: data
./2cc1076f3c6e65d0a59792b75370b04613258ffa: PE32 executable for MS
Windows (GUI) Intel 80386 32-bit
./4089097915b5de378c9ffb0180f02790f48d4d21: DOS executable (COM)
./5c286793eb1ed4ef94932b8b1ef0fd03795d083b: data
./a11719211d886dbe060ebc6348f6f60c603cc40c: data
./5a7f37bc8481bd35863debfc113e19381c2d9fb4: data
./7c0dcde7e13dbc350eb8fc45100edcf526633be2: data
./ef7c1a5991f95ed3c61f6f88bc0d03cd2a0f2d32: data
./10b512c811fa173d2dcfb7796d5b312e2e91d629: data
./bfcc02219321d1047cc0330454a61f6b276d06f6: PE32 executable for MS
Windows (GUI) Intel 80386 32-bit
./c155efdb8e846076fc7ecc44006556f0974bcace: DOS executable (COM)
./0cc6215d31e5e639a19b4ceb3d57ce64d62e9b2c: PE32 executable for MS
Windows (GUI) Intel 80386 32-bit
./9b259bc255fef873f1e41629fb67c30f0c40e5dc: PE32 executable for MS
Windows (GUI) Intel 80386 32-bit
./22b4eccfc0fb59acefa2140992e12d0d6f5defc2: data
./cd195e5943b68637b57eac9a916cc742b2599e89: data
./eab52fcdcf2f7d875a0fd6b41f7842cde93ebb: data
./fd6f8c968854e9ca3d336e03cd3221c25be8cd5d: data
./8b591e9324afb0c641b4c0e68c0c0e7ae9ddc2fb: data

Some of them are HTML, though none appear to have any unusual or suspect traits.