“Cyber Analytics”
Malware and Static Analysis
Lecture 4

John Cavazos

Dept of Computer & Information Sciences

University of Delaware

CISC 849 : CyberAnalytics

What's Malware

- **Malware=Mal**icious soft**ware**
- Software with malicious intent
  - Different from software with bugs

Viruses

- Programs that attach themselves to another program to gain access to your machine
  - *Insertion phase* is inserting itself into file
  - *Execution phase* is performing some (possibly null) action
- May do nothing on your machine or may destroy all your files
- Seek to use your machine as a launching point to infect other machines

Worms

- Like a virus but they are self-contained programs (they don't need a host)
- Copy themselves from from machine-to-machine
- Scan for other vulnerable machines

Trojan Horse

- Program with an overt purpose (known to the user) and a covert purpose (not known to user)
  - *Often called a Trojan*

Adware

- Some programs are “free” but they support their costs by sending ads to your machines
  - E.g., Kazaa

Spyware (example)

- You download a music player
- The music player includes another program that is installed and running continuously
- This program records the websites you visit and send them to a database

How Bad is The Threat? (BAD!)

- Internet Security Emerging Threats List
  - Hackers use Instant Messaging to Spread Viruses And Worms
  - Phishing fraud is prevalent and sophisticated
  - Viruses attack any mobile device
  - Hackers target online banking accounts
  - Internet crimes continue to go unreported

What is Malware Analysis?

- Taking malware apart to study it
  - Static analysis
    - Looking at malware internals
    - NOT running
  - Dynamic analysis
    - Looking at malware behavior
    - Running it and capture behavior

Program Analysis

- Given an executable, how do we find out what it does?
  - Try to find the program online
    - Analyze source code to find clues
    - Search for the name of the program
  - **Perform source code review**
  - Execute the program in a sandbox
    - Some programs can break out of a sandbox / jail

Program Compilation

- Compiler
  - Translates HLL code to Assembly / ILL
- Assembler
  - Translates Assembly code to machine language
- Linker
  - Creates object code out of several modules
  - A program usually makes library calls (stdio)

Program Compilation

- Statically Linked: All library code is part of the object code
- Dynamically Linked: Program calls library functions. (DLL)
- Stripping: Removes all human-readable symbols from object code.
  - Combats reverse engineering.
- Packing with UPX, etc.
  - upx.sourceforge.net
  - Compresses source code (achieves ratios of 20% - 40%)

Program Analysis

- Static Analysis:
  - Determine the type of executable
    - ELF file in Unix
    - Exe-type in Windows
  - Symbol Extraction:
    - Use a program like “strings” to find symbols left in object code
    - Names give hints on program
    - Will not work for stripped files

strings /bin/mkdir

@(#) Copyright (c) 1983, 1992, 1993

The Regents of the University of California. All rights reserved.

$FreeBSD: src/bin/mkdir/mkdir.c,v 1.26 2002/06/30 05:13:54 obrien Exp $

@(#)PROGRAM:mkdir PROJECT:file_cmds-251

m:pv

invalid file mode: %s

mkdir: created directory '%s'

usage: mkdir [-pv] [-m mode] directory ...

Static Program Analysis

- Investigate source code
- Use Reversing Tools:
  - Disassembler:
    - Decodes binary machine code into a readable assembly language text
      - IDA-Pro
      - ILDasm (Microsoft .Net IL disassembler)
      - RADARE2

Static Program Analysis

Decompilers

- Attempt to produce a high-level language source-code-like representation from a binary
- Never completely possible because
  - The compiler removes some information
  - The compiler optimizes the code

Static Program Analysis

- Artifacts to look for:
  - Names of functions
    - Especially API functions
  - Data strings
    - Names of constant strings
    - Names of directories
    - Identification of compiler

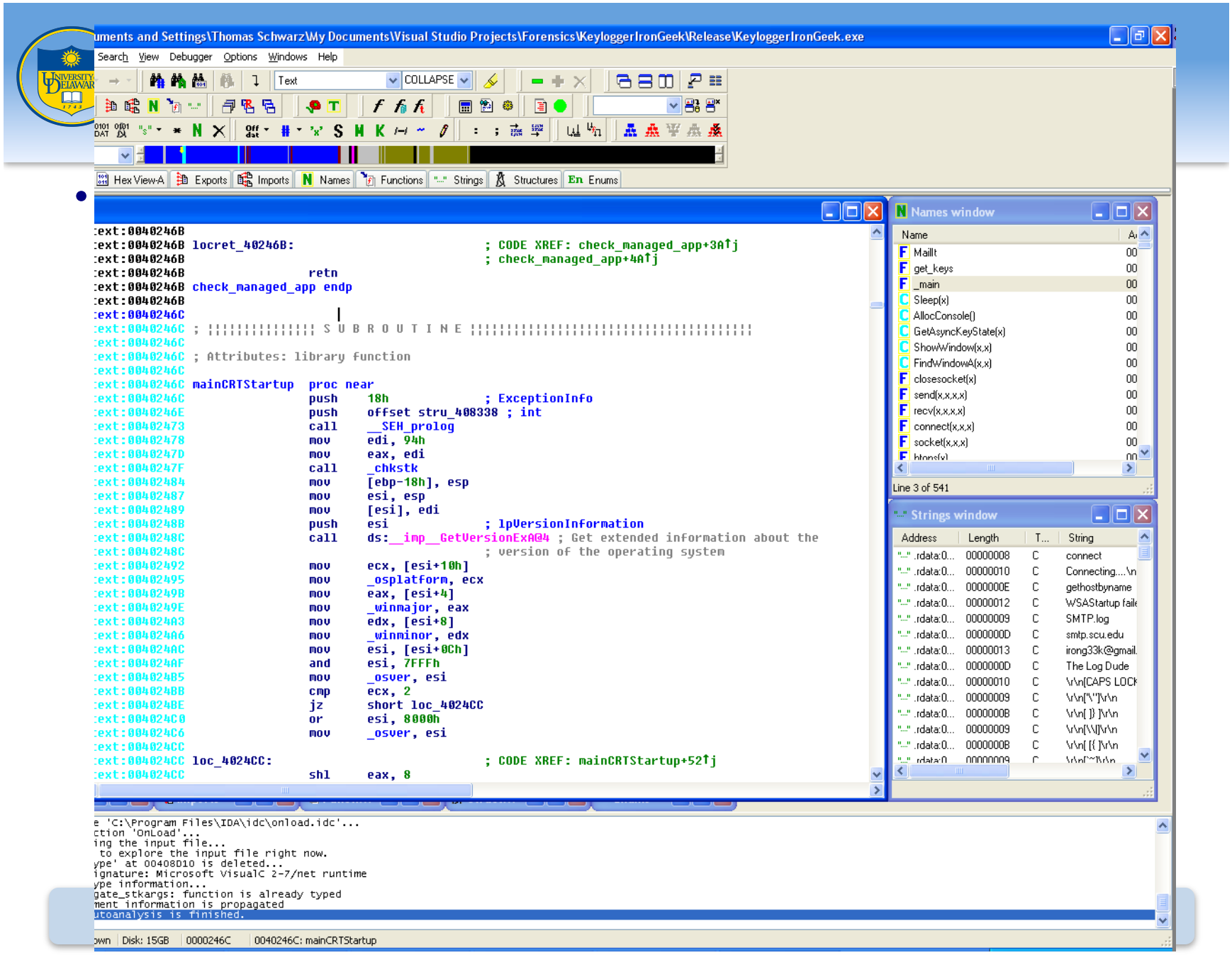
Static Program Analysis

- Compilers generate different types of code for the same high-level language features
  - Function Calls:
    - Order in which parameters are pushed on stack
    - Use of certain registers to pass variables
    - Use of stack / registers to return a value
    - Division of labor between callee and caller

Static Program Analysis

- This allows us to recognize the compiler with which an executable was created
- Programmers using assembly will not follow the same standards throughout the code
  - Hence, we can recognize assembly writers as well

Break


uments and Settings\Thomas Schwarz\My Documents\Visual Studio Projects\Forensics\KeyloggerIronGeek\Release\KeyloggerIronGeek.exe
Search View Debugger Options Windows Help
Hex View-A Exports Imports Names Functions Strings Structures Enums
:ext:0040246B
:ext:0040246B locret_40246B: ; CODE XREF: check_managed_app+3A↑j
:ext:0040246B ; check_managed_app+4A↑j
:ext:0040246B retn
:ext:0040246B check_managed_app endp
:ext:0040246B
:ext:0040246C
:ext:0040246C ; ¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦ S U B R O U T I N E ¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦
:ext:0040246C
:ext:0040246C ; Attributes: library function
:ext:0040246C
:ext:0040246C mainCRTStartup proc near
:ext:0040246C push 18h ; ExceptionInfo
:ext:0040246E push offset stru_408338 ; int
:ext:00402473 call __SEH_prolog
:ext:00402478 mov edi, 94h
:ext:0040247D mov eax, edi
:ext:0040247F call _chkstk
:ext:00402484 mov [ebp-18h], esp
:ext:00402487 mov esi, esp
:ext:00402489 mov [esi], edi
:ext:0040248B push esi ; lpVersionInformation
:ext:0040248C call ds:__imp__GetVersionExA@4 ; Get extended information about the
:ext:0040248C ; version of the operating system
:ext:00402492 mov ecx, [esi+10h]
:ext:00402495 mov _osplatform, ecx
:ext:0040249B mov eax, [esi+4]
:ext:0040249E mov _winmajor, eax
:ext:004024A3 mov edx, [esi+8]
:ext:004024A6 mov _winminor, edx
:ext:004024AC mov esi, [esi+0Ch]
:ext:004024AF and esi, 7FFFh
:ext:004024B5 mov _osver, esi
:ext:004024BB cmp ecx, 2
:ext:004024BE jz short loc_4024CC
:ext:004024C0 or esi, 8000h
:ext:004024C6 mov _osver, esi
:ext:004024CC
:ext:004024CC loc_4024CC: ; CODE XREF: mainCRTStartup+52↑j
:ext:004024CC shl eax, 8
Names window
Name
Maillt
get_keys
_main
Sleep(x)
AllocConsole()
GetAsyncKeyState(x)
ShowWindow(x,x)
FindWindowA(x,x)
closesocket(x)
send(x,x,x,x)
recv(x,x,x,x)
connect(x,x,x)
socket(x,x,x)
Line 3 of 541
Strings window
Address Length T... String
.rdata:0... 00000008 C connect
.rdata:0... 00000010 C Connecting....\n
.rdata:0... 0000000E C gethostbyname
.rdata:0... 00000012 C WSAStartup fail
.rdata:0... 00000009 C SMTP.log
.rdata:0... 0000000D C smtp.scu.edu
.rdata:0... 00000013 C irong33k@gmail.
.rdata:0... 0000000D C The Log Dude
.rdata:0... 00000010 C \r\n[CAPS LOCK
.rdata:0... 00000009 C \r\n[\"]\r\n
.rdata:0... 0000000B C \r\n[ ]}]\r\n
.rdata:0... 00000009 C \r\n[\\|]\r\n
.rdata:0... 0000000B C \r\n[[{]\r\n
e 'C:\Program Files\IDA\idc\onload.idc'...
ction 'OnLoad'...
ing the input file...
to explore the input file right now.
ype' at 00408D10 is deleted...
ignature: Microsoft VisualC 2-7/net runtime
ype information...
gate_stkargs: function is already typed
ment information is propagated
utoanalysis is finished.
own Disk: 15GB 0000246C 0040246C: mainCRTStartup

Program Analysis

- Malware writers can use anti-reversing techniques
  - Eliminate symbolic information
  - Encrypt code
  - Code obfuscation
    - Make high level constructs difficult to understand

Program Analysis

- Anti-debugger Methods:
  - Use the IsDebuggerPresent API to protect against user-level debuggers
  - Use the NTQuerySystemInformation API to determine if a kernel debugger is attached to the system
  - Set a trap flag and check whether it is still there
    - A debugger would “swallow” it
  - Put in bogus bytes over which the code jumps
    - Does not work for all disassemblers

Anti-debugging using traps

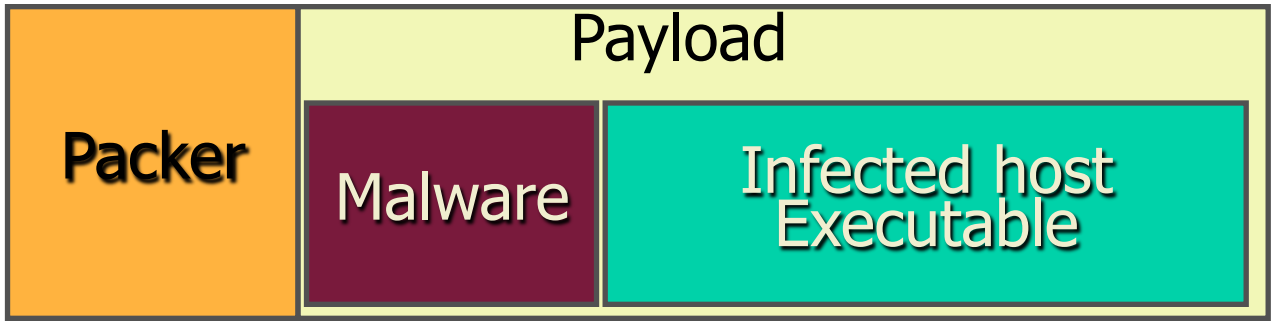
- Set a trap flag and check whether it is still there
    - A debugger would “swallow” it

Note: A good explanation of what this means is found here (http://leons.im/posts/anti-debug-with-trap-flag-register/). Briefly, this means is that an application can set up a “trap” (also known as an “exception”) that must be handled during the execution of the program. “An exception is an event that occurs during the execution of a program, and requires execution of code outside the normal flow of control.”[1]

During normal execution, the exception can be set up to be “handled” in a way that execution of the code continues along one path of the code, allowing the program to proceed. However during debug mode, “exceptions” or “traps” will not be “handled” and therefore execution will go down a different path of the code. A malware writer can write their program so that it exits when execution goes down this different path. Thus, executing the program during normal versus debug mode can take two different paths and the malware writer can cause the program to exit when execution goes down the different debug path.

[1] https://msdn.microsoft.com/en-us/library/windows/desktop/ms680657(v=vs.85).aspx

Packers



Packer functionalities

- Compress
- Encrypt (custom)
- Randomize (polymorphism)
- Anti-debug technique (fake jmps)
- Add junk code
- Anti-VM (next week!)

RADARE2 Tutorial

CISC 849 : CyberAnalytics