Application Security

Tom Chothia
Computer Security, Lecture 14
Introduction

• Compiled code is really just data…
  – which can be edit and inspected.

• By examining low level code protections can be removed and the function of programs altered.

• Good protection tends to slow down this process, not stop it.
This lecture and next:

• Java Byte code:
  – High level overview
  – Inspecting the byte code
  – Decompiling back to Java

• x86 assembly:
  – High level overview
  – Inspecting and altering binaries in IDA

• Buffer overflow attacks.
Java Byte Code

• Java compiles to *Java Byte Code*.  
  – Type: “javap -c <ClassName>” to see the byte code.

• Every computer must have its own Java Virtual Machine (JVM) which runs the byte code.

• Every different OS must have it’s own JVM
Compile Java to Byte Code Using "javac"

Run Byte Code On JVM using "java"

Java Program .java

Java Byte Code .class

Windows JVM

Windows Computer

Linux JVM

Linux Computer

Phone JVM

Mobile Phone
A stack machine has a stack to hold data and a small number of registers.

Data pushed onto the stack or “popped” off.

The registers are fast, but there are only a few of them.
Java Byte Code

• `iconst_0`: push 0 onto the stack
• `istore_1`: pop the top of the stack as variable 1
• `goto`: jump to line:
• `iload_1`: push variable 1 onto the stack
• `iadd`: add the top two numbers on the stack.
• `if_icmpge`: if 1\textsuperscript{st} item on stack $\leq$ 2\textsuperscript{nd} jump
• `ifeq`: if 1\textsuperscript{st} item on stack $> 2\textsuperscript{nd}$ jump to line
Example code starts off by loading 0s into registers 1 and 2.

These are i & j in the code.

- 0: iconst_1
- 1: istore_1
- 2: iconst_1
- 3: istore_2
A Stack Machine

Next the code checks the for loop guard:

- 4: iload_2
- 5: iconst_4
- 6: if_icmpge 26

The program doesn’t jump
A Stack Machine

The loop continues:

\[ \ldots \]

- 4: iload_2
- 5: iconst_4
- 6: if_icmpge
- 26

\[ \ldots \]

\[ \ldots \]

- 20: iinc
- 23: goto
- 26: return
Decompile

• Wouldn’t it be much easier to work with the source code, rather than the byte code?

• JD-GUI is a Java de-compiler, it transforms Java Byte Code into Java Code.

• Not perfect, e.g. confuses 0,1 and true, false.
Bypassing the password check.

• De-compilation makes it much easier to understand what a program is doing.

• It also makes it easy to alter and recompile the code.

• All code that is used to protect the code can be removed.
Binaries

- Binaries are written in assembly
- Much lower level than Java byte code,
- Assembly compiled for one type of machine won’t run on another.
- But the same techniques apply.
Some x86 Commands

PUSH: add to top of stack

CALL: execute a function

RET, RETN, RETF: end a function and restart calling code.

POP: read and remove from top of stack

JMP: jump to some code (like writing to EIP)

MOV: move value between registers

MOV $r1, $r2 = PUSH $r1

POP $r2
Jumps

To jump in x86 you first compare the values and then jump.

TEST: does a bitwise “and”.
CMP: subtracts 2 values

The result isn’t stored but flags are set.

Following TEST:
JZ: jump if result was 0
JNZ: jump if result isn’t zero
JE: jump if equal
JNE: jump if not equal
JL: jump if less than.
For Loop Program x86 (64 bit):

```
(gdb) disassemble
Dump of assembler code for function main:
0x000000000000000400524 <+0>: push %rbp
0x000000000000000400525 <+1>: mov %rsp,%rbp
=> 0x000000000000000400528 <+4>: sub $0x10,%rsp
0x00000000000000040052c <+8>: movl $0x1,-0x4(%rbp)
0x000000000000000400533 <+15>: movl $0x1,-0x8(%rbp)
0x00000000000000040053a <+22>: jmp 0x40055d <main+57>
0x00000000000000040053c <+24>: mov -0x8(%rbp),%eax
0x00000000000000040053f <+27>: add %eax,-0x4(%rbp)
0x000000000000000400542 <+30>: mov $0x40065c,%eax
0x000000000000000400547 <+35>: mov -0x4(%rbp),%edx
0x00000000000000040054a <+38>: mov %edx,%esi
0x00000000000000040054c <+40>: mov %rax,%rdi
0x00000000000000040054f <+43>: mov $0x0,%eax
0x000000000000000400554 <+48>: callq 0x400418 <printf@plt>
0x000000000000000400559 <+53>: addl $0x1,-0x8(%rbp)
0x00000000000000040055d <+57>: cmpl $0x3,-0x8(%rbp)
0x000000000000000400561 <+61>: jle 0x40053c <main+24>
0x000000000000000400563 <+63>: leaveq
0x000000000000000400564 <+64>: retq
End of assembler dump.
```
Can we De-Compile?

• Not really …
  – There is no clear distinction between data and code.
  – Code can be constructed dynamically.
  – Parts of the code can rewrite other parts.

• So it’s quite easy to stop fully automated disassembly.
IDA pro

• IDA pro is an Interactive DisAssembler.

• It helps a human understand binaries.

• This is the best tool for malware binary analysis, security analysis of firmware and reverse engineering.

• There is a free & demo version:
  – http://www.hex-rays.com/
Function preamble:
sets up the stack space

Set I & j to 1:
i is at stack location: exp+18
j is at stack location: exp+1C

For loop check:
Compare i to 3,

Add i to j

Print j

Add 1 to i

End
Common Techniques

• Look for strings.

• Identify key tests and check the values in the register using a debugger.

• Swap JEQ and JNEQ.

• Jump over the instructions that perform checks.
Defenses

• Dynamically construct the key
  – Attacker can run code.

• Encrypt the binary,
  – Your program must include the key in plain text, so the attacker can find it.

• Obfuscate the code, e.g. mix data and code, so it’s not clear which is which
  – Can slow down attacks by months or years! (e.g. Skype).
Defense

• Require online activation.
  – Activation can be completely disabled, users don’t like this.

• Require online content, e.g. WoW, BlueRay

• Hardware based protection, i.e., store part of the code in tamper resistant hardware.
Summary

• Machine code can be inspected and edited.

• Many tools exist to inspect, debug and decompile code.

• Most software protection can be removed.

• But slowing this down by months or years can save a business.
Next Lecture

• Buffer overflow attacks
• More on x86 assembly.
• More on IDA