Plan for Today

Java Security
Java Byte Codes (JVML) and Verification

Reminder:
Project Team Requests are due before midnight tomorrow
Project Idea Proposals are due in class Tuesday

Buzzword Description

“A simple, object-oriented, distributed, interpreted, robust, secure, architecture neutral, portable, high-performance, multithreaded, and dynamic language.” [Sun95]

As the course proceeds, we will discuss how well it satisfies these “buzzwords”. You should especially be able to answer how well it satisfies each of the blue ones in your final interview.

Safe Programming Languages

Type Safety
Compiler and run-time environment ensure that bits are treated as the type they represent

Memory Safety
Compiler and run-time environment ensure that program cannot access memory outside defined storage

Control Flow Safety
Can’t jump to arbitrary addresses

What happens if you don’t have type/memory safety?

Sometimes people use “type safety” to mean all of these.

Is Java the first language to have them?
No way! LISP had them all before 1960.
Lack of Safety in C++

```c++
#include <iostream>
using namespace std;

int main(void) {
    int x = 9;
    char s[4];
    cin >> s;
    cout << "s is: " << s << endl;
    cout << "x is: " << x << endl;
}
```

When things go really bad...

If person entering input is clever and mean, they can put what they want in the return address, and their own code after that to jump to!

**Buffer Overflow Attack**

"Stack Smashing"

Buffer Overflows

- Code Red: exploited buffer overflow in Microsoft's IIS (web server)
- Attacker sends excessively long request to web server, overflows buffer and puts virus code on stack
- Until about 5 years ago: cause of most security problems
- Now: still a serious problem

Is the Java Programming Language safe?

**Type Safety**

Compiler and run-time environment ensure that bits are treated as the type they represent

**Memory Safety**

Compiler and run-time environment ensure that program cannot access memory outside defined storage

**Control Flow Safety**

Can't jump to arbitrary addresses
Is the Java Programming Language safe?

**Type Safety**
Compiler and run-time environment ensure that bits are treated as the type they represent

**Memory Safety**
Compiler and run-time environment ensure that program cannot access memory outside defined storage

**Control Flow Safety**
Can’t jump to arbitrary addresses

---

Malicious Code
Can a safe programming language protect you from malicious code?

1. Code your servers in it to protect from buffer overflow bugs
2. Only allow programs from untrustworthy origins to run if the are programmed in the safe language

---

Safe Languages?

- But how can you tell program was written in the safe language?
  - Get the source code and compile it (most vendors, and all malicious attackers refuse to provide source code)
  - Special compilation service cryptographically signs object files generated from the safe language (SPIN, [Bershad96])
  - Verify object files preserve safety properties of source language (Java)

---

JVML

- javac
  - malcode.java
  - Java Source Code
  - malcode.class
  - JVML Object Code
  - Alice User
  - Alice wants to know JVML code satisfies Java PL’s safety properties.

---

Java Virtual Machine

- Small and simple to implement
- All VMs will run all programs the same way
- Secure

---

Does JVML satisfy Java PL’s safety properties?
Implementing the JavaVM

load class into memory
set the instruction pointer to point to the
beginning of main
while (there is more to do) {
    fetch the next instruction
    execute that instruction
}

Some other issues we will talk about later... (e.g., Garbage
collection – need to reclaim unused storage)

Java Byte Codes

- **Stack-based** virtual machine
- Small instruction set: 202 instructions (all are
  1 byte opcode + operands)
  - Intel x86: ~280 instructions (1 to 17 bytes long!)
- **Memory is typed** (but imprecisely)
- Every Java class file begins with magic number
  3405691582

  \[= 0xCAFEBABE \text{ in base 16} \]

Stack-Based Computation

**push** – put something on the top of the stack
**pop** – get and remove the top of the stack

```
push 2
push 3
add

Stack

2
5
3
```

Some Java Instructions

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Mnemonic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>nop</td>
<td>Does nothing</td>
</tr>
<tr>
<td>1</td>
<td>aconst_null</td>
<td>Push null on the stack</td>
</tr>
<tr>
<td>3</td>
<td>iconst_0</td>
<td>Push int 0 on the stack</td>
</tr>
<tr>
<td>4</td>
<td>iconst_1</td>
<td>Push int 1 on the stack</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**ldc** \(<\text{value}>\)
Push a one-word (4 bytes)
constant onto the stack

```
ldc "Hello"
```

```
ldc 2220
```

Constant may be an int, float or String

The String is really a reference to an entry in
the string constant table! The strange String
semantics should make more sense now.

Some Java Instructions

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Mnemonic</th>
<th>Description</th>
</tr>
</thead>
</table>
| 18     | ldc \(<value>\) | Push a one-word (4 bytes)
constant onto the stack |

Arithmetic

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Mnemonic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>96</td>
<td>iadd</td>
<td>Pops two integers from the stack and pushes their sum</td>
</tr>
</tbody>
</table>

```
iadd
```

```
iconst_2
```

```
iconst_3
```

```
iadd
```
Arithmetic

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Mnemonic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>96</td>
<td>iadd</td>
<td>Pops two integers from the stack and pushes their sum</td>
</tr>
<tr>
<td>97</td>
<td>ladd</td>
<td>Pops two long integers from the stack and pushes their sum</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>106</td>
<td>fmul</td>
<td>Pops two floats from the stack and pushes their product</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>119</td>
<td>dneg</td>
<td>Pops a double from the stack, and pushes its negation</td>
</tr>
</tbody>
</table>

Java Byte Code Instructions

0: nop
1-20: putting constants on the stack
96-119: arithmetic on ints, longs, floats, doubles

What other kinds of instructions do we need?

Other Instruction Classes

Control Flow (~20 instructions)
if, goto, return
Method Calls (4 instructions)
Loading and Storing Variables (65 instructions)
Creating objects (1 instruction)
Using object fields (4 instructions)
Arrays (3 instructions)

Does JVML satisfy Java PL’s safety properties?

iconst_2 push integer constant 2 on stack
istore_0 store top of stack in variable 0 as int
aload_0 load object reference from variable 0

No! This code violates Java’s type rules.

javap -c Sample1
Compiled from Sample1.java
public class Sample1 extends java.lang.Object {
    public Sample1();
    public static void main(java.lang.String[]);
}

Method Sample1()
0 aload_0
1 invokespecial #1 <Method java.lang.Object();>
4 return

Method void main(java.lang.String[])
0 getstatic #2 <Field java.io.PrintStream err>
1 ldc #3 <String "Hello!">
5 invokevirtual #4 <Method void println(java.lang.String)>
8 iconst_1
9 invokestatic #5 <Method void exit(int)>
12 return

public class Sample1 {
    static public void main (String args[])
    {
        System.err.println("Hello!");
        System.exit (1);
    }
}
Charge

- Next: what the verifier does, security policies in Java

Remember to send your team requests by Friday, and be ready to present your project ideas next class.