Programming Languages and Techniques (CIS120)

Bonus Lecture

November 25, 2015

“Code is Data”
Code is Data

M.C. Escher, Drawing Hands, 1948
A Java source file is just a sequence of characters.
We can represent programs with Strings!

```java
String p_0 = "class C { public static void main(String args[]) {...}}"
String p_1 = "class D { public static void main(String args[]) {...}}"
...
String p_12312398445 = "..." // solution to HW08
...
String p_93919113414 = "..." // code for Eclipse
...
Consequence 1: Programs that manipulate programs
Interpreters

We can create programs that manipulate programs
An interpreter is a program that executes other programs
interpret ("3 + 4") $\rightarrow$ 7
Example 1: javascript
• Example 2: Eclipse
  – Note that Eclipse manipulates a representation of Java programs
  – Eclipse itself is written in Java
  – So you could use Eclipse to edit the code for Eclipse... ?!

• Example 3: Compiler
  – The Java compiler takes a representation of a Java program
  – It outputs a “low-level” representation of the program as a .class file (i.e. Java byte code)
  – Can also compile to other representations, e.g. x86 “machine code”
Example Compilation: Java to X86

class Point {
    int x;
    int y;
    Point move(int dx, int dy) {
        x = x + dx;
        y = y + dy;
        return this;
    }
}

.globl __fun__Point.move
__fun__Point.move:
    pushl %ebp
    movl %esp, %ebp
    subl $4, %esp
    _5:
        movl 8(%ebp), %eax
        movl 4(%eax), %eax
        movl %eax, 4(%eax)
        movl 8(%ebp), %eax
        movl 0(%eax), %eax
        movl %eax, -4(%ebp)
        movl 16(%ebp), %ecx
        addl %ecx, -4(%ebp)
        movl -4(%ebp), %ecx
        movl 8(%ebp), %eax
        movl %ecx, 0(%eax)
        movl 8(%ebp), %eax
        movl %ebp, %esp
    popl %ebp
    ret
Consequence 2: Malware

Rene Magritte, The Human Condition, 1933
Consequence 2: Malware

• Why does Java do array bounds checking?

• *Unsafe* language like C and C++ don’t do that checking;
  – They will happily let you write a program that “writes past” the end of an array.

• Result:
  – viruses, worms, “jailbreaking” mobile phones, Spam, botnets, ...

• Fundamental issue:
  – Code is data.
  – Why?
void m() {
    char[2] buffer;

    char c = read();
    int i = 0;
    while (c != -1) {
        buffer[i] = c;
        c = read();
        i++;
    }
    process(buffer);
}

void main() {
    m();
    // do some more stuff
}
Abstract Stack Machine

“Stack Smashing Attack”
Abstract Stack Machine

Call to main() to start the program...
Abstract Stack Machine

Workspace

```c
char[2] buffer;
char c = read();
int i = 0;
while (c != -1) {
    buffer[i] = c;
    c = read();
    i++;
}
process(buffer);
```

Stack

```c
;
// do some more stuff
```

Push the saved workspace, run m()
Allocate space for buffer on the stack.
Allocate space for `c`.
Read the first user input... 'z'.

```c
int i = 0;
while (c != -1) {
    buffer[i] = c;
    c = read();
    i++;
}
process(buffer);
```
Abstract Stack Machine

Workspace

```
while (c != -1) {
    buffer[i] = c;
    c = read();
    i++;
}
process(buffer);
```

Allocate space for i.

Stack

```
// do some more stuff

buffer

<table>
<thead>
<tr>
<th>c</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>0</td>
</tr>
</tbody>
</table>
```
Abstract Stack Machine

while (c != -1) {
  buffer[i] = c;
  c = read();
  i++;
}
process(buffer);

Copy (contents of) c to buffer[0]
Abstract Stack Machine

Workspace

while (c != -1) {
    buffer[i] = c;
    c = read();
    i++;
}
process(buffer);

Stack

ds;  // do some more stuff

<table>
<thead>
<tr>
<th>buffer</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>y</td>
</tr>
<tr>
<td>i</td>
<td>0</td>
</tr>
</tbody>
</table>

Read next character ... 'y'
while (c != -1) {
    buffer[i] = c;
    c = read();
    i++;
}
process(buffer);

Increment i
Copy (contents of) c to buffer[1]
while (c != -1) {
    buffer[i] = c;
    c = read();
    i++;
}
process(buffer);

Read next character ... 'N'
while (c != -1) {
    buffer[i] = c;
    c = read();
    i++;
}
process(buffer);

Increment i
Abstract Stack Machine

Workspace

while (c != -1) {
  buffer[i] = c;
  c = read();
  i++;
}
process(buffer);

Stack

N  do some more stuff

<table>
<thead>
<tr>
<th>buffer</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>N</td>
</tr>
<tr>
<td>i</td>
<td></td>
</tr>
</tbody>
</table>

Copy (contents of) c to buffer[2]?!?

Overwrites the saved workspace!?
Abstract Stack Machine

while (c != -1) {
    buffer[i] = c;
    c = read();
    i++;
}
process(buffer);

Keep going… read 'o'...
while (c != -1) {
    buffer[i] = c;
    c = read();
    i++;
}
process(buffer);

Keep going… read 'o'…increment i…
Start reading the buffer:

```java
while (c != -1) {
    buffer[i] = c;
    c = read();
    i++;
}
process(buffer);
```

Keep going... read 'o'...increment i...write 'o' into saved workspace...
Abstract Stack Machine

Workspace

Stack

Now I pwn U!!!!

<table>
<thead>
<tr>
<th>buffer</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>0</td>
</tr>
<tr>
<td>i</td>
<td>3</td>
</tr>
</tbody>
</table>

Later...

POP!
Now I pwn U!!!!

The stack smashing attack successfully wrote *arbitrary* code into the program's workspace...
Other Code Injection Attacks

```java
void registerStudent() {
    print("Welcome to student registration.");
    print("Please enter your name: ");
    String name = readLine();
    evalSQL("INSERT INTO Students(" + name + ")");
}

http://xkcd.com/327/
```
Consequence 3: Undecidability
Theorem: It is impossible to write a method
\begin{verbatim}
boolean halts(String prog)
\end{verbatim}
such that, for any valid Java program P represented as a string p_P,
\begin{verbatim}
halts(p_P)
\end{verbatim}
returns true exactly when the program P halts, and false otherwise.

Alonzo Church, April 1936

Alan Turing, May 1936
• Suppose we could write such a program:

```java
class HaltDetector {
    public static boolean halts(String javaProgram) {
        // ...do some super-clever analysis...
        // return true if javaProgram halts
        // return false if javaProgram does not
    }
}
```

• A correct implementation of HaltDetector.halts(p) always returns either true or false
  – i.e., it never raises an exception or loops

• HaltDetector.halts(p) ⇒ true means “p halts”
• HaltDetector.halts(p) ⇒ false means “p loops forever”
Do these methods halt?

“boolean m(){ return false; }”
⇒ YES

“boolean m(){ return m(); }”
⇒ NO  (assuming infinite stack space)

“boolean m(){
   if ("abc".length() == 3) return true;
   else return m(); }
⇒ YES

“boolean m(){
   String x = "";
   while (true) {
      if (x.length() == 3) return true;
      x = x + \"a\";
   }
   return false;
}”
⇒ YES
Consider this Program called Q:

class HaltDetector {
    public static boolean halts(String javaProgram) {
        // ...do some super-clever analysis...
        // return true if javaProgram halts
        // return false if javaProgram does not
    }
}

class Main {
    public static void Q() {
        String p_Q = ???;  // string representing method Q
        if (HaltDetector.halts(p_Q)) {
            while (true) {}  // infinite loop!
        }
    }
}

What happens when we run Q?

```java
public static void Q() {
    String p_Q = ???;  // string representing method Q
    if (HaltDetector.halts(p_Q)) {
        while (true) {}  // infinite loop!
    }
}
```

if \(\text{HaltDetector.halts}(p_{Q}) \Rightarrow \text{true}\) then \(Q \Rightarrow \text{infinite loop}\)

if \(\text{HaltDetector.halts}(p_{Q}) \Rightarrow \text{false}\) then \(Q \Rightarrow \text{halts}\)

**Contradiction!**

- Russell’s Paradox (1901)
- Gödel’s Incompleteness Theorem (1931)
- Both rely on *self reference*

Bertrand Russell, 1901
Kurt Gödel, 1931
Potential Hole in the Proof

• What about the ??? in the program Q?
• It is supposed to be a String representing the program Q itself.
• How can that be possible?
• Answer: code is data!
  – And there's more than one representation for the same data.

• See Quine.java
**Profound Consequences**

- The “halting problem” is *undecidable*
  - There are problems that cannot be solved by a computer program!

- Rice’s Theorem:
  - Every “interesting” property about computer programs is undecidable!

- You can’t write a perfect virus detector!
  (whether a program is a virus is certainly interesting)
  1. virus detector might go into an infinite loop
  2. it gives you false positives (i.e. says something is a virus when it isn’t)
  3. it gives you false negatives (i.e. it says a program is not a virus when it is)

- Also: You can’t write a perfect autograder!
  (whether a program is correct is certainly interesting)
Recommended Courses

• Programs that manipulate Programs
  – CIS 341: Compilers and interpreters

• Malware
  – CIS 331: Intro to Networks and Security

• Undecidability
  – CIS 262: Automata, Computability and Complexity
Recommended Reading

- Logicomix
- Secure Coding in C and C++
- Building Secure Software
- Gödel, Escher, Bach: An Eternal Golden Braid
- I Am A Strange Loop