

CS 6V81-002: Language-based Security

Instructor: Dr. Kevin W. Hamlen

Spring 2008

Outline

- Course Logistics
 - Course objectives
 - Homework, grading, etc.
 - About me
- What is Language-based Security?
- Tentative Course Schedule
 - List of topics
 - Example security vulnerabilities, policies, and enforcement strategies

Course Information

- Course webpage:
 - <http://www.utdallas.edu/~hamlen/cs6V81sp08.html>
 - Course number: CS 6V81-002
- Prerequisites:
 - PhD & Masters students
 - Undergraduates who have passed CS 6371 (APL)
 - If neither of the above, please see me
- Instructor:
 - Dr. Kevin W. Hamlen
 - hamlen at utdallas dot edu
 - ECSS 3.704
 - Office hours: Wed 2:00-4:00
 - email me for additional appointments

Course Objectives

- Survey Course
 - general familiarity with current research in the field of language-based security
 - solid understanding of at least one topic
 - learn some type theory
 - generate research ideas/directions
 - turn the security folks into PL geeks and the PL folks into security geeks...

Grading

- Homework – assigned readings (~2 papers per class)
- Short Quizzes (25%)
 - one quiz given at start of each class with an assigned reading
 - quiz based on FIRST of two assigned readings
 - consist of ~5 short answer or multiple-choice questions
- Presentations (25%)
 - 1 or 2 per student
 - ~45 min. each; cover assigned readings for the day
 - exemption from quiz on papers you present
- In-class Discussion (10%)
- Projects (40%) (letter grade only)
 - project proposals due mid-semester
 - work in teams of 2-3 students
 - I will suggest project ideas as we go...

Texts

- No Required Textbook
- Two very useful textbooks to have:
 - Both available online from UTD computers!
(See course website for links)
 - Advanced Topics in Types and Programming Languages by Benjamin C. Pierce
 - Computer Security: Art and Science by Matt Bishop

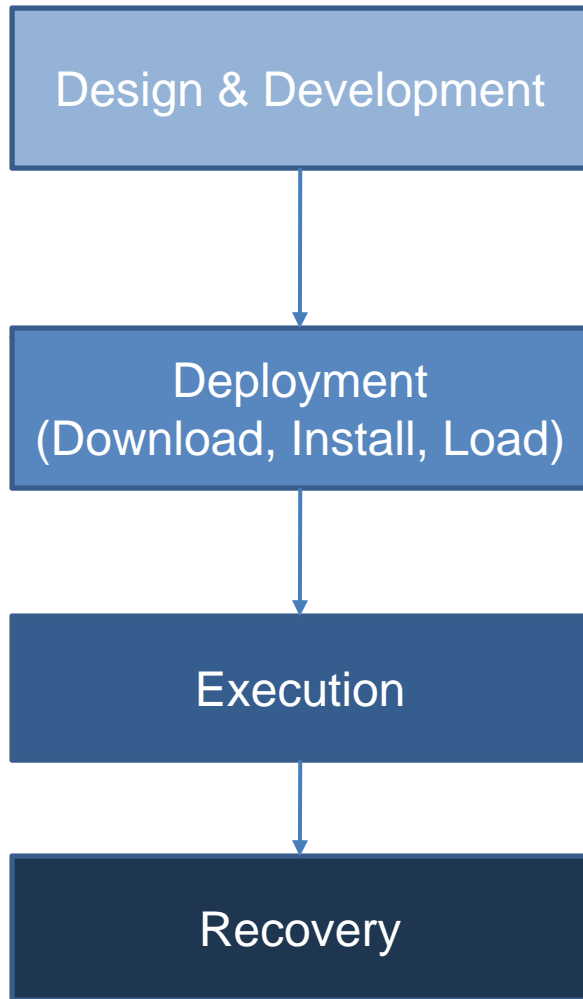
About Me

- originally from the northeast (Buffalo, NY)
- Undergrad: Carnegie Mellon University
 - Proof-Carrying Code
- Masters & PhD: Cornell University (Aug '06)
- Microsoft Research
 - Redmond, WA and Cambridge, England facilities
 - Language-based security for .NET
 - Designed and built the Mobile Certified In-lined Reference Monitoring system for .NET F#

What is Language-based Security?

- Utilize programming-language and compiler design techniques to enforce software security
- Not just about designing better languages!
 - Legacy code (C, Fortran, Cobol)
 - Low-level code (device drivers, matrix multipliers)
- Analyze existing source codes
- Analyze object codes when sources unavailable

Software Lifecycle



- ▶ Security vulnerabilities in non-malicious code
 - ▶ type-safe programming languages
 - ▶ formal verification
 - ▶ code synthesis
- ▶ Malicious code (viruses, worms, etc.)
- ▶ Antivirus scanning
- ▶ Code-signing
- ▶ Type-safe target codes (e.g., Java bytecode)
- ▶ Independently verifiable certificates
- ▶ Runtime monitoring
- ▶ Automatically generated self-monitoring code
- ▶ Auditing (logging)
- ▶ Rollback (reversible computation, restore points)
- ▶ Legal action

C/C++: Unsafe Languages

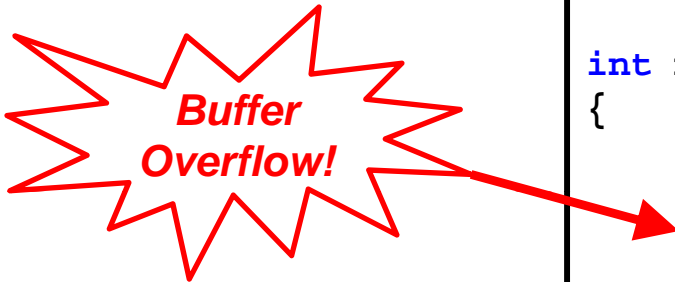
- Find the bug:

```
#include <stdio.h>

int main()
{
    char name[1024];
    printf("Enter your name: ");
    gets(name);
    printf("Your name is: %s\n", name);
    return 0;
}
```

C/C++: Unsafe Languages

- Find the bug:



```
#include <stdio.h>

int main()
{
    char name[1024];
    printf("Enter your name: ");
    gets(name);
    printf("Your name is: %s\n", name);
    return 0;
}
```

- C/C++ lets you write programs that **seg fault**
- Some language features *cannot* be used safely!
- Most of the software crashes you experience are a direct result of the unsafe design of C/C++

Java: A Type-safe, Imperative Language

- Find two bugs:

```
import java.io.*;
import java.util.*;

class Summation {
    public static void main(String[] args) {
        List list = new LinkedList();

        for (int i=0; i<args.length; ++i)
            list.add(args[i]);

        int sum = 0;
        while (!list.isEmpty())
            sum += ((Integer)list.remove(1)).intValue();

        System.out.println(sum);
    }
}
```

Java: A Type-safe, Imperative Language

- Find two bugs:

```
import java.io.*;
import java.util.*;

class Summation {
    public static void main(String[] args) {
        List list = new LinkedList();

        for (int i=0; i<args.length; ++i)
            list.add(args[i]);

        int sum = 0;
        while (!list.isEmpty())
            sum += (Integer)list.remove(1)).intValue();

        System.out.println(sum);
    }
}
```

**Cast
Exception!**

**OutOfBounds
Exception!**

Problems with Java

- Every Java cast operation is a potential crash
 - In Java, crash=exception instead of seg fault
- Some typecasting issues can be solved with Generics, but not all (e.g., list emptiness check)
- Problem: Java relies far too much on programmer-supplied typing annotations

Goals of Functional Languages

- In an “Advanced” Programming Language:
 - The compiler should tell you about typing errors in advance (not at runtime!)
 - The language structure should make it difficult to write programs that might crash (no unsafe casts!)
 - 80% of your time should be spent getting the program to compile, and only 20% on debugging
- For (much) more on this, take my APL course

But what if I'm an End-user?

- How to distinguish malicious/buggy code from non-malicious/non-vulnerable code?
 - No source code (sorry, no open source utopia)
 - Digital signatures don't help if developer is fallible
- System-specific, user-specific, role-specific security policies
- Language-based philosophy:
 - machine language is just another programming language
 - analyze it, rewrite it, verify producer-supplied proofs

An Example: Memory Safety

- Memory safety: ?
- Traditional security model:
 - program is a black box
 - OS/hardware intercepts every memory access
- Language-based security model:
 - program is a sequence of instructions
 - analyze the sequence to identify potential violations
 - insert dynamic memory checks into program

An Example: Memory Safety

- Memory safety: Programs may not access unallocated memory addresses
- Traditional security model:
 - program is a black box
 - OS/hardware intercepts every memory access
- Language-based security model:
 - program is a sequence of instructions
 - analyze the sequence to identify potential violations
 - insert dynamic memory checks into program

Reasons for Language-based Approach

- Efficiency
 - fewer context switches
 - avoid unnecessary security checks
- Flexibility
 - no need for custom OS/hardware
 - apply user-supplied security policies
- Richer Security Policies
 - history-based policies
 - information flow policies

History-based Policies

- policy: no network-sends after file-reads
- Traditional approach: none?
- Language-based approach: In-lined Reference Monitors
 - perform an automated program transformation:
 - inject a new state bit: `send_ok:=1`
 - after each file-read instruction, add an assignment: `send_ok:=0`
 - before each send instruction, add a guard:
`if (!send_ok) then throw SecurityException`
 - new program satisfies policy!

Information-flow Policies

- policy: don't divulge my credit card number
- Traditional approach:
 - monitor outgoing network traffic
 - block any transmission containing the relevant bit sequence
- Language-based approach:
 - analyze dataflow of program
 - reject flows from high-security sources to low-security sinks (e.g., network sends)

Decidability

- Is this really possible with a static analysis?
- The halting problem
 - Exercise: Reduce memory safety to halting
- Escape hatches:
 - conservative rejection
 - limit the domain (e.g., Java bytecode only)
 - dynamic checks

Where is Language-based Security Research Hot?

- Academia
 - Cornell, Carnegie Mellon, MIT, Princeton, Harvard, U. Penn, U.C. Berkeley, Yale, U. Illinois, U. Virginia, ... now UTD!
- Industry
 - Microsoft Research
 - Intel Research
 - Mobile phone companies (e.g., DoCoMo)

Tentative List of Topics

- This week and next: Intro to type theory & computer security (instructor lectures)
- Jan 23rd: Begin student presentations
- 1/23-1/30: Memory safety (SFI, CFI)
- 2/4-2/20: Minimizing the Trusted Computing Base (PCC, TAL)
- 2/25-2/27: Securing legacy (C++) code (Cyclone, CCured)

Tentative List of Topics (cont)

- 3/3-3/19: History-based Policies (IRM's, SASI, Mobile, Polymer)
- 3/24-3/31: Confidentiality / Information Flow
- 4/2-4/9: Obfuscation / Randomization
- Other topics?

Homework

- Read (see link on course webpage):

Fred B. Schneider, Greg Morrisett, Robert Harper. **A language-based approach to security**. *Informatics: 10 Years Back, 10 Years Ahead*. Lecture Notes in Computer Science, Vol. 2000, Springer-Verlag, Heidelberg, 86-101.

- First quiz Monday (on paper above)
- If you haven't already registered, do so!