18732: Secure Software Systems

Course Recap

Lujo Bauer

Spring 2015
Course Goals

1. Understand current software attacks and defenses
2. Understand general principles of secure software system design, implementation, and analysis
Topics

0. Attacks
1. Software Security Architectures
2. Security Analysis of Software
3. Language-based Security
4. Run-time Security Enforcement
Buffer Overflow Attacks

- One cause: Array abstraction not protected in C
  - Suppose array is 128 bytes
  - Possible to write 200 bytes into the array and overwrite return address of function

Program P: \texttt{exec( "/bin/sh" )}
Questions for the class

• What are some possible defenses?
  – Run-time Enforcement
  – Security Analysis of Software
  – Language-based Security

• What general security property is violated?
  Control Flow Integrity
Control Hijacking Attacks

- Attacker causes arbitrary attack code (e.g. root shell) to execute on target machine by hijacking the control flow of an application program

- Examples
  - Buffer overflow attacks
  - Integer overflow attacks
  - Format string vulnerabilities
A Reflected XSS Attack

1. Visit web site
2. Receive malicious page
3. Click on link
4. Echo user input
5. Send valuable data


Attack server

Victim client

Victim server
Questions for the class

• What are some possible defenses?
  – Run-time enforcement

• What general security property is violated?

  Information Flow
Web-based attacks
(Top 3 web site vulnerabilities)

1. Cross-site scripting (XSS)
2. Cross-site request forgery
3. SQL injection
Learning Outcome

• Understanding of the state-of-the-art in
  – control hijacking attacks and associated defenses
  – web-based attacks and associated defenses

• Note: These two classes of attacks currently top the charts of common security vulnerabilities databases
Topics

0. Attacks
1. Software Security Architectures
2. Security Analysis of Software
3. Language-based Security
4. Run-time Security Enforcement
Isolation (is good for security)

- Virtual machine-style architecture aims to provide isolation between multiple VM’s running on the same physical machine.

- Will cover a number of related approaches and their implications for security:
  - Java VM, Web security, Android security
Trusted Computing

How do we trust remote servers?
I'll ask what programs they have run!

How do I know if I can trust that server?

Hey server!

What code have you run?

Since I last rebooted, I’ve run
0: Operating System
1: Web Server
2: Shopping Cart Server

I trust those programs to protect my data!

Attestation to Code Integrity
Learning Outcome

• Systematic thinking about system security
  – What is the security mechanism?
  – What is the adversary model?
  – What is the desired security property?
  – Does system guarantee security property in the presence of adversaries?
  – What is the trusted computing base (TCB)? What are our assumptions about the TCB?

• Understanding of specific security architectures for isolation and attestation
Topics

0. Attacks
1. Software Security Architectures
2. Security Analysis of Software
3. Language-based Security
4. Run-time Security Enforcement
Security Analysis of Software

#include <stdio.h>

main(argc, argv)
int argc;
char * argv[];

/* the number of arguments in argv */
/* of null-terminated strings also known */
/* as an array of arrays of char. It */
/* holds the contents of the command line */
{
    int loopcounter;

    /* NOTE: The name with which the program */
    /* was invoked is held in argv[0]. */

    for (loopcounter=1; loopcounter < argc; loopcounter++) {
        switch (argv[loopcounter][0]) {
        case "-e":
            printf("OPTION \n",
            argv[loopcounter][1]);
            break;
        default : { printf("FILENAME \n",
                    argv[loopcounter]);
            break;
        }
        }
    }
}
Learning Outcome

• Understanding of
  – general methods of software analysis (model checking, static analysis, dynamic analysis)
  – their application to identifying security vulnerabilities in legacy code (buffer overflows, race conditions, setuid, chroot, malware) and for verifying higher-level security properties (e.g., memory isolation)

• Understanding of strengths and limitations of various methods

• Hands-on experience with a state-of-the-art industrial static analysis tool – Coverity Code Advisor

• Hands-on experience with a practical program verifier (and language) – SPARK

Assignment 3

Assignment 4
Topics

0. Attacks
1. Software Security Architectures
2. Security Analysis of Software
3. Language-based Security
4. Run-time Security Enforcement
Security Typed Languages

- **Type safe programming languages**
  - Think Java (as opposed to C)
  - Better protection for language abstractions such as arrays
    - No buffer overflow attacks on Java programs

- **Security-typed programming languages**
  - Language designed so that a program that compiles correctly does not incur security violation at run-time
What Is a Type System?

- Jif Example

```
int {alice\rightarrow bob, alice; bob\leftarrow alice} y;
int {bob\rightarrow bob} x;
int {alice\rightarrow bob; bob\rightarrow alice} z;
if (x == 0)
  z = y;
```

- Well-typed program won’t have security violations
- Type checking done by compiler
Learning Outcome

• Understanding of basic concepts of type systems for programming languages: syntax, static and dynamic semantics, type safety

• Understanding the security property of *non-interference* (an information flow property) and how type soundness can imply non-interference

• Understanding of type systems to improve security of mobile (PCC) and assembly code (TAL)

• Hands-on experience with building a secure interpreter that dynamically tracks information flows
Topics

0. Attacks
1. Software Security Architectures
2. Security Analysis of Software
3. Language-based Security
4. Run-time Security Enforcement
Run-time Security Enforcement

- Enforce security using mechanisms that monitor systems as they *execute* (as opposed to *static* analysis of code)

- Some examples
  - *Stackguard*: Dynamic checks to prevent buffer overflows
  - *Taint Analysis*: Information flow control, malware detection
  - Stack inspection, firewalls, applet sandboxing, displaying security warnings, …
Learning Outcome

• Understanding of fundamentals of run-time enforcement of security properties
  - What class of security policies can be enforced using run-time monitoring?
  - How do we specify policies for run-time monitors to enforce?

• Understanding of a general run-time mechanism for guaranteeing control flow integrity (CFI)
  - CFI guarantees absence of control hijacking attacks
Summary: Course Topics & Goals

Course Topics:
0. Attacks
1. Software Security Architectures
2. Security Analysis of Software
3. Language-based Security
4. Run-time Security Enforcement

Course Goals:
1. Understand current software attacks and defenses
2. Understand general principles of secure software design, implementation and analysis

- Control Flow Integrity
- Information Flow
A Final Comment

Do not try attacks at home! Our goal is to educate so you can *defend*, not *attack*.