Theory and Practice of Secure Systems

Vitaly Shmatikov

http://www.cs.utexas.edu/~shmat/courses/cs380s/
Course Logistics

- Lectures: Tuesday and Thursday, 2-3:30pm
- Instructor: Vitaly Shmatikov
  - Office: TAYLOR 4.115C
  - Office hours: Tuesday, 3:30-4:30pm (after class)
  - Open door policy – don’t hesitate to stop by!
- TA: Rolf Rolles
  - Office hours: Wed, 1-2pm in ENS 31NQ, desk #4
- No textbook; we will read a fair number of research papers
- Watch the course website for lecture notes, assignments, and reference materials
Grading

- **Homeworks**: 40% (4 homeworks, 10% each)
  - Homework problems will be based on research papers

- **Midterm**: 15%

- **Project**: 45%
  - Computer security is a contact sport – the best way to understand it is to get your hands dirty
  - Projects can be done individually or in small teams
  - *Project proposal* due in late September
    - More details later
  - I will provide a list of potential project ideas, but don’t hesitate to propose your own
Prerequisites

◆ Basic understanding of operating systems and memory management
  • At the level of an undergraduate OS course
◆ Some familiarity with cryptography is helpful
  • Cryptographic hash functions, public-key and symmetric cryptosystems
◆ Undergraduate course in complexity and/or theory of computation
◆ Ask me if you are not sure whether you are qualified to take this course
What This Course is **Not** About

- **Not** a comprehensive course on computer security
- **Not** a course on cryptography
  - We will cover some crypto when talking about provable security
- **Not** a seminar course
  - We will read and understand state-of-the-art research papers, but you’ll also have to do some actual work 😊
- Focus on several specific research areas
  - Mixture of theory and systems (very unusual!)
- You have a lot of leeway in picking your project
Correctness vs. Security

◆ Program or system **correctness**: program satisfies specification
  • For reasonable input, get reasonable output

◆ Program or system **security**: program properties preserved in face of attack
  • For unreasonable input, output not completely disastrous

◆ **Main difference**: adversary
  • Active interference from a malicious agent
  • It is very difficult to come up with a model that captures all possible adversarial actions
    – Look at how adversary is modeled in “systems” and in “theory”
The Meaning of the Lock

What does this lock mean?

How about this lock?

What does this lock mean?
Theme #1: Software Security

◆ Vulnerabilities and attacks
  - Memory corruption attacks
  - Access control violations and concurrency attacks
  - Web security: browsers and Web applications
  - Side-channel attacks (if time permits): timing, power

◆ Detecting and containing malicious behavior
  - Isolation, reference monitors, intrusion detection

◆ Preventing attacks
  - Memory protection
  - Applications of static analysis to security
  - Information flow control
Theme #2: Privacy

◆ Theoretical models
  • Semantic security
  • Secure multi-party computation
  • Introduction to zero knowledge
  • Key concept: provable security

◆ Data privacy
  • Query auditing and randomization
  • Privacy-preserving data mining
  • Differential privacy
And Now
Our Feature Presentation
Famous Internet Worms

◆ Morris worm (1988): overflow in fingerd
  • 6,000 machines infected (10% of existing Internet)

  • 300,000 machines infected in 14 hours

◆ SQL Slammer (2003): overflow in MS-SQL server
  • 75,000 machines infected in 10 minutes (!!!)

◆ Sasser (2004): overflow in Windows LSASS
  • Around 500,000 machines infected

◆ Conficker (2008-09): overflow in Windows Server
  • Around 10 million machines infected (estimates vary)
Why Are We Insecure?

- Of these, 87 are memory corruption vulnerabilities
- 73 are in applications providing remote services
  - 13 in HTTP servers, 7 in database services, 6 in remote login services, 4 in mail services, 3 in FTP services

Most exploits involve **illegitimate control transfers**
  - Jumps to injected attack code, return-to-libc, etc.
  - Therefore, most defenses focus on control-flow security

But exploits can also target **configurations, user data and decision-making values**
Memory Exploits

◆ **Buffer** is a data storage area inside computer memory (stack or heap)
  - Intended to hold pre-defined amount of data
  - If executable code is supplied as “data”, victim’s machine may be fooled into executing it
    - Code will self-propagate or give attacker control over machine

◆ **Attack can exploit any memory operation**
  - Pointer assignment, format strings, memory allocation and de-allocation, function pointers, calls to library routines via offset tables
**Stack Buffers**

- Suppose Web server contains this function
  
  ```c
  void func(char *str) {
    char buf[126];
    strcpy(buf, str);
  }
  ```

- When this function is invoked, a new frame is pushed onto the stack
  
  - Allocate local buffer (126 bytes reserved on stack)
  - Copy argument into local buffer

- Stack grows this way
  
  - buf
  - sfp
  - ret addr
  - str

  **Frame of the calling function**

  - Local variables
  - Pointer to previous frame
  - Execute code at this address after `func()` finishes
  - Arguments

  Top of stack
What If Buffer Is Overstuffed?

◆ Memory pointed to by str is copied onto stack...

```c
void func(char *str) {
    char buf[126];
    strcpy(buf,str);
}
```

◆ If a string longer than 126 bytes is copied into buffer, it will overwrite adjacent stack locations.

*strcpy does not check whether the string at *str contains fewer than 126 characters.*

This will be interpreted as return address!
Executing Attack Code

- Suppose buffer contains attacker-created string
  - For example, *str contains a string received from the network as input to some network service daemon

  ![Diagram showing code, ret, str, and stack]

  - Attacker puts actual assembly instructions into his input string, e.g., binary code of `execve("/bin/sh")`
  - In the overflow, a pointer back into the buffer appears in the location where the system expects to find return address

- When function exits, code in the buffer will be executed, giving attacker a shell
  - Root shell if the victim program is setuid root
Buffer Overflow Issues

- **Basic exploit**: executable attack code is stored on stack, in the buffer containing attacker’s string
  - Stack memory usually contains only data, but...

- **For the basic exploit**, overflow portion of the buffer must contain **correct address of attack code** in the RET position
  - The value in the RET position must point to the beginning of attack assembly code in the buffer
    - Otherwise application will crash with segmentation violation
  - Attacker must correctly guess in which stack position his buffer will be when the function is called
int bar (int val1) {
    int val2;
    foo (a_function_pointer);
}

int foo (void (*funcp)()) {
    char* ptr = point_to_an_array;
    char buf[128];
    gets (buf);
    strncpy(ptr, buf, 8);
    (*funcp)();
}
Attack #1: Return Address

1. Change the return address to point to the attack code. After the function returns, control is transferred to the attack code.

2. … or return-to-libc: use existing instructions in the code segment such as system(), exec(), etc. as the attack code.
Problem: No Range Checking

◆ **strcpy does not check input size**
  
  - **strcpy(buf, str)** simply copies memory contents into buf starting from *str until “\0” is encountered, ignoring the size of area allocated to buf

◆ Many C library functions are unsafe
  
  - **strcpy(char *dest, const char *src)**
  - **strcat(char *dest, const char *src)**
  - **gets(char *s)**
  - **scanf(const char *format, …)**
  - **printf(const char *format, …)**
Does Range Checking Help?

◆ `strncpy(char *dest, const char *src, size_t n)`
  - If `strncpy` is used instead of `strcpy`, no more than `n` characters will be copied from `*src` to `*dest`
    - Programmer has to supply the right value of `n`

◆ Potential overflow in `htpasswd.c` (Apache 1.3)

```
... strcpy(record, user);
    strcat(record, ":");
    strcat(record, cpw); ...
```

Copies username ("user") into buffer ("record"), then appends ":" and hashed password ("cpw")

◆ Published “fix” (do you see the problem?)

```
... strncpy(record, user, MAX_STRING_LEN-1);
    strcat(record, ":");
    strncat(record, cpw, MAX_STRING_LEN-1); ...
```
Published “fix” for Apache htpasswd overflow:

```c
... strncpy(record, user, MAX_STRING_LEN-1);  
    strcat(record, "\n:"");  
    strncat(record, cpw, MAX_STRING_LEN-1); ...
```

MAX_STRING_LEN bytes allocated for record buffer

- **contents of *user** : **contents of *cpw**
- Put up to MAX_STRING_LEN-1 characters into buffer
- Put "\n:"
- **Again** put up to MAX_STRING_LEN-1 characters into buffer
Off-By-One Overflow

● Home-brewed range-checking string copy

```c
void notSoSafeCopy(char *input) {
    char buffer[512]; int i;
    for (i=0; i<512; i++)
        buffer[i] = input[i];
}
void main(int argc, char *argv[]) {
    if (argc==2)
        notSoSafeCopy(argv[1]);
}
```

● 1-byte overflow: can’t change RET, but can change pointer to previous stack frame

  • On little-endian architecture, make it point into buffer
  • RET for previous function will be read from buffer!
Attack #2: Frame Pointer

1. Change the caller’s saved frame pointer to point to attack-controlled memory. Caller’s return address will be read from this memory.
C uses function pointers for callbacks: if pointer to F is stored in memory location P, then another function G can call F as (*P)(...)
Attack #3: Pointer Variables

① Change a function pointer to point to the attack code.

② Any memory, even outside the stack, can be modified by the statement that stores a value into the compromised pointer.

```
strncpy(ptr, buf, 8);
*ptr = 0;
```
Heap Overflow

- Overflowing buffers on heap can change pointers that point to important data
  - Sometimes can also transfer execution to attack code
  - Can cause program to crash by forcing it to read from an invalid address (segmentation violation)

- Illegitimate privilege elevation: if program with overflow has sysadm/root rights, attacker can use it to write into a normally inaccessible file
  - For example, replace a filename pointer with a pointer into buffer location containing name of a system file
    - Instead of temporary file, write into AUTOEXEC.BAT
Start Thinking About a Project

◆ A few ideas are on the course website
◆ Several ways to go about it
  • Build a tool that improves software security
    – Analysis, verification, attack detection, attack containment
  • Apply an existing tool to a real-world system
  • Demonstrate feasibility of some attack
  • Do a substantial theoretical study
  • Invent something of your own
◆ Start forming teams and thinking about potential topics early on!
Some Ideas (More Later)

- Sandboxes and reference monitors
- Enforcing security policies with transactions
- E-commerce protocols
  - Micropayment schemes, secure electronic transactions
- Wireless security
  - Ad-hoc routing, WiFi security, location security
- Enforcing legally mandated privacy policies
- Security for voice-over-IP
- Choose something that interests you!
Reading Assignment

◆ Read “Smashing the Stack for Fun and Profit” and “Blended Attacks”
  • Links on the course website

◆ For better understanding, read other reference materials on buffer overflow on the course site
  • Sotirov and Dowd’s “Bypassing Browser Memory Protections”
  • This will help when we talk about defenses later on