SMASHING THE STACK: OFFENSE AND DEFENSE

SMASHING THE STACK

RECAP SO FAR
- Threat Model
  - Enforcement mechanisms should anticipate potential attacks
- Security Principles
  - Guidelines for secure design
- Reasonable Assumptions
  - Assume the most powerful attacker possible for a given threat model
- Security Policies
  - Trust vs Trustworthiness
  - Dimensions of security: Confidentiality, Integrity, Availability (CIA)
- Enforcement Mechanisms
  - Isolation, Monitoring, Recovery

SMASHING THE STACK

TODAY
- Buffer Overflows
- Code Injection
- Fun with printf()
- Stack Smashing

SMASHING THE STACK

ALICE TAKES A TRIP
- Prof. Alice Smith is a young assistant professor who just got a paper accepted to a big prestigious conference overseas.
- She has been so busy preparing her talk that she's barely slept and will probably fall asleep during her talk unless she gets some sleep on the plane.
- Unfortunately, since the government has been shut down, all her grant money is frozen and she can barely afford a middle seat in coach class.
- A first-class ticket is out of the question, but then she remembered a funny thing that happened the last time she was booking a flight and fell asleep on the keyboard.

SMASHING THE STACK

AT HOME

To comply with the TSA Secure Flight program, the traveler information listed here must exactly match the information on the government-issued photo ID that the traveler presents at the airport.

Traveler 1 - Adult (age 18 to 64)

Name:
First Name: Alice
Middle Name:
Last Name: Smith
Date of Birth: 01/24/93
Gender: Female

To prevent identity theft, if the name on the government-issued photo ID does not match the name on your ticket, please complete the ID correction form in the bag you received.

Known Traveler Number/Pass ID (optional): 
Redress Number (optional): 

Seat Request: No Preference,isle or window
At the Check-in Desk

Alice Enters: "Dr. Alice Smith"
Airline Enters: "Economy"

Confused Ticketing Agent

How Might Alice Exploit this?

Alice Decides to Cheat the Airline

Airline Enters: “Economy”
Alice Enters: “Dr.” “Alice” “Smith”

Buffer Overflow
Contents:
- Overflling Protected Data
- Overflling Pointers to Code
- Vulnerabilities by Type (1998-2012)
- Famous Examples:
  - Sasser Worm (2000) [Windows]: Shutdown X-Ray machines at Swedish Hospital and caused Delta Airlines to cancel flights.
  - Stuxnet (2010) [Siemens Step 7 on PLCs via Windows]: Targeted Iran’s nuclear program and is believed to have caused damage to centrifuges.
  - Heartbleed (2012) [All platforms, but mainly UNIX-Like]: Error in the OpenSSL cryptography library—a widely used implementation of the Transport Layer Security (TLS) Protocol.

NIST Common Vulnerability and Exposure database (CVE)
Common Vulnerability Scoring System (CVSS) for 1988 to 2012 contains ~53,000 known vulnerabilities

- Buffer Overflow:
  - Total: 7,908
  - High: 5,528
  - Critical: 1,391

- Cross Site Scripting (XSS):
  - Total: 7,006
  - High: 141
  - Critical: 0

Heardleed (2012) [All platforms, but mainly UNIX-Like]: Error in the OpenSSL cryptography library—a widely used implementation of the Transport Layer Security (TLS) Protocol.
void main()
{
    int x, y, z;
    x = rand();
    y = x / rand();
    z = y / 2.47;
}

// January 1, 1970 is the UNIX Epoch, i.e. UNIX's D.O.B.
// call another function to write log message to disk
// append it to the log message after the date
// buffer where we'll construct a log message
// copy date into the first 'len' chars of the log message
// get user to type in some text
// append it to the log message after the date
// call another function to write log message to disk
// buffer into which we'll read user input
// buffer where we'll construct a log message
// find out how many characters are in the date
// copy data into the first 'len' chars of the log message
// get user to type in some text
// append it to the log message after the date
// call another function to write log message to disk

// Infinite loop!

// Place a sacrificial random number (canary) on the stack just below the function return address
// Inject code to check canary health
// If canary is dead (random number has changed), the stack is corrupt
// corrective action can be taken
SMASHING THE STACK

DEFENSE: DATA EXECUTION PREVENTION

▷ Problem: Code injected as data can be executed
▷ Solution: Tag memory as either code or data
▷ Data Execution Prevention (DEP)
 ▷ Data memory can not be executed
 ▷ Implementations:
    ▷ Hardware: Separate physical memory locations for data and code
    ▷ Software: Separate logical locations for data and code (e.g., page permissions)
 ▷ All modern OSes and CPU architectures have some form of DEP

SMASHING THE STACK

ATTACK: BINARY CODE REUSE

▷ But why bother injecting new code when there are millions of lines of it lying around in existing binaries?
▷ Sneaky idea: Subvert existing code to defeat DEP!
▷ Lots of abuse of common libraries like libc — almost always there
▷ Use system function in libc to execute arbitrary commands
▷ Load stack with command and libc addresses

SMASHING THE STACK

ATTACK: COMMAND INJECTION

/*
 * Seriously sloppy program to copy one file to another.
 */
int main() {
    char from[64];
    char to[64];
    char cmd[96];
    printf("Enter name of file to copy from: "); // user enters "foo"
    gets(from);
    printf("Enter name of file to copy to: "); // User enters "bar"
    gets(to);
    strcat(cmd, "cp ");
    strcat(cmd, from);
    strcat(cmd, " ");
    strcat(cmd, to);
    system(cmd);
    exit(1);
}

What we expect to execute is: 
cp foo bar
But what if the user types bar; rm -rf / at the second prompt?

SMASHING THE STACK

DEFENSE: ADDRESS RANDOMIZATION

▷ Prevent ROP by randomizing addresses of functions and data each time the program is run
▷ Address Space Layout Randomization (ASLR)
▷ Unfortunately, not every system implements ASLR at the same granularity
  ▷ Kernel pages are often skipped
  ▷ Some approaches re-link libraries nightly
▷ Linux has had automatic ASLR since 2005. Can be switched off by root.
▷ Windows has had ASLR since Vista (2007), developers must link to library.
PICKING ON GETS() AGAIN

void vulnerable() {
    char buf[64];
    ... 
    gets(buf);
    ... 
}

void still_vulnerable() {
    char *buf = malloc(64);
    ... 
    gets(buf);
    ... 
}

void safe() {
    char buf[64];
    ... 
    fgets(buf, 64, stdin);
    ... 
}

void safer() {
    char buf[64];
    ... 
    fgets(buf, sizeof(buf), stdin);
    ... 
}

void vulnerable(char *data, int len) {
    char buf[64];
    if (len <= 64)
        memcpy(buf, data, len);
}

void safe(char *data, size_t len) {
    char buf[64];
    if (len <= 64)
        memcpy(buf, data, len);
}

void foo(char *data, size_t len) {
    char *buf = malloc(len+2);
    if (buf != null) {
        memcpy(buf, data, len);
        buf[len] = '\n';
        buf[len+1] = '\0';
    }
}

Discuss with your seatmates... Is this safe?
printf("100% sir!");
printf("100% sir!");
printf("%d %d %d %d\n", score);
printf("%d %s\n", score);
printf("%d %d %d %d\n", score);
printf("%d %s\n", score);
printf("%d %d %d %d\n", score);
printf("%d %s\n", score);

void vulnerable() {
    char buf[64];
    if (fgets(buf, sizeof(buf), stdin) != NULL) {
        printf("100% dude!");
        printf("100% sir!");
        printf("%d %d %d %d\n", score);
        printf("%d %s\n", score);
        printf("%d %d %d %d\n", score);
        printf("%d %s\n", score);
        printf("%d %d %d %d\n", score);
        printf("%d %s\n", score);
    }
}

EXPLOITING BUFFER OVERFLOWS: STACK SMASHING

- "Smashing the Stack for Fun and Profit" by AlephOne. Phrack 49, 1996.
- "Smashing the Stack For Fun and Profit (Today)" by Travis Finkenauer, 2016.
  - https://travisf.net/smashing-the-stack-today
- "Smashing the Stack for Fun & Profit : Revised" by avicoder, Feb. 01, 2016.
  - https://avicoder.me/2016/02/01/smashstack-revived/