CMPS 122: COMPUTER SECURITY

SMASHING THE STACK: OFFENSE AND DEFENSE
SMASHING THE STACK

TODAY

▸ Buffer Overflows
▸ Code Injection
▸ Fun with printf()
▸ Stack Smashing
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

TECHNIQUES FOR BUILDING SECURE SYSTEMS

- Goals: try to prevent, otherwise mitigate, at least detect
- Run-time checks / monitoring
- Address randomization (ASLR)
- Non-executable stack and heap
- Defensive programming
- Coding standards/reviews
- Bug-finding tools
- Use a safe language
- Constrain user inputs
- Contain damage
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.
## Traveler 1 - Adults (age 18 to 64)

To comply with the [TSA Secure Flight program](https://www.tsa.gov/about-tsa), the traveler information listed here must exactly match the information on the government-issued photo ID that the traveler presents at the airport.

<table>
<thead>
<tr>
<th>Title (optional):</th>
<th>First Name:</th>
<th>Middle Name:</th>
<th>Last Name:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dr.</td>
<td>Alice</td>
<td></td>
<td>Smith</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Gender:</th>
<th>Date of Birth:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Female</td>
<td>01/24/93</td>
</tr>
</tbody>
</table>

Travelers are required to enter a middle name/initial if one is listed on their government-issued photo ID.

Some younger travelers are not required to present an ID when traveling within the U.S. [Learn more](#).

- **Known Traveler Number/Pass ID (optional):** ✉️
- **Redress Number (optional):** ✉️

**Seat Request:**
- ☐ No Preference
- ☑️ Aisle
- ☐ Window
SMASHING THE STACK

AT THE CHECK IN DESK

NZ001 LHR-AUK 010817 1230
DR ALICE SMITH
ECONOMY
SEAT: ANY
NOTES: NONE
# SMASHING THE STACK

## ASLEEP ON THE KEYBOARD

### Traveler 1 - Adults (age 18 to 64)

To comply with the [TSA Secure Flight program](https://www.tsa.gov), the traveler information listed here must exactly match the information on the government-issued photo ID that the traveler presents at the airport.

<table>
<thead>
<tr>
<th>Title (optional):</th>
<th>First Name:</th>
<th>Middle Name:</th>
<th>Last Name:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dr.</td>
<td>Alice</td>
<td></td>
<td>Smithhhhhhhhhhhhh</td>
</tr>
</tbody>
</table>

Gender: Male

Date of Birth: 01/24/93

Travelers are required to enter a middle name or initial if one is listed on their government-issued photo ID.

Some younger travelers are not required to present an ID when traveling within the U.S. [Learn more](https://www.tsa.gov).

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

Seat Request:
- ☐ No Preference
- ☒ Aisle
- ☐ Window
Back at the Airport
NZ001 LHR-AUK 010817 1230
DR ALICE SMITHHHHHHHHH
HHHHHOMY
SEAT: ANY
NOTES: NONE

How Might Alice Exploit this?
To comply with the [TSA Secure Flight program](https://www.tsa.gov), 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 - Adults (age 18 to 64)**

<table>
<thead>
<tr>
<th>Title (optional):</th>
<th>First Name:</th>
<th>Middle Name:</th>
<th>Last Name:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dr.</td>
<td>Alice</td>
<td></td>
<td>Smith</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Gender:</th>
<th>Date of Birth:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Female</td>
<td>01/24/93</td>
</tr>
</tbody>
</table>

Travellers are required to enter a middle name, if listed on their government-issued photo ID.

Some younger travellers are not required to present an ID when traveling within the U.S. [Learn more](https://www.tsa.gov).
SMASHING THE STACK

...AND SCORES AN UPGRADE

NZ001 LHR-AUK 140218 1230
DR ALICE SMITH
FIRST
SEAT: ANY
NOTES: NONE
SMASHING THE STACK

BUFFER OVERFLOW

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

Name Buffer

Class Buffer

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

Name Buffer

Class Buffer
char name[16+1];
char class[7+1];

void vulnerable() {
    ...
    gets(class);
    ...
    gets(name);
    ...
}
void vulnerable() {
    ... 
    gets(name);
    ...
}
SMASHING THE STACK

VULNERABILITIES BY TYPE (1998–2012)

SMASHING THE STACK

BUFFER VULNERABILITIES

- 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
- Famous Buffer Overflows:
  - **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
SMASHING THE STACK

HEARTBLEED (À LA XKCD)

SERVER, ARE YOU STILL THERE?
IF SO, REPLY "POTATO" (6 LETTERS).

Secure connection using key "453853874224".
User Meg wants these 6 letters: POTATO. User
wants pages about "lili games". Unlocking
secure records with master key 513098573341.

User Meg wants these 6 letters: POTATO. User
wants pages about "lili games". Unlocking
secure records with master key 513098573341.
SMASHING THE STACK

HEARTBLEED (À LA XKCD)

SERVER, ARE YOU STILL THERE? IF SO, REPLY "BIRD" (4 LETTERS).

HMM...

User Meg wants these 4 letters: BIRD. There are currently 34 connections open. User Brendan uploaded the file /usr/files-3843.
SMASHING THE STACK

HEARTBLEED (À LA XKCD)

User Meg wants these 500 letters: "HAT. Lucas requests the "missed connections" page. Eve (administrator) wants to set server's master key to "14835038534". Isabel wants pages about snakes but not too long". User Karen wants "HAT. Lucas requests the "missed connections" page. Eve (administrator) wants to set server's master key to "14835038534". Isabel wants pages about snakes but not too long".
GETS(3) Linux Programmer's Manual GETS(3)

NAME

gets - get a string from standard input (DEPRECATED)

SYNOPSIS

einclude <stdio.h>

char *gets(char *s);

DESCRIPTION

Never use this function.

gets() reads a line from stdin into the buffer pointed to by s until either a terminating newline or EOF, which it replaces with a null byte ('\0'). No check for buffer overrun is performed (see BUGS below).

RETURN VALUE

gets() returns s on success, and NULL on error or when end of file occurs while no characters have been read. However, given the lack of buffer overrun checking, there can be no guarantees that the function will even return.

BUGS

Never use gets(). Because it is impossible to tell without knowing the data in advance how many characters gets() will read, and because gets() will continue to store characters past the end of the buffer, it is extremely dangerous to use. It has been used to break computer security. Use fgets() instead.

For more information, see CWE-242 (aka 'Use of Inherently Dangerous Function') at http://cwe.mitre.org/data/definitions/242.html
C library function `gets()` is the problem; if user enters more than 127 characters, `buf` will overflow.
SMASHING THE STACK

DEFENSE: STACK CANARIES

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

DEFENSE: STACK CANARIES

```c
void dodgier(char *date) {
  int len;
  char buf[64]; // buffer into which we’ll read user input
  char msg[96]; // buffer where we’ll construct a log message

  len = strlen(date); // find out how many characters are in the date
  strcpy(msg, date); // copy date into the first ‘len’ chars of the log message
  gets(buf); // get user to type in some text
  strcpy(msg+len, buf); // append it to the log message after the date
  write_log(msg); // call another function to write log message to disk
}

int main() {
  int x, y, z;
  dodgier("01/01/1970"); // January 1, 1970 is the UNIX Epoch, i.e. UNIX’s D.O.B.
  x = rand();
  y = x / rand();
  z = y / 2.47;
  exit(1);
}
```
SMASHING THE STACK

ATTACK: AVOIDING STACK CANARIES

gets() reads attack string and overflows buf, but not by so much it overwrites (kills) the canary

Instead, it sneakily overwrites the value of len and installs new code into buf

strcpy() now uses a subverted start address (msg+X+Y) and the real return address is replaced with a fake

⇒ Infinite loop! 😎

X = size of msg + size of buf + size of len
Y = size of canary

epoch

\[
\begin{align*}
\text{len} &= 0 \\
\text{strncpy(msg, epoch) \\
\text{gets(buf) \\
\text{strncpy(msg+len, buf)} \\
\text{write_log(msg) \\
\text{check_canary()}} \\
\text{dodgier(&epoch) \\
\text{exit(1)}}
\end{align*}
\]

\[
\begin{align*}
\text{len} &= 0 \\
\text{strncpy(msg, epoch) \\
\text{gets(buf) \\
\text{strncpy(msg+len, buf)} \\
\text{write_log(msg) \\
\text{check_canary()}} \\
\text{dodgier(&epoch) \\
\text{exit(1)}}
\end{align*}
\]

\[
\begin{align*}
\text{len} &= 0 \\
\text{strncpy(msg, epoch) \\
\text{gets(buf) \\
\text{strncpy(msg+len, buf)} \\
\text{write_log(msg) \\
\text{check_canary()}} \\
\text{dodgier(&epoch) \\
\text{exit(1)}}
\end{align*}
\]
SMASHING THE STACK

IS THIS FINE?
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 cannot 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
Example:

Return Oriented Programming (ROP)

When good instructions go bad: generalizing return-oriented programming to RISC

Erik Buchanan, Ryan Roemer, Hovav Shacham, and Stefan Savage
Department of Computer Science & Engineering, University of California San Diego

https://dl.acm.org/citation.cfm?id=1455776
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.
/*
 * Seriously sloppy program to copy one file to another.
 */
int main() {
    char from[64];
    char to[64];
    char cmd[96];

    /*
       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);
}"
PICKING ON GETS() AGAIN

**Linux**

```bash
$ gcc -o sloppy sloppy.c 
/tmp/ccK7LeFB.o: In function ‘main’:
  sloppy.c:(.text+0x24): warning: the ‘gets’ function is dangerous and should not be used.
```

**macOS**

```bash
$ ./sloppy
warning: this program uses gets(), which is unsafe.
Enter name of file to copy from:
```
void vulnerable() {
    char buf[64];
    ...
    gets(buf);
    ...
}

void still_vulnerable?() {
    char *buf = malloc(64);
    ...
    gets(buf);
    ...
}
IE’s Role in the Google-China War

By Richard Adhikari
TechNewsWorld
01/15/10 12:25 PM PT

The hack attack on Google that set off the company’s ongoing standoff with China appears to have come through a zero-day flaw in Microsoft’s Internet Explorer browser. Microsoft has released a security advisory, and researchers are hard at work studying the exploit. The attack appears to consist of several files, each a different piece of malware.

Computer security companies are scrambling to cope with the fallout from the Internet Explorer (IE) flaw that led to cyberattacks on Google (Nasdaq: GOOG) and its corporate and individual customers.

The zero-day attack that exploited IE is part of a lethal cocktail of malware that is keeping researchers very busy.

“We’re discovering things on an up-to-the-minute basis, and we’ve seen about a dozen files dropped on infected PCs so far,” Dmitri Alperovitch, vice president of research at McAfee Labs, told TechNewsWorld.

The attacks on Google, which appeared to originate in China, have sparked a feud between the Internet giant and the nation’s government over censorship, and it could result in Google pulling away from its business dealings in the country.

Pointing to the Flaw

The vulnerability in IE is an invalid pointer reference, Microsoft (Nasdaq: MSFT) said in a security advisory 9798552, which it issued on Thursday. Under certain conditions, the invalid pointer can be accessed after an object is deleted; the advisory states. In specially crafted attacks, like the ones launched against Google and its customers, IE can allow remote execution of code when the flaw is exploited.
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?
Broward County, Fla. -- The Broward County Elections Department has egg on its face today after a computer glitch misreported a key amendment race, according to WPLG-TV in Miami.

Amendment 4, which would allow Miami-Dade and Broward counties to hold a future election to decide if slot machines should be allowed at racetracks, was thought to be tied. But now that a computer glitch for machines counting absentee ballots has been exposed, it turns out the amendment passed.

"The software is not geared to count more than 32,000 votes in a precinct. So what happens when it gets to 32,000 is the software starts counting backward," said Broward County Mayor Ilene Lieberman.

That means that Amendment 4 passed in Broward County by more than 240,000 votes rather than the 166,000-vote margin reported Wednesday night. That increase changes the overall statewide results in what had been a neck-and-neck race, one for which recounts had been going on today. But with news of Broward's error, it's clear amendment 4 passed.
void vulnerable() {
    char buf[64];
    if (fgets(buf, sizeof(buf), stdin) != NULL) {
        printf(buf);
    }
}

printf("New high score! %d\n", score);
SMASHING THE STACK

FUN WITHPRINTF()

```c
printf("100% dude! ");

printf("100% sir! ");

printf("%d %d %d %d ...");

printf("%d %s");
```
$ gcc -o fun-with-printf fun-with-printf.c
fun-with-printf.c:23:12: warning: format '%d' expects a matching 'int' argument [-Wformat=
  printf("100% dude!\n");
  ^

fun-with-printf.c:24:12: warning: format '%d' expects a matching 'int' argument [-Wformat=
  printf("%d %s\n");
  ^

fun-with-printf.c:24:12: warning: format '%s' expects a matching 'char *' argument [-Wformat=
fun-with-printf.c:25:12: warning: format '%d' expects a matching 'int' argument [-Wformat=
  printf("%d %d %d %d\n");
  ^

fun-with-printf.c:25:12: warning: format '%d' expects a matching 'int' argument [-Wformat=
fun-with-printf.c:25:12: warning: format '%d' expects a matching 'int' argument [-Wformat=
fun-with-printf.c:25:12: warning: format '%d' expects a matching 'int' argument [-Wformat=
fun-with-printf.c:25:12: warning: format '%d' expects a matching 'int' argument [-Wformat=
  printf("100% sir!");
  ^

fun-with-printf.c:26:12: warning: format '%s' expects a matching 'char *' argument [-Wformat=}

38
void vulnerable() {
    char buf[64];
    if (fgets(buf, sizeof(buf), stdin) != NULL) {
        printf(buf);
    }
}

void safe() {
    char buf[64];
    if (fgets(buf, sizeof(buf), stdin) != NULL) {
        printf("%s", buf);
    }
}
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](https://travisf.net/smashing-the-stack-today)
- “Smashing the Stack for Fun & Profit : Revived” by avicoder, Feb. 01, 2016.
  - [https://avicoder.me/2016/02/01/smashsatck-revived/](https://avicoder.me/2016/02/01/smashsatck-revived/)
CMPS 122: COMPUTER SECURITY

SMASHING THE STACK: LIVE DEMO