Overflows, Injection, & Memory Safety

Internet of Shit...
- A device produced by the lowest bidder...
  - That you then connect through the network
- This has a very wide **attack surface**
  - Methods where an attacker might access a vulnerability
- And its often incredibly **cost sensitive**
  - Very little support after purchase
  - So things don’t get patched
  - No way for the user to tell what is "secure" or "not"
  - But they can tell what is cheaper!
  - And often it is *insanely* insecure:
    - Default passwords on telnet of admin/admin...
    - Trivial buffer overflows

Net Of A Million Spies...
- Device only communicates through a central service
  - Greatly reduces the attack surface but...
- Most of the companies running the service are "Data Asset" companies
  - Make their money from advertising, not the product themselves
  - May actually subsidize the product considerably
  - Some you know about: Google, Amazon
  - Some you may not: Salesforce
- Only exception of note is Apple:
  - I’ll talk about HomeKit later...
    - But you still have to trust that the HomeKit product doesn’t report to a third party.
Traveler Information

Traveler 1 - Adults (age 18 to 64)

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.

Title (optional): First name: Middle name: Last name:
Dr. Alice Smith
Gender: Date of Birth: Travelers are required to enter a middle name/initial if one is listed on their government-issued photo ID.
Female 5/12/93
Some younger travelers are not required to present an ID when traveling within the U.S. Look/Check

Known Traveler Number/Pass ID (optional) [ ]

Redress Number (optional) [ ]

Sofl Request:
[ ] No Preference [ ] Audio [ ] Window
#293 HRE-THR 850 1930
ALICE SMITH
SPECIAL INSTRUX: NONE

How could Alice exploit this?
Find a partner and talk it through.
char name[20];
char instrux[80] = "none";

void vulnerable() {
    ...
    gets(name);
    ...
}
char name[20];
int authenticated = 0;
void vulnerable()
{
    ...
    gets(name);
    ...
}

char line[512];
char command[] = "/usr/bin/finger";

void main()
{
    ...
    gets(line);
    ...
    execv(command, ...);
}
void vulnerable() {
    char buf[64];
    ...
    gets(buf);
    ...
}

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

IE's Role in the Google-China War

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 a dozen files dropped on infected PCs as of now," Dimitri 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.

The vulnerability in IE is an invalid pointer reference, Microsoft (Nasdaq: MSFT) said in security advisory 779555, which it issued on Thursday. Under certain conditions, the invalid pointer can be accessed after an object is deleted, the advisory said. In some cases, crafted attacks that exploit this vulnerability in certain versions of IE can allow remote execution of code when the flaw is exploited.

Linux (32-bit) process memory layout

- Reserved for Kernel: -0xFFFFFFFF
- user stack: 0xC0000000
- shared libraries: 0x40000000
- run time heap: 0x08048000
- static data segment: -0x00000000
- unused: -0x00000000
- text segment (program): -0x00048000
- brk: -0x00048000
- $esp: -0x00000000
- loaded from exec: -0x00000000
void safe() {
    char buf[64];
    ...
    fgets(buf, 64, stdin);
    ...
}

void safer() {
    char buf[64];
    ...
    fgets(buf, sizeof(buf), stdin);
    ...
}
void vulnerable(int len, char *data) {
    char buf[64];
    if (len > 64)
        return;
    memcpy(buf, data, len);
}

memcpy(void *s1, const void *s2, size_t n);

Assume these are both under the control of an attacker.

size_t is unsigned. What happens if len == -1?

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

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

Is it safe? Talk to your partner.

Vulnerable! If len = 0xffffffff, allocates only 1 byte.

---

**Broward Vote-Counting Blunder Changes Amendment Result**

POSTED: 1:34 pm EST November 4, 2004

**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.
```c
void vulnerable()
{
    char buf[64];
    if (fgets(buf, 64, stdin) == NULL)
        return;
    printf(buf);
}
```

```c
printf("you scored %d\n", score);
```

```c
printf("a %s costs $%d\n", item, price);
```
Fun With `printf` format strings...

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

Format argument is missing!

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

⇒ prints value 4 bytes above retaddr as integer

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

⇒ prints bytes pointed to by that stack entry up through first NUL

```c
printf("%d %d %d ...\n");
```

⇒ prints series of stack entries as integers

```c
printf("%d %s");
```

⇒ prints value 4 bytes above retaddr plus bytes pointed to by preceding stack entry

```c
printf("100% nuke\’m!");
```

What does the `%n` format do??
Fun With `printf` format strings...

```c
printf("100% dude!\n");
⇒ prints value 4 bytes above retaddr as integer
printf("100% sir!\n");
⇒ prints bytes pointed to by that stack entry
up through first NUL
printf("%d %d %d ...\n");
⇒ prints series of stack entries as integers
printf("%d %s\n");
⇒ prints value 4 bytes above retaddr plus bytes
pointed to by preceding stack entry
printf("100% nuke’em!\n");
⇒ writes the value 3 to the address pointed to by stack entry
```

And Now:

**Lets Walk Through A Stack Overflow**

- Idea: We override a buffer on the stack...
  - In the buffer we place some code of our choosing
    - "Shellcode"
  - Override the return address to point to code of our choosing
- Lets step through the process on an x86...