# CSE 127:
# Buffer Overflow

**George Obaido, Ph.D.**

UCSD

Spring 2022 Lecture 2

UC San Diego

# How did you spend your day?

Go to **www.menti.com** and use the code **4959 1104**

# Stack Buffer Overflows

My nephew got angry when I told him: **"The sky is the limit for you".**

Why:

My nephew got angry when I told him: **"The sky is the limit for you".**

Why:

He wants to be an astronaut.

# When is a program secure?

- Formal approach: When it does exactly what it should
  - Not more
  - Not less
- But how do we know what it is supposed to do?

# When is a program secure?

- Formal approach: When it does exactly what it should
  - Not more
  - Not less
- But how do we know what it is supposed to do?
  - Somebody tells us? (Do we trust them?)
  - We write the code ourselves? (What fraction of the software you use have you written?)

# When is a program secure?

- Pragmatic approach: When it doesn't do bad things
- Often easier to specify a list of "bad" things:
  - Delete or corrupt important files
  - Crash my system
  - Send my password over the internet
  - Send threatening email to the professor

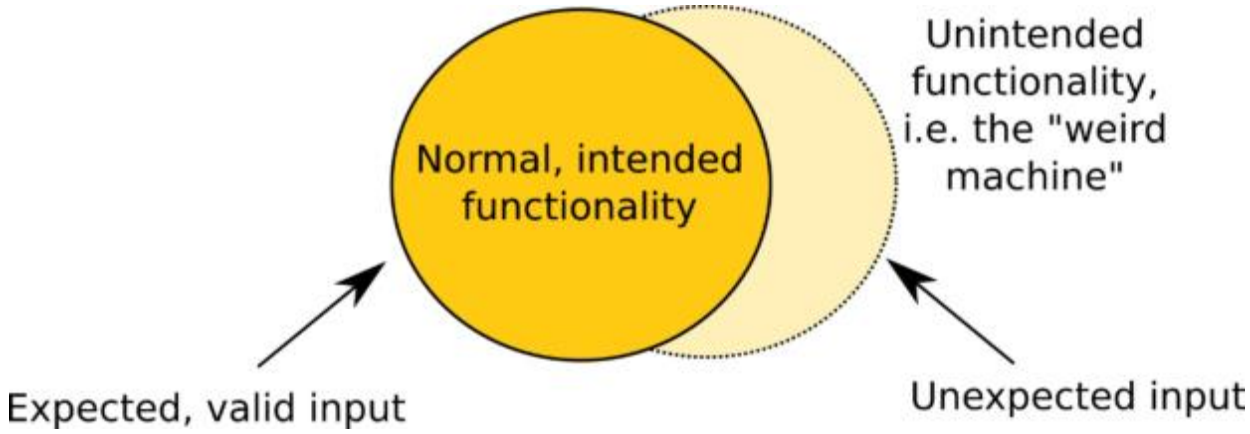
# When is a program secure?

What if the program doesn't do bad things, but could?

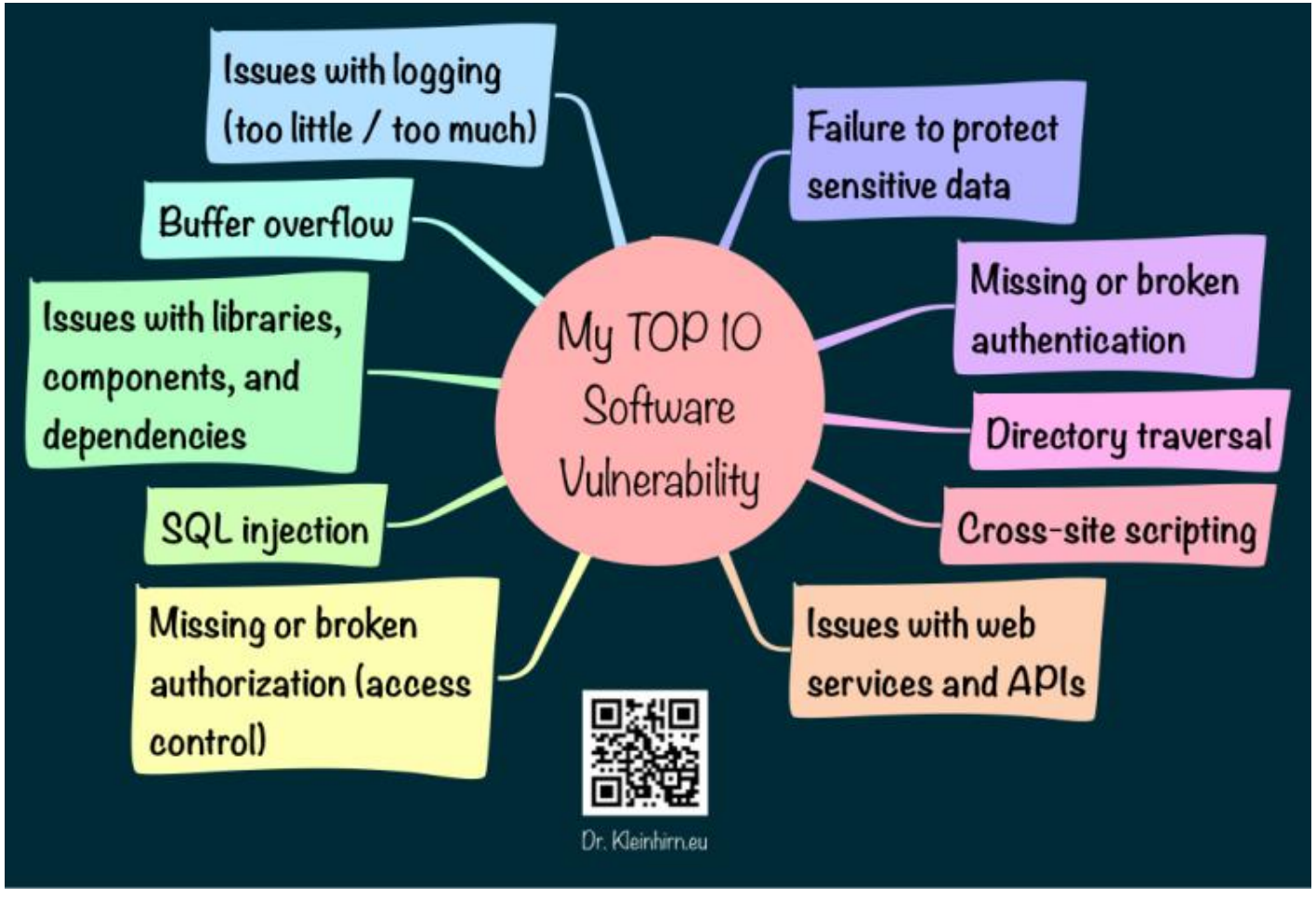
Is it secure?

# Weird machines

- Complex systems contain unintended functionality




- Attackers can trigger this unintended functionality
  - i.e. they are exploiting vulnerabilities




**Source:** https://www.biggerplate.com/mindmaps/84Hrd05h/my-top-10-software-vulnerability

# What is a software vulnerability?

# What is a software vulnerability?

- A bug is a program that allows an unprivileged user capabilities that should be denied to them

# What is a software vulnerability?

- A bug is a program that allows an unprivileged user capabilities that should be denied to them
- There are many types of vulnerabilities
- Today: bugs that violate "control *fl*ow integrity"
  - Why? This lets an attacker run code on your computer!

# What is a software vulnerability?

- A bug is a program that allows an unprivileged user capabilities that should be denied to them
- There are many types of vulnerabilities
- Today: bugs that violate "control flow integrity"
  - Why? This lets an attacker run code on your computer!
- Typically these involve violating *assumptions* of the programming language or its runtime

# Exploiting vulnerabilities (the start)

- Dive into low-level details of how exploits work
    - How can a remote attacker get a victim program to execute their code?
- Threat model: Victim code is handling input that comes from across a security boundary
    - What are some examples of this?
- Security policy: Want to protect integrity of execution and confidentiality of data from being compromised by malicious and highly skilled users of our system.

# Scenario 1

As a proud script kid, Bob enjoys adding SQL code to an application's input form to gain access to resources and make changes to data. What kind of attack is this?

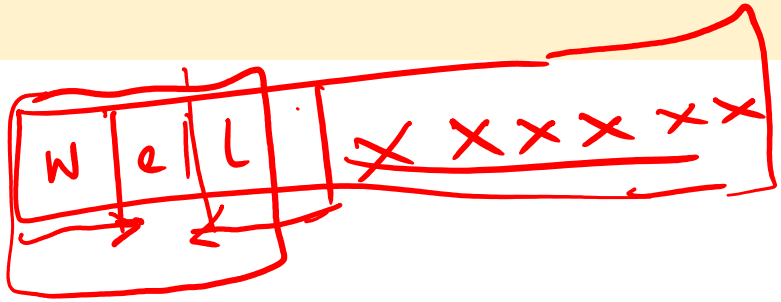
a. SQL Injection
b. Buffer Overflow
c. Cross Site Scripting
d. None of the above

Go to **www.menti.com** and use the code: **9041 8629**

# Scenario 2

As a new intern at ABC Inc, Alice erroneously performed the below into a memory for an array with 3 elements. What kind of vulnerability is this?

```
int main() {
    char s[3];
    strcpy(s, "welcome to my program");
}
```




a. SQL Injection
b. Buffer Overflow
c. Cross Site Scripting
d. None of the above

Go to **www.menti.com** and use the code: **9523 0539**

# Today: Stack buffer overflows

Lecture objectives:

- Understand how buffer overflow vulns can be exploited
- Identify buffer overflow and assess their impact
- Avoid introducing buffer overflow vulnerabilities
- Correctly fix buffer overflow activities

# Buffer overflows

- Definition: An anomaly that occurs when a program writes data beyond the boundary of a buffer
- Archetypal software vulnerability
  - Ubiquitous in system software (C/C++)
  - OSes, web servers, web browsers, etc.
  - If your program crashes with memory faults, you probably have a buffer overflow vulnerability

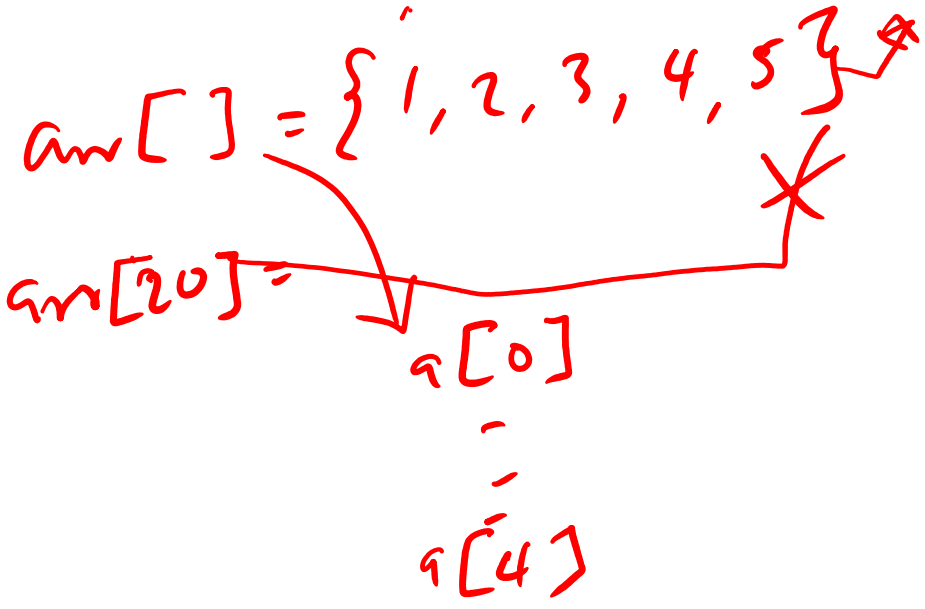
# Why are they interesting?

- Core concept → broad range of possible attacks
  - Sometimes a single byte is all an attacker needs
- Ongoing arms race between defenders and attackers
  - Co-evolution of defenses and exploitation techniques

# How are they introduced?

# How are they introduced?

- No automatic bounds checking in C/C++
  - C/C++ fails to detect whether a variable is within some bounds.



# How are they introduced?

- No automatic bounds checking in C/C++
  - C/C++ fails to detect whether a variable is within some bounds.

- The problem is made more acute by the fact that many **C stdlib** functions make it easy to go past bounds.

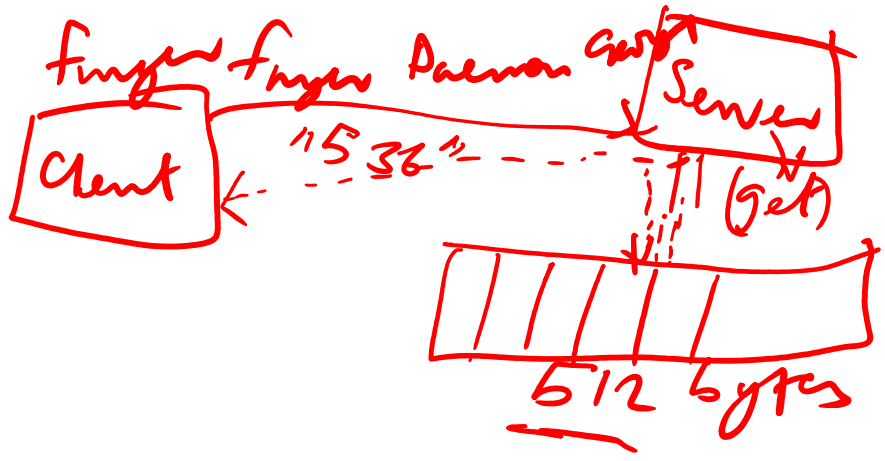
- String manipulation functions like **gets()**, **strcpy()**, and **strcat()** all write to the destination buffer until they encounter a terminating '\0' byte in the input

# How are they introduced?

- No automatic bounds checking in C/C++
  - C/C++ fails to detect whether a variable is within some bounds.
- The problem is made more acute by the fact that many **C stdlib** functions make it easy to go past bounds.
- String manipulation functions like **gets(), strcpy()**, and **strcat()** all write to the destination buffer until they encounter a terminating '\0' byte in the input
- Whoever is providing the input (often from the other side of a security boundary) controls how much gets written

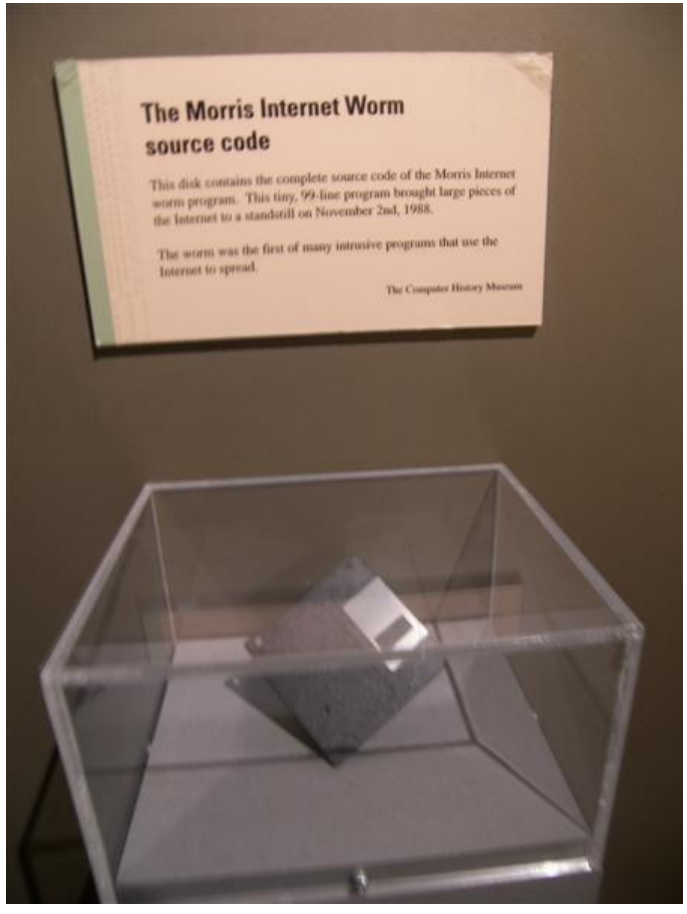
# Let's look at the finger daemon in BSD 4.3

```
/*
 * Finger server.
 */
#include <sys/types.h>
#include <netinet/in.h>

#include <stdio.h>
#include <ctype.h>

main(argc, argv)
        char *argv[];
{
        register char *sp;
        char line[512];
        struct sockaddr_in sin;
        int i, p[2], pid, status;
        FILE *fp;
        char *av[4];

        i = sizeof (sin);
        if (getpeername(0, &sin, &i) < 0)
                fatal(argv[0], "getpeername");
        line[0] = '\0';
        gets(line);
        sp = line;
        av[0] = "finger";
        i = 1;
        while (1) {
                while (isspace(*sp))
                        sp++;
                if (!*sp)
                        break;
                if (*sp == '/' && (sp[1] == 'W' || sp[1] == 'w')) {
                        sp += 2;
                        av[i++] = "-l";
                }
                if (*sp && !isspace(*sp)) {
                        av[i++] = sp;
                        while (*sp && !isspace(*sp))
                                sp++;
                        *sp = '\0';
                }
        }
```



# Morris worm

- This fingerd vuln was one of several exploited by the Morris worm in 1988
- Created by Robert Morris, graduate student at Cornell.
- One of the first internet worms
- Devastating effect on the internet
- Took over thousands of computers and shut down large chunks of the internet
- First conviction under CFAA



That was over 30+ years ago!

Surely buffer overflows are no longer a problem...

# Project Zero

News and updates from the Project Zero team at Google

Thursday, July 16, 2020

## MMS Exploit Part 1: Introduction to the Samsung Qmage Codec and Remote Attack Surface

Posted by Mateusz Jurczyk, Project Zero

*This post is the first of a multi-part series capturing my journey from discovering a vulnerable little-known Samsung image codec, to completing a remote zero-click MMS attack that worked on the latest Samsung flagship devices. New posts will be published as they are completed and will be linked here when complete.*

- [this post]
- MMS Exploit Part 2: Effective Fuzzing of the Qmage Codec
- MMS Exploit Part 3: Constructing the Memory Corruption Primitives
- MMS Exploit Part 4: MMS Primer, Completing the ASLR Oracle
- MMS Exploit Part 5: Defeating Android ASLR, Getting RCE

### Introduction

In January 2020, I reported a large volume of crashes in a custom Samsung codec called "Qmage", present in all Samsung phones since late 2014 (Android version 4.4.4+). This codec is written in C/C++ code, and is baked deeply into the Skia graphics library, which is in turn the underlying engine used for nearly all graphics operations in the Android OS. In other words, in addition to the well-known formats such as JPEG and PNG, modern Samsung phones also natively support a proprietary Qmage format, typically denoted by the .qmg file extension. It is automatically enabled for all apps which display images, making it a prime target for remote attacks, as sending pictures is the core functionality of some of the most popular mobile apps.

# In Wild Critical Buffer Overflow Vulnerability in Solaris Can Allow Remote Takeover — CVE-2020-14871

**November 04, 2020 | by Jacob Thompson**

EXPLOIT VULNERABILITY FLARE

FireEye Mandiant has been investigating compromised Oracle Solaris machines in customer environments. During our investigations, we discovered an exploit tool on a customer's system and analyzed it to see how it was attacking their Solaris environment. The FLARE team's Offensive Task Force analyzed the exploit to determine how it worked, reproduced the vulnerability on different versions of Solaris, and then reported it to Oracle. In this blog post we present a description of the vulnerability, offer a quick way to test whether a system may be vulnerable, and suggest mitigations and workarounds. Mandiant experts from the FLARE team will provide more information on this vulnerability and how it was used by UNC1945 during a Nov. 12 webinar. Register today and start preparing questions, b em from the audience at the end of the session.

https://www.fireeye.com/blog/threat-research/2020/11/critical-buffer-overflow-vulnerability-in-solaris-can-allow-remote-takeover.html

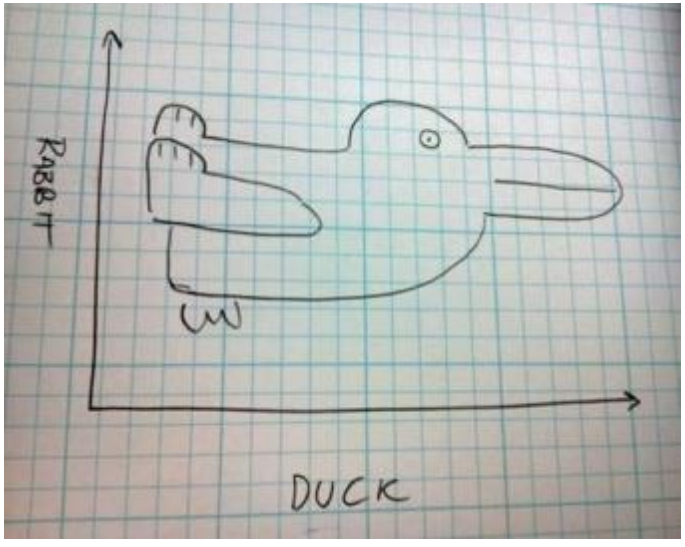
## Vulnerability Discovery

The security vulnerability occurs in the Pluggable Authentication Modules (PAM) library. PAM enables a Solaris application to authenticate users while allowing the system administrator to configure authentication parameters (e.g., password complexity and expiration) in one location that is consistently enforced by all applications.

The actual vulnerability is a classic stack-based buffer overflow located in the PAM `parse_user_name` function. An abbreviated version of this function is shown in Figure 1.

# How does a buffer over*fl*ow let you take over a machine?

- Your program manipulates data
- Data manipulates your program



# What we need to know

- How C arrays work
- How memory is laid out
- How the stack and function calls work
- How to turn an array overflow into an exploit

# How do C arrays work?

- What does a[idx] get compiled to?
  - *((a)+(idx))
- What does the spec say?
  - 6.5.2.1 Array subscripting in ISO/IEC 9899:2017
  - There is no concept of bounds!

# Linux process memory layout

- **Stack:** Stores local variables.
- **Heap**: Dynamic memory for programmer to allocate.
- **Data segment**: Stores global variables, separated into initialized and uninitialized.
- **Text segment**: Stores the code being executed.