

# CSIT 5740 Introduction to Software Security

Note set 1

Dr. Alex LAM



DEPARTMENT OF  
**COMPUTER SCIENCE & ENGINEERING**

The set of note is adopted and converted from a software security course at the Purdue University by Prof. Antonio Bianchi

# *The course instructor*

---

- Dr. Alex Lam



- Personal Website: <http://www.cse.ust.hk/~lamngok>
- Email: [lamngok@cse.ust.hk](mailto:lamngok@cse.ust.hk)
- Office: Room 3548 (Lift 27/28)

# Our Teaching assistant (TA)

- We are lucky to have a really nice and experienced TA for this class:



- CHEN Yanzuo
- [ychenjo@cse.ust.hk](mailto:ychenjo@cse.ust.hk)

- When you have questions, instead of sending us emails, you could raise them at Piazza, and we will answer them there. In that way, Piazza could become a collection of “knowledge base” of the course. (Piazza registration is at <https://piazza.com/class/lz9trfnytis4ao> , more on this in 30 mins)
- Piazza would allow you to remain anonymous while posting questions. Canvas would not.

# Course Focus

- Common security pitfalls in software (with illustration examples\*)
  - Memory safety and exploitations (x86 assembly using GDB\*)
  - Bug exploitation (x86 assembly using GDB\*)
  - Web safety and exploitations (understand how your web browser interacts with remote web servers\*)
  - Bug finding, etc
- 
- Focus on compiled software and memory corruption
  - Focus on Linux (as it is open-source), the general software security concepts apply to other operating systems.

\* You all need to install **Kali Linux**, we will show it briefly at the end of today's class

# Modules

---

- Topics (subject to time and other constraints)
  - Linux and its security mechanisms
  - Memory safety and exploitations
  - Code protection mechanisms
  - Program analysis
  - Fuzzing
  - Web safety and exploitations

# Prerequisites

- Basic programming skills (knowing C/C++, **C/C++ pointers**, Python, etc)
- Basic operating system concepts (we will be sticking with Linux for many of the topics, as it is open source and provides the best resource to study security. We will be using the industrial standard Kali Linux to illustrate many of the concepts )
- The ability to pick up new programming languages reasonably quick
- Basic logical reasoning skills
- Knowledge of assembly language (x86/x64) will be helpful, but this is not compulsory

# ***This course does not do the below***

- We will not teach any programming language
- We will not help you or your company develop secured software system (we won't have the time and won't be able to take the risk of possible legal consequences), etc
- Explain every detail → we will try our best to explain as much as we can, but we probably won't be able to explain every single detail. In case you are still not clear about some of the concepts after our explanations, feel free to do the following:
  - Google on related materials
  - Do programming experiments yourself
  - Ask us through the piazza

# Logistics

- Material
  - Slides on Canvas
    - we will try to upload them before the class, but
      - It may not always be possible
      - I may update them after the class, if I do that I will let you know and ask you to download the latest version
  - Extra files also on Canvas
- Piazza
  - We will mainly use Piazza for Q&A, try not to send emails as they could be overlooked because of the volume of emails we receive daily
  - You are supposed to check Canvas and Piazza frequently



# Logistics

- Grading
  - **Two Individual Homework assignments:** (each accounts for 7.5% of the grade, for a total of 15% of the grade)
  - Written exams
    - **Midterm:** 25% of the grade
      - in class, on 30<sup>th</sup> October, likely to be 1.5-hour, open book, open note
    - **Final:** 60% of the grade
      - Open book, open note
      - The University will arrange it, we do not know the date and time perhaps before early November

# Homework Submission

---

- Double check what you submitted (download it and test it after submitting)
  - We will grade the homework zero if:
    - It does not follow exactly the specified format
      - Including file name capitalization
    - It does not work at all
- The TA is responsible for marking the homework and exams, so questions about grading should be first directed to the TA

# Cheating

- Homework must be done **individually**
- **Sharing code** (even 1 line) is cheating
- **Sharing solutions** is cheating
- **Asking someone else** to do your homework is cheating
- **Do not copy from StackOverflow or other online resources** (unless you mention it explicitly in the homework assignments)
- **Do not use generative AI (e.g., ChatGPT), it won't be helpful anyway** 😊
- **Do not copy even a single line from your classmates**
  - Not even a “fragment” of a line
- **You will automatically fail the course if you are caught cheating**



# Cheating

< Undergraduate Student Guide

## Academic Integrity

Academic integrity and honesty are key values at HKUST.

---

28 JUL 2021 •  ALL

---

All students are required to uphold the University's [Academic Honor Code](#).

Some kinds of academic misconduct are obvious but others are not so clear. It is your responsibility to learn where the University draws the line between academic honesty and cheating.

### Academic Misconduct

Students are required to maintain the highest standards of academic integrity. The University has zero tolerance for academic misconduct.

- Please take a careful at the HKUST web for Academic Integrity  
<https://registry.hkust.edu.hk/resource-library/academic-integrity>

# *More about this course*

---

- This is probably not “the usual” class
  - Assignments could be very different from what you have done for other classes
- We probably will not explain in details how to use mentioned tools, techniques, ...
  - You may need to read online documentation

# *More about this course*

---

- You may need to write code in:
  - C, Python, bash shell script, ...
  - we probably will not have enough time to teach them, and you may need to learn yourself

# *We will teach simple software “hacking”*

- To understand the security threats, we will teach a little bit about some basic hacking knowledge, but it is for illustrating the concepts - so that you know the approaches that would be employed to break your software
- The hacking knowledge, in general, is not enough for you to launch attacks against well-maintained systems, but sometimes it could be employed to attack older systems
- You are not allowed to use the knowledge to attack any infrastructure (our infrastructure included, of course!). Attacking an infrastructure is considered a crime in Hong Kong!
- **Everything has to be done ethically (see the next slide).**

# *You have to do hacking ethically*

- Here is what you should do when you are done with the Cybersecurity concentration and are asked to do security testing:
  - **Always seek authorization and approval** before performing any assessment on the computer system
  - **Let the system administrator know** clearly the **scope of assessment and the plan** for identifying potential vulnerabilities in the system
  - **Report every single security breach and system vulnerability found** to the system administrator
  - **Keep all the learned system vulnerabilities confidential.** Moreover, you should **never utilize the learned knowledge** on system vulnerabilities **in a way detrimental to the owner of the system**
  - You should **erase all the hack traces after assessing the system vulnerabilities**, so that the attackers will not be able to identify the system loopholes through the hack traces



# Why you would like to study for this course?

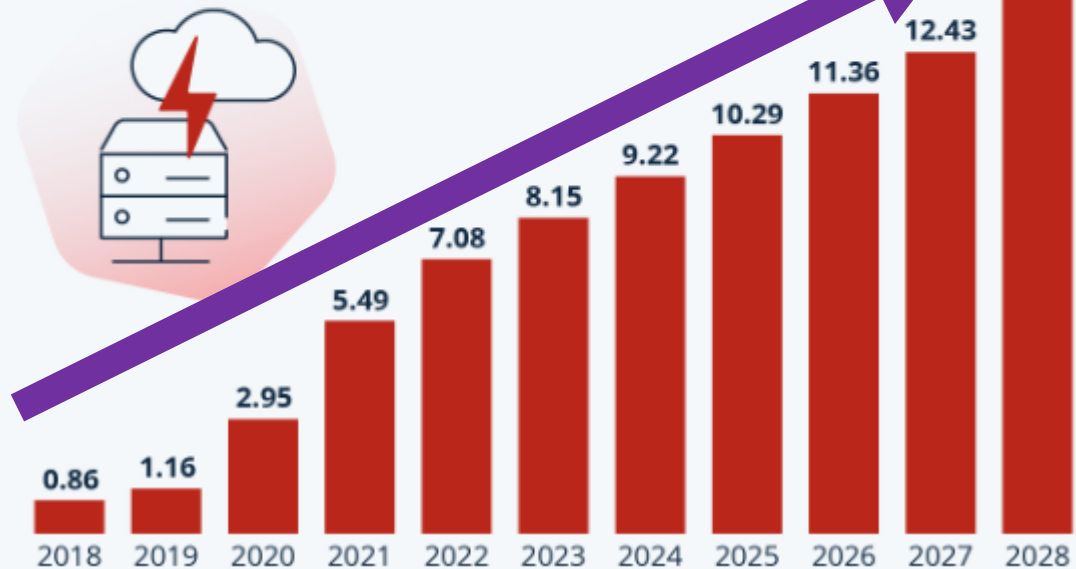
CYBERCRIME

## Cybercrime Expected To Skyrocket in Coming Years

by [Anna Fleck](#), Feb 22, 2024

### Cybercrime Expected To Skyrocket

Estimated annual cost of cybercrime worldwide  
(in trillion U.S. dollars)



\* How much is 1 trillion? It is **1000 billions**

# *Why you would like to study for this course?*

- Want to better protect our assets!



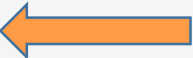
# Things you probably should know, a simple trace of a C program

- Memory content tracing: what does this print?

```
#include <stdio.h>
#include <malloc.h>
#include <string.h>
```

```
void funct(char* m){
//    m[4] = '\x00';
    memcpy(&(m[0]), "Chewing gum", 4);
    memcpy(&m[4], " is delicious!", 15);
    m[3]++;
}
```

```
int main(){
    char a[] = "CSIT5740";
    long* buffer = (long*)malloc(20);
    memcpy(buffer, a, 9);
    funct((char*)buffer);
    printf("%s\n", (char*)buffer);
}
```

 What is in `a[]`?

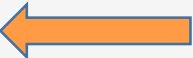
# Things you probably should know, a simple trace of a C program

- Memory content tracing: what does this print?

```
#include <stdio.h>
#include <malloc.h>
#include <string.h>
```

```
void funct(char* m){
//    m[4] = '\x00';
    memcpy(&(m[0]), "Chewing gum", 4);
    memcpy(&m[4], " is delicious!", 15);
    m[3]++;
}
```

```
int main(){
    char a[] = "CSIT5740";
    long* buffer = (long*)malloc(20);
    memcpy(buffer, a, 9);
    funct((char*)buffer);
    printf("%s\n", (char*)buffer);
}
```

 What is in `a[]`?

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]
C	S	I	T	5	7	4	0	\x00


# Things you probably should know, a simple trace of a C program

- Memory content tracing: what does this print?

```
#include <stdio.h>
#include <malloc.h>
#include <string.h>
```

```
void funct(char* m){
//    m[4] = '\x00';
    memcpy(&(m[0]), "Chewing gum", 4);
    memcpy(&m[4], " is delicious!", 15);
    m[3]++;
}
```

```
int main(){
    char a[] = "CSIT5740";
    long* buffer = (long*)malloc(20);
    memcpy(buffer, a, 9);
    funct((char*)buffer);
    printf("%s\n", (char*)buffer);
}
```

 What does `buffer` points to?

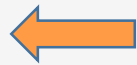
# Things you probably should know, a simple trace of a C program

- Memory content tracing: what does this print?

```
#include <stdio.h>
#include <malloc.h>
#include <string.h>
```

```
void funct(char* m){
//    m[4] = '\x00';
    memcpy(&(m[0]), "Chewing gum", 4);
    memcpy(&m[4], " is delicious!", 15);
    m[3]++;
}
```

```
int main(){
    char a[] = "CSIT5740";
    long* buffer = (long*)malloc(20);
    memcpy(buffer, a, 9);
    funct((char*)buffer);
    printf("%s\n", (char*)buffer);
}
```



What does `buffer` points to?

`buffer` points to a storage space 20 bytes in size

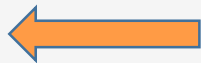
# Things you probably should know, a simple trace of a C program

- Memory content tracing: what does this print?

```
#include <stdio.h>
#include <malloc.h>
#include <string.h>
```

```
void funct(char* m){
//    m[4] = '\x00';
    memcpy(&(m[0]), "Chewing gum", 4);
    memcpy(&m[4], " is delicious!", 15);
    m[3]++;
}
```

```
int main(){
    char a[] = "CSIT5740";
    long* buffer = (long*)malloc(20);
    memcpy(buffer, a, 9);
    funct((char*)buffer);
    printf("%s\n", (char*)buffer);
}
```



What memcpy does?

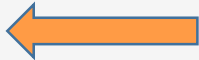
# Things you probably should know, a simple trace of a C program

- Memory content tracing: what does this print?

```
#include <stdio.h>
#include <malloc.h>
#include <string.h>
```

```
void funct(char* m){
//   m[4] = '\x00';
    memcpy(&(m[0]), "Chewing gum", 4);
    memcpy(&m[4], " is delicious!", 15);
    m[3]++;
}
```

```
int main(){
    char a[] = "CSIT5740";
    long* buffer = (long*)malloc(20);
    memcpy(buffer, a, 9);
    funct((char*)buffer);
    printf("%s\n", (char*)buffer);
}
```

 What memcpy does?

memcpy copies the first 9 bytes ("CSIT5740") from a[] to the storage space the buffer pointer points to




# Things you probably should know, a simple trace of a C program

- Memory content tracing: what does this print?

```
#include <stdio.h>
#include <malloc.h>
#include <string.h>
```

```
void funct(char* m){
//    m[4] = '\x00';
    memcpy(&(m[0]), "Chewing gum", 4);
    memcpy(&m[4], " is delicious!", 15);
    m[3]++;
}
```

```
int main(){
    char a[] = "CSIT5740";
    long* buffer = (long*)malloc(20);
    memcpy(buffer, a, 9);
    funct((char*)buffer);
    printf("%s\n", (char*)buffer);
}
```

 What memcpy does?

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]
C	S	I	T	5	7	4	0	\x00

buffer[0]	buffer[1]	buffer[2]	buffer[3]	buffer[4]	buffer[5]	buffer[6]	buffer[7]	buffer[8]	...
C	S	I	T	5	7	4	0	\x00	

# Things you probably should know, a simple trace of a C program

- Memory content tracing: what does this print?

```
#include <stdio.h>
#include <malloc.h>
#include <string.h>
```

```
void funct(char* m){
//    m[4] = '\x00';
    memcpy(&(m[0]), "Chewing gum", 4);
    memcpy(&m[4], " is delicious!", 15);
    m[3]++;
}
```

```
int main(){
    char a[] = "CSIT5740";
    long* buffer = (long*)malloc(20);
    memcpy(buffer, a, 9);
    funct((char*)buffer);
    printf("%s\n", (char*)buffer);
}
```

← What memcpy does?

This end of string char `\x00` is very important, and must be copied. Why?

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]
C	S	I	T	5	7	4	0	\x00

buffer[0]	buffer[1]	buffer[2]	buffer[3]	buffer[4]	buffer[5]	buffer[6]	buffer[7]	buffer[8]	...
C	S	I	T	5	7	4	0	\x00	

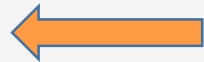
# Things you probably should know, a simple trace of a C program

- Memory content tracing: what does this print?

```
#include <stdio.h>
#include <malloc.h>
#include <string.h>
```

```
void funct(char* m){
//    m[4] = '\x00';
    memcpy(&(m[0]), "Chewing gum", 4);
    memcpy(&m[4], " is delicious!", 15);
    m[3]++;
}
```

```
int main(){
    char a[] = "CSIT5740";
    long* buffer = (long*)malloc(20);
    memcpy(buffer, a, 9);
    funct((char*)buffer);
    printf("%s\n", (char*)buffer);
}
```



What the `function()` does?

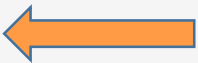
# Things you probably should know, a simple trace of a C program

- Memory content tracing: what does this print?

```
#include <stdio.h>
#include <malloc.h>
#include <string.h>
```

```
void funct(char* m){
//   m[4] = '\x00';      The commented line puts the end of string character to buffer[4]. It is commented, so has no effect
    memcpy(&(m[0]), "Chewing gum", 4);
    memcpy(&m[4], " is delicious!", 15);
    m[3]++;
}
```

```
int main(){
    char a[] = "CSIT5740";
    long* buffer = (long*)malloc(20);
    memcpy(buffer, a, 9);
    funct((char*)buffer);
    printf("%s\n", (char*)buffer);
}
```

 What the `function()` does?

# Things you probably should know, a simple trace of a C program

- Memory content tracing: what does this print?

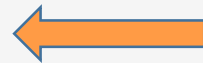
```
#include <stdio.h>
#include <malloc.h>
#include <string.h>
```

```
void funct(char* m){
//    m[4] = '\x00';
    memcpy(&(m[0]), "Chewing gum", 4);
    memcpy(&m[4], " is delicious!", 15);
    m[3]++;
}
```

1<sup>st</sup> memcpy copies 4 bytes from "Chewing gum" to buffer[0], so the string in buffer becomes "Chew5740"

buffer[0]	buffer[1]	buffer[2]	buffer[3]	buffer[4]	buffer[5]	buffer[6]	buffer[7]	buffer[8]	...
C	h	e	w	5	7	4	0	\00	

```
int main(){
    char a[] = "CSIT5740";
    long* buffer = (long*)malloc(20);
    memcpy(buffer, a, 9);
    funct((char*)buffer);
    printf("%s\n", (char*)buffer);
}
```



What the `function()` does?

# Things you probably should know, a simple trace of a C program

- Memory content tracing: what does this print?

```
#include <stdio.h>
#include <malloc.h>
#include <string.h>
```

```
void funct(char* m){
```

```
// m[4] = '\x00';
```

```
memcpy(&(m[0]), "Chewing gum", 4);
```

```
memcpy(&m[4], "is delicious!", 15);
```

```
m[3]++;
```

```
}
```

```
int main(){
```

```
char a[] = "CSIT5740";
```

```
long* buffer = (long*)malloc(20);
```

```
memcpy(buffer, a, 9);
```

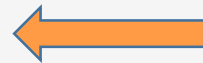
```
funct((char*)buffer);
```

```
printf("%s\n", (char*)buffer);
```

```
}
```

2<sup>nd</sup> memcpy copies 15 bytes from "is delicious!" to buffer[4], so the string in buffer becomes "Chew is delicious!"

buffer[0]	buffer[1]	buffer[2]	buffer[3]	buffer[4]	buffer[5]	buffer[6]	buffer[7]	buffer[8]	...
C	h	e	w		i	s		d	...



What the `function()` does?

# Things you probably should know, a simple trace of a C program

- Memory content tracing: what does this print?

```
#include <stdio.h>
#include <malloc.h>
#include <string.h>
```

```
void funct(char* m){
//    m[4] = '\x00';
    memcpy(&(m[0]), "Chewing gum", 4);
    memcpy(&m[4], " is delicious!", 15);
    m[3]++;    buffer[3] is added by 1, so 'w' becomes 'x' and the string becomes
}              "Chex is delicious!"
```

```
int main(){
    char a[] = "CSIT5740";
    long* buffer = (long*)malloc(20);
    memcpy(buffer, a, 9);
    funct((char*)buffer);    ← What the function() does?
    printf("%s\n", (char*)buffer);
}
```

buffer[0]	buffer[1]	buffer[2]	buffer[3]	buffer[4]	buffer[5]	buffer[6]	buffer[7]	buffer[8]	...
C	h	e	x		i	s		d	...

# Things you probably should know, a simple trace of a C program

- Memory content tracing: what does this print?

```
#include <stdio.h>
#include <malloc.h>
#include <string.h>
```

```
void funct(char* m){
//  m[4] = '\x00';
    memcpy(&(m[0]), "Chewing gum", 4);
    memcpy(&m[4], " is delicious!", 15);
    m[3]++;
}
```

```
int main(){
    char a[] = "CSIT5740";
    long* buffer = (long*)malloc(20);
    memcpy(buffer, a, 9);
    funct((char*)buffer);
    printf("%s\n", (char*)buffer); ← "Chex is delicious!" is printed
}
```



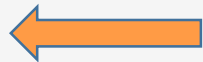


# Things you probably should know, a simple trace of a C program

- Memory content tracing: what does this print?

```
#include <stdio.h>
#include <malloc.h>
#include <string.h>
```

```
void funct(char* m){
    m[4] = '\x00';
    memcpy(&(m[0]), "Chewing gum", 4);
    memcpy(&m[4], " is delicious!", 15);
    m[3]++;
}
```



What will happen if we uncomment this line?  
Try it if you don't know the answer

```
int main(){
    char a[] = "CSIT5740";
    long* buffer = (long*)malloc(20);
    memcpy(buffer, a, 9);
    funct((char*)buffer);
    printf("%s\n", (char*)buffer);
}
```



# Homework 0 - installation 1

- Not graded, but **essential** to be able to participate in the class (the future homework assignments will depend on it, revision for exams also needs it)
- Register to Piazza <https://piazza.com/class/lz9trfnytis4ao>
- Get a Kali Linux on the Virtualbox (hypervisor) running on your PC (we are assuming Windows platform, for Mac machines, please refer to slide 37, **please be warned that you may run into problem in running our x64 programs for homework 1/2**).
  1. Download the Kali image here: <https://cdimage.kali.org/kali-2024.2/kali-linux-2024.2-virtualbox-amd64.7z> , this image is for Intel x64. Unzip the file to get the image
    - a) You may need to download and install the 7-zip utility to unzip the Kali image (<https://www.7-zip.org/> )
    - b) When you unzip the Kali image, make sure you put it in a folder that you have the full access right (for example, for windows it could be c:\users\yourUserName\VM\_image, where “yourUserName” is the user name you use to log on windows, and “VM\_image” is a directory made by you)

# Homework 0 - installation 2

- Get a Kali Linux on the Virtualbox (hypervisor) running on your PC
  2. Download the Virtualbox here: <https://www.virtualbox.org/wiki/Downloads>
  3. Install Virtualbox, choose all the default options when you install
  4. The Virtualbox installer may require you to install the “Microsoft Visual C++ 2019 Redistributable Package”, you may find this package here <https://learn.microsoft.com/en-us/cpp/windows/latest-supported-vc-redist?view=msvc-170>
    - Please pick the 64-bit (i.e. x64) version of the package
  5. Once Virtualbox is installed, you can go to the folder in step 1a), double-click the “Virtual Machine Definition” file, Kali Linux will start

# Homework 0 - installation 3

---

- Get a Kali Linux on the Virtualbox (hypervisor) running on your PC
  7. Please do it right after today's class, because downloading the file could take days on some of the networks...
  8. After successfully installing and running the Kali Linux in the Virtualbox, you may want to change the passwords of the default user "Kali" and also set up the password for the super user "root", see slide 38 for the details.
  9. It may be possible to use other OSes for some of the homework assignments, but it is highly discouraged, because Kali Linux is considered the industrial standard for security experts

# Homework 0 - installation Apple Silicon

- If you are doing this on Apple Silicon (M1/M2/M3) Macs, you need to refer to the following for instructions to get the Kali Linux working (**please be warned that you may run into problem in running our x64 programs for homework 1/2**):

<https://www.kali.org/docs/virtualization/install-vmware-silicon-host/>

# Homework 0: setting up Kali users

- The default user of the Kali Linux is “kali” and the password is also “kali”.
- We call this pair of username and password the “credential for login”
- You will be using this “kali” user to do a lot of things, please remember to change its password to a different one. You can do that by issuing “**passwd**” at the prompt

```
(kali@kali)-[~]  
$ passwd  
Changing password for kali.  
Current password:  
New password:  
Retype new password:  
passwd: password updated successfully
```

- The most privileged user (aka the super user) in a Linux system is the “root”. In the Kali image, the “root” is not enabled by default. To enable the root, you need set the password for it using “**sudo passwd**”, notice the difference between this password change and the previous one. You will know the reason in the next few classes.

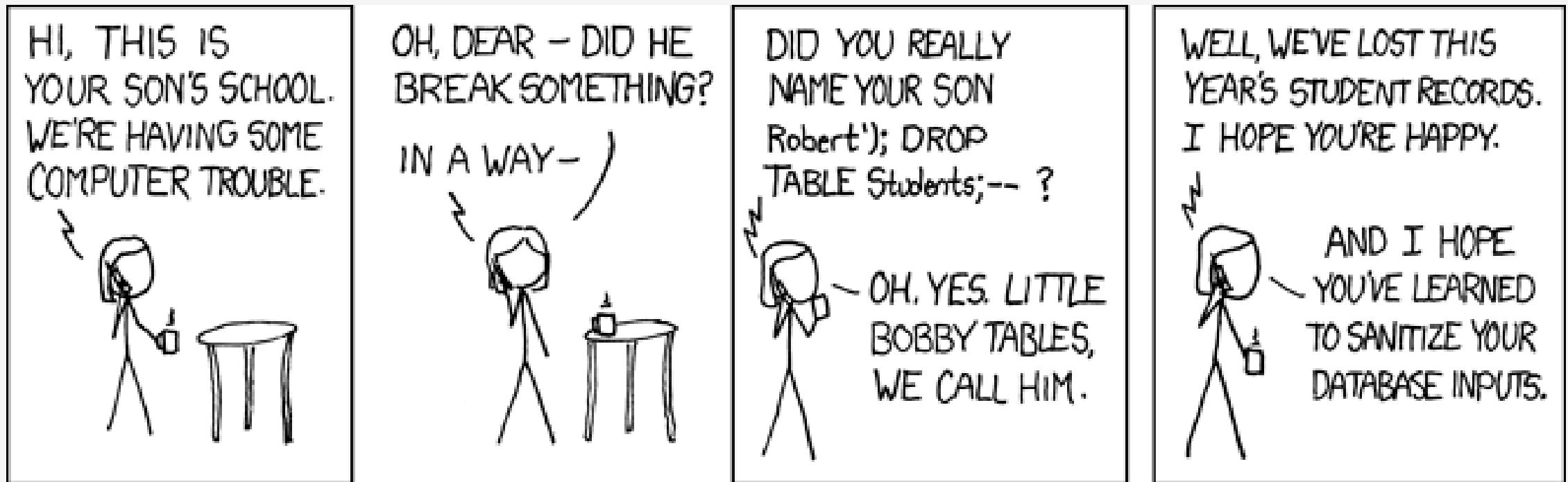
```
(kali@kali)-[~]  
$ sudo passwd  
New password:  
Retype new password:  
passwd: password updated successfully
```

# Cap'n Crunch

- In 1960s, people found that the whistle that came with the Cap'n Crunch allows free long-distance calls
- Why?
  - The whistle produced a 2600 Hz sound
  - AT&T used a 2600 Hz signal to allow long-distance calls
  - Ambiguity between “transmitted data” (*the sound*) and “control information” (*allow long-distance calls*)



# What is it in the comic?





# SQL Injections

If this is the SQL query passed to the backend server:

```
" (SELECT * FROM Students WHERE username = ' " + userid + "' AND  
password = ' " + password + " ' ) "
```

If userid = "Robert'); DROP TABLE Students ; -- " and  
password = "abc", then the full SQL query will be:

```
" (SELECT * FROM Students WHERE username = 'Robert') ; DROP TABLE  
Students ; -- ' AND password = 'abc' ) "
```

# SQL Injections

- Let's take a closer look by dividing the query into 3 rows:
  1. `(SELECT * FROM Students WHERE username = 'Robert' ) ;`
  2. `DROP TABLE Students ;`
  3. `-- ' AND password = 'abc' ) "`
- Line 1 is valid SQL code that will legitimately insert data about a student named Robert.
- Line 2 is valid injected SQL code that will delete the whole Students data table from the database.
- Line 3 is a valid code comment ('--' denotes a comment), which will cause the rest of the line to be ignored by the SQL server.

# The 'NULL' license plate

- Security researcher Joseph Tartaro set NULL as his license plate value
- “any time a traffic cop forgot to fill in the license plate number on a citation, the fine automatically got sent to Joseph Tartaro”
- <https://www.wired.com/story/null-license-plate-landed-one-hacker-ticket-hell/>



# Memory Corruption (buffer overflow)

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
```

```
int function(char* arg){
    char buf[10];
    strcpy(buf, arg);
    printf("buf is: %s\n", buf);
    return 0;
}

int main(int argc, char **argv) {
    function(argv[1]);
    return 0;
}
```

Calling the program with argv[1] equal to gives us arbitrary code execution in the context of this program:

abcdefghijkl\x9f\xff\xff\xf7

The red part above could be the start address of a potentially malicious set of instructions! More details on this in 2-3 weeks.

You will be able to launch a similar buffer overflow attack too!

# *Lessons learned*

---

- Always sanitize user inputs
- Mixing data and control (e.g., code) is bad

# Threat Model

---

- Asking “is this secure?” is a “bad” question.
- What can our attackers do? What we consider as possible?
  - Remote Attacker, Local Attacker, Attacker in Physical Proximity, Your Friend, Evil Maid, Hacker, Malware, Foreign Government, ...
- What do we consider as “bad”?

# *The CIA triad and more*

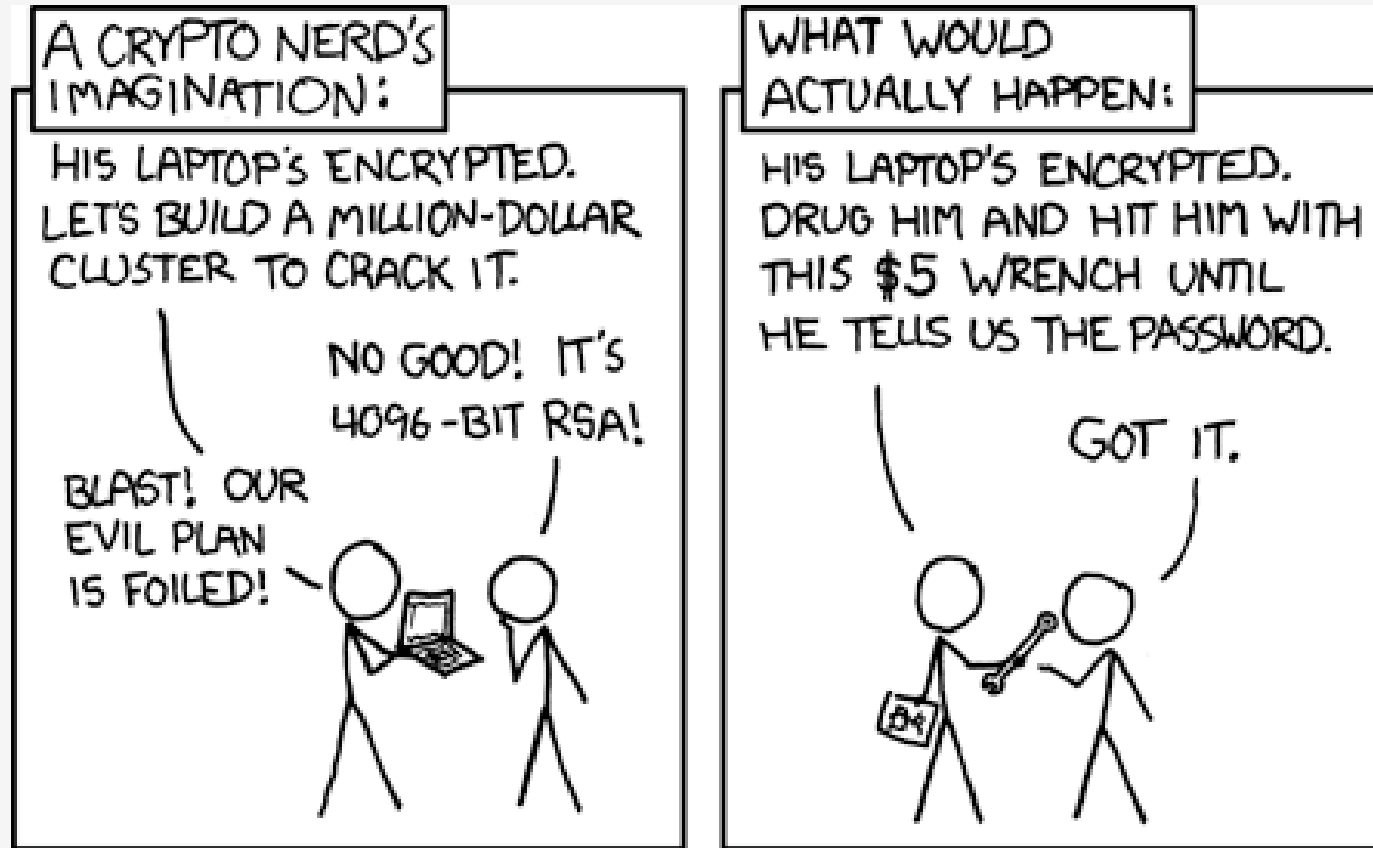
- CIA triad
  - Confidentiality
    - It refers to the need to protect information from unauthorized accesses, so that the data is kept safe and secret.
  - Integrity
    - It refers to the need to make sure data is trustworthy, complete and has not been modified intentionally or unintentionally by unauthorized users
  - Availability
    - It refers to the need to ensure the service/data is available when you need them
- Other properties
  - Authenticity
    - We know the origin of some data
  - Privacy

# *Examples, are these security issues?*

- On a smartphone
  - An app can know where the user is located
  - The user did not setup any PIN to unlock a device
- When visiting a webpage
  - The webpage can know where you are located
  - The webpage can know your IP address
  - The webpage can execute Javascript code on your computer
  - The webpage can execute arbitrary code on your computer
- On a server, shared among multiple users
  - A user can delete other users' files
  - A user can see other users' account name
  - A user can execute arbitrary code



# xkcd: Security



*“Actual actual reality: nobody cares about his secrets.  
(Also, I would be hard-pressed to find that wrench for \$5.)”*

xkcd – <https://xkcd.com/538/>

# ***The Linux System***

# Users

---

- Linux (Unix) is a multiuser OS
  - `/etc/passwd`
  - Every user is identified by a number (user ID) and a name
  - lists all the users with their corresponding user ID (uid)
- Main security
  - Enables protection from other users
  - Enables protection of system code from users

# Users

---

- System code runs as the super-user, also called **root** user
  - The root user has user ID equal to zero
- The system code is responsible (among other things) to ensure its own protection and the protections between users
  - System code is the kernel code plus code in user-space processes, run by the root user

# Users

- Every process has an owner (the “user ID” of a process shows the owner)
  - The owner is one of the users in the system
  - Normally, the process owner is the user that started its execution
  - More details on this later...
- In general, what a process can do is determined by who its owner is and what its owner can do (again, more details later...)
  - You can see owners using `htop`

# Users

---

- Typical setup:
  - In a desktop/laptop: one main user
  - In a server: one user for every person allowed to access it
  - Every user has a “home” folder
  - In addition, special users running specific processes and root
  - Android: typically one user per app

# Users

---

- Every user belongs to one or multiple groups
- `/etc/group` contains all the available groups and the owners belonging to it
- The command `groups` lists all the groups which the current user belongs to
- Groups are useful if we want to allow multiple users to perform some operation, for instance:
  - The program `docker` can be used only by the root user or any user in the “docker” group
  - If we want to allow a user to use `docker`, we can just add it to the “docker” groups

# Users

- Some commands (some require root privileges):
  - `id`:  
show the current user and its groups
  - `adduser`:  
add a new user to the system
  - `useradd`:  
add a new user and also create its home folder, ...
  - `usermod -a -G group user`:  
add the user “user” to the group “group”
  - `sudo command`:  
run command as root
  - `sudo -u user command`:  
run a command as “user”
  - `sudo [-u user] -i`:  
open a shell running commands as root [or “user”]



# *Users: Sudoers*

---

- Normal users have an associated password
- Typically asked for “login”
  - System’s startup
  - Remote login (ssh)
- Users belonging to the group sudo can switch to another user using the sudo command
- By default, it requires inserting the user’s password

# Permission Checking

- In general, to perform system-level operations a process invokes kernel code, by using system calls
- The called kernel code checks
  - Which process called it
  - The owner of the process
  - Is the owner of the process authorized to perform the requested operation?
    - Some operations may be restricted only to:
    - Users belonging to specific groups
    - The root user
    - ...

# *File Permissions*

- Every file has an associated owner user and an associated owner group
  - Use: `ls -al`  
to list files and their owners
  - `chown`: change owner user/group
- Every file has permissions associated to it
  - `chmod`: change file permissions

# File Permissions

File Type      # of Hard Links      File size

Permissions      Owners      Last Modify Time

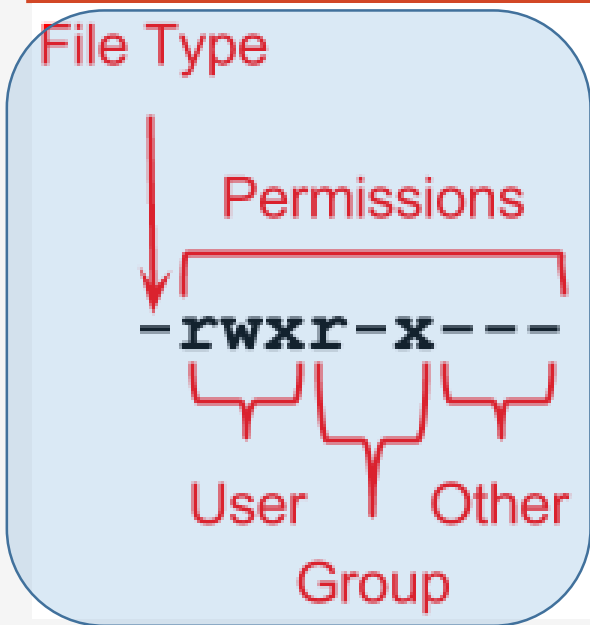
`-rwxr-x---`      `1`      `walbert support`      `0`      `Oct 31 11:06`      `test`

User      Group      User      Group      File name

Group

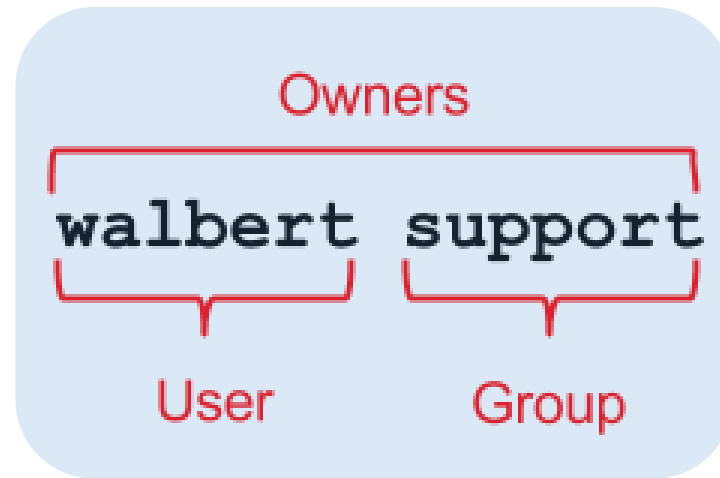
Owner	Group	Other
<code>rw</code>	<code>r-x</code>	<code>r-x</code>
$4+2+1$	$4+0+1$	$4+0+1$
<b>7</b>	<b>5</b>	<b>5</b>

# File Permissions



Owner	Group	Other
<code>rwx</code>	<code>r-x</code>	<code>r-x</code>
$4+2+1$	$4+0+1$	$4+0+1$
<b>7</b>	<b>5</b>	<b>5</b>

# *File Permissions*



# *File Permissions*

Last Modify Time  
Oct 31 11:06 test  
File name

# *File Permissions*

# of Hard Links



**1**

File size



**0**



# File Permissions

File Type      # of Hard Links      File size

Permissions      Owners      Last Modify Time

`-rwxr-x---`      `1`      `walbert support`      `0`      `Oct 31 11:06`      `test`

User      Group      User      Group      File name

Group

Owner	Group	Other
<code>rw</code>	<code>r-x</code>	<code>r-x</code>
$4+2+1$	$4+0+1$	$4+0+1$
<b>7</b>	<b>5</b>	<b>5</b>

# Directory Permissions

---

## Read permission

- : The directory's contents cannot be shown.
- r**: The directory's contents can be shown (listed).

## Write permission

- : The directory's contents (the list of files) cannot be modified.
- w**: The directory's contents can be modified (create new files or folders, rename or delete existing files, ...). It requires the execute permission, otherwise it has no effect

# Directory Permissions

---

## Execute permission

–: The directory cannot be accessed (cannot cd inside the directory)

**x**: The directory can be accessed using cd

(this is the only permission bit that in practice can be considered to be "inherited" from the ancestor directories, in fact if *any* folder in the path does not have the x bit set, the final file or folder cannot be accessed either)

→ You can create directories from which a user can read files, but you cannot list them → remove the “r” permission, but keep the “x” permission

# Directory Permissions

## Sticky Bit

- files in the directory may only be deleted or renamed by
  - root
  - the directory owner
  - the file owner.

→ *Typically used for /tmp*

Every user can add files to /tmp or list the content of /tmp, but a user cannot delete/rename other users' files in /tmp

# Links

## Hard links

- The file system interprets two paths as “the same file”
- When you delete a file that was hard linked to the data, the data is still on the harddrive as long as there are still hard-linked files existing. The data is deleted only when all the hard-linked files are deleted

## Soft links

- More used, a link is a special file “pointing to” another file

```
ln -s <target> <link_name>
```

```
ls -la
```

- When the source file is deleted, the soft link is still there, though it will be pointing to an invalid file.