CS 344: Operating Systems I
01.11: Preliminaries

Mon/Wed 12:00 – 1:50 PM (LINC #2000)

Sanghyun Hong
sanghyun.hong@oregonstate.edu

SAIL
Secure AI Systems Lab

Recap

- Introduction to OS
  - What is an OS?
  - What are the functionalities of OS?
  - What are the tips for studying OS?
  - What are the course topics?

Topics overview

- Part I: How OS runs programs?
  - Processes
  - Threads
  - Scheduling basics
- Part II: How OS loads/stores data?
  - Files
  - I/Os
  - Filesystem internals
- Part III: How OS support comm.?
  - Signals and PIPEs
  - Sockets
  - Networking
- Part IV: How OS manages programs running on limited resources *safely*?
  - Synchronization
  - Rust

Topics for today

- Preliminaries
  - Connect to OS I server
  - Shell + script
  - Version control and editors (vim)
  - C Reviews
  - Debugging (GDB)

Connect to OS1 server

- Tools to connect: any SSH client
  - Terminal (Mac OSX), Terminal in VS Code (Mac OS / Windows)

- How to access the OS 1 server?
  - On-campus: ssh <ONID>@os1.engr.oregonstate.edu
  - Off-campus:
    - First, ssh to those: ssh <ONID>@access/flip.engr.oregonstate.edu
    - Second, ssh to the OS 1 server: ssh <ONID>@os1.engr.oregonstate.edu
  - Note: do *not* run any program on the access/flip servers

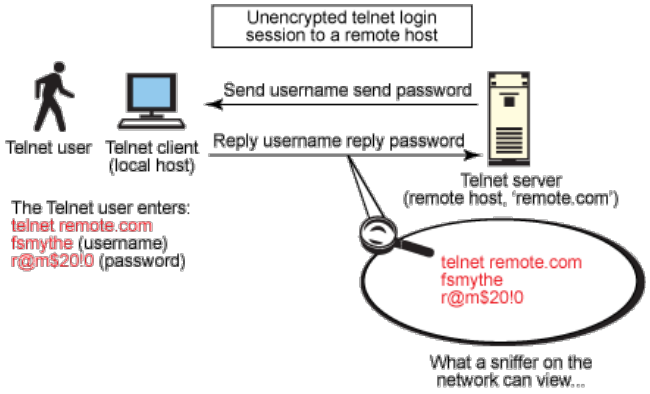
Connect to OS1 server – cont'd

- SSH without password
  - Authentication using an SSH key
  - **Pro:** don't need to type password in every SSH log-in

- How to?
  - Generate a *private* and *public* key pair on your PC/laptop
    - Command: ssh-keygen –t ed25519 -C "<ONID>@oregonstate.edu"
    - Output: you will have <keyname> and <keyname>.pub under a specified folder
  - Copy the public key to the OS 1 server
    - Open <keyname>.pub and copy the content
    - Paste it into *authorized_keys* file in OS 1 server's <your home>/.ssh folder
    - Update the permission of authorized_keys file to *700*: chmod 0700 authorized_key
  - Try SSH command again
    - ssh <ONID>@access/flip.engr.oregonstate.edu (It won't ask the password again)

Connect to OS1 server – cont'd

- How does it work and why do we do?
  - Password login is not secure against man-in-the-middle attackers
  - Potential solutions:
    - Encrypt login information
    - Encrypt all the communications (like German's Enigma)



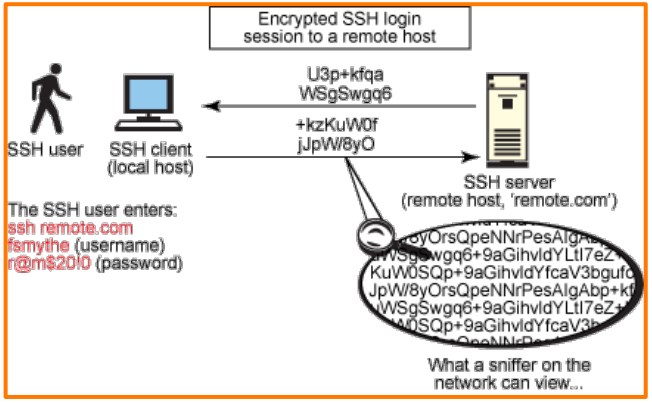
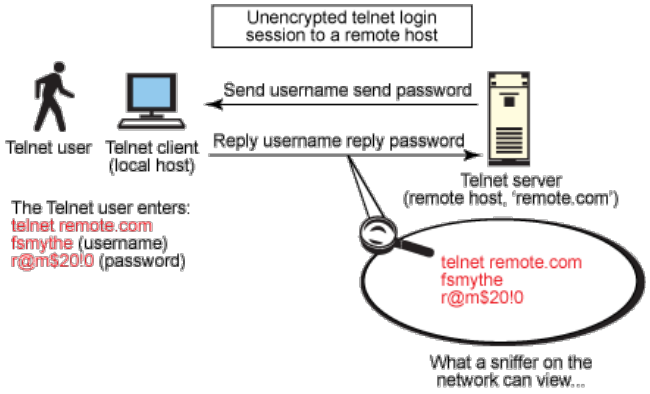


Image source: https://developer.ibm.com/articles/au-sshsecurity/

Connect to OS1 server – cont'd

- How does it work and why do we do?
  - Password login is not secure against man-in-the-middle attackers
  - Potential solutions:
    - ~~Encrypt login information~~
    - Encrypt all the communications (like German's Enigma)
    - Encrypt, but not with a shared mechanism (not with a shared key)



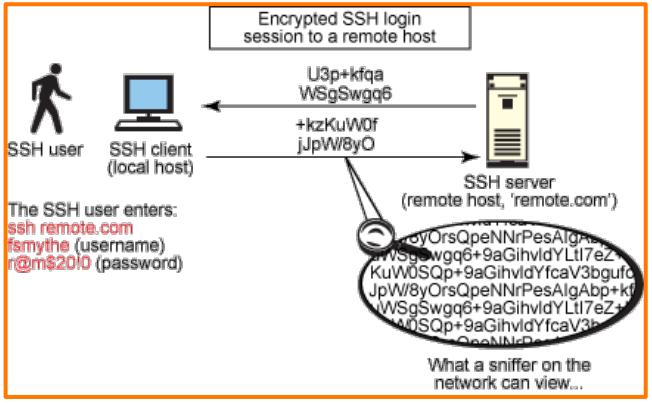


Image source: https://developer.ibm.com/articles/au-sshsecurity/

Connect to OS1 server – cont'd

- How does it work and why do we do?
  - Password login is not secure against man-in-the-middle attackers
  - Potential solutions:
    - **Asymmetric (public-key-based) encryption**
    - An authentication protocol with the asymmetric encryption

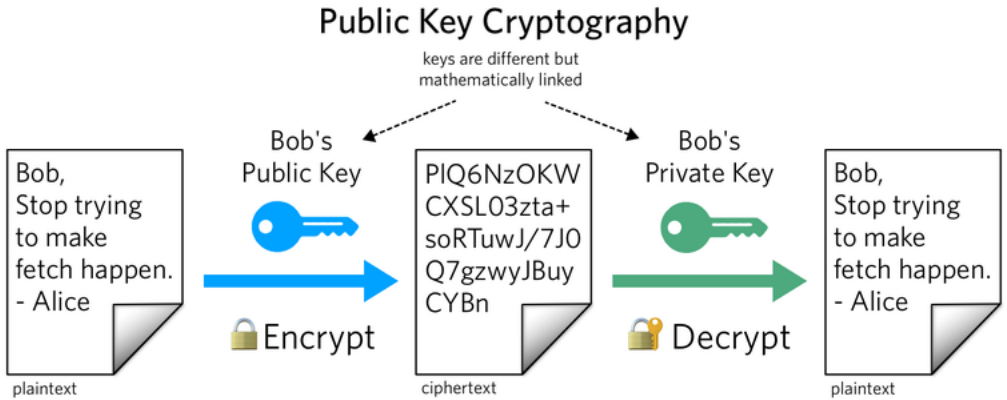


Image source: https://www.twilio.com/blog/what-is-public-key-cryptography

Connect to OS1 server – cont'd

- How does it work and why do we do?
  - Password login is not secure against man-in-the-middle attackers
  - Potential solutions:
    - Asymmetric (public-key-based) encryption
    - **An authentication protocol with the asymmetric encryption**
      - 1) You put your public key to the server (manually)
      - 2) You ask the connection
      - 3) The server encrypts a challenge with your public key and send it to you
      - 4) You decrypts the package with your private key
      - 6) You solve the challenge and encrypt the answer with your private key
      - 7) The server decrypts the package with your public key and verifies the answer
      - 8) Both establish the *safe* connection and communicate with the encryption

Image source: https://www.twilio.com/blog/what-is-public-key-cryptography

Topics for today

- Preliminaries
  - Connect to OS I server
  - Shell + script
  - Version control and editors (vim)
  - C Reviews
  - Debugging (GDB)

Shell

- What is shell:
  - **Formal:** A program witch exposes OS's services to users or to other programs
  - **Informal:** That you will see after the SSH log-in

- What are the types?
  - ***Bourne shell (bash)**, Korn shell (ksh), Z shell (zsh), C shell (csh)…

- What are the (basic) features?
  - Print a message(s)
  - Launch a program
  - Create, rename, or remove files and directories
  - See what programs running on OS

Basic bash shell commands

- Basic commands
  - Print a message(s): echo
  - Launch a program: ./<program name> <arguments>
  - Create, rename, or remove files (and directories)
    - Create a dir: mkdir (-p) <directory>
    - Create a file: touch <filename>
    - Move a dir/file: mv <file/directory> <destination>
    - Copy a dir/file: cp -rf <file/directory>
    - Remove files/dirs: rm (-r <empty directory> / -f <file> / -rf <all files and subdirectories>)
    - Others:
      - List files or directories: ls (-al / -alh / -t) <file or directories>
      - Go to a certain directory: cd <directory path>
      - Print out a file content: cat <filename>
      - Print out a (log) file content being updated: tail (-f) <filename>

Basic bash shell commands – cont'd

- Basic commands
  - Others
    - See what programs are running on OS: ps (-ef)
    - See who runs what programs on OS: ps -ef | grep <username>
    - See the OS version and distribution: uname (-r / -a)
    - See the CPU/mem.: cat /proc/cpuinfo (or cat /proc/meminfo)
    - See the directories where you are: pwd (absolute path)
    - …

Basic bash shell commands – cont'd

- Data wrangling
  - You can run multiple commands at once: <command 1>; <command 2>; <command 3>…
  - You can combine multiple commands at once:
    - Sequential executions: <command 1> && <command 2>
    - Store execution results of a command to a file: <command 1> > <output file>
    - Run a program background: <command 1> &
    - Example) run in background and store the results to a file: <command 1> > <output file> &
    - Example) see the output file in real-time: tail -f <output file>
  - More commands (with previous commands)
    - Search for files or directories: find <directory> -name <token like *sanghyun*>
    - Search for a string in files or directories: grep -nr <token like sanghyun> <directory>

Tips: Be Creative with Your Combinations!

Basic bash shell setup

- Customization
  - Use the configuration file: ~/.bashrc or ~/.profile (~/ indicates your home dir.)
  - Add commands you want to run when you log-in: echo “Hell-o-world”
  - Create an alias of your command(s): alias os1=“ssh <ONID>@server-addr”
  - … (more)

Topics for today

- Preliminaries
  - Connect to OS I server
  - Shell + script
  - Version control and editor (vim)
  - C Reviews
  - Debugging (GDB)

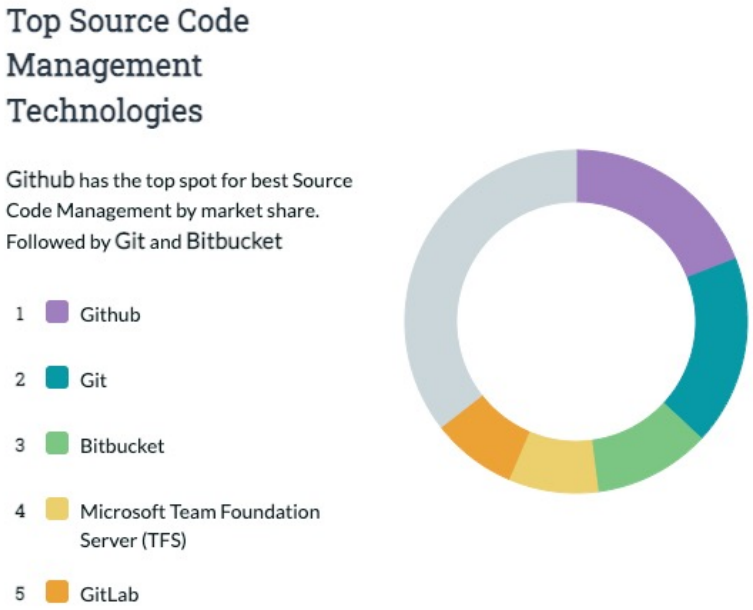
Version control

- Problems we may face
  - What if we do accidentally rm -rf <project dir>?
  - What if our computer suddenly not working?
  - What if we remove a piece of code that was correct?

- Consequences
  - Give up
  - Re-write the code from scratch

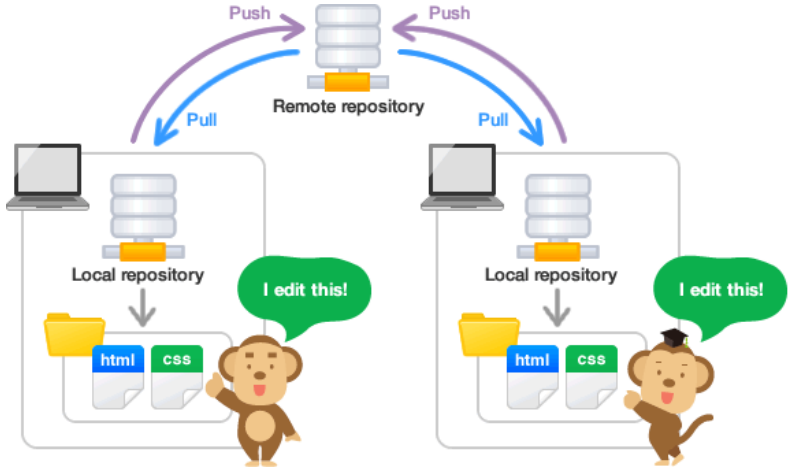
Version control

- Solution: use version control tools
  - **Definition:** the practice of tracking and managing changes to source code
  - **Available tools:**
    - Github
    - Gitlab
    - Git
    - Bitbucket
    - Microsoft Team Foundation
    - …



Version control

- How can we use?
  - Let's do an exercise with GitHub
  - Create a repository for the homework
  - Create a file and modify its content
  - Save the file and push to the repository
  - Git commands we will use:
    - $ git clone <a remote repository url>
    - $ git add <files or a dir>
    - $ git commit -m "message"
    - $ git push
    - $ git pull



Editors (vim)

- Basic functions
  - Open a file: vim <filename>
  - Two modes – command and edit modes
  - Command modes
    - Store the file: :w / Exit: :q / Store and exit: :wq
    - Insert mode: i / Insert in a newline: o
    - Remove texts: d <up-arrow|down-arrow>, dd, u
    - Undo the edits: u
    - Search texts: /<rexpr>
    - Replace texts: :%s/<old-rexpr>/<new-rexpr>/g
    - Copy and paste texts: yy, d<#lines> + <up/down>, p
  - Edit modes
  - Others
    - Split screens: :sp, ctrl+w+v / Move a cursor between screens: ctrl+w+w

Editors (vim) – cont'd

- Let's write an example program "overflow.c"

Version control + editor (vim)

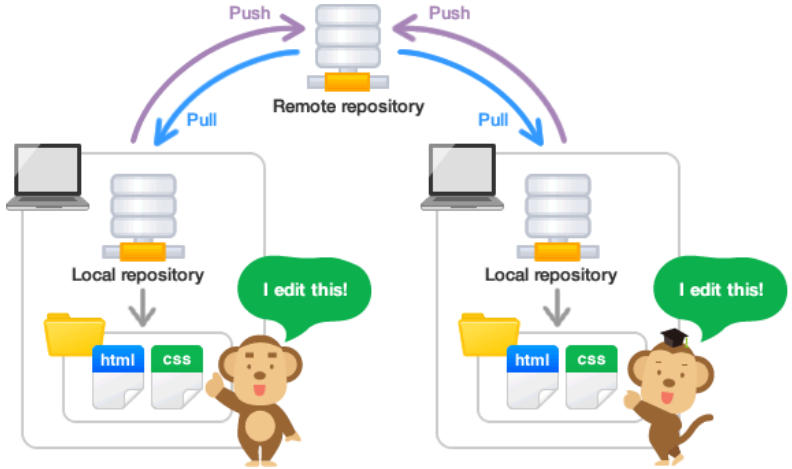
- A sample C program: overflow.c

```c
#include <stdio.h>
#include <errno.h>
#include <stdlib.h>
#include <string.h>

static int buffer_size = 10;

int store_name_and_print(char *buffer, char *sinput)
{
   if (sinput == NULL) {
      printf("Error: the argument string is NULL, abort.\n");
      return -1;
   }

   // copy the string to my buffer
   strcpy(buffer, sinput);

   // check what's in the buffer
   printf("My buffer holds: %s\n", buffer);

   // Here, as a CS student, we will do something with buffer...

   return 0;
}
```

```c
// continue from the left…

int main(int argc, char *argv[])
{
   char *buffer = (char *) malloc(buffer_size);
   int ret = 0;

   // print your name in the argument
   if (argc == 2) {
      ret = store_name_and_print(buffer, argv[1]);
   }
   else if (argc > 2) {
      printf("Error: too many arguments are given - %d, abort.\n", argc);
      return -E2BIG;
   }
   else {
      printf("Error: no name given, abort.\n");
      return -1;
   }

   return ret;
}
```

Version control

- How can we use?
  - Let's do an exercise with GitHub
  - Create a repository for the homework
  - Create a file and modify its content
  - Save the file and push to the repository
  - Git commands we will use:
    - $ git clone <a remote repository url>
    - $ git add <files or a dir>
    - $ git commit -m "message"
    - $ git push
    - $ git pull



Topics for today

- Preliminaries
  - Connect to OS I server
  - Shell + script
  - Version control and editors (vim)
  - C Reviews
  - Debugging (GDB)

C REVIEW

- Revisit the sample C Program: overflow.c

```c
#include <stdio.h>
#include <errno.h>
#include <stdlib.h>
#include <string.h>

static int buffer_size = 10;

int store_name_and_print(char *buffer, char *sinput)
{
   if (sinput == NULL) {
      printf("Error: the argument string is NULL, abort.\n");
      return -1;
   }

   // copy the string to my buffer
   strcpy(buffer, sinput);

   // check what's in the buffer
   printf("My buffer holds: %s\n", buffer);

   // TODO: as a CS student, we will do something with buffer...

   return 0;
}
```

The function takes mem. addrs of the arguments, not the contents (call by reference, not call by values)

Copy the string argument to the buffer we allocated (10 bytes)

```c
// continue from the left...

int main(int argc, char *argv[])
{
   char *buffer = (char *) malloc(buffer_size);
   int ret = 0;

   // print your name in the argument
   if (argc == 2) {
      ret = store_name_and_print(buffer, argv[1]);
   }
   else if (argc > 2) {
      printf("Error: too many arguments are given - %d, abort.\n", argc);
      return -E2BIG;
   }
   else {
      printf("Error: no name given, abort.\n");
      return -1;
   }

   return ret;
}
```

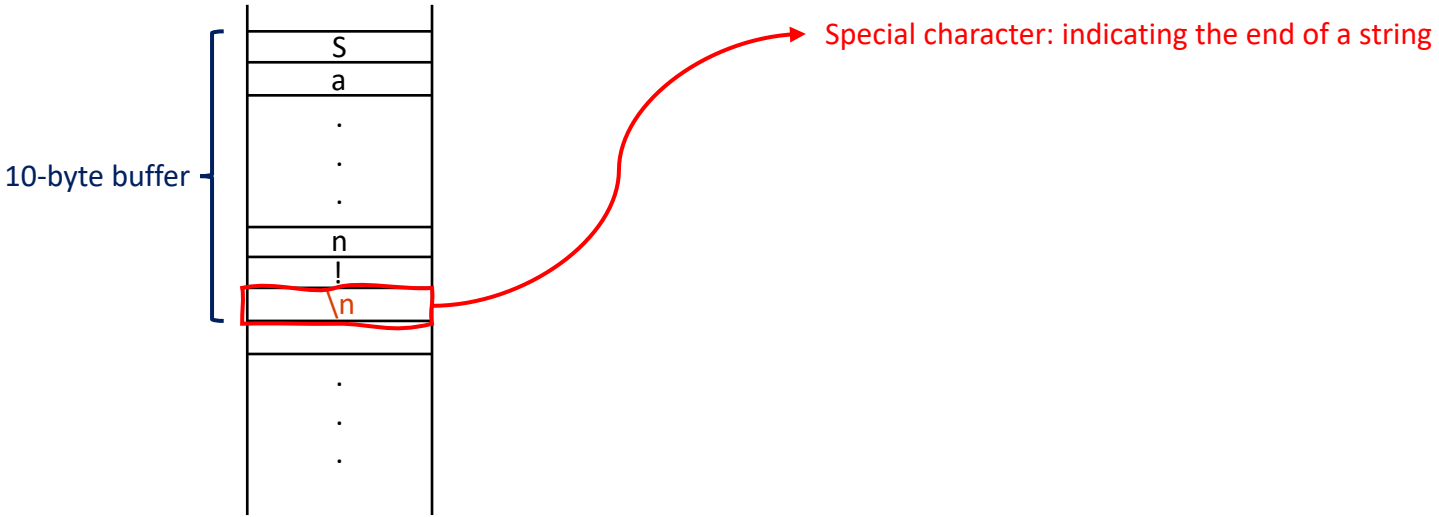
Request OS to allocate mem. the size is equal to 10 bytes (malloc system call)

This program gets a single argument

If it has more than one arg., return error.

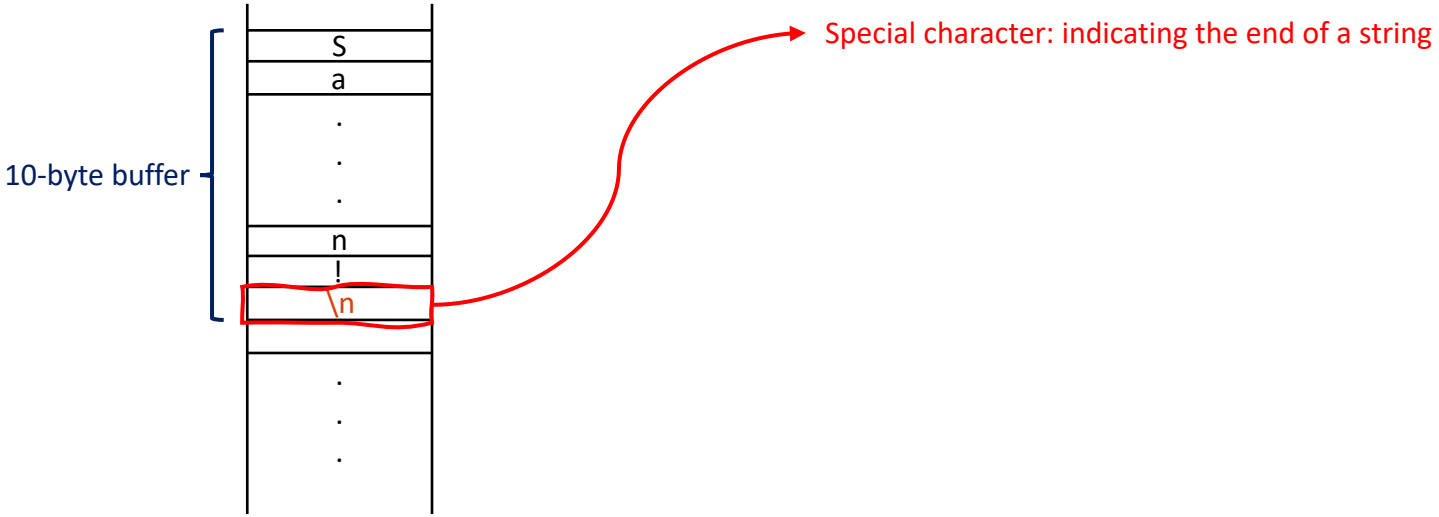
C REVIEW: HOW C STORE STRING IN MEMORY

- Sample C program: overflow.c



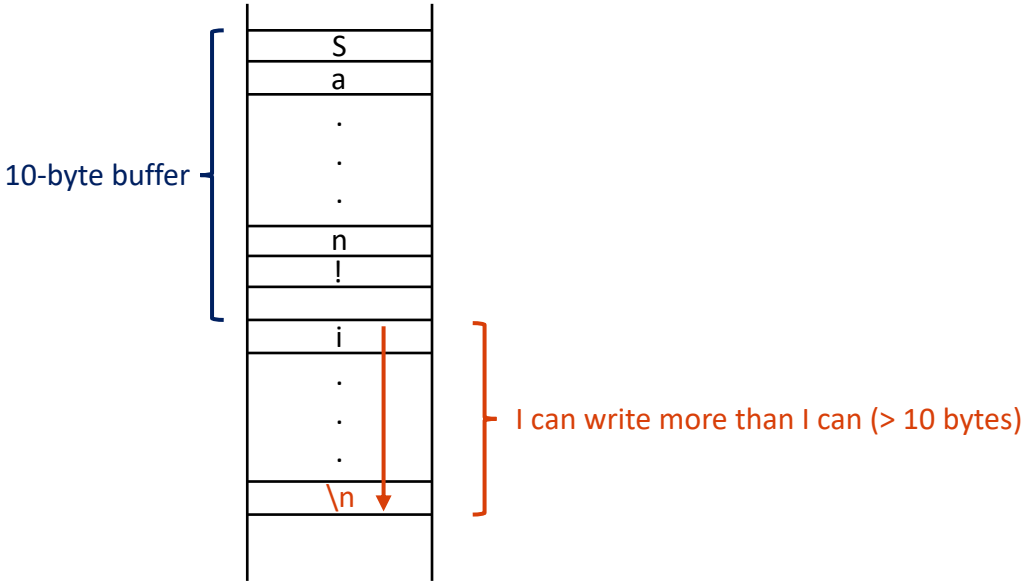
C REVIEW: C DOES NOT MANAGE MEMORY AUTOMATICALLY

- Sample C program: overflow.c
  - len(sinput) < 10: We're okay
  - len(sinput) >= 10: It overwrites some unknown memory locations



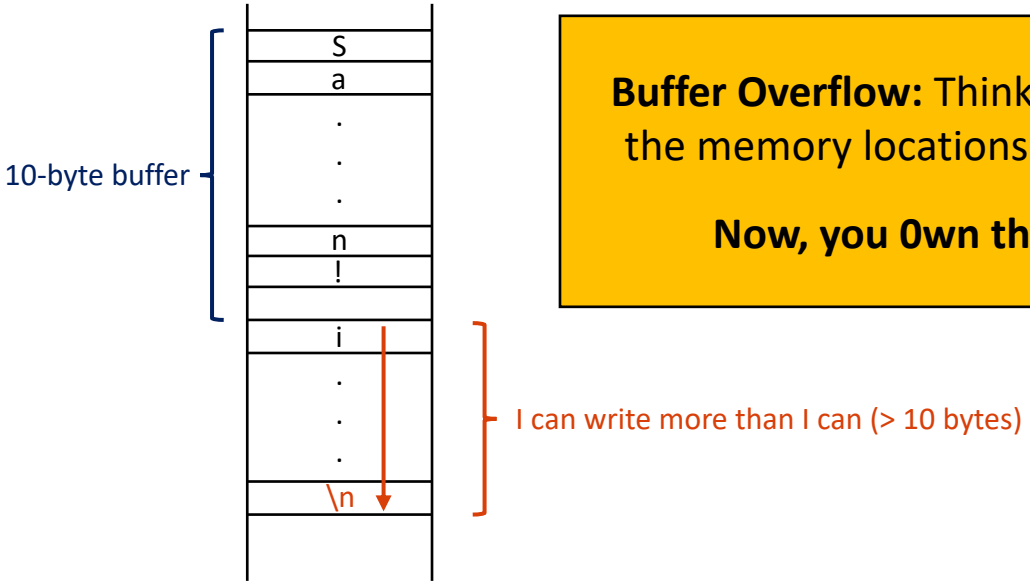
C REVIEW: BUFFER OVERFLOW SECURITY VULNERABILITY

- Sample C program: overflow.c
  - len(sinput) < 10: We're okay
  - len(sinput) >= 10: It overwrites some unknown memory locations



C REVIEW: BUFFER OVERFLOW SECURITY VULNERABILITY

- Sample C program: overflow.c
  - len(sinput) < 10: We're okay
  - len(sinput) >= 10: It overwrites some unknown memory locations



Buffer Overflow: Think about you wrote a program [?!] at the memory locations > 10 bytes, and that's executable!

Now, you 0wn the system (Security Problem)!

C REVIEW: SOLUTION

- Secure programming practices

```c
#include <stdio.h>
#include <errno.h>
#include <stdlib.h>
#include <string.h>

static int buffer_size = 10;

int store_name_and_print(char *buffer, char *sinput)
{
   if (sinput == NULL) {
      printf("Error: the argument string is NULL, abort.\n");
      return -1;
   }

   // copy the string to my buffer
   strncpy(buffer, sinput, buffer_size);

   // check what's in the buffer
   printf("My buffer holds: %s\n", buffer);

   // TODO: as a CS student, we will do something with buffer...

   return 0;
}
```

Copy the string exactly 10 bytes and then truncate the rest of it!

```c
// continue from the left…

int main(int argc, char *argv[])
{
   char *buffer = (char *) malloc(buffer_size);
   int ret = 0;

   // print your name in the argument
   if (argc == 2) {
      ret = store_name_and_print(buffer, argv[1]);
   }
   else if (argc > 2) {
      printf("Error: too many arguments are given - %d, abort.\n", argc);
      return -E2BIG;
   }
   else {
      printf("Error: no name given, abort.\n");
      return -1;
   }

   return ret;
}
```

Topics for today

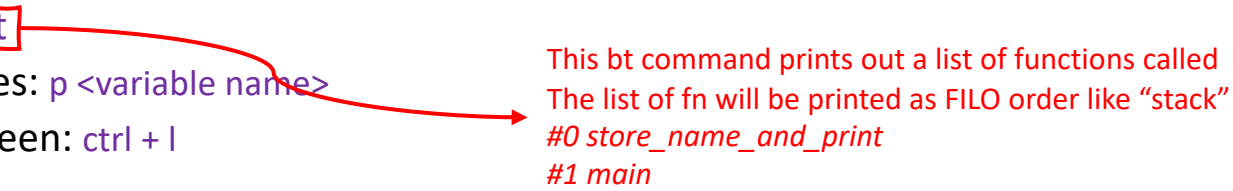
- Preliminaries
  - Connect to OS I server
  - Shell + script
  - Version control and editors (vim)
  - C Reviews
  - Debugging (GDB)

Debugging with GDB

- Types of errors we will face:
  - Static errors, such as syntax errors
    - Relatively easy to fix; GCC provides error messages
  - Runtime errors, such as buffer overflow:
    - Hard to fix
    - Program runs, but does not provide the expected outputs
    - …

Debugging with GDB

- GDB: a tool for debugging C programs in runtime
  - **Pre-requisite:**
    - Compile our program with debug symbols (-g): gcc -g <source file> -o <output file>
    - Run the executable with gdb: gdb ./<output file>
  - **Useful commands:**
    - See lines of codes: list <line #>, list
    - Breakpoints: break <line #>
    - Run: run / step (if you want to execute one line of code at a time)
    - Backtrace: bt
    - Print variables: p <variable name>
    - Clear the screen: ctrl + l
    - … More



Buffer overflow exploit

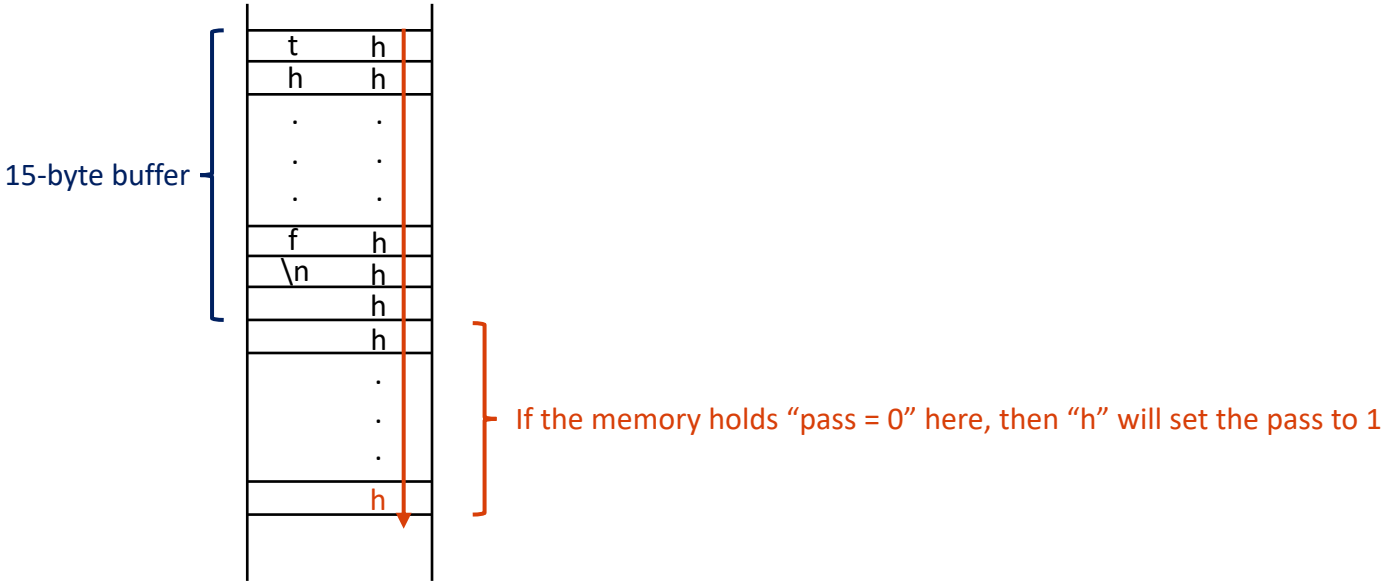
- Sample exploit: subvert.c

```
(base) os1 ~/temp 1012$ ./subvert
Enter your password:
thesecretbuff
Correct password, login!
Now you are allowed to run some private queries
(base) os1 ~/temp 1013$ ./subvert
Enter your password:
hackerishere
[Error]  incorrect password
(base) os1 ~/temp 1013$ ./subvert
Enter your password:
hhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhh
[Error]  incorrect password
Now you are allowed to run some private queries
Segmentation fault
(base) os1 ~/temp 1013$
```

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

int main(int argc, char *argv[]) {
   char buf[15];
   int  pass = 0;

   // read a name from the command line
   printf("Enter your password: \n");
   gets(buf);

  if (strcmp(buf, "thesecretbuff")) {
      printf("[Error]  incorrect password\n");
  }
  else {
      printf("Correct password, login!\n");
      pass = 1;
  }

  // read a ssn from the command line
  if (pass) {
     printf("Now you are allowed to run some private queries\n");
  }

   return 0;
}
```

Buffer overflow exploit

- Sample C program: subvert.c
  - **Normal:** the password I type will be stored into the 15-byte buffer
  - **Attack:** the password “hhhhh…hhh” will go over the 15-byte limit
  - **Real-world cases:** Heartbleed, Shellshock



Debug this code with GDB

- Let's inspect the buffer status with GDB

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

int main(int argc, char *argv[]) {
   char buf[15];
   int   pass = 0;

   // read a name from the command line
   printf("Enter your password: \n");
   gets(buf);

  if (strcmp(buf, "thesecretbuff")) {
      printf("[Error]  incorrect password\n");
  }
  else {
      printf("Correct password, login!\n");
      pass = 1;
  }

  // read a ssn from the command line
  if (pass) {
     printf("Now you are allowed to run some private queries\n");
  }

   return 0;
}
```

Topics for today

- Preliminaries
  - Connect to OS I server
  - Shell + script
  - Version control and editors (vim)
  - C Reviews
  - Debugging (GDB)

Thank You!

Mon/Wed 12:00 – 1:50 PM (LINC #2000)

Sanghyun Hong

sanghyun.hong@oregonstate.edu

Oregon State University

SAIL

Secure AI Systems Lab