Agenda

- The Obligatory Hacking Story
- C
- gcc
- gdb
The Obligatory Hacking Story

- MIT vs CIA
C

- Basics
- Pointers
- Dynamic Memory Allocation
Basics

- Types
  - char, int, short, long
- Arrays, Strings, Structs
- Conditionals
  - if/else, switch/case
- Loops
  - for, while, do-while
- Functions
  - int main()
  - int main(int argc, char **argv);
Pointers Operators

- * operator
  - pointer declaration operator
  - contents-of operator
- & operator
  - returns address of variable
Some Pointers

- `int *ip;`
  - creates pointer to `int`
- `int i = 5;`
- `ip = &i;`
  - `*ip = ?`
- `*ip = 7;`
  - `i = ?`
More Pointers

- `int j = 3;`
- `ip = &j`
  - `*ip = ?`
- `int *ip2`
- `ip2 = ip;`
  - `*ip2 = ?`
- `ip = &i;`
  - `*ip = ?, *ip2 = ?`
- `*ip = *ip2;`
  - `i = ?, j = ?`
Pointers and Arrays

```
int *ip;
int[] a[10];
ip = &a[3];
```

- `a` → returns **ADDRESS** of first element of array `a`
  - `*(ip + 1) = 5;`
  - `*(ip + 3) = 7;`
  - `*(ip - 2) = 1;`
- What do these do?
Pointers and Strings

- Strings are just char[], so pointers work like they do in arrays
- `strcpy` with pointers

```c
char *dp = &dest[0], *sp = &src[0];
while(*sp != '\0'){
    *dp++ = *sp++;
    *dp = '\0';
}
```
NULL Pointers

- Pointers that don't point to anywhere
- Use predefined constant NULL
- Why declare NULL pointers?
Dynamic Memory Allocation

● For when you don't know how much memory you actually need

● Allocates memory on heap

● malloc

● calloc

● realloc

● free
malloc

- Allocates the specified number of bytes
- Returns a pointer to the block of allocated memory

```c
int *ip = malloc(100 * sizeof(int));
```

- What happens when `malloc` can't allocate the memory?
  - Returns `NULL` pointer
calloc/realloc

- **calloc**
  - allocates the specified number of bytes and initializes them to zero
    
    ```c
    int *ip = calloc(100 * sizeof(int));
    ```

- **realloc**
  - increases or decrease the size of the specified block of memory. Reallocates it if needed
    
    ```c
    ip = realloc(ip, 200 * sizeof(int));
    ```
free

• Frees allocated memory when you're done using it.

free(ip);
Dynamic Memory in C++

- **new**
  - `int *ip1 = new int;`
  - `int *ip2 = new int [10];`

- **delete**
  - `delete ip1;`
  - `delete [] ip2;`
Pointer Safety

- What happens when you store a pointer from `malloc` in a local variable when the function returns?
  - Local variable disappears → that pointer disappears
  - Memory allocated doesn't AND you can't access it → MEMORY LEAK
- What happens when you assign a value to a pointer that points to 'nowhere'?
  - It points `somewhere`, just not where you want
- Be careful with pointers.
For more information

- The C Programming Language (K & R)
gcc

- Basics
- Compiling to assembly
- Options to know
- Demos
gcc Basics

- Compile to executable
- `gcc filename.c`
  - `a.out`
  - `./a`
- `gcc -o <program name> filename.c`
  - `./<program name>`
#include <stdio.h>

int main() {
    printf("Hello, world!\n");
    return 0;
}

gcc Compilation Process

- Preprocessing
  - `cpp hello.c > hello.i`

- Compilation
  - `gcc -S hello.i`

- Assembly
  - `as -o hello.o hello.s`

- Linker
  - `ld -o hello hello.o ...libraries...`
gcc and File Types

- **file.c** → C source code which must be preprocessed.
- **file.i** → C source code which should not be preprocessed.
- **file.h** → C header file to be turned into a precompiled header.
- **file.o** → Object code made by assembler.
- **file.s** → Assembler code.
Basic gcc Options

- `-E` → Preprocess but do not compile
- `-S` → Compile but do not assemble
- `-c` → Assemble but do not link
- `-o file` → Place output in `file`
Basic gcc Options

- `v` → Print (on standard error output) the commands executed to run the stages of compilation.
- `--help`
- `-O` → Optimize code
- `-Wall` → Turns on all warnings
- `-g` → Hooks for gdb
Compiling more than one file

```
gcc -o out foo.c bar.c baz.c
```

- OR

```
gcc -c foo.c bar.c baz.c
gcc -o out foo.o bar.o baz.o
```

- Second option → only recompile files with changes. Useful in large projects.
More gcc Options

- C language options
- Warning options
- Debugging options
- Optimization options
- Machine dependent options
- ...and much more all in the gcc manual
Headers and Libraries

- **Default paths**
  
  - `cpp -v` → shows default path for header files
  
  - `gcc -v -o hello hello.c` → shows default paths for libraries

- **-I dir** → search for header files in **dir**

- **-L dir** → search for libraries in **dir**

- **-l library** → search **library** when linking
Environmental Variables

- **PATH** → executables and run-time shared libraries.
- **CPATH** → paths for headers.
- **LIBRARY_PATH** → paths for link libraries.
More **gcc** Options

- C language options
- Warning options
- Debugging options
- Optimization options
- Machine dependent options
- ...and much more all in the **gcc** manual
gcc Failsafes

- `-fno-stack-protector`
  - turns off stack protection
- `-z execstack`
  - allows execution of code in stack
g++

- Use for c++ files
- Many of the same options as gcc
  - -c, -S, -o, -g, -Wall, etc.
  - -fno-stack-protector
- Debugging in C
- Debugging in assembly
- Demos
What **gdb** does

- **gcc -g** → includes debugger information
- General purpose high level language debugger
  - breakpoints
  - examine memory
- Also use to debug C code at the assembly level
A little demo code

```
#include <stdio.h>

void hello()
{
    printf("Hello, world!\n");
}

int main() {
    hello()
    return 0;
}
```
Using **gdb** to debug assembly

- Starting **gdb**
  - `gdb`
  - `gdb <file>`

- Viewing that C program as assembly
  - `disassemble main`

- Run the program
  - `run`
  - `kill`

- Exiting **gdb**
  - `quit`
Setting Breakpoints

- `break FUNCTION`
  - set breakpoint at entry to `FUNCTION`
- `break *ADDRESS`
  - set breakpoint at specified address
- `disable <NUM>`
  - disable breakpoint with that number
- `enable <NUM>`
  - enable breakpoint with that number
Clearing Breakpoints

- **clear FUNCTION**
  - clear breakpoints at entry to FUNCTION
- **delete <NUM>**
  - delete breakpoint with that number
- **delete**
  - delete all breakpoints
Working with Breakpoints

- **stepe**
  - execute one machine code instruction
- **stepe <NUM>**
  - execute NUM machine code instructions
- **step**
  - execute one C statement
- **nexti, nexti <NUM>, next**
  - like step, but with subroutines
Working with Breakpoints

- **until LOCATION**
  - continue running until LOCATION is reached

- **continue**
  - resume execution

- **continue <NUM>**
  - continue, ignoring this break point NUM times

- **finish**
  - run until the current function returns
Watchpoints

• Stop execution whenever value of an expression changes.

• Software or hardware
  – Software: single step through program and test value each step, much slower than normal execution
  – Hardware: no delays
  – GDB uses hardware watchpoints wherever possible
Watchpoint Commands

- **watch expr**
  - set watchpoint, break when *expr* is written to by the program and its value changes

- **rwatch expr**
  - set watchpoint that breaks when *expr* is read by the program

- **awatch expr**
  - set watchpoint that breaks when *expr* is read by or written to by the program
Stack Traces

- Prints current address and stack backtrace
- backtrace
- where
Examining Code

- **print /a $pc**
  - prints program counter
- **print $sp**
  - prints stack pointer
- **disass**
  - displays function around current line
- **disass ADDR**
  - displays function around **ADDR**
- **disass ADDR1 ADDR2**
  - displays function between **ADDR1** and **ADDR2**
Examining Data

- `print $<REG>` → print contents of REG
  - `\x` → print as HEX
  - `\a` → print as address
  - `\d` → print as decimal
  - `\t` → print as binary
  - `\c` → print as character

- `print 0x<HEX>` → print HEX as DEC

- `print \x DEC` → print DEC as HEX
Examining Memory

- **x ADDR**
  - prints contents of ADDR in memory

- **x /NFU ADDR**
  - **N** → number of units to display
  - **F** → display format
  - **U** → b (bytes), h (2 bytes), w (4 bytes)
Autodisplay Information

- **display $<\text{REG}>$**
  - prints contents of $\text{REG}$ every time the program stops

- **display**
  - print the auto displayed items

- **delete display $<\text{NUM}>$**
  - stop displaying item $\text{NUM}$
Useful Information Commands

- `help info`
- `info program` → current status of program
- `info functions` → functions in the program
- `info stack` → backtrace of stack
- `info frame` → current stack frame
- `info scope` → variables local to scope
Useful Information Commands

• `info registers` → contents of registers
• `info variables` → global and static variables
• `info breakpoints` → user defined breakpoints
• `info watchpoints` → user defined watchpoints
• `info address SYMBOL` → look up address of functions
ddd

- Graphical front end for gdb and other command line debuggers

- www.gnu.org/software/ddd/manual/