VIT UNIVERSITY
SCHOOL OF COMPUTER SCIENCE ENGINEERING

UNIT 03

Programming Survival Skills

Dr. Ravi Verma

Output of this Presentation

- Students will be able to define the basic survival problems relevant to the programming fundamentals and basics .
- This presentation will include the content of programming relevant with different types of functioning.

Programming Survival Skills

- Why study programming? Ethical gray hat hackers should study programming and learn as much about the subject as possible in order to find vulnerabilities in programs and get them fixed before unethical hackers take advantage of the
- In this chapter, we cover the following topics:
- • C programming language • Computer memory • Intel processors • Assembly language basics • Debugging with gdb • Python survival skillsm.

C Programming Language

- The C programming language was developed in 1972 by Dennis Ritchie from AT&T Bell Labs. The language was heavily used in Unix and is thereby ubiquitous. In fact, much of the staple networking programs and operating systems are based in C.

Basic C Language Construct

- **main()**
- All C programs contain a main() structure (lowercase) that follows this format:
  - <optional return value type> main (<optional arguments>)
    - {
  - Optional procedural statements or functions calls
    - }

Functions

- Functions are self-contained bundles of algorithms that can be called for execution by main() or other functions. Technically, the main() structure of each C program is also a function; however,

```
<optional return value type> function name (<optional function argument>){
}
```

Variables

- Variables are used in programs to store pieces of information that may change and may be used to dynamically influence the program. Table 10-1 shows some common types of variables. When the program is compiled, most variables are pre-allocated memory of a fixed size according to system-specific definitions of size. Sizes in the table are considered typical; there is no guarantee that you will get those exact sizes. It is left up to the hardware implementation to define this size. However, the function sizeof() is used in C to ensure that the correct sizes are allocated by the compiler.

Variables Types

Variable Type	Use	Typical Size
int	Stores signed integer values such as 314 or –314	4 bytes for 32-bit machines 2 bytes for 16-bit machines
float	Stores signed floating-point numbers such as –3.234	4 bytes
double	Stores large floating-point numbers	8 bytes
char	Stores a single character such as "d"	1 byte

For example: int a = 0; where an integer (normally 4 bytes) is declared in memory with a name of a and an initial value of 0.

printf

- The C language comes with many useful constructs for free (bundled in the libc library). One of the most commonly used constructs is the printf command, generally used to print output to the screen. There are two forms of the printf command:

printf(String);
printf(<format of string><List of variables>)

Printf Cont..

Table 10-2 printf Format Symbols

Format Symbol	Meaning	Example
\n	Carriage return/new line	printf("test\n");
%d	Decimal value	printf("test %d", 123);
%s	String value	printf("test %s", "123");
%x	Hex value	printf("test %x", 0x123);

scanf

- The scanf command complements the printf command and is generally used to get input from the user. The format is as follows:
- scanf(<format of string>,<List of variables>); where the format string can contain format symbols such as those shown for printf in above Table For example, the following code will read an integer from the user and store it into the variable called number: scanf("%d", &number);

for and while Loops

- Loops are used in programming languages to iterate through a series of commands multiple times. The two common types are for and while loops. for loops start counting at a beginning value, test the value for some condition, execute the statement, and increment the value for the next iteration. The format is as follows:

- for(<Beginning value; Test Value; Change Value> ;)

{

<Statements>

; }

Therefore, a for loop like for(i=0; i<10;i++)

if/else

- The if/else construct is used to execute a series of statements if a certain condition is met; otherwise, the optional else block of statements is executed. If there is no else block of statements, the flow of the program will continue after the end of the closing if block bracket (}). The format is as follows:

if(condition) {

<statement should be execute if , if condition is met> }

Else

{statement should be execute if , if condition is met; }

Sample Progra

- You are now ready to review your first program. We will start by showing the program with // comments included, and will follow up with a discussion of the

```
//hello.c                    //customary comment of program name
#include <stdio.h>           //needed for screen printing
main ( ) {                   //required main function
    printf("Hello haxor");   //simply say hello
}                            //exit program
```

Compiling with gcc

Compiling is the process of turning human-readable source code into machine-readable binary files that can be digested by the computer and executed. More specifically, a compiler takes source code and translates it into an intermediate set of files called object code. These files are nearly ready to execute but may contain unresolved references to symbols and functions not included in the original source code file. These symbols and references are resolved through a process called linking, as each object file is linked together into an executable binary file. We have simplified the process for you here.

When programming with C on Unix systems, the compiler of choice is GNU C Compiler (gcc). gcc offers plenty of options when compiling. The most commonly used flags are listed and described in Table

Option	Description
–o <filename>	Saves the compiled binary with this name. The default is to save the output as a.out.
–S	Produces a file containing assembly instructions; saved with a .s extension.
–ggdb	Produces extra debugging information; useful when using GNU debugger (**gdb**).
–c	Compiles without linking; produces object files with a .o extension.

Computer Memory

- In the simplest terms, computer memory is an electronic mechanism that has the ability to store and retrieve data. The smallest amount of data that can be stored is 1 bit, which can be represented by either a 1 or a 0 in memory.
- There are many types of computer memory; we will focus on random access memory (RAM) and Register.

Random Access Memory (RAM)

- In RAM, any piece of stored data can be retrieved at any time—thus the term “random access.” However, RAM is volatile, meaning that when the computer is turned off, all data is lost from RAM.

Segmentation of Memory

- The basic concept is simple. Each process (oversimplified as an executing program) needs to have access to its own areas in memory. After all, you would not want one process overwriting another process's data. So memory is broken down into small segments and handed out to processes as needed.

Programs in Memory

- When processes are loaded into memory, they are basically broken into many small sections. There are six main sections that we are concerned with, and we'll discuss them in the following sections.

Programs in Memory Cont..

- **.text Section**
- The .text section basically corresponds to the .text portion of the binary executable file. It contains the machine instructions to get the task done. This section is marked as read only and will cause a segmentation fault if written to. The size is fixed at runtime when the process is first loaded.

Programs in Memory Cont..

.data Section

The .data section is used to store global initialized variables, such as:

int a = 0;

The size of this section is fixed at runtime.

.bss Section

The below stack section (.bss) is used to store global noninitialized variables, such as: int a;

The size of this section is fixed at runtime.

Programs in Memory Cont..

Heap Section

The heap section is used to store dynamically allocated variables and grows from the lower-addressed memory to the higher-addressed memory. The allocation of memory is controlled through the malloc() and free() functions. For example, to declare an integer and have the memory allocated at runtime, you would use something like:

```
int i = malloc (sizeof (int)); //dynamically allocates an integer, contains
                               //the pre-existing value of that memory
```

Stack Section

The stack section is used to keep track of function calls (recursively) and grows from the higher-addressed memory to the lower-addressed memory on most systems.

Buffers

The term buffer refers to a storage place used to receive and hold data until it can be handled by a process. Since each process can have its own set of buffers, it is critical to keep them straight. This is done by allocating the memory within the .data or .bss section of the process’s memory

Buffers

The term buffer refers to a storage place used to receive and hold data until it can be handled by a process. Since each process can have its own set of buffers, it is critical to keep them straight. This is done by allocating the memory within the .data or .bss section of the process's memory. Remember, once allocated, the buffer is of fixed length. The buffer may hold any predefined type of data; however, for our purpose, we will focus on string-based buffers, used to store user input and variables

Strings in Memory

- Simply put, strings are just continuous arrays of character data in memory. The string is referenced in memory by the address of the first character. The string is terminated or ended by a null character (\0 in C)
- String A[7];
-

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	\0

Pointer

- Pointers are special pieces of memory that hold the address of other pieces of memory. Moving data around inside of memory is a relatively slow operation. It turns out that www.it-ebooks.info Chapter 10: Programming Survival Skills 183 PART III instead of moving data, it is much easier to keep track of the location of items in memory through pointers and simply change the pointers. Pointers are saved in 4 bytes of contiguous memory because memory addresses are 32 bits in length (4 bytes).

Pointer cont..

- **For example,**

As mentioned, strings are referenced by the address of the first character in the array. That address value is called a pointer. So the variable declaration of a string in C is written as follows:

```
char * str; //this is read, give me 4 bytes called str which is a pointer
            //to a Character variable (the first byte of the array).
```

Question ?