

THOMSON



COURSE TECHNOLOGY

Hands-On Ethical Hacking and Network Defense 3rd edition



Chapter 7 Programming for Security Professionals

Last modified 1-11-17

Objectives

- Explain basic programming concepts
- Write a simple C program
- Explain how Web pages are created with HTML
- Describe and create basic Perl programs
- Explain basic object-oriented programming concepts

Introduction to Computer Programming

- Computer programmers must understand the rules of programming languages
 - Programmers deal with syntax errors
- One minor mistake and the program will not run
 - Or worse, it will produce unpredictable results
- Being a good programmer takes time and patience

Computer Programming Fundamentals

- Fundamental concepts
 - Branching, Looping, and Testing (BLT)
 - Documentation
- Function
 - Mini program within a main program that carries out a task

Branching, Looping, and Testing (BLT)

- Branching
 - Takes you from one area of the program to another area
- Looping
 - Act of performing a task over and over
- Testing
 - Verifies some condition and returns true or false

A C Program

```
GNU nano 1.3.12 File: dem01.c

#include <stdio.h>

main()
{
printf("Hello World!");
}
```

- Filename ends in .c
- It's hard to read at first
- A single missing semicolon can ruin a program

Comments

```
GNU nano 1.3.12           File: demo2.c

#include <stdio.h>        /* Standard libraries like printf    */

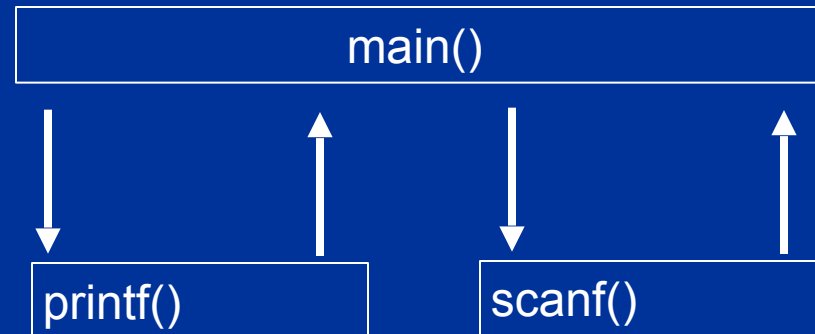
main()                   /* Every C program needs a main()    */
                        /* Processing starts with main()     */
{
printf("Hello World!\n\n"); /* Calls the function printf        */
                        /* \n is a newline character        */
}
```

- Comments make code easier to read

Branching and Testing

```
GNU nano 1.3.12 File: agedemo.c
#include <stdio.h>
main()
{
    int age; /* Declaring a variable */
    printf("Enter your age: "); /* Branching--Calling a function */
    scanf("%d", &age);
    if (age > 0) /* Testing the value of age */
        printf("You are %d years old\n", age);
}
```

Diagram of branches
See links Ch 7b, 7c



Looping

GNU nano 1.3.12

File: loopdemo.c

```
#include <stdio.h>
main()

{
    int i;
    for (i=0; i<11; i++)    /* Looping: repeat ten times */
        printf("I will not hack in class\n");
}
```

Branching, Looping, and Testing (BLT)

- Algorithm
 - Defines steps for performing a task
 - Keep it as simple as possible
- Bug
 - An error that causes unpredictable results
- Pseudocode
 - English-like language used to create the structure of a program

Pseudocode For Shopping

- PurchaseIngredients Function
 - Call GetCar Function
 - Call DriveToStore Function
 - Purchase Bacon, Bread, Tomatoes, Lettuce, and Mayonnaise
- End PurchaseIngredients Function

Documentation

- Documenting your work is essential
 - Add comments to your programs
 - Comments should explain what you are doing
- Many programmers find it time consuming and tedious
- Helps others understand your work

Bugs

- Industry standard
 - 20 to 30 bugs for every 1000 lines of code (link Ch 7f)
 - Textbook claims a much smaller number without a source
- Windows 2000 contains almost 50 million lines
 - And fewer than 60,000 bugs (about 1 per 1000 lines)
 - See link Ch 7e for comments in the leaked Win 2000 source code
- Linux has 0.17 bugs per 1000 lines of code
 - (Link Ch 7f)

Learning the C Language

- Developed by Dennis Ritchie at Bell Laboratories in 1972
- Powerful and concise language
- UNIX was first written in assembly language and later rewritten in C
- C++ is an enhancement of the C language
- C is powerful but dangerous
 - Bugs can crash computers, and it's easy to leave security holes in the code

Compiling C in Ubuntu Linux

- Compiler
 - Converts a text-based program (source code) into executable or binary code
- To prepare Ubuntu Linux for C programming, use this command:

```
sudo apt-get install build-essential
```
- Then you compile a file named "program.c" with this command:

```
gcc program.c -o program
```


Anatomy of a C Program

- The first computer program a C student learns "Hello, World!"

```
GNU nano 1.3.12 File: dem01.c

#include <stdio.h>

main()
{
printf("Hello World!");
}
```

Comments

- Use `/*` and `*/` to comment large portions of text
- Use `//` for one-line comments

```
GNU nano 1.3.12 File: agedemo.c
#include <stdio.h>
main()
{
    int age;                /* Declaring a variable */
    printf("Enter your age: "); /* Branching--Calling a function */
    scanf("%d", &age);
    if (age > 0)            /* Testing the value of age */
        printf("You are %d years old\n", age);
}
```

Include

- #include statement
 - Loads libraries that hold the commands and functions used in your program

```
#include <stdio.h>

main()
{
printf("Hello World!");
}
```

Functions

```
#include <stdio.h>

main()
{
printf("Hello World!");
}
```

- A Function Name is always followed by parentheses ()
- Curly Braces { } shows where a function begins and ends
- main() function
 - Every C program requires a main() function
 - main() is where processing starts

Functions

- Functions can call other functions
 - Parameters or arguments are optional
- `\n` represents a line feed

```
GNU nano 1.3.12           File: agedemo.c

#include <stdio.h>
main()
{
    int age;                /* Declaring a variable          */
    printf("Enter your age: "); /* Branching--Calling a function */
    scanf("%d", &age);
    if (age > 0)            /* Testing the value of age      */
        printf("You are %d years old\n", age);
}
```

Declaring Variables

- A variable represents a numeric or string value
- You must declare a variable before using it

```
GNU nano 1.3.12 File: agedemo.c

#include <stdio.h>
main()
{
    int age;                /* Declaring a variable */
    printf("Enter your age: "); /* Branching--Calling a function */
    scanf("%d", &age);
    if (age > 0)            /* Testing the value of age */
        printf("You are %d years old\n", age);
}
```

Variable Types in C

Table 7-3 Variable types in C

Variable Type	Description
int	Use this variable type for an integer (positive or negative number).
float	This variable type is for a real number that includes a decimal point, such as 1.299999.
double	Use this variable type for a double-precision floating point.
char	This variable type holds the value of a single letter.
string	This variable type holds the value of multiple characters or words.

Mathematical Operators

- The `i++` in the example below adds one to the variable `i`

```
GNU nano 1.3.12                               File: loopdemo.c
#include <stdio.h>
main()
{
    int i;
    for (i=0; i<11; i++)    /* Looping: repeat ten times */
        printf("I will not hack in class\n");
}
```


Mathematical Operators

Table 7-5 Mathematical operators in C

Operator	Description
+ (unary)	Doesn't change the value of the number. Unary operators use a single argument; binary operators use two arguments. Example: +(2).
- (unary)	Returns the negative value of a single number.
++ (unary)	Increments the unary value by 1. For example, if a is equal to 5, the ++a command changes the value to 6.
-- (unary)	Decrements the unary value by 1. For example, if a is equal to 5, the --a command changes the value to 4.
+ (binary)	Addition. For example, a + b.
- (binary)	Subtraction. For example, a - b.
* (binary)	Multiplication. For example, a * b.
/ (binary)	Division. For example, a / b.
% (binary)	Modulus. For example, 10 % 3 is equal to 1 because 10 divided by 3 leaves a remainder of 1.

Logical Operators

- The `i<11` in the example below compares the variable `i` to 11

```
GNU nano 1.3.12                               File: loopdemo.c
#include <stdio.h>
main()
{
    int i;
    for (i=0; i<11; i++)    /* Looping: repeat ten times */
        printf("I will not hack in class\n");
}
```

Logical Operators

Table 7-6 Logical operators in C

Operator	Description
==	Used to compare the equality of two variables. In <code>a == b</code> , for example, the condition is true if variable <code>a</code> is equal to variable <code>b</code> .
!=	Not equal. The exclamation mark negates the equal sign. For example, the statement <code>if a != b</code> is read as "if <code>a</code> is not equal to <code>b</code> ."
>	Greater than.
<	Less than.
>=	Greater than or equal to.
<=	Less than or equal to.
&&	The AND operator; evaluated as true if both sides of the operator are equal. For example, <code>if ((a > 5) && (b > 5)) printf ("Hello, world!");</code> prints only if both <code>a</code> and <code>b</code> are greater than 5.
	The OR operator; evaluated as true if either side of the operator is equal.
!	The NOT operator; the statement <code>a != 5</code> , for example, means that variable <code>a</code> is not equal to the number 5.

Demonstration: Buffer Overflow

```
#include <stdio.h>

main()
{
  char name[10];
  printf("What is your name?");
  scanf("%s",name);
  printf("Hi, %s\n\n",name);
}
```

```
yourname@S214-01u:~$ ./hello2.exe
What is your name?12345678901234567890
Hi, 12345678901234567890

*** stack smashing detected ***: ./hello2.exe terminated
Aborted (core dumped)
```

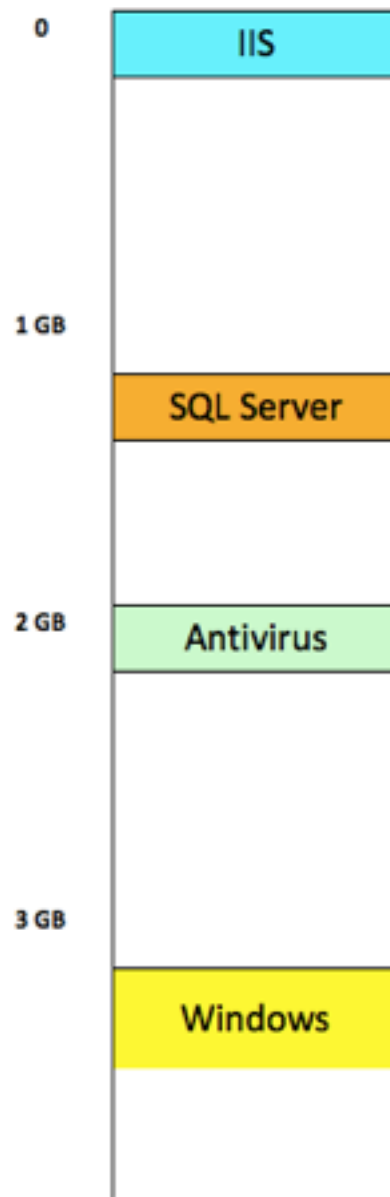

Buffer Overflow Defenses

RAM

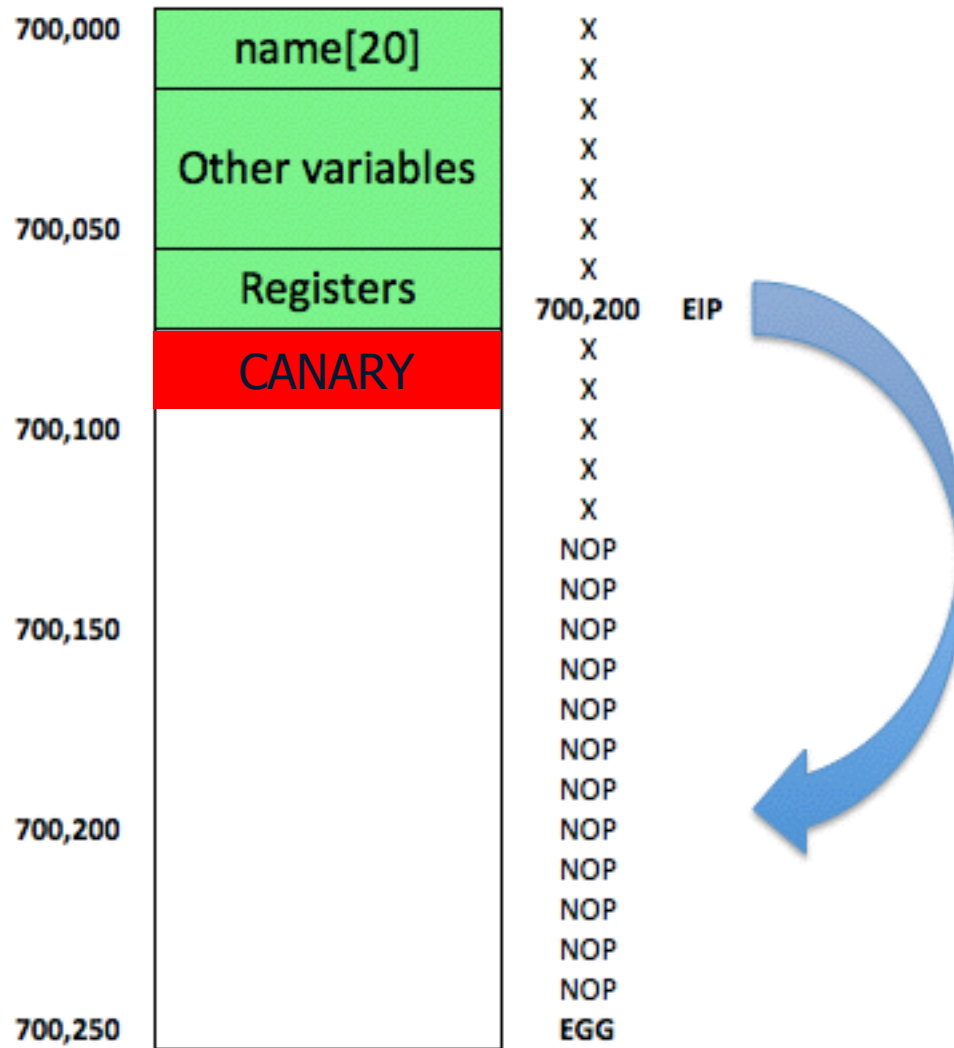
No ASLR



With ASLR



Detecting stack smashing with a canary value



Understanding HTML Basics

- HTML is a language used to create Web pages
- HTML files are text files
- Security professionals often need to examine Web pages
 - Be able to recognize when something looks suspicious

Creating a Web Page Using HTML

- Create HTML Web page in Notepad
- View HTML Web page in a Web browser
- HTML does not use branching, looping, or testing
- HTML is a static formatting language
 - Rather than a programming language
- < and > symbols denote HTML tags
 - Each tag has a matching closing tag
 - <HTML> and </HTML>

Table 7-7 HTML formatting tags

Opening Tag	Closing Tag	Description
<H1>	</H1>	Formats text as a level 1 heading. (Level 1 is the largest font size, and level 6 is the smallest.)
<H2>, <H3>, <H4>, <H5>, and <H6>	</H2>, </H3>, </H4>, </H5>, and </H6>	Formats text as a level 2, 3, 4, 5, or 6 heading.
<P>	</P>	Used to mark the beginning and end of a paragraph.
 	</BR>	Used to insert a carriage return.
		Formats enclosed text in bold.
<I>	</I>	Formats enclosed text in italics.

```
<!-- This HTML web page has many tags -->
<HTML>
<HEAD>
  <TITLE> HTML For Security Testers </TITLE>
</HEAD>
<BODY>

<H2>Security Tester Web Site </H2>

<P><B>There are many good web sites to visit for security testers. For vulnerabilities click</B>
<A HREF="HTTP://www.cve.mitre.org"><FONT COLOR="red">here!</FONT></A>
</P>
<BR><FONT SIZE="-1">Copyright 2005 Security Testers, Incorporated. </FONT></BR>

</BODY>
</HTML>
```

Figure 7-4 HTML source code

Security Tester Web Site

There are many good Web sites to visit for security testers. For vulnerabilities click [here!](#)

Copyright 2005 Security Testers, Incorporated.

Figure 7-5 An HTML Web page

Understanding Practical Extraction and Report Language (Perl)

- PERL
 - Powerful scripting language
 - Used to write scripts and programs for security professionals

Background on Perl

- Developed by Larry Wall in 1987
- Can run on almost any platform
 - *NIX-base OSs already have Perl installed
- Perl syntax is similar to C
- Hackers use Perl to write malware
- Security professionals use Perl to perform repetitive tasks and conduct security monitoring


```
# This is my first Perl script program
# I should always have documentation in my scripts-- no matter
# how easy I think the script is to understand!

print "Hello security testers!\n\n";
```

Figure 7-9 Creating the first.pl Perl script

Understanding the Basics of Perl

- perl -h command
 - Gives you a list of parameters used with perl

```
yourname@S214-01u:~$ perl -h

Usage: perl [switches] [--] [programfile] [arguments]
  -0[octal]      specify record separator (\0, if no arg
  -a            autosplit mode with -n or -p (splits $_
  -C[number/list] enables the listed Unicode features
  -c            check syntax only (runs BEGIN and CHECK
  -d[:debugger] run program under debugger
  -D[number/list] set debugging flags (argument is a bit
  -e program    one line of program (several -e's allow
  -f            don't do $sitelib/sitecustomize.pl at s
```

Table 7-9 Using printf to format output

Formatting Character	Description	Input	Output
%c	Character	<code>printf '%c', "d"</code>	d
%s	String	<code>printf '%s', "This is fun!"</code>	This is fun!
%d	Signed integer in decimal	<code>printf '%+d %d', 1, 1</code>	+1 1
%u	Unsigned integer in decimal	<code>printf '%u', 2</code>	2
%o	Unsigned integer in octal	<code>printf '%o', 8</code>	10
%x	Unsigned integer in hexadecimal	<code>printf '%x', 10</code>	a
%e	Floating point in scientific notation	<code>printf '%e', 10;</code>	1.000000e+001 (OS dependent)
%f	Floating point in fixed decimal notation	<code>printf '%f', 1;</code>	1.000000

Understanding the BLT of Perl

- Some syntax rules
 - Keyword “sub” is used in front of function names
 - Variables begin with the \$ character
 - Comment lines begin with the # character
 - The & character is used when calling a function

Branching in Perl

`& speak ;`

- Calls the subroutine
- `sub speak`
- Defines the subroutine

```
# Perl program demonstrating branching

$word = "Hack";
&speak;

$word = "Or";
&speak;

$word = "Die!";
&speak;

sub speak {
    print"$word\n";
}
```

For Loop in Perl

- For loop

```
# Prepares a shell script that does a ping sweep
for ($a = 1; $a <100; $a++)
{
    print "ping 192.168.1.$a -w1\n";
}
```

Testing Conditions in Perl

```
# Leet test
print "How smart are you (1-10): ";
open (INFO, "-");
$IQ = <INFO>;
if ($IQ > 7) {
    print "1337!\n";
}
else {
    print "1u53r!\n";
}
```

Understanding Object-Oriented Programming Concepts

- New programming paradigm
- There are several languages that support object-oriented programming
 - C++
 - C#
 - Java
 - Perl 6.0
 - Object Cobol

Components of Object-Oriented Programming

- Classes
 - Structures that hold pieces of data and functions
- The :: symbol
 - Used to separate the name of a class from a member function
 - Example:
 - `Employee::GetEmp()`

Example of a Class in C++

```
class Employee
{
public:
    char firstname[25];
    char lastname[25];
    char PlaceOfBirth[30];
    [code continues]
};

void GetEmp()
{
    // Perform tasks to get employee info
    [program code goes here]
}
```

Ruby Example

```
require 'msf/core'
require 'zlib'

class Metasploit3 < Msf::Exploit::Remote
  Rank = GreatRanking # aslr+dep bypass, js heap spray, rop, stack bof

  include Msf::Exploit::FILEFORMAT

  def initialize(info = {})
    super(update_info(info,
      'Name' => 'Adobe CoolType SING Table "uniqueName" Stack Buffer Overflow',
      'Description' => %q{
(SING) table
          This module exploits a vulnerability in the Smart INdependent Glyplets
          handling within versions 8.2.4 and 9.3.4 of Adobe Reader. Prior versions are
          assumed to be vulnerable as well.
        },
      'License' => MSF_LICENSE,
      'Author' =>
        [
          'Unknown', # 0day found in the wild
          '@sn0wfl0w', # initial analysis
          '@v1ck3ck', # initial analysis
          'jduck' # Metasploit module
        ]
    )
  end
end
```

- Metasploit is written in Ruby
- See link Ch 7u

LOLCODE

```
HAI 1.2
CAN HAS STDIO?
I HAS A VAR
IM IN YR LOOP
    UP VAR!!!
    VISIBLE VAR
    IZ VAR BIGGER THAN 10? KTHX
IM OUTTA YR LOOP
KTHXBYE
```

Links Ch 7x, Ch 7y



Can Haz
LOLCODE
by Code School

SIGN IN

1 LERN HOW 2 LOLCODE

O HAI! YR KITTEH, TELL ME BOUT DEMZ. 2 TELL ME UR KITTEH'S NAYM, U TYPE:

VISIBLE "<KITTEH NAYM GOEZ HERE>"


REMEMBR 2 PUT YR KITTEH'S NAYM IN QUOTES.

JUS GIMMEH ANSWER



1

Yr Input

SUBMIT CODE 

Yr Output

Brainfuck

Character	Meaning
>	increment the data pointer (to point to the next cell to the right).
<	decrement the data pointer (to point to the next cell to the left).
+	increment (increase by one) the byte at the data pointer.
-	decrement (decrease by one) the byte at the data pointer.
.	output the byte at the data pointer.
,	accept one byte of input, storing its value in the byte at the data pointer.
[if the byte at the data pointer is zero, then instead of moving the instruction pointer forward to the next command, jump it <i>forward</i> to the command after the <i>matching</i> <code>]</code> command.
]	if the byte at the data pointer is nonzero, then instead of moving the instruction pointer forward to the next command, jump it <i>back</i> to the command after the <i>matching</i> <code>[</code> command.

"Hello, World!" in Brainfuck

```
+++++++[>+++++>+++++++>+++>+
<<<<-]>+>+.+++++.++>+<<+++++++>+.++>----->-----
----->+>.
```