



University of Dayton

Department of Computer Science

CPS 475/575

Secure Application Development

Lecture 5 – Secure Socket Programming in C & Lab 2

Phu Phung

1/28/2020

Today's agenda

- Lab 1 review
 - Reminder: Lab 1 submission is due tonight at 11:55 pm
 - Please re-read Slide 27 - Lecture 1 for submission policy.
- Secure Programming in C
- Hands-on: Lab 2 - Task 1
 - Exploiting and fixing vulnerable C programs
- (Tentative) Socket Programming in C
 - Hands-on: A simple TCP Client

Lab 1 Review

- Released on Isidore on Tuesday, January 21
 - Submission deadline: 11:55 PM January 28
 - Instruction: <http://bit.ly/secad-s20-lab1>
 - Update: 6/32 submissions by Sunday night, 13/32 by this class
- Network traffic with Wireshark
- Experiments with HTTP
- Experiments with HTTPS

Lab 1 – Task 3: HTTPS

c. Application data the browser sent

You can use `$ nslookup phu-udayton.bitbucket.io` to double-check the IP address

ssl

No.	Time	Source	Destination	Protocol	Length	Info
19	2020-01-26 07:02:25.584367780	10.0.2.15	18.205.93.11	TLSv1.2	573	Client Hello
21	2020-01-26 07:02:25.613305537	18.205.93.11	10.0.2.15	TLSv1.2	1504	Server Hello
23	2020-01-26 07:02:25.613906762	18.205.93.11	10.0.2.15	TLSv1.2	1504	Certificate[TCP segment of a reas...
25	2020-01-26 07:02:25.614028134	18.205.93.11	10.0.2.15	TLSv1.2	338	Certificate Status, Server Key Ex...
27	2020-01-26 07:02:25.630970695	10.0.2.15	18.205.93.11	TLSv1.2	182	Client Key Exchange, Change Ciph...
29	2020-01-26 07:02:25.657691565	18.205.93.11	10.0.2.15	TLSv1.2	314	New Session Ticket, Change Ciph...
30	2020-01-26 07:02:25.657707822	18.205.93.11	10.0.2.15	TLSv1.2	143	Application Data
32	2020-01-26 07:02:25.698993570	10.0.2.15	18.205.93.11	TLSv1.2	233	Application Data
34	2020-01-26 07:02:25.699298801	10.0.2.15	18.205.93.11	TLSv1.2	301	Application Data
36	2020-01-26 07:02:25.699658277	10.0.2.15	18.205.93.11	TLSv1.2	94	Application Data
38	2020-01-26 07:02:25.740284325	18.205.93.11	10.0.2.15	TLSv1.2	94	Application Data
40	2020-01-26 07:02:25.812027078	18.205.93.11	10.0.2.15	TLSv1.2	1248	Application Data
42	2020-01-26 07:02:26.118703424	10.0.2.15	18.205.93.11	TLSv1.2	133	Application Data
44	2020-01-26 07:02:26.126609261	10.0.2.15	18.205.93.11	TLSv1.2	135	Application Data
46	2020-01-26 07:02:26.171511589	18.205.93.11	10.0.2.15	TLSv1.2	224	Application Data
48	2020-01-26 07:02:26.176913820	10.0.2.15	18.205.93.11	TLSv1.2	98	Application Data
50	2020-01-26 07:02:26.196268664	18.205.93.11	10.0.2.15	TLSv1.2	170	Application Data

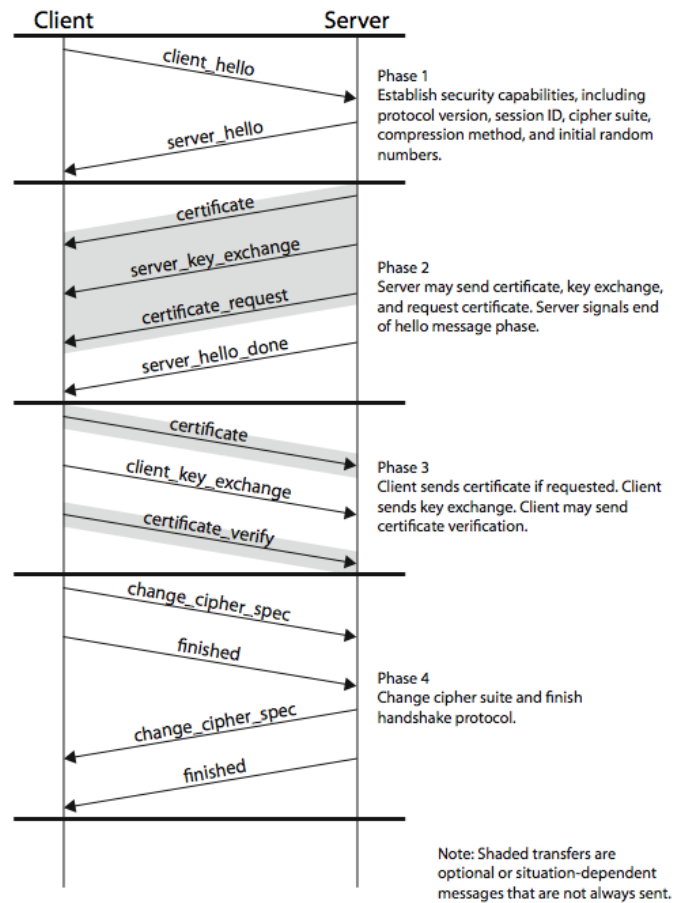
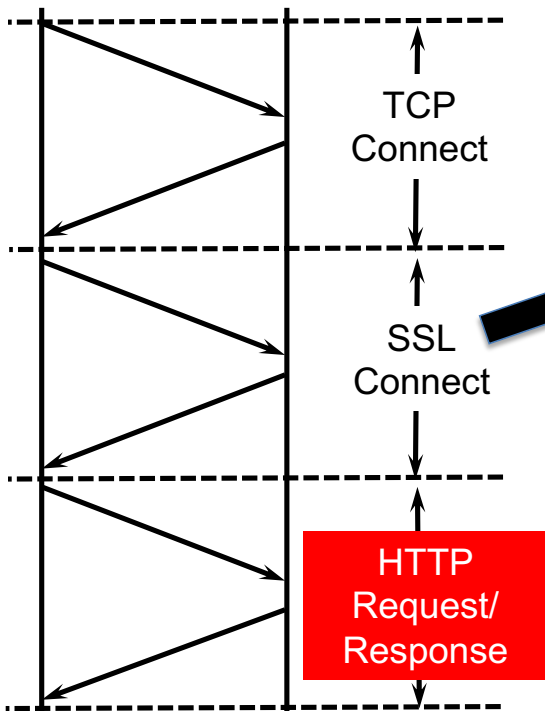
▶ Frame 32: 233 bytes on wire (1864 bits), 233 bytes captured (1864 bits) on interface 0

- ▶ Linux cooked capture
- ▶ Internet Protocol Version 4, Src: 10.0.2.15, Dst: 18.205.93.11
- ▶ Transmission Control Protocol, Src Port: 49788, Dst Port: 443, Seq: 4129927709, Ack: 4026755525, Len: 177
- ▼ Secure Sockets Layer
 - ▼ TLSv1.2 Record Layer: Application Data Protocol: http2
 - Content Type: Application Data (23)
 - Version: TLS 1.2 (0x0303)
 - Length: 172
 - Encrypted Application Data: 0000000000000000132875eed856bc7d3cf3bd34b2a41273c...

Payload is encrypted application data (ssl.app_data), 172 bytes

Packets: 53 · Displayed: 17 (32.1%) · Dropped: 0 (0.0%) · Profile: Default

Review: HTTPS Transaction



Lab 1 – Task 3: HTTPS

e. TCP Stream



Today's agenda

- Lab 1 review
 - Reminder: Lab 1 submission is due tonight at 11:55 pm
 - Please re-read Slide 27 - Lecture 1 for submission policy.
- Secure Programming in C
- Hands-on: Lab 2 - Task 1
 - Exploiting and fixing vulnerable C programs
- (Tentative) Socket Programming in C
 - Hands-on: A simple TCP Client

Introduction to C



- A general-purpose, imperative procedural programming language
 - Developed at Bell Labs (1969-1973) for system programming (to re-implement the Unix OS)
 - Standardized by American National Standards Institute (ANSI C – 1989) and then by International standard (ISO) in 1990
 - C++ : an extension of C to support object-oriented

Key features of C

- Has low-level features of assembly language
 - can directly access memory through pointer manipulation
 - Concise syntax
- A high-level programming language
 - Block structure
 - Functions with some code encapsulation
 - Weak type checking

A C program structure

```
1 /* include libraries */
2 #include <stdio.h>
3 #include <stdlib.h>
4
5 int main (int argc, char *argv[])
6 {
7     printf("Hello World\n");
8 }
```

- You can save in any **filename.c**
 - There must be the main function with void or two arguments (int argc, char *argv[])
 - argc: the number of arguments from user
 - *argv[]: value of arguments in array
 - argv[0]: the program name
 - argv[1]: the first argument value

Hands-on: **helloworld.c** program

- Pull the course repository to get the source code of the **helloworld.c** program

```
$ cd ~/secad && git pull
```

- Copy the code to your private repository folder.

```
$ cp lectures/lecture5* ~/secad-yourUDID/lectures
```

```
$ cd ~/secad-yourUDID/lectures/lecture5<tab>
```

- View/edit the file:

```
$ subl helloworld.c
```

Compilation and Execution a C program

- gcc is a C compiler and linker in Ubuntu (Linux) to compile and generate binary code.

Usage:

```
$ gcc <sourcefile.c>
```

- a.out will be generated

- Execution

```
$ ./a.out
```

```
1 /* include libraries */
2 #include <stdio.h>
3 #include <stdlib.h>
4
5 int main (int argc, char *argv[])
6 {
7     printf("Hello World\n");
8 }
```

- Hands-on:

- Compile and run the **helloworld.c** program:

```
$ gcc helloworld.c
```

```
$ ./a.out
```

```
seed@UbuntuVM:~/.../lecture5-secure-c$ ls
bufferoverflow helloworld.c
seed@UbuntuVM:~/.../lecture5-secure-c$ gcc helloworld.c
seed@UbuntuVM:~/.../lecture5-secure-c$ ls
a.out bufferoverflow helloworld.c
seed@UbuntuVM:~/.../lecture5-secure-c$ ./a.out
Hello World
seed@UbuntuVM:~/.../lecture5-secure-c$
```

A C program taking inputs

- Recall in the function main

```
int main (int argc, char *argv[])
```

- `*argv[]` : value of arguments in array

- `argv[0]` : the program name
- `argv[1]` : the first argument value

- Exercise:

- Instead printing “Hello World” as in the previous program, let’s print the first argument from the input, i.e.,:

```
$ ./a.out secad
```

```
secad
```

- *This is a typical example of echo application: just reprint the input*

Introduction to Lab 2

Lab description (available): <http://bit.ly/secad-s20-lab2>
(will be opened on Isidore at 11:55 PM January 28 - tonight)

Deadline: 11:55 PM Tuesday February 4

- Task 1 (10 points). Exploiting and fixing vulnerable C programs
 - Develop simple programs, perform attacks and fix them (**Today's lecture**)
- Task 2 (6 points). Develop a simple TCP client application in C.
 - Practices on C Socket programming
 - Consider the security aspects
- Task 3 (14 points). Develop a simple yet secure HTTP client application in C.
- Task 4 (100 points – 1% Extra credit). Full functional secure HTTP Client application

Hands-on: **myecho0.c** program

(Lab 2 – Task 1.a.i)

- Create a new folder "**lab2**" under the "**labs**" folder in your repository's local folder
- Copy the file **helloworld.c** program to that new folder, rename it to **myecho0.c** and revise it following the previous exercise
 - An insecure and incorrect version:

```
1 /* include libraries */
2 #include <stdio.h>
3 #include <stdlib.h>
4
5 int main (int argc, char *argv[])
6 {
7     printf(argv[1]);
8 }
```

Compilation and Execution a C program

- You can also specify the output program name. Usage:
`gcc <sourcefile.c> -o <program-name>`
- **Hands-on:** Compile and run the `myecho0.c` program developed previously

```
$ gcc myecho0.c -o myecho0
```

```
$ myecho0 secad
```

```
secad
```

```
[01/26/20]seed@VM:~/secad-pphung1$ gcc -o myecho0 myecho0.c
myecho0.c: In function 'main':
myecho0.c:6:5: warning: format not a string literal and no format arguments [-Wformat-security]
    printf(argv[1]);
    ^
[01/26/20]seed@VM:~/secad-pphung1$ myecho0 secad
secad[01/26/20]seed@VM:~/secad-pphung1$
```


The myecho0.c program revisit

```
$/myecho0 %x
```

?

```
$/myecho0 "%x %s"
```

?

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

Input from user

```
[05/27/19]seed@VM:~/.../lab2$ myecho0 %x
bfe46aa4[05/27/19]seed@VM:~/.../lab2$ myecho0 "%x %s"
bfe1ad94
```

```
0000007000I000y0ï000o00000000000000H000y0
000000000'000a000'00000006000z0ï000000$000E000Z0ñ0
x0H0p0000000000000000000000000000=0000000j0H0ï000000
000000D000f0E0û0`000[05/27/19]seed@VM:~/.../lab2$
```

This is a format string vulnerability

This is a wrong usage of the `printf` function
Correct usage: `printf("%s", argv[1]);`

myecho0.c revisit:

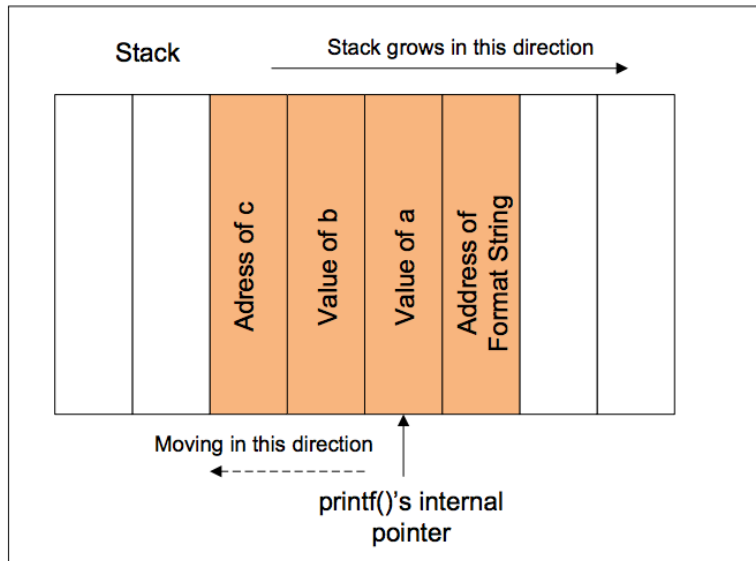
Format String Vulnerability

- Happens if your program:
 - uses functions such as `printf`, `snprintf` ... directly, or indirectly through system services (such as `syslog`) or other AND
 - the use of such functions allows input from the user to contain **control information** interpreted by the function itself

Format String at Runtime

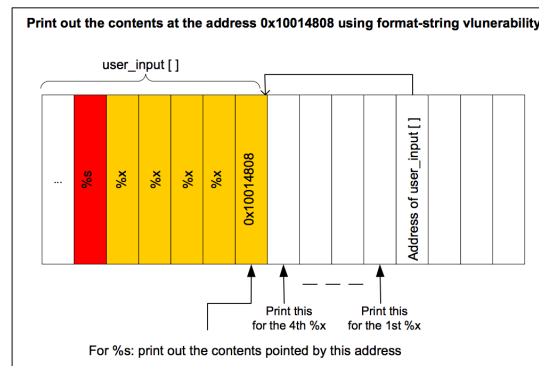
Example:

```
printf("a has value %d, b has value %d, c is at address: %08x\n",  
      a, b, &c);
```



Attacks on Format String Vulnerability

- Crash of the program
 - E.g.,: `printf ("%s%s%s%s%s%s%s%s%s%s%s%s");`
 - '%s' displays memory from an address on the stack
- Viewing the process memory
 - Viewing the stack, e.g.:
`printf ("%08x %08x %08x %08x %08x\n");`
 - Viewing memory at any location, e.g.:
`printf ("\x10\x01\x48\x08 %x %x %x %x %s");`



Attacks on Format String Vulnerability

- Overwriting of arbitrary memory
 - similar to common buffer overflows
`%497\x3c\xd3\xff\xbf<nops><shellcode>`
 - through pure format strings
`"\xc0\xc8\xff\xbf_%08x.%08x.%08x.%08x.%08x.%n"`
- Using this attack, attackers can do the following:
 - Overwrite important program flags that control access privileges
 - Overwrite return addresses on the stack, function pointers, etc.

Revised myecho0.c (Lab 2 – Task 1.a.ii)

- Step 0: Commit and push the current (vulnerable) code to your remote repository
- Step 1: Fixing the program
 - Use the correct format string in the `printf` function:
`printf("%s\n", argv[1]);`
- Step 2: Test the revised program again to ensure it is not vulnerable to format string
- Step 3: commit and push the revised code (you need the screenshot of the code difference to include to your report)