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

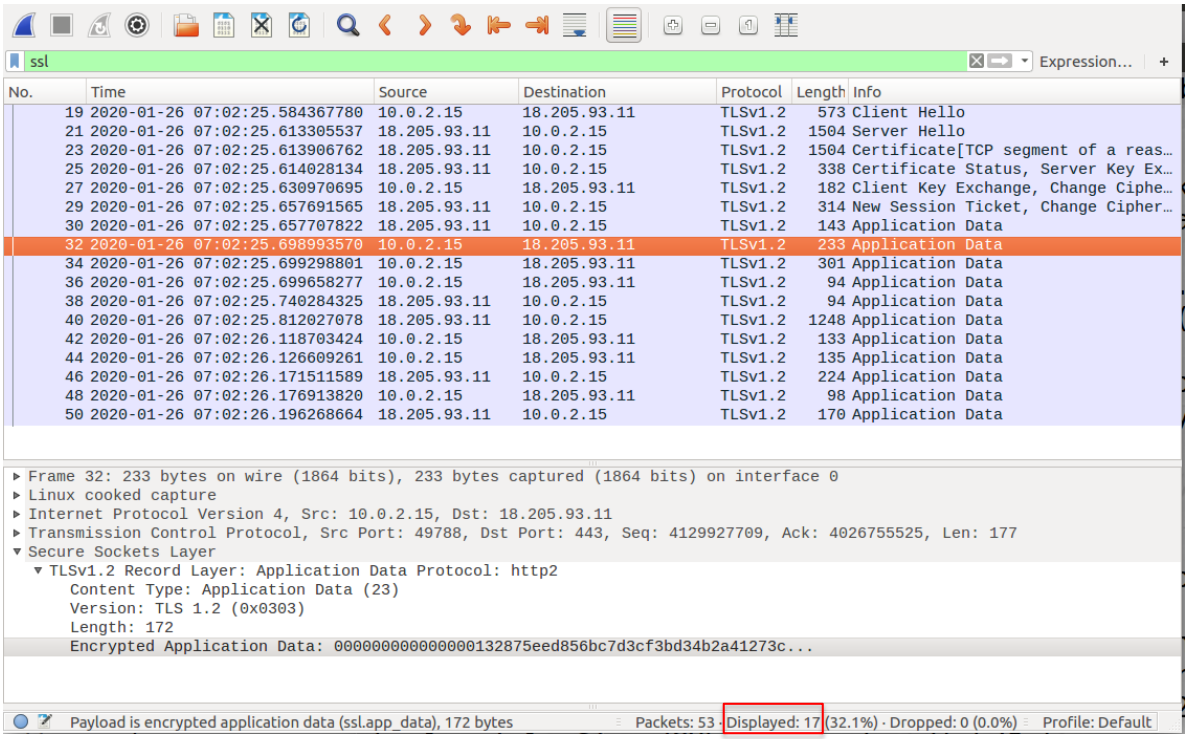
Lab 1 Review

- 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

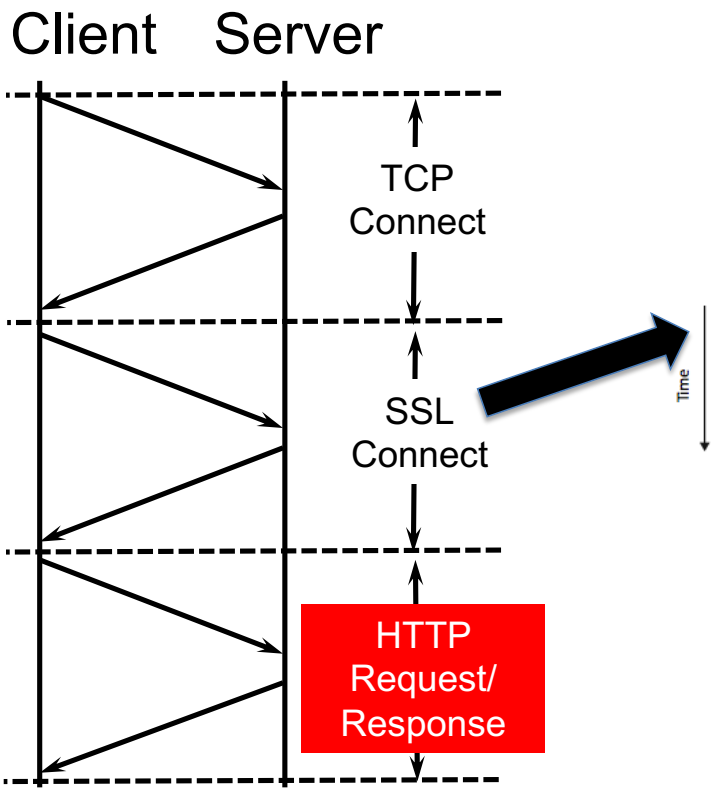
Lab 1 – Task 3: HTTPS

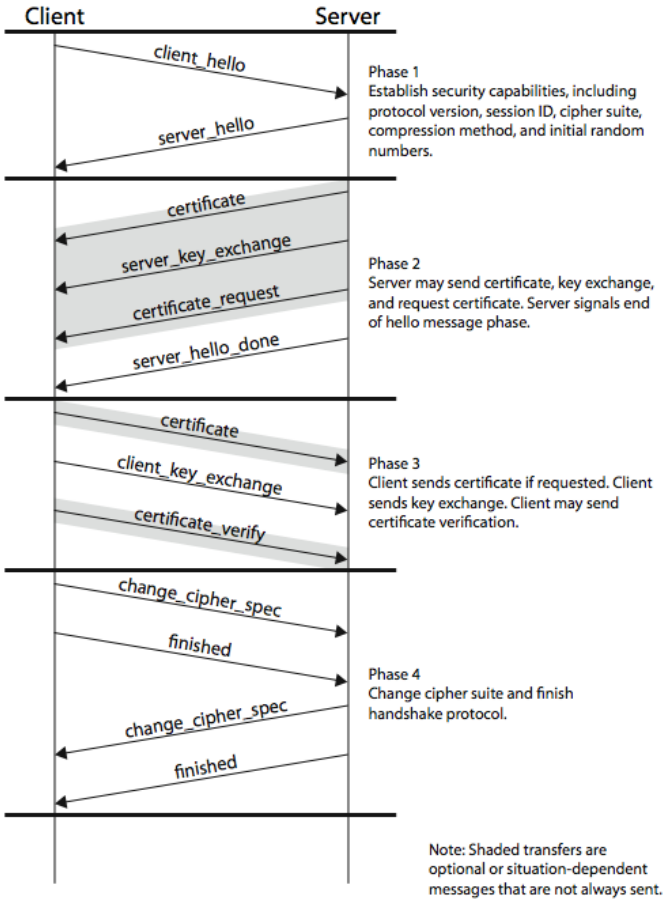
c. Application data the browser sent

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



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

```c
/* include libraries */
#include <stdio.h>
#include <stdlib.h>

int main (int argc, char *argv[])
{
   printf("Hello World\n");
}
```

- 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

```c
/* include libraries */
#include <stdio.h>
#include <stdlib.h>

int main (int argc, char *argv[])
{
   printf("Hello World\n");
}
```

- 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:

```c
/* include libraries */
#include <stdio.h>
#include <stdlib.h>

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

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 [-Wf
ormat-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
          ▯▯▯▯▯▯▯7▯▯▯I▯▯▯y▯ĭ▯▯▯o▯▯▯▯▯▯▯▯▯▯▯▯▯▯H▯▯▯y▯
▯▯▯▯▯▯▯▯▯'▯▯▯a▯▯▯'▯▯▯▯▯▯▯6▯▯▯z▯ĭ▯▯▯▯▯▯▯$▯▯▯E▯▯▯Z▯ñ▯

x▯H▯ȷ▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯=▯▯▯O▯▯▯j▯H▯ĩ▯▯▯▯▯
▯▯▯▯▯▯▯D▯▯▯f▯E▯ŭ▯`▯▯▯[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]);`

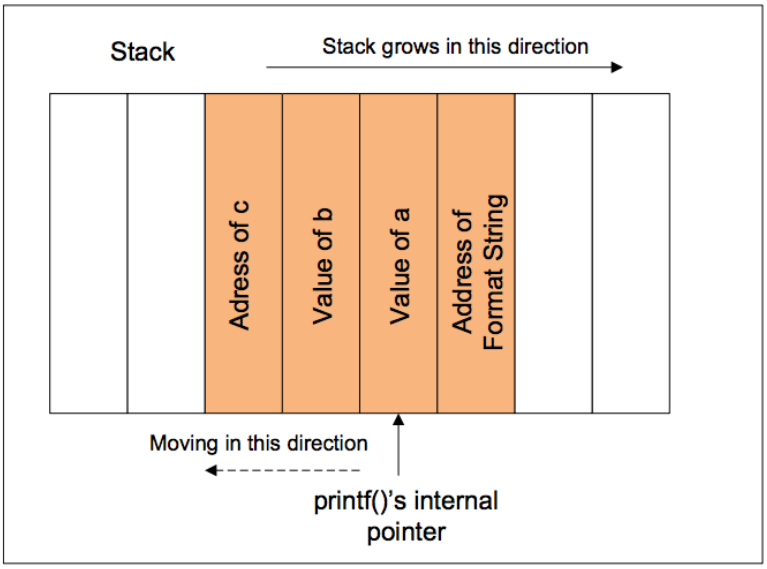
myecho0.c revisit: Format String Vulnerability

- 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

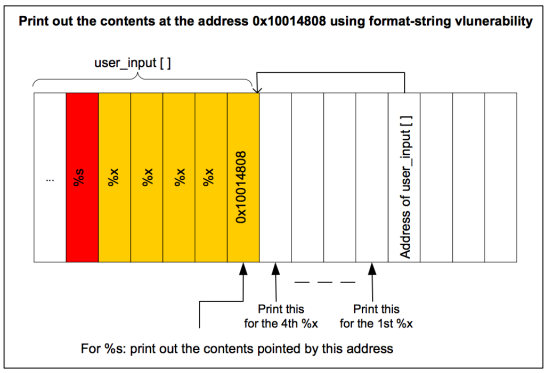
Example:

```
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 revise code (you need the screenshot of the code different to include to your report)