Fun with Information Engineering and Security Summer 2024

Day 3 (Aug-02, Fri)

Department of Information Engineering
香港中文大學
The Chinese University of Hong Kong

Why does input validation matter?

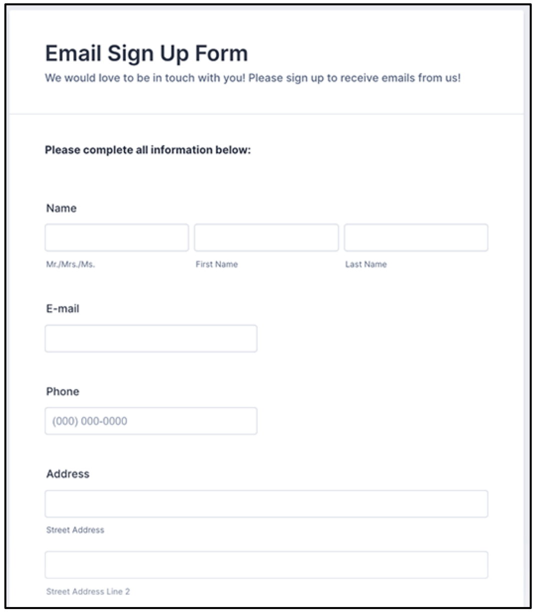
- Well, there’s only so much cryptography can do
- In fact, even cryptographic guarantees often depend upon the correctness of software **implementations**
  - Likewise for network (security) protocols
- Vulnerable software not only **breaks** the desired security control **policies**; sometimes allow attackers to completely **take over** a device
  - This is often how **botnets** are being built

How does software get exploited?

- A lot of the times, a process (a running software) gets **a purposely crafted malicious input** from an attacker
- And because of an **insufficiently thorough** (or a complete lack of) **input validation**, such inputs get through
  - Some can then crash the process (breaks availability)
  - Some can then extract sensitive information (breaks confidentiality)
  - Some can then execute arbitrary code (attackers can do whatever the want)

How does software get exploited?

- **Inputs** can come in many **different forms**, e.g.,
  - Inputs from Web forms
  - Inputs from command line
  - Email attachments
  - Configuration files
  - Environment variables
  - Parameters from Inter-Process Communication (IPC) communication
  - PDUs (frames/packets/segments) from network
  - …



Another example

- Web contents can be dynamically generated by server-side scripts (e.g., PHP)
  - “PHP: Hypertext Preprocessor” is a recursive acronym
- In PHP, passthru(string) executes string as command

passthru

(PHP 4, PHP 5, PHP 7, PHP 8)
passthru — Execute an external program and display raw output

Description

```
passthru(string $command, int &$result_code = null): ?false
```

The **passthru()** function is similar to the exec() function in that it executes a **command**. This function should be used in place of exec() or system() when the output from the Unix command is binary data which needs to be passed directly back to the browser. A common use for this is to execute something like the pbmplus utilities that can output an image stream directly. By setting the Content-type to `image/gif` and then calling a pbmplus program to output a gif, you can create PHP scripts that output images directly.

Another example

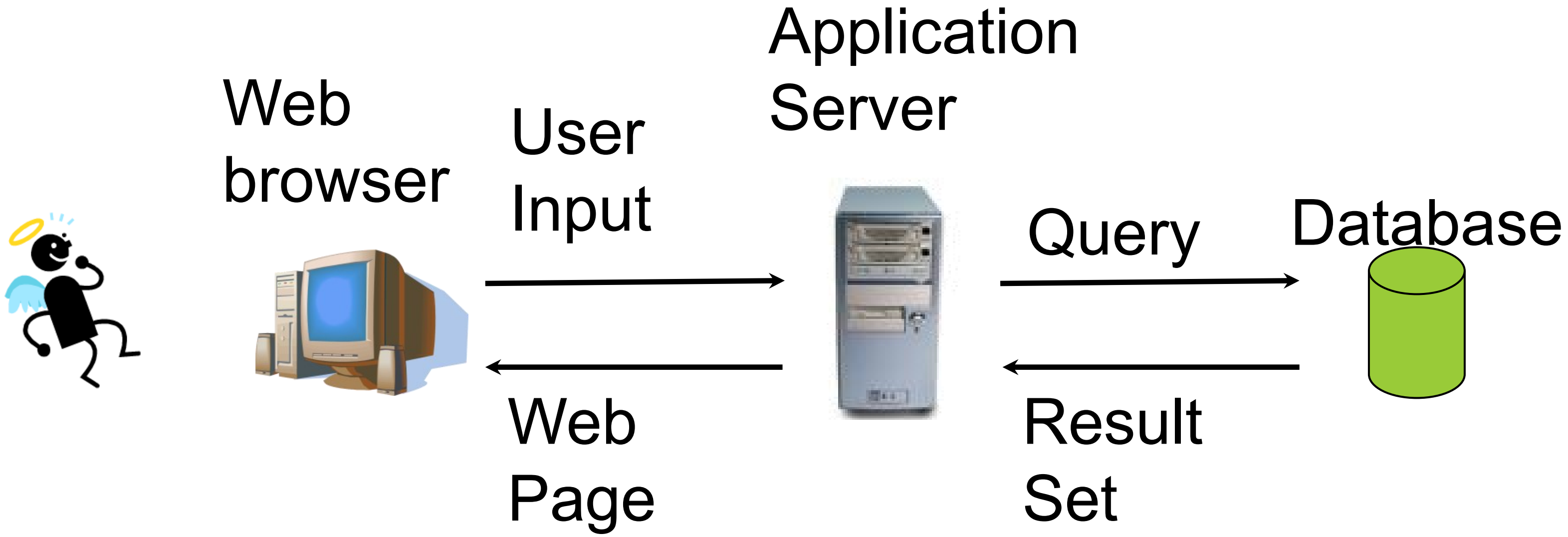
- Imagine a simple PHP-based Web page that displays a user's usage log (e.g., a Web portal for checking printing quota)

```
echo 'Your usage log:<br />';
$username = $_GET['username'];
passthru("cat /logs/usage/$username");
```

- Since attacker can choose the username on the Web page, it can put ";" in the input to run additional command with the PHP daemon's user privilege (depending on the system config, can be root), e.g.,
    - username = john; rm -rf /home

This type of threat is not hypothetical

- A very common threat to Websites, especially those that uses a relational database



Cross Site Scripting (XSS)

- A common security threat to Web due to problems in input validation
- It is a bit different from SQL injection, in the sense that XSS is more about attacking users of a Website
  - SQL injection, on the other hand, tends to be more about attacking the Website itself

A historic example of Stored XSS

- Back in the days, myspace.com was a popular social network site, and users can post custom HTML code on their personal pages
- To prevent abuse, myspace blocks many things, e.g.
  - Tags: <scripts>, <body>, <a href=javascript:foo()>
  - As well as the onclick event
- However, a clever guy figured out how to workaround these blocks

A historic example of Stored XSS

- Result = the “Samy worm” aka “JS.Spacehero worm”
  - Infects anyone who **visits an infected myspace page**, and **adds Samy as a friend** on myspace
  - Samy **got millions of friends** within 24 hours

- This kind of problem still occasionally haunts online forums/message boards, where users are free to type in long inputs, which will then be **stored** on the server and served to other users at a later time

How to defend against these?

- Need to properly encode + validate user inputs
  - Sometimes this is easier said than done; creative attackers (see the myspace example)

- Learn and use features from the underlying programming language to help you

How to defend against these?

- Encoding: “escapes” (preprocess) the user input so that it will be interpreted as data, not code
- Validation: checks that the user input is expected w.r.t your assumptions
    - E.g., not longer than a certain length, not containing malicious commands

How to defend against these?

- Examples of encoding functions
  - In PHP: htmlspecialchars(string, flags)

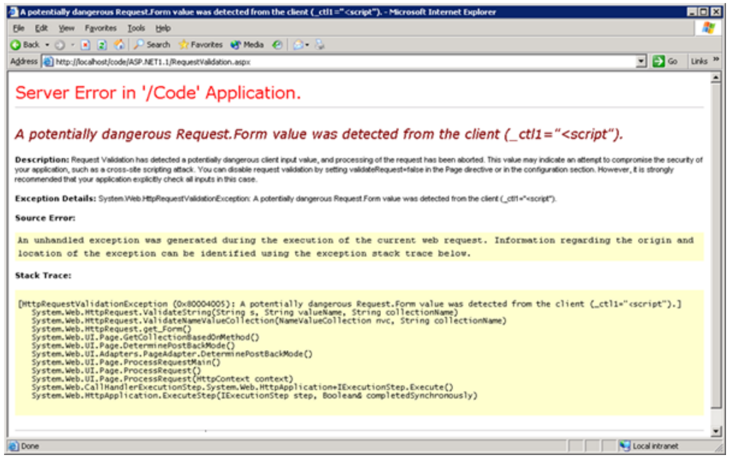
Performed translations

Character	Replacement
& (ampersand)	`&`
" (double quote)	`"`, unless **ENT_NOQUOTES** is set
' (single quote)	`'` (for **ENT_HTML401**) or `'` (for **ENT_XML1**, **ENT_XHTML** or **ENT_HTML5**), but only when **ENT_QUOTES** is set
< (less than)	`<`
> (greater than)	`>`

    - htmlspecialchars("Test")
      - Test
  - In ASP.NET: Server.HtmlEncode(string)
    - Similar to PHP’s htmlspecialchars()

How to defend against these?

- In some programming languages, for **specific scenarios** (e.g., Web request), some built-in validation features exist, e.g.,
  - In ASP.NET: validateRequest
    - Looks for a hardcoded list of patterns (blacklist)
    - Crashes the page if it finds un-encoded HTML code (such as the <script> tag) in the request content



SQL Injection

- Many Websites **assemble user inputs** as part of the SQL query, without thorough input validation
  - SQL = structured query language, is a domain-specific language commonly used to manage data in relational database systems

Phonebook Record Manager

Username: John

Password: open_sesame

◉ Display ○ Delete

Submit

SELECT * FROM phonebook WHERE username = 'John' AND password = 'open_sesame'

John's phonebook entries are displayed

SQL Injection

- An attacker can then inject additional logic/command to bypass checks and/or cause harm
- Use "--" to render rest of the original query ineffective (turn it into comment)

Phonebook Record Manager

Username: John' OR 1=1 --

Password: not needed

◉ Display ○ Delete

Submit

SELECT * FROM phonebook WHERE username = 'John' OR 1=1 --' AND password = 'not needed'

All phonebook entries are displayed

SQL Injection

- An attacker can then inject additional logic/command to bypass checks and/or cause harm
- Use "--" to render rest of the original query ineffective (turn it into comment)

DO NOT TRY THIS ON REAL WEBSITES
(they should have defenses, but what if they don't?)

Phonebook Record Manager

Username: J' ; DROP TABLE phonebook; --

Password: not needed

Display Delete

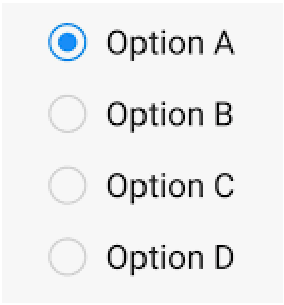
Submit

SELECT * FROM phonebook WHERE username = 'J' ; DROP TABLE phonebook; --' AND password = 'not needed'

All phonebook entries are removed

How to defend?

- Be very careful when you **assemble user inputs** into SQL commands
    - Sanitize/filter inputs, strip special symbols
    - Instead of allowing free text, if possible, perhaps use elements like radio buttons to **limit the input space**

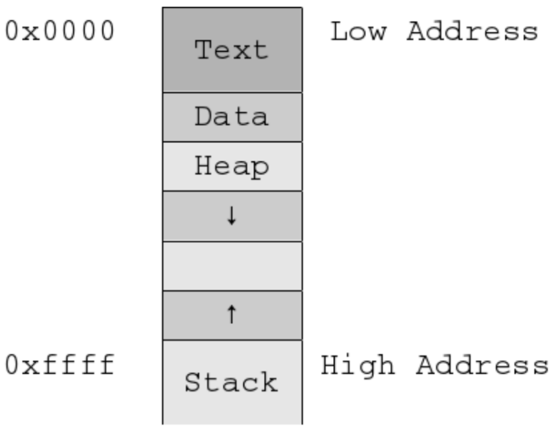


What is buffer overflow?

- Can think of it as another input validation problem
- In programming languages (e.g., C/C++) where memory management is done by the programmer, inputs related to memory access are not properly validated
- Because of that, various exploits exist
  - Sometimes the program can be induced into reading from unexpected addresses
  - Sometimes can overwrite existing values stored on the memory

Memory layout of a C program

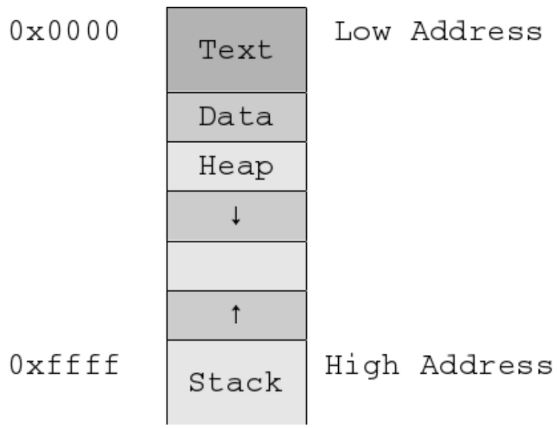
- To better understand (and exploit) buffer overflow, it's useful to first look at the memory layout
- Text segment: program code
  - Begins at low address
  - Normally read-only
- Data segment: global/static variables + constants
- Heap and stack: dynamically variables



Memory layout of a C program

- Heap: dynamically allocated storage space via the likes of malloc() and new()

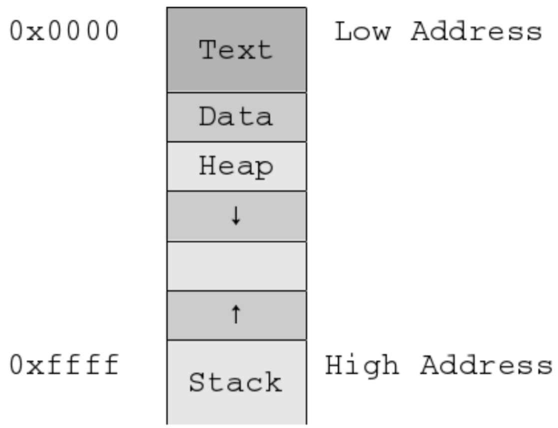
```
#include <stdio.h>
int main()
{
    char *p=(char*)malloc(sizeof(char));    /* memory allocating in heap segment */
    return 0;
}
```



- Stack: **local variables** + passing **parameters** to functions + return address of a **function call**
  - Each function call results in an activation record being added to the stack
  - This is also why too many recursions can lead to segmentation fault

Memory layout of a C program

- Heap vs stack
  - They share the same space
  - Grow **towards** each other



- Note: sometimes the diagram will be drawn in reverse (upside down)
  - pay attention to the position of **high** and **low addresses**

Stack

- Stack is “last in, first out” (LIFO)
- Basic operation: push, pop
  - Both on to/from the top of the stack
- Each function call results in an activation record being pushed to the top of the stack
  - Each function return results in an activation record being popped from the top of the stack

(Lack of) Input Validation strikes again

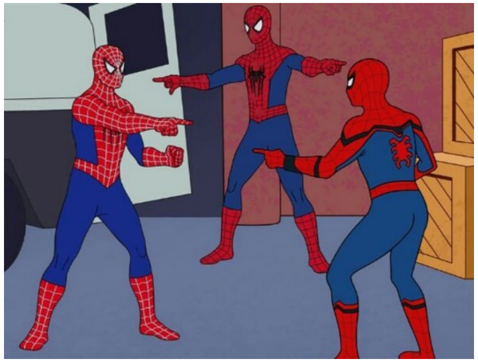
- Now we can look at buffer overflow (BOF) attacks
- Imagine a function has buffers of limited sizes on its stack
- Due to **input validation** problems, e.g., no boundary checks, one could read/write beyond the boundary of some buffers

```
void foo(int a, int b, int c) {
   char buffer1[13];
   … …
}
void main() {
   foo(1,2,3);
}
```

(Lack of) Input Validation strikes again

- In some languages, programmers are expected to manage memory themselves, and perform their own boundary checks
  - In C, many functions **DO NOT** have built-in length checks
    - strcpy (char *dest, const char *src)
    - strcat (char *dest, const char *src)
    - gets (char *s)
    - scanf (const char *format, ...)
    - sprintf (conts char *format, ...)
    - … …

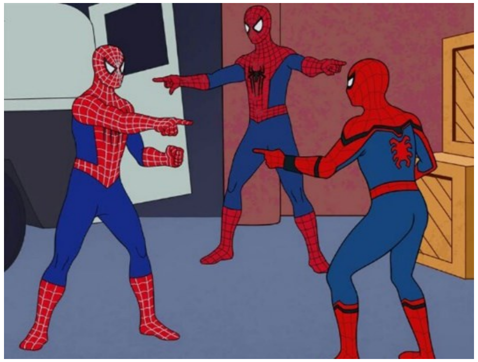
No boundary checks … who’s to blame?



(Lack of) Input Validation strikes again

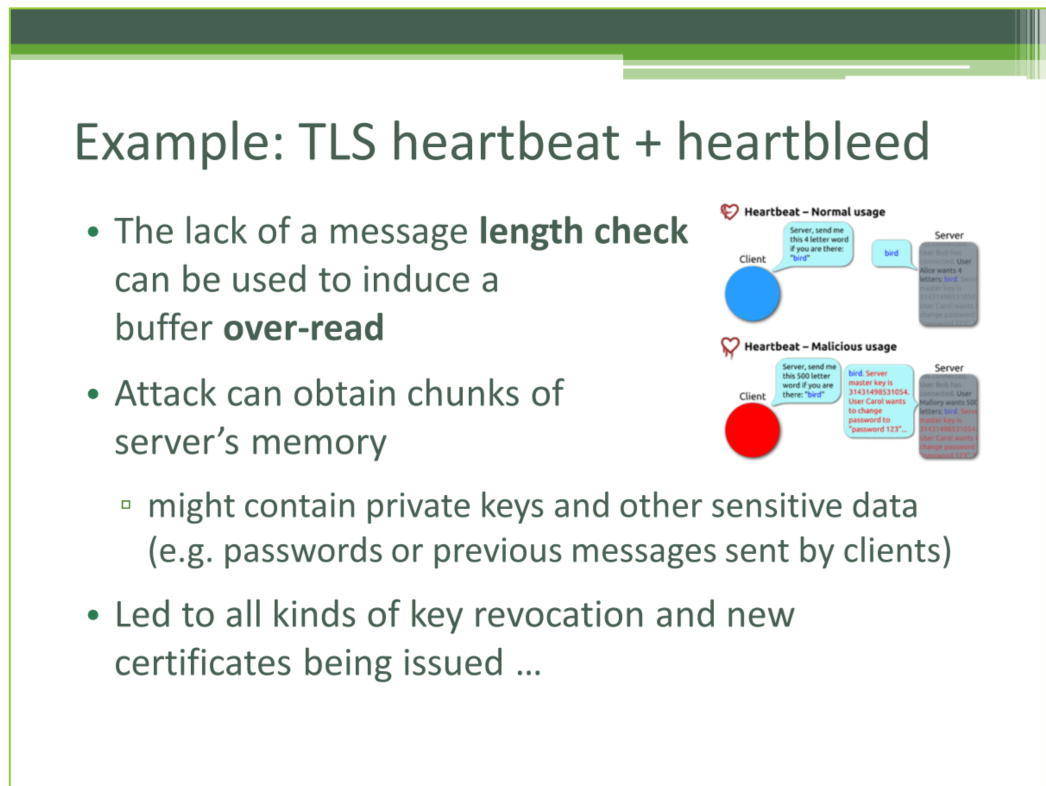
- Increasingly, many new languages take matter into their own hands, and feature boundary checks without bothering the programmer
  - E.g., Python, Java, Rust, …
  - Perhaps a better fit for programmer's expectation
- But loads of important software still written in C/C++
  - E.g., OSes, IoT firmware, …

No boundary checks …
who's to blame?



What's the harm?

- Here are some possibilities
  - **Overread** can lead to **exposure** of sensitive info
  - E.g., OpenSSL heartbleed bug
    - Server's private key can be stolen



What’s the harm?

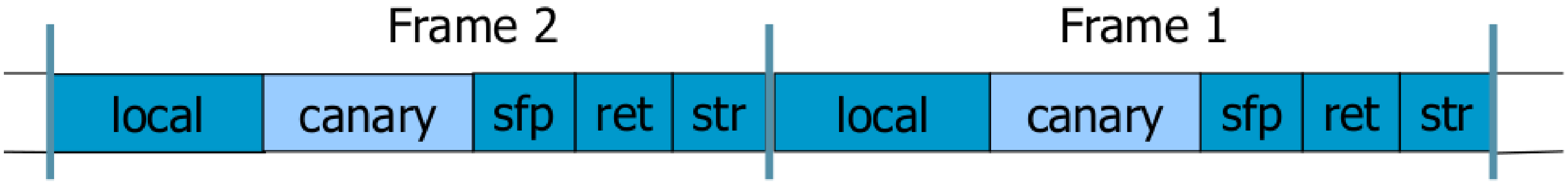
- Here are some possibilities
  - **Overwrite** some critical **variable** and change the program behaviour
    - We will try this in the challenge
  - **Overwrite return address** with invalid address => can crash the program (DoS; breaks availability)
  - **Overwrite return address** with meaningful address => may lead to arbitrary code execution

How to defend?

- First and foremost, do boundary checks
  - But programmers keep forgetting
- What else can we do?
  - StackGuard
  - NX bit
  - ASLR

How to defend? (StackGuard)

- A runtime checking technique
- Compiler adds code to a function call's prologue and epilogue, to add and check a special "canary" value
  - "canary in a coal mine"
    - detect the presence of carbon monoxide
  - While smashing the stack, if the "canary" value doesn't look right, then the program knows something is wrong



Stack guard in action

```
#include <stdio.h>
#include <string.h>

void echo(char *inp) {
   int a = 10;
   char b1[2];
   strcpy(b1, inp);
   printf("echo:\n");
   printf("%s\n", b1);
}

int main() {
   char input[10];
   printf("what is your input?\n");
   scanf("%s", input);

   echo(input);
}
```

Copy and paste this code to
https://www.onlinegdb.com/online_c_compiler

then click “run”

Now try to give a long input

```
*** stack smashing detected ***: terminated
Aborted
```

How to defend? (StackGuard)

- The presence of canary makes it **more difficult** to smash the stack, but **not impossible**
  - There are different ways of choosing the canary value
    - Random? How random?
    - Sometimes if the attacker can guess the canary value, then stack smashing can still work
  - In some cases, can target the error handler (called when the canary is dead)
  - If interested, more technical details can be found at https://www.coresecurity.com/sites/default/files/private-files/publications/2016/05/StackguardPaper.pdf