Aalto University

Software security

Tuomas Aura
CS-C3130 Information security

Aalto University 2023

Outline

- Untrusted input
- SQL injection
- Buffer overrun
- Web vulnerabilities: CSRF, XSS
- Input validation

There is no one simple solution that would make programs secure. → A competent programmer must learn about all the things that can go wrong. This lecture is only a starting point.

Untrusted input

- User and network input is untrusted
- Does my software get input from the Internet?
  - Documents, streams, messages, photos may all be untrusted
  - All modern applications are Internet clients or servers
  - Intranet hosts, backend databases, office appliances etc. are directly or indirectly exposed to input from the Internet

→ All software must be able to handle malformed and malicious input safely

Example: format string vulnerability

- Vulnerable C code:

```
int process_text(const char *input){
  char buffer[1000];
  snprintf(buffer, 1000, input);
  ...
}
```

- User input in the format string:
  - Input "**%x%x%x%x%x…**" will print data from the memory
  - Input "**%s%s%s%s%s…**" will probably crash the program
  - Input "**ABC%n**" will write value 3 to somewhere in the memory

SQL INJECTION

SQL injection example

- SQL query:

```
SELECT * FROM users WHERE username = 'Alice';
```

- Code with embedded SQL:

```
"SELECT * FROM users WHERE username = '" + input + "';"
```

- Attacker sends input:

```
input = "Bob'; DROP TABLE users; --"
```

- The query evaluated by the SQL database:

```
SELECT * FROM users WHERE username = 'Bob';
  DROP TABLE users; --';
```

SQL injection example 2

- Application greets the user by first name:

```
"SELECT firstname FROM users WHERE username = '" + input + "';"
```

- Attacker enters username:

```
input = "nobody' UNION SELECT password FROM users WHERE username
= 'alice'; --"
```

- The query evaluated by the SQL database:

```
SELECT firstname FROM users WHERE username = 'nobody' UNION
SELECT password FROM users WHERE username = 'alice'; --';
```

- This is why we should always assume that the attacker can read the password database

Mitigating SQL injection

- Minimum privilege: set tables as read-only; run different services as different users
- Sanitize input: allow only the necessary characters and string formats – but it is hard to do correctly!
- Escape input strings with safe library functions, e.g.
  - mysql_real_escape_string() in PHP
  - MySQLdb.escape_string(), MySQLdb.execute(), sqlalchemy.text() in Python

Mitigating SQL injection

- Prepared statements and stored procedures: precompiled SQL queries that can be executed many times with different parameter values

Use these!

- Disable SQL error messages to normal users
  → harder to build exploits

Do not make this mistake

- Jonne heard prepared statements are good for security:

```
$stmt = $conn->prepare("SELECT * FROM users WHERE username
= '" + input + "';");
$stmt->execute();
```

- Why is this wrong?

Swedish parliamentary election 2010

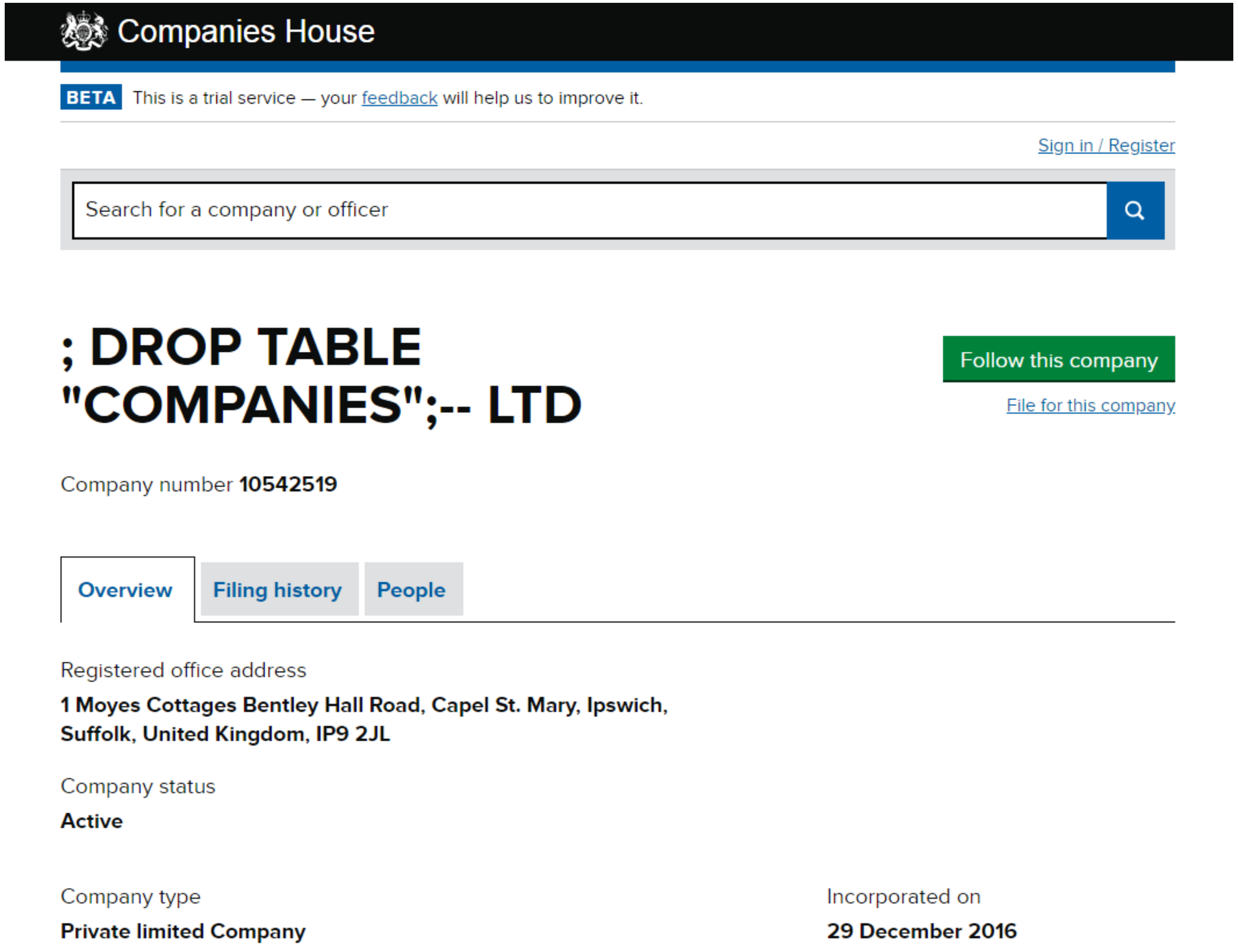
Some hand-written votes scanned by machine:

```
Halmstads västra valkrets;0903;Söndrum 3;Feministiskt initiativ;3
Halmstads västra valkrets;0903;Söndrum 3;Piratpartiet;1
Halmstads västra valkrets;0903;Söndrum 3;Syndikalisterna;1
Halmstads västra valkrets;0904;Söndrum 4;pwn DROP TABLE VALJ;1
Halmstads västra valkrets;0904;Söndrum 4;pwn DROP TABLE VALJ;1
Halmstads västra valkrets;0905;Söndrum 5;Feministiskt initiativ;1
Halmstads västra valkrets;0906;Söndrum 6;Feministiskt initiativ;1
Halmstads västra valkrets;1001;Holm-Vapnö;Raggarpartiet;1
Halmstads västra valkrets;1001;Holm-Vapnö;Raggarpartiet;1
```

```
Centrum, Övre Johanneberg;Klassiskt liberala partiet;1
Centrum, Övre Johanneberg;Svenskarnas parti;1
Centrum, Övre Johanneberg;Ett fristående frisinnat parti med fokus på miljö
Centrum, Övre Johanneberg;(Script src=http://hittepa.webs.com/x.txt);1
Centrum, Landalabergen m fl;Tillit;1
Centrum, Landalabergen m fl;SPI;1
Centrum, Landalabergen m fl;TilliT;1
```

http://www.val.se/val/val2010/handskrivna/handskrivna.skv

List of UK Registered Companies


Companies House

BETA This is a trial service — your feedback will help us to improve it.

Sign in / Register

Search for a company or officer

; DROP TABLE "COMPANIES";-- LTD

Follow this company

File for this company

Company number **10542519**

Overview | Filing history | People

Registered office address

1 Moyes Cottages Bentley Hall Road, Capel St. Mary, Ipswich, Suffolk, United Kingdom, IP9 2JL

Company status

Active

Company type

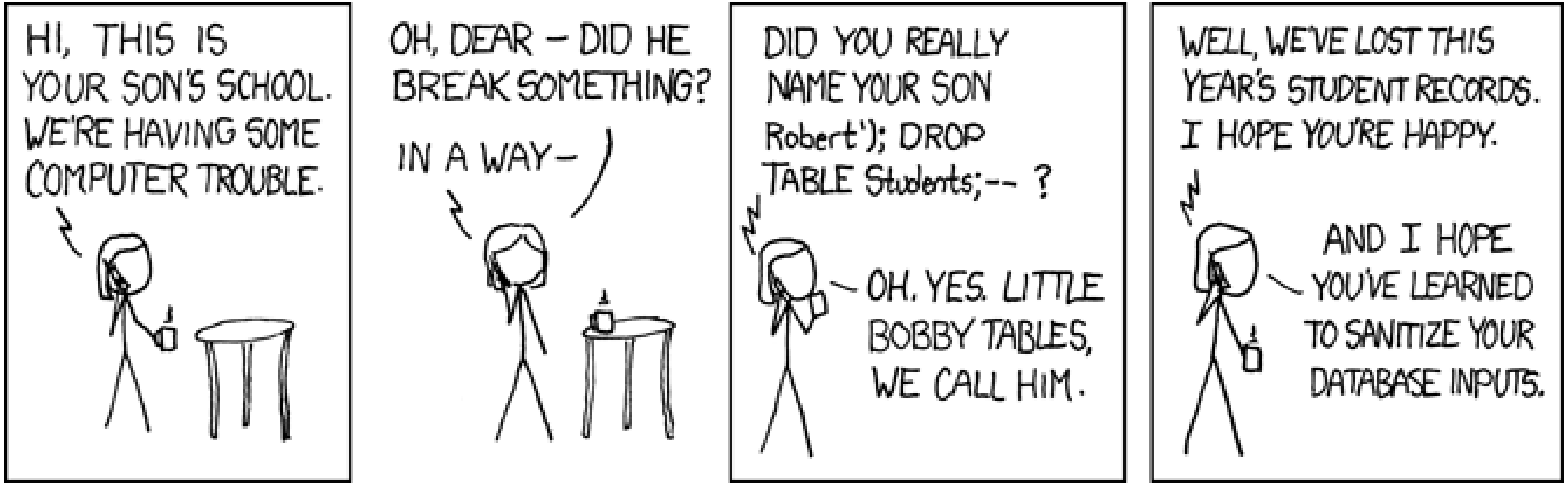
Private limited Company

Incorporated on

29 December 2016

https://web.archive.org/web/20171107023155/https://beta.companieshouse.gov.uk/company/10542519

XKCD: Exploits of a Mom



https://xkcd.com/327/

BUFFER OVERRUN

Used to be the number one software security problem. Still common in embedded devices and the Internet of Things.

Buffer overrun

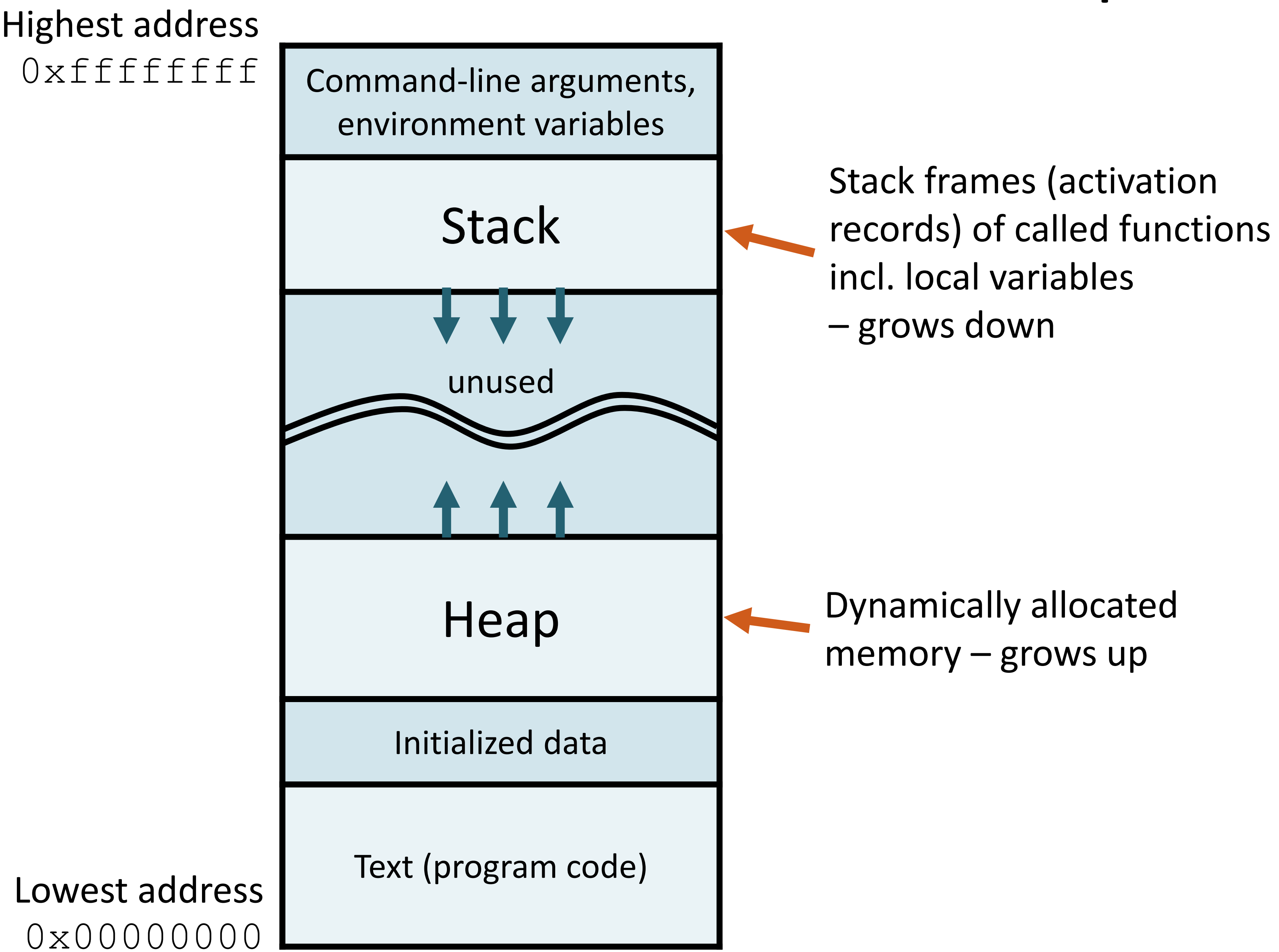
- Bug: failure to check for array boundary

```
#define MAXLEN 1000

char *process_input (char *input) {
  char buffer[MAXLEN];
  int i;
  for (i = 0; input[i] != 0; i++) {
    buffer[i] = input[i];
    ...
  }
```

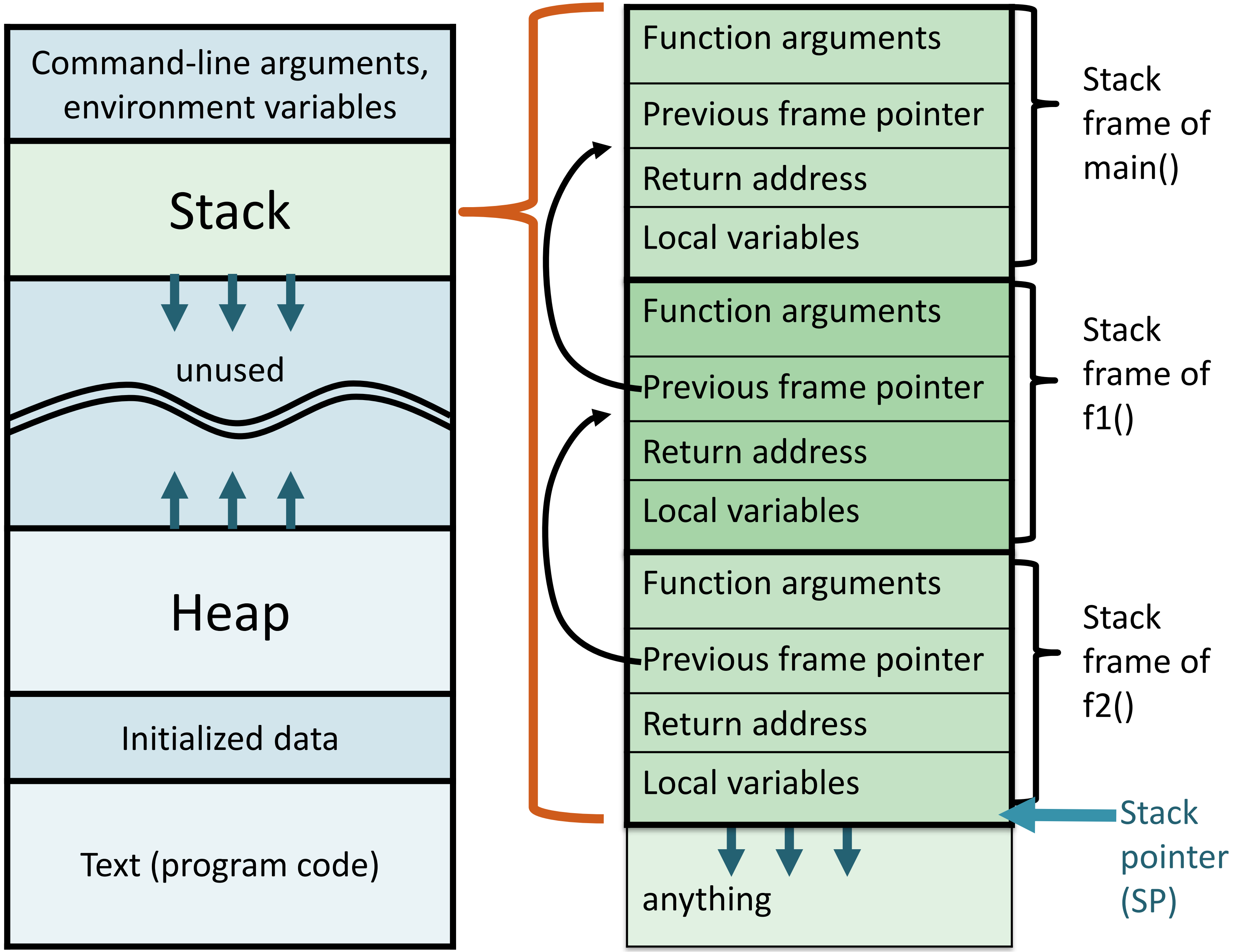
Loops until a null character found;
should check also for `i < MAXLEN`

Process virtual address space



Call stack (single-threaded program)

Details depend on processor architecture and programming language



Stack smashing

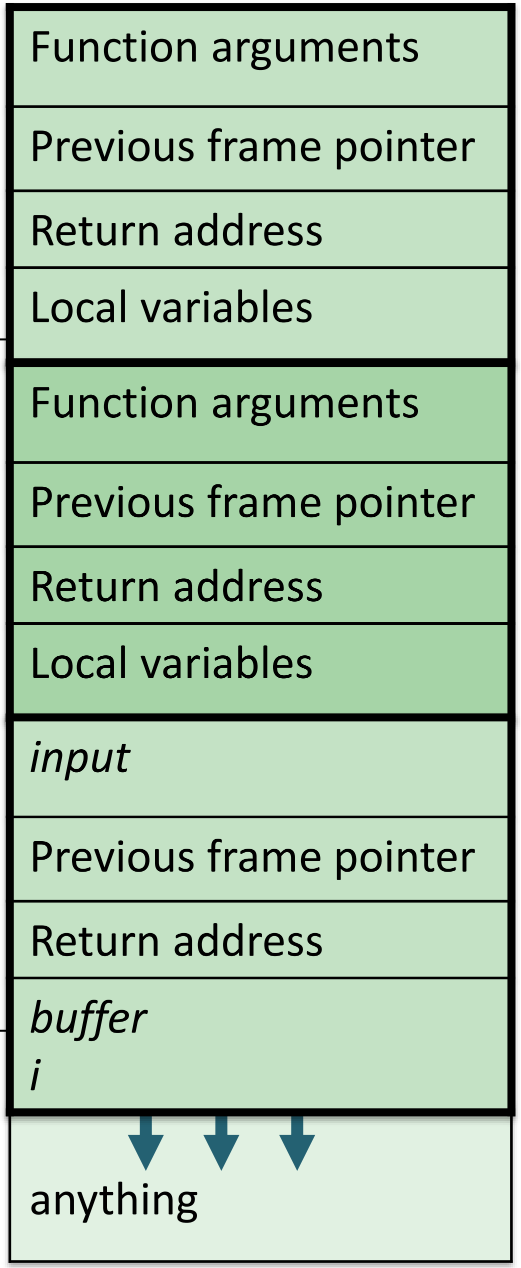
- Why are buffer overruns a security issue?

```
#define MAXLEN 1000

char *process_input (char *input) {
  char buffer[MAXLEN];
  int i;
  for (i = 0; input[i] != 0; i++) {
    buffer[i] = input[i];
    ...
  }
```

Loops until a null character found; should check also for **`i < MAXLEN`**

Call stack



Stack smashing

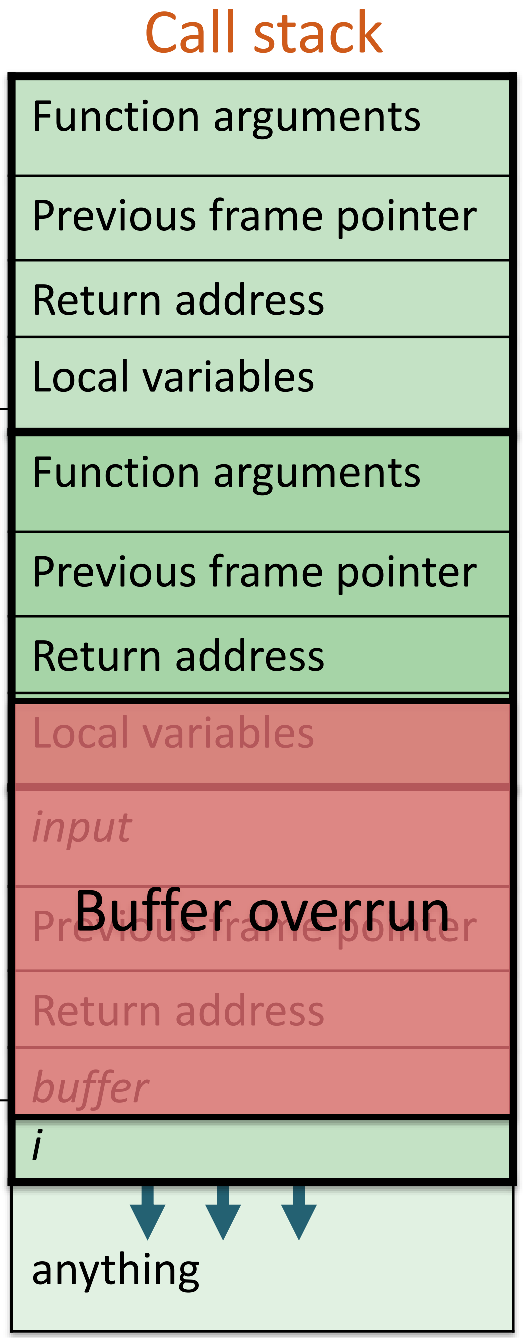
- Why are buffer overruns a security issue?

Too long input overwrites variables in the stack

```
#define MAXLEN 1000

char *process_input (char *input) {
  char buffer[MAXLEN];
  int i;
  for (i = 0; input[i] != 0; i++) {
    buffer[i] = input[i];
    ...
  }
```

Loops until a null character found; should check also for **i < MAXLEN**



Malicious code execution

- Why are buffer overruns a security issue?

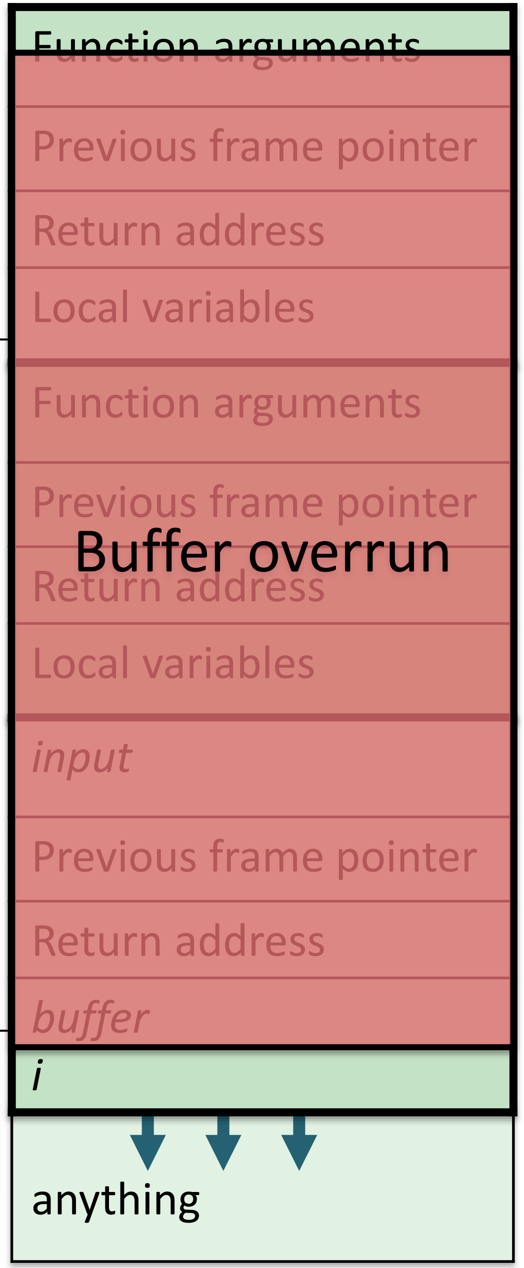
```
#define MAXLEN 1000

char *process_input (char *input) {
  char buffer[MAXLEN];
  int i;
  for (i = 0; input[i] != 0; i++) {
    buffer[i] = input[i];
    ...
  }
```

Loops until a null ch
should check also fo

Much too long input overwrites the function return address and previous stack frames

Call stack



Malicious code execution

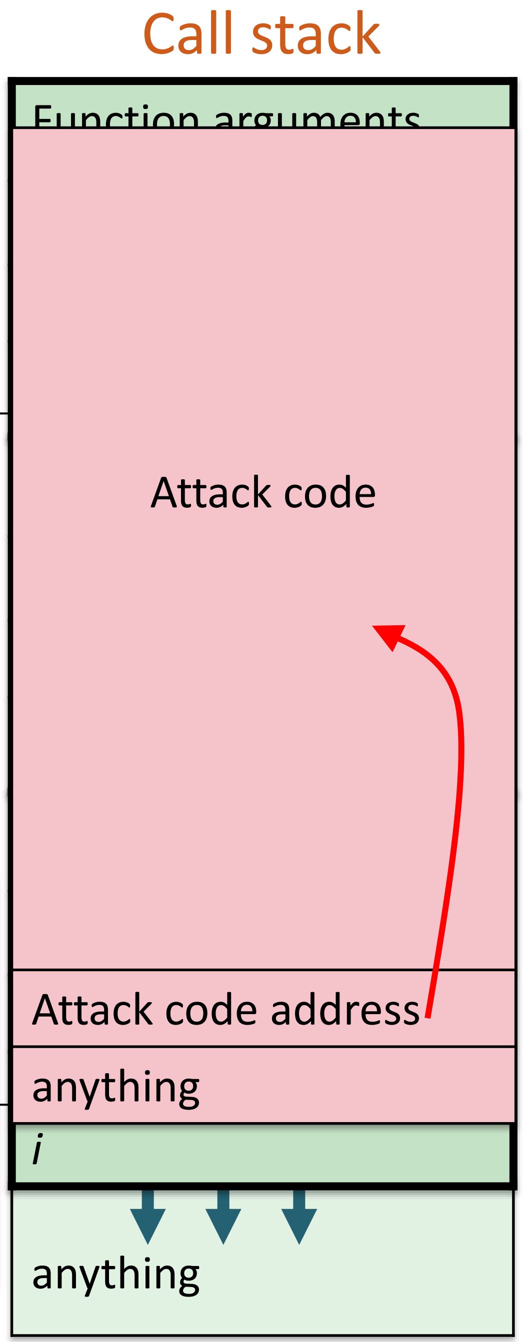
- Why are buffer overruns a security issue?

```
#define MAXLEN 1000

char *process_input (char *input) {
  char buffer[MAXLEN];
  int i;
  for (i = 0; input[i] != 0; i++) {
    buffer[i] = inp
    ...
  }
```

Loops until a null
should check als

When the function returns, execution will jump to the new return address, which points to attack code → malicious code execution with the process's permissions



Buffer overruns

- Buffer overruns may cause
  - data modification → unstable program behavior
  - access violation “segmentation fault” → process crashing
  - malicious data modification
  - code injection → attacker gains full control of the process

Running exploit code

- How attacker gains control:
  - Stack overruns may overwrite function return address or exception handler address
  - Heap overruns may overwrite function pointer or virtual method table
- How difficult is writing an exploit?
  - Instructions and code widely available
  - There are people and companies actively developing exploits

Another example

- Vulnerabilities can be difficult to spot

```
#define BUFLEN 4

void vulnerable(void) {
  wchar_t buf[BUFLEN];
  int val;

  val = MultiByteToWideChar(
    CP_ACP, 0, "1234567", -1, buf, sizeof(buf));
  printf("%d\n", val);
}
```

Expected size of the destination buffer in wide characters , but sizeof gives bytes

Should calculate target buffer size as
`sizeof(buf)/sizeof(buf[0])`

[Code Red vulnerability, example thanks to Ulfar Erlingsson]

Buffer overrun prevention

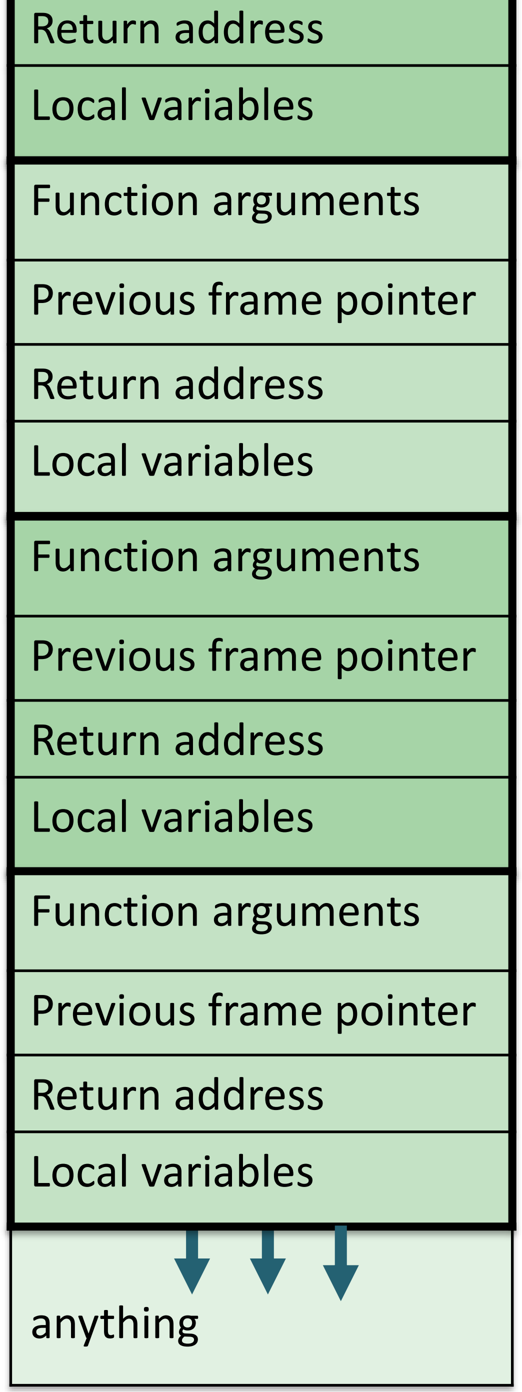
- Preventing buffer overruns:
  - Type-safe languages (e.g. Java, C#)
  - Programmer training, code reviews
  - Avoiding unsafe and difficult-to-use libraries: **`strcpy`**, **`gets`**, **`scanf`**, **`MultiByteToWideChar`**, etc.
  - Fuzz testing
  - Static code analysis, symbolic model checking: proving safety
- No reliable way to find all buffer overrun vulnerabilities
  → need to also mitigate consequences

Buffer overrun mitigation

- Stack canary
  - Store an unguessable value on the top of the stack frame in function prologue; check before returning
  - GCC -fstack-protector-all, Visual Studio /GS
- Non-executable (NX) bit
  - Set the stack and heap memory pages as non-executable
  - Breaks some code, e.g. JIT compilation
  - Often combined with memory layout randomization

Return to libc

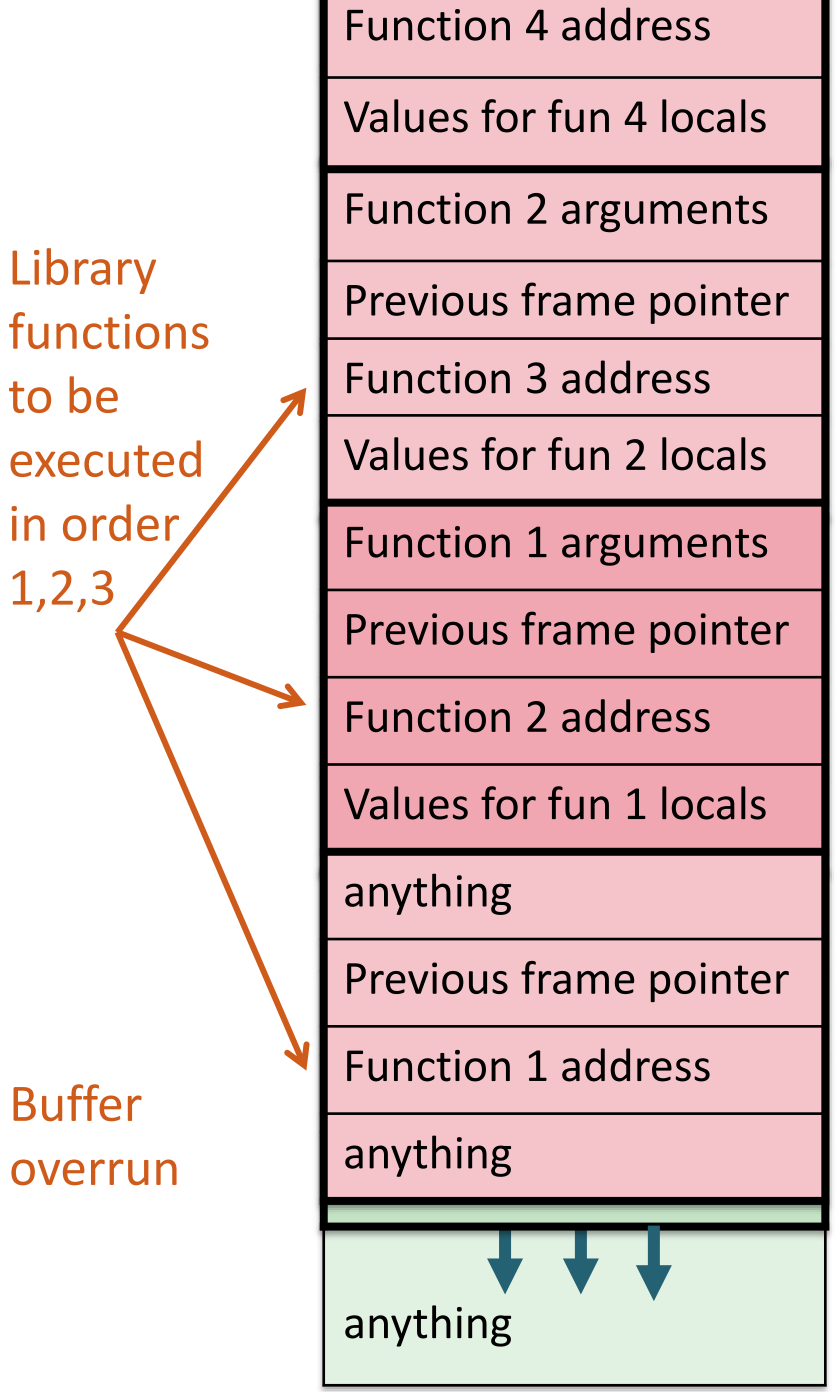
- NX prevents machine code insertion to stack
- However, there is another attack: return to libc, which uses existing executable code in the memory
  - e.g. standard library functions in libc



Extra material

Return to libc

- When function returns, execution jumps to the return address in stack
  - Point the return addresses to the beginnings of library (libc) functions
  - Set arguments as desired
- Typical exploit creates an executable page, copies attack code there, and runs it

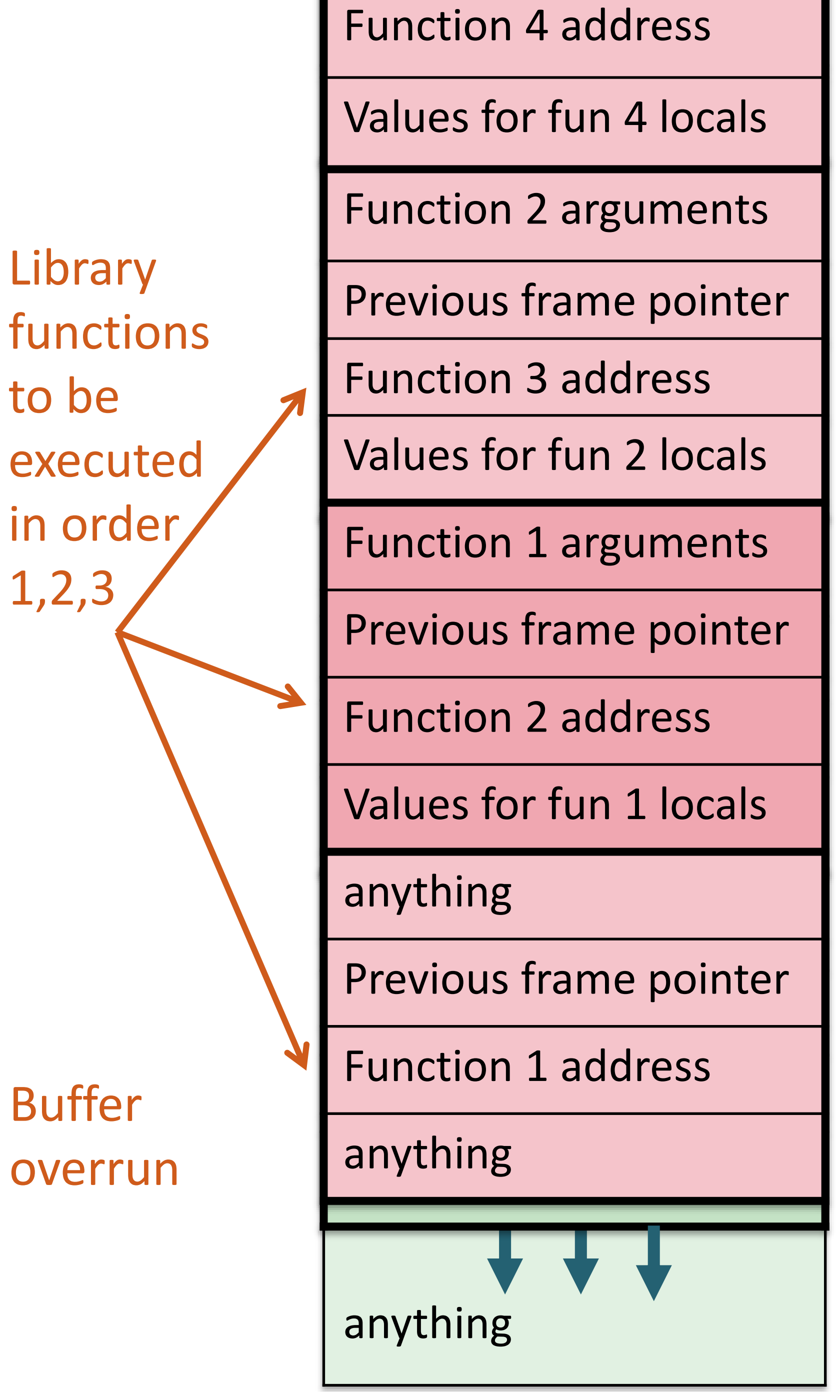


Extra material

Extra material

Return to libc

- Solution: combine NX with memory layout randomization
  - Load libc and other library code at a random memory offset → attacker does not know where to jump
- New solutions:
  e.g. Windows control flow guard



Integer overflow

- Integers in programming languages are not ideal mathematical integers
- Integer overflow can cause buffer overrun

Vulnerable code:

```
nBytes = (nBits + 7) >> 3;
if (nBytes <= bufferSize)
  copyBits(input, buffer, nBits);
```

Attacker input:

```
nBits = UINT_MAX
```

Evaluation

```
nBytes = (UINT_MAX + 7) >> 3
→ 6 >> 3 → 0

nBytes <= bufferSize
→ (0 <= bufferSize) → 1
```

WEB VULNERABILITIES: CSRF, XSS

Cross-site request forgery (CSRF)

- Fictional example:
  - Users on *social.net* stay logged in with a session cookie
  - JavaScript on Bob's web page *bobs.org*:

```
<script type="text/javascript">
frames['hidden'].window.location =
'http://social.net/AddFriend.php?name=Bob';
</script>
```

- Why possible? Interaction between web sites is usually prevented by the same origin policy. However, web links and HTTP GET and POST redirection to another site is allowed.

Preventing CSRF

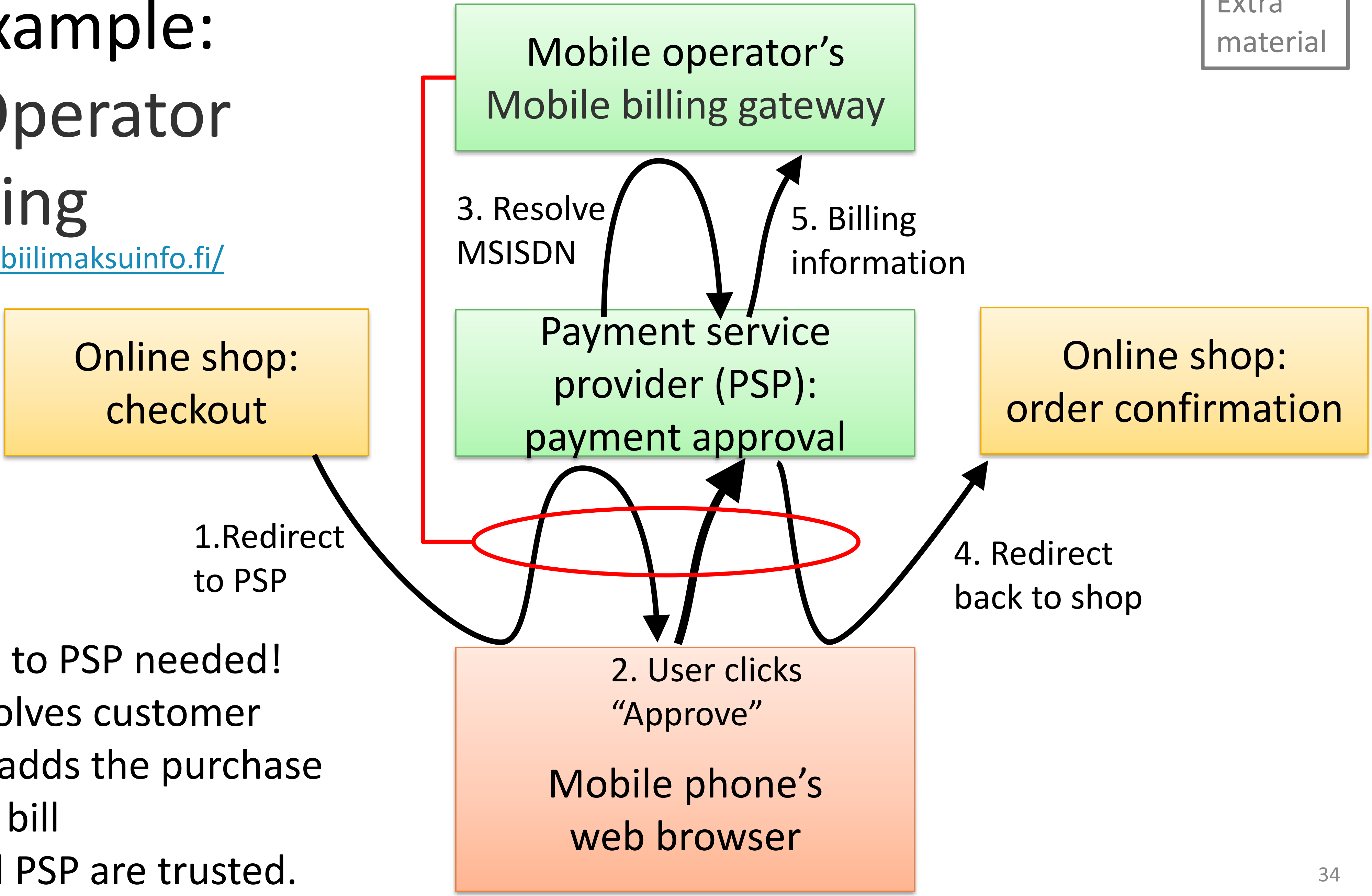
- Server checks Referer (sic) field in HTTP requests
- CSRF token i.e. secret session identifier in all GET URLs or POST payloads; attacker would need to guess it

Modern web application frameworks prevent CSRF with good session management (including CSRF token)

CSRF example: Direct Operator Billing

http://www.mobiilimaksuinfo.fi/

Extra material

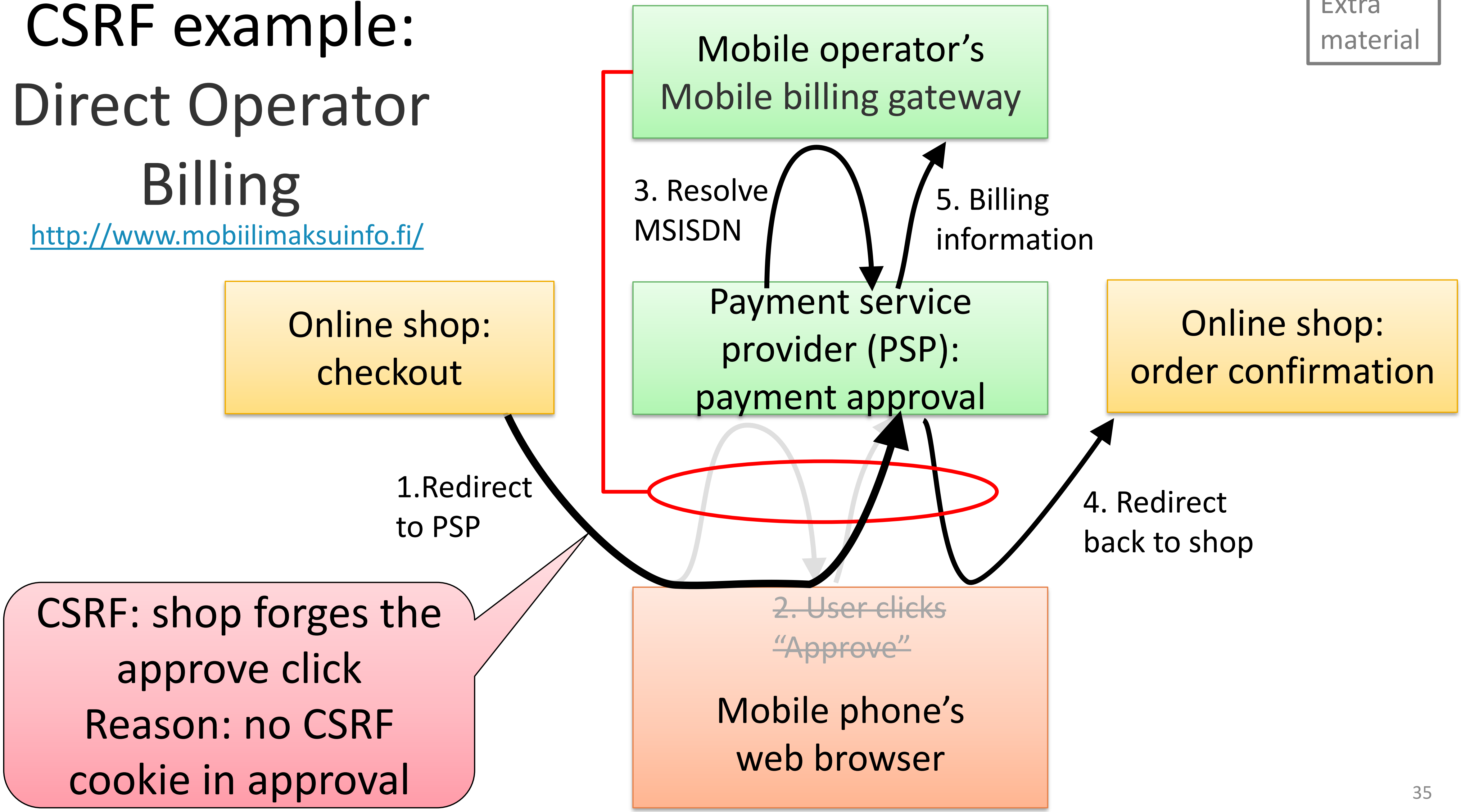


- No user login to PSP needed! Operator resolves customer MSISDN and adds the purchase to the phone bill
- Operator and PSP are trusted.

Extra material

CSRF example: Direct Operator Billing

http://www.mobiilimaksuinfo.fi/



Cross-site scripting (XSS)

- User-posted content on web sites may contain malicious JavaScript
- Fictional example:
  - Social.net allows users to post comments. Bob’s comment:

```
<b onmouseover="$.get('
http://social.net/AddFriend.php?name=Bob′);">
Be my friend!<b>
```

  - Another user reads the blog and moves mouse over the text
- This is stored XSS: malicious script stored on the server

Reflected XSS

- PHP code on a web server:

```
<html><body><?php
print "Page not found: ". urldecode($_SERVER["REQUEST_URI"]);
?></body></html>
```

- Typical output: `Page not found: /foo.html`

- Bob tricks Alice to follow this URL-encoded link:

```
http://social.net/%3Cscript%3E%3D%22%24.get%28%27http%3A%2F%2Fsoci
al.net%2FAddFriend.php%3Fname%3DBob%E2%80%99%29%3B%3C%2Fscript%3E
```

- The error page on social.net will contain this:

```
Page not found: /<script>="$.get('http://social.net/AddFriend.php
?name=Bob’);</script>
```

- This is reflected XSS: malicious script sent via the server but not stored there

Preventing XSS on web servers

- Browsers try to detect XSS, but they are not perfect. A lot still depends on the web application programmer

- Avoid embedding input into output; generate the output from scratch when possible
- Filter <tags>, Javascript or all angled brackets from user-posted content
- When you need to embed untrusted data into a web page, encode it first as HTML entities
- Do not embed untrusted data into <script>, <style>, URL, tag attributes or other unusual places
- Content Security Policy (CSP) allows web servers to declare in response headers what types of active content the response may contain, e.g. to exclude scripts
- Set HttpOnly flag on session cookies to prevent scripts from stealing them

What is untrusted input to web server?

- Inputs that may contain XSS or other malicious content:
  - Input from web client or user, or REST API client
  - Data read from database
  - Messages between servers
- Should escape or validate all these inputs

INPUT VALIDATION

Example: File path vulnerability

Vulnerable code:

```
char docRoot[] = "/usr/www/";
char fileName[109];
strncpy(fileName, docRoot, 9);
strncpy(fileName+9, input, 100);
file = fopen(fileName, "r");
// Next, send file to client
```

User input:

```
input =
"../../etc/passwd"
```

- The same file path has many representations → use the realpath(3) function to obtain the canonical representation
- Online services should be executed in a sandbox to limit their access: chroot(2), virtual machine or container

Sanitizing input

- Sanitizing input is not easy
- Escape sequences enable many encodings for the same character and string:
  - URL escapes: `%2e%2e%2f2e%2e%2f = ../../`
  - HTML entities:

    `<SCRIPT>` =<SCRIPT>

- Not sufficient to filter out `..` or `<`

SUMMARY

Why security failures?

- Why does software have security failures?
  – Greedy business sells dangerous products?
  – Lack of professional pride and ethics?

- Software is specified with use cases that describe the desired functionality. Security is about undesired functionality
- Market forces and software development practices encourage releasing a minimum viable product (MVP) – security not included
- Threats and attacks evolve. Software security is never ready

Other types of security bugs

- Injection of untrusted input into the command line, JavaScript, HTML, XML, format string, file path etc.
- Logical errors, e.g. time of check—time of use, use after free
- Integer overruns or signed/unsigned confusion
- Crypto mistakes, bad random numbers
- Insecure direct references
- Most software bugs first seem harmless but eventually someone figures out how to build an exploit

How to produce secure code?

- Programming:
  - Learn about bugs and vulnerabilities by example
  - Adopt secure coding guidelines
  - Use safe languages, libraries and tools
  - Code reviews, static checkers
  - Fuzz testing, penetration testing
- Software process:
  - Threat modelling
  - Define security requirements
  - Define quality assurance process

Security principles

- Keep it simple
- Minimize attack surface
- Sanitize input and output
- Least privilege
- Defense in depth
- Isolation
- Secure defaults
- Secure failure modes
- Separation of duties
- No security through obscurity
- Fix even potential bugs

List of key concepts

- Untrusted input, input validation or sanitization
- Buffer overrun call stack, heap, stack frame, malicious code execution, integer overrun, safe language, stack cookies, NX bit, return to libc, memory layout randomization
- SQL injection, code injection, stored procedure or prepared statement, escaping,
- Same-origin policy, cross-site request forgery CSRF, Referer, CSRF token, cross-site scripting XSS
- Input validation, canonical form, isolation
- Fuzz testing, penetration testing

Reading material

- Dieter Gollmann: Computer Security, 2nd ed. chapter 14; 3rd ed. Chapters 10, 18, 20.5–20.6
- Stallings and Brown: Computer security, principles and practice, 4th ed., chapter 10-11
- Michael Howard and David LeBlanc, Writing Secure Code, 2nd ed.
- Online:
  - Top 25 Most Dangerous Software Errors, http://cwe.mitre.org/top25/
  - SQL Injection Attacks by Example, http://unixwiz.net/techtips/sql-injection.html
  - OWASP, https://www.owasp.org/, see especially Top Ten
  - CERT Secure Coding Standards, https://www.securecoding.cert.org/confluence/display/seccode/SEI+CERT+Coding+Standards
  - Aleph One, Smashing The Stack For Fun And Profit (classic paper) http://inst.eecs.berkeley.edu/~cs161/fa08/papers/stack_smashing.pdf

Exercises

- Find examples of actual security flaws in different categories. Try to understand how they can or have been exploited.
- Which features in code may indicate poor quality and potential security vulnerabilities?
- When you find a security vulnerability, it is worth the trouble to write an exploit to prove how serious it is?
- How can error messages help an attacker?
- Buffer overrun in a type-safe language like Java will raise an exception. Problem solved — or can there still be a security issue?
- What is insecure direct object reference? Find some examples.
- What security bugs can occur in concurrent systems, e.g. multiple web servers that use one shared database?
- Find out what carefully designed string sanitization functions, such as *mysql_real_escape_string* or the OWASP Enterprise Security API, actually do.