CSci 4271W
Development of Secure Software Systems
Day 9: Defensive programming 2

Stephen McCamant (he/him)
University of Minnesota, Computer Science & Engineering

Outline

Defensive coding

Defensive design principles

Midterm, other announcements

More buggy examples

Defensive programming

How do we design and write programs so that security bugs are less likely to happen in the first place?
Like in threat modeling, it's important to ask "what could go wrong" in a program and write code that plans for it.

- Undefined behavior: what can go wrong with the compiler?
- Input validation: what can go wrong with our inputs?
- Safe call-outs: what can go wrong when calling out?
- Safe return values: what can go wrong with my outputs?

Outputs

The outputs of a program can potentially cause information disclosure, sometimes leading to EoP.

- "The third letter of your password is wrong"
- Query FAILED with output. . .

Feedback and errors

Examples that may reveal too much information:

- Displaying incorrect password/secret
- Returning program line/check causing an error
- Returning "hidden" information in comments or fields
- Returning error message from a library or external program

Formatting and encoding

- Will the consumer of output be another program?
  - What text encoding should be used?
  - Are tags, control characters, etc., important?
- Don't use user-specified formats
  - Especially not format strings
  - Limited choices can be safe

Outline

Defensive coding

Defensive design principles

Midterm, other announcements

More buggy examples

Defensive programming (2)

General guidelines. . .

- CODeMAP (Saltzer and Schroeder)
- Deploy defense in depth
- Separate data and control
- Test error-handling code

Saltzer and Schroeder

Don't forget your CODeMAP:

	C.omplete mediation
	O.pen design
Safe	De.faults
	M.echanism
Psychological	A.cceptability
	P.rivileges

Review at: `https://shostack.org/blog/the-security-principles-of-saltzer-and-schroeder`

Complete mediation

Check every access to every object...

- Race conditions (e.g., TOCTTOU):
  - Employ locks, `mkstemp`, atomic filesystem operations...
- Caching
- Server-side vs. client-side input validation

Open design

No "security by obscurity": the security of a system should not depend on secrecy of the design. Examples:

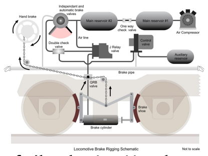
- NT password file in registry: format reverse-engineered
- "Backdoors" hidden in code: found with debuggers
- "Roll-your-own" cryptography

Being open source can improve security, but is not a guarantee (c.f. GnuPG, X Windows, ...)

Safe Defaults

A system should default to a secure state.

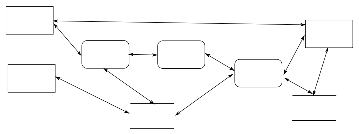
- Deny read/write by default
- No auto-execute
- Compare: fail-safe, fail-stop



Users will notice if the application fails, but attackers don't inform devs/ops if security fails.

Image source: `https://www.tsb.gc.ca/eng/lab/rail/2013/lp1872013/LP1872013.html`
Transportation Safety Board of Canada, TSB Laboratory Report LP187/2013, Fig. 4

Mechanism

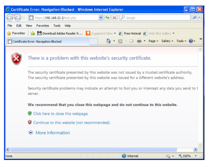


Economy of Mechanism: Each edge is a bug opportunity. Minimize edges.
Least Common Mechanism: Minimize complexity of high-degree components.

Psychological Aceptability

Security mechanisms must not:

- Prevent users from doing their work
- Force users to make important choices



Certificate Error: Navigation Blocked - Windows Internet Explorer

https://192.168.21.2/test.php

There is a problem with this website's security certificate.

The security certificate presented by this website was not issued by a trusted certificate authority.
The security certificate presented by this website was issued for a different website's address.

Security certificate problems may indicate an attempt to fool you or intercept any data you send to the server.

We recommend that you close this webpage and do not continue to this website.

Click here to close this webpage.

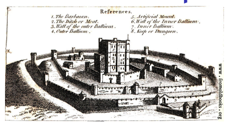
Continue to this website (not recommended).

More information

Privilege

- Least Privilege: A process/entity should have only have access to the resources it needs to function
- Separation of Privilege: Distribute important roles between multiple processes or entities

Defense in depth



Multiple, orthogonal defenses decrease the probability of security failures:

- Belt and suspenders
- ASLR + CFI + Stack Cookies + DEP
- Network firewall and software firewall

Image source: Francis Grose, "The Antiquities of England and Wales Vol I" (1783)

Separate data and control

- Avoid embedded code/scripts
- Don't serialize/interpret across system components
- Send query fields to stored procedures
- Send arguments to compiled programs

Test error handling code

```
if rare_error_condition:
  s1 = statement(1)
  s2 = action(s1)
  ...# how do I test this?
else:
  compute_as_normal()
  ...
```

$\longrightarrow$

```
def handle_error(state):
  state.s1 = statement(state)
  state.s2 = action(state)
  ...

if rare_error_condition:
  handle_error(state)
else:
  compute_as_normal()
  ...
```

Outline

Defensive coding

Defensive design principles

Midterm, other announcements

More buggy examples

Upcoming assignments

- Homework 2 is due tonight
- Homework 3 will be out soon, mostly about OS security
- Have you reached out to groups and started looking at Project 1?

Midterm 1 information

- In class (normal time and place) this Thursday
- Open book, open notes, any paper materials OK, but no electronics
- Pencil or erasable pen recommended
- Sorry, no sample old midterms released (changing format)
- Structure:
  - 3 homework-like questions (50 points total)
  - 10 short-answer questions (50 points total)

Midterm 1 topics

Covers from the beginning of the course through today

- Threat models, risk assessment
- DFDs and other diagrams
- STRIDE and other threat modeling
- Memory corruption attacks and mitigations
- Defensive programming and design

Outline

Defensive coding

Defensive design principles

Midterm, other announcements

More buggy examples

Non-defensive example 3

```
char *username = getenv("USER");
char *buf = malloc(strlen(username)+7);
sprintf(buf, "mail %s", username);
FILE *f = popen(buf, "w");
fprintf(f, "Hi.\n");
fclose(f);
```

Non-defensive example 4

```
try:
  connect_to_db(dbuser, dbpassword)
except Exception as e:
  return str(e)

if user not in userlist:
  return "User not found!"
if hashlib.sha256(password) != pw_hash_list[user]:
  return "Incorrect password!"
if account_number not in accounts[user]:
  return "Account number " + account_number + " not found!"

do_the_thing(user,account)
```

Non-defensive example 5

```
int makedir (char *newdir); {
  int len = (int)strlen(newdir);
  char *p, *buffer = (char*)malloc(len+1);
  strcpy(buffer,newdir);
  if (buffer[len-1] == '/') buffer[len-1] = '\0';
  if (mkdir(buffer,0775) == 0) goto done;
  p = buffer+1;
  while (1) {
    char hold;
    while(*p && *p != '/') p++;
    hold = *p; *p = 0;
    mkdir(buffer, 0775);
    if (hold == 0) goto done;
    *p++ = hold; }
  done: free(buffer);
  return 1; }
```