

Web Security

50.520 Systems Security
Paweł Szalachowski

Web

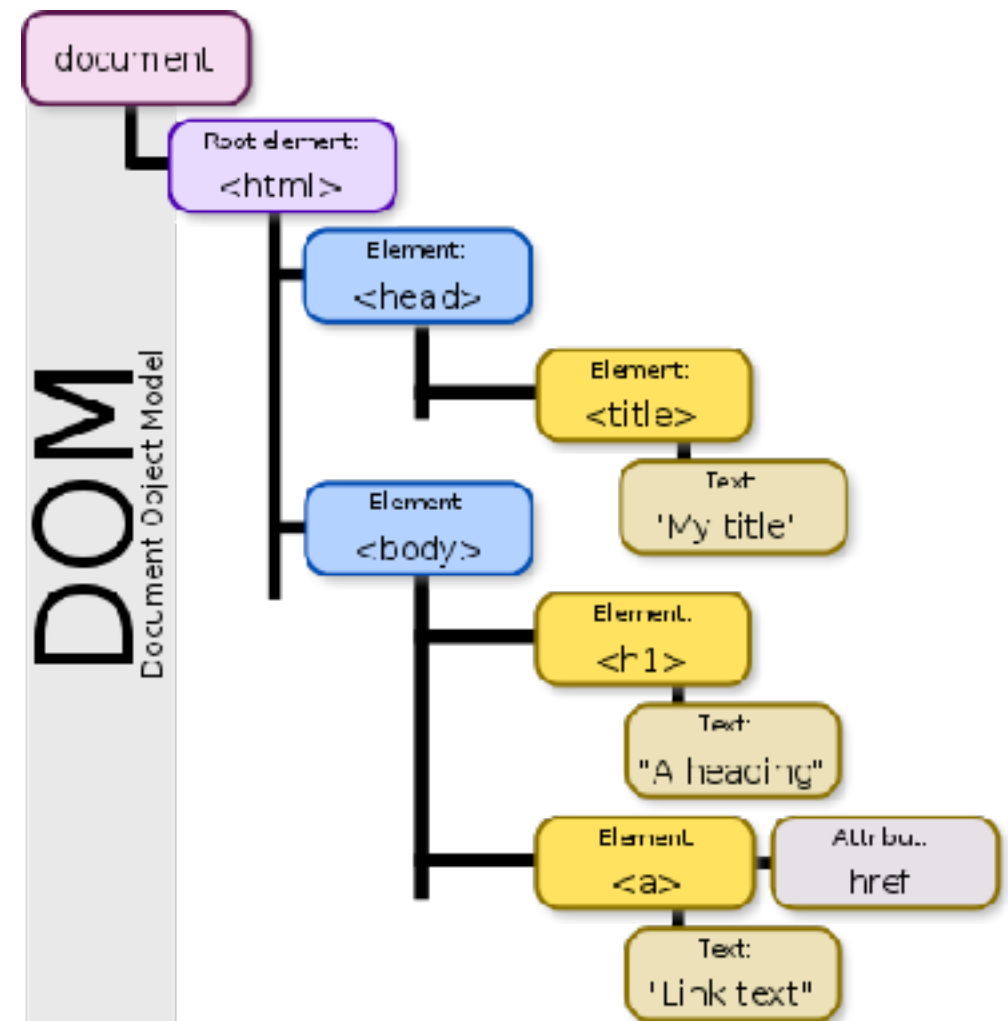
- Platform for deploying applications and sharing resources
 - Portable
 - Secure ?
- Uniform Resource Locator (URL)
- The Hypertext Transfer Protocol (HTTP)

Webpage

- Hypertext Markup Language (HTML)
 - Language to create webpages
 - Structured documents, objects, images, forms, ...
- Cascading Style Sheets (CSS)
 - Used for describing presentation of webpages
- JavaScript (JS)
 - Programming language
 - Primarily used to manipulate webpages

Webpage Rendering

- Parsers
 - HTML and CSS
- JS engine
 - Executes JS code
- Document Object Model (DOM)
 - Interface for representing and interacting with objects (HTML, XML, ...)
- Painter



Security

- Another system designed w/o security in mind
- Became very complex
 - Distributed (resources from other websites)
 - Many applications and services moved to web
 - Dynamic content, Audio, Video, I/O, ...
- Malicious websites should not be able to
 - modify not-related information of the computer or other websites
 - steal confidential information from the computer or other websites
 - spy on the computer activities or other websites

Same-origin Policy

- Isolation mechanism
 - Scripts from one webpage can access data of another webpage only if both webpages have the same **origin**
- Origin is defined as triple: (protocol, hostname, port)
- Origin is defined basing on a loaded URL
 - JS loads within the origin of the webpage that loaded it
- Cross-origin communication (CORS, postMessage(), ...)

Server-side Attacks

| OWASP Top 10 - 2013 | | → | OWASP Top 10 - 2017 | |
|--|---|---|--|--|
| A1 – Injection | → | | A1:2017-Injection | |
| A2 – Broken Authentication and Session Management | → | | A2:2017-Broken Authentication | |
| A3 – Cross-Site Scripting (XSS) | ↘ | | A3:2017-Sensitive Data Exposure | |
| A4 – Insecure Direct Object References [Merged+A7] | U | | A4:2017-XML External Entities (XXE) [NEW] | |
| A5 – Security Misconfiguration | ↘ | | A5:2017-Broken Access Control [Merged] | |
| A6 – Sensitive Data Exposure | ↗ | | A6:2017-Security Misconfiguration | |
| A7 – Missing Function Level Access Contr [Merged+A4] | U | | A7:2017-Cross-Site Scripting (XSS) | |
| A8 – Cross-Site Request Forgery (CSRF) | ⊗ | | A8:2017-Insecure Deserialization [NEW, Community] | |
| A9 – Using Components with Known Vulnerabilities | → | | A9:2017-Using Components with Known Vulnerabilities | |
| A10 – Unvalidated Redirects and Forwards | ⊗ | | A10:2017-Insufficient Logging&Monitoring [NEW,Comm.] | |

Code Injection Attacks

- Execute malicious user's input that is not sanitized properly
 - Always whitelist (not blacklist)
 - Never trust client-side validation
- Not only web
 - `system()`, `eval()`, ...
- Lack of or wrong separation (user and execution contexts)
 - Similar to buffer overflows ?

SQL Injection

- Web Services
 - Browser, Web Server, Database
 - Depending on browser's input, the server prepares a query and queries the database, and returns results to the browser
 - The database can be also modified
- Database
 - Collection of data (structured in tables)

SQL Injection

- SQL commands
 - SELECT *column* FROM *table* WHERE *condition*
 - INSERT INTO *table* VALUES (*v1*, *v2*, *v3*, ...)
 - DROP TABLE *table*
- Commands can be separated by semicolon

Example

- Server-side code

```
$student_name = $_POST['student_name']  
$sql = "SELECT StudID FROM Student  
      WHERE name='$student_name' ";  
$ret = $db->executeQuery($sql)
```

- What is executed for “?student=Alice”?
 - \$student_name is used directly in SQL query
- What if \$student_name = Alice'; SELECT * FROM Student;

Other examples

- `SELECT * FROM Users WHERE user='$user' AND pwd='$pwd'`
 - `$user = "` or 1=1 --"`
 - Everything after `--` (two dashes) is ignored
- Speed camera system
 - Photo
 - OCR
 - Database query





SQL Injection Prevention

- Do not build raw queries
 - Use secure tools and frameworks
 - Consider using abstractions
- Always validate user input on server-side
 - Reject special character
 - Escape input string, e.g., ' -> \'
 - Site-specific input (e.g., an integer or email)

Cross-Site Scripting (XSS)

JavaScript

- Widely used
 - De facto a standard programming language for web
- Development, increasing performance, libraries, ...
- Scripts (executed by browsers)
 - Can alter webpage content (DOM)
 - Track I/O events (mouse and keyboard)
 - Send/receive web requests/replies
- Same-origin policy separates code at different origins

XSS

- Webpages are not read-only and are not static
 - Users can create content (articles, comments, entries, ...)
 - JavaScript can be used to express webpages
- Idea: adversary can inject a malicious code into webpage viewed by a victim user
 - The code can access the user's page data
 - Does the same-origin policy prevent that?

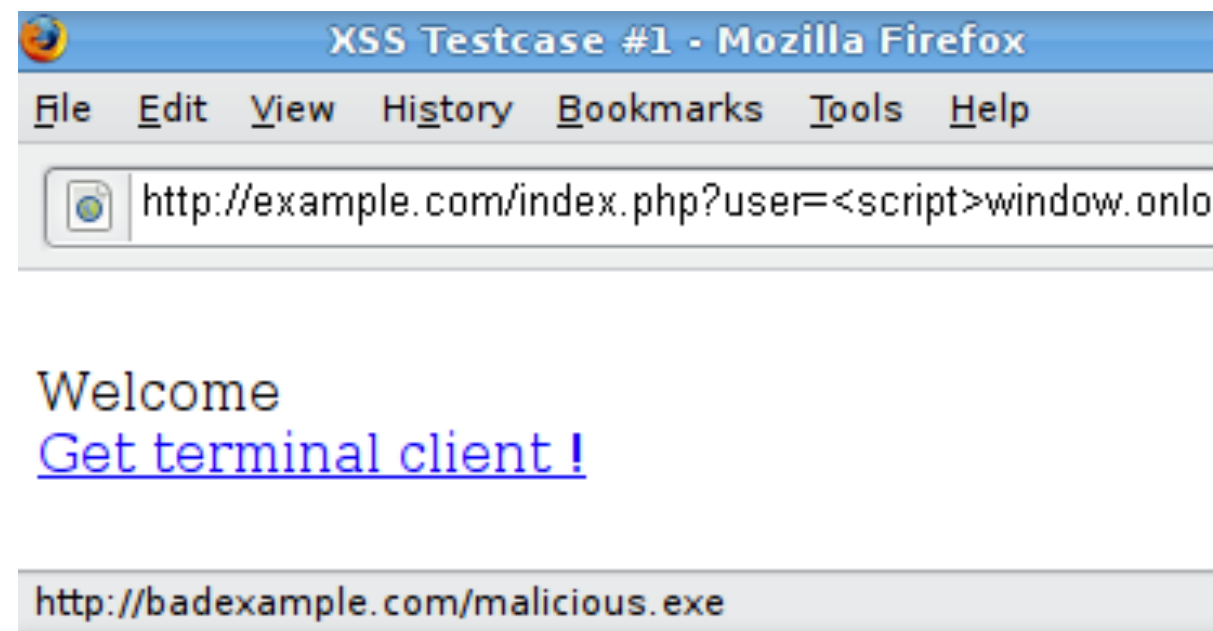
Stored XSS

- Adversary injects a malicious code at a server
- Then the server presents the stored code to next visitors
 - Code runs in the same origin (it is served by the server)
- Example



Reflected XSS

- The adversary causes the user to visit a URL for a website
 - The URL contains a malicious JS code
- The user clicks on the link and executes the code
 - Code is executed within the website's origin



XSS Prevention

- Input validation (again)
 - Whitelisting
 - Tags: <html>, <div>, <script>
 - Special characters: <, >, &, “, ‘, ...
 - Escape data before putting it into HTML
 - Secure tools & frameworks
- Content-security policy (CSP)
 - Web operator specifies scripts (domains) that are allowed to be executed on its pages
 - Inline scripts can be disallowed

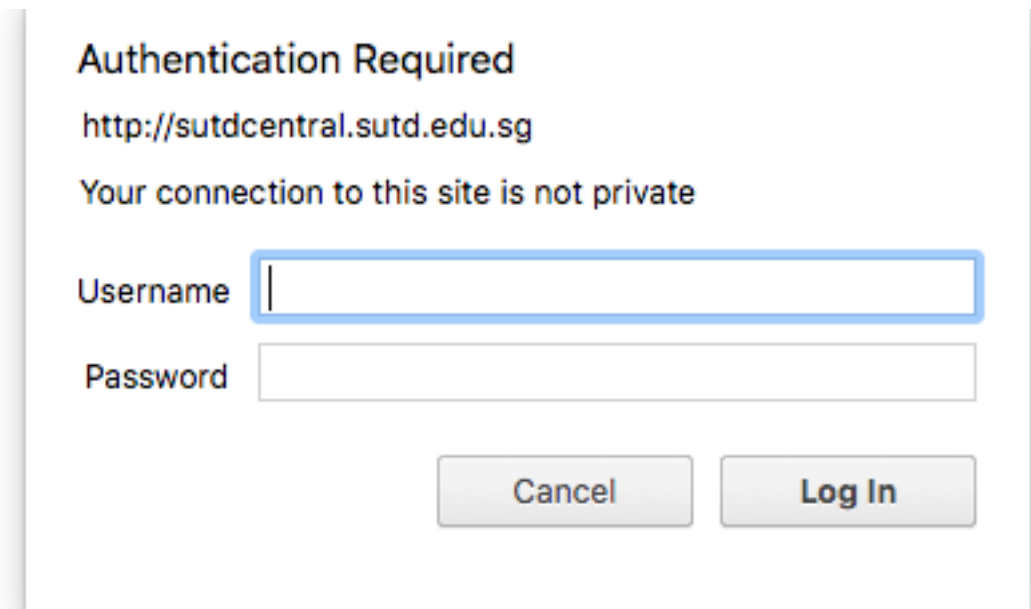
HTTP Sessions

HTTP Sessions

- HTTP is stateless
 - No built-in notion of session
 - Requests & responses
 - How to tie them to users?
- Cookies
 - Set/removed by web sites
 - Name-value pairs used to implement sessions
 - Scope

HTTP Authentication

- HTTP auth
 - Not very popular anymore (except .edu.* ;-)
 - Cannot log out, cannot customize dialog,
- Session tokens
 - Stored in cookies
 - Sent with every request
 - URLs
 - Can be leaked (HTTP Referer header)
 - Hidden forms
 - Short sessions only
 - Can be combined together



Authentication Required

http://sutdcentral.sutd.edu.sg

Your connection to this site is not private

Username

Password

Cross-Site Request Forgery (CSRF)

CSRF

- Sessions implemented via cookies
- User log-ins to a benign website and visits a malicious one
 - The malicious website contains e.g., a form that is submitted automatically to a benign website
- The browser submits this form together with its cookie
 - The form will be accepted as authentication is ok

CSRF Prevention

- Secret Validation Tokens
 - `<input type=hidden value=random>`
 - *random* value is set and checked by the server
 - *random* can be cryptographically generated for a session
- Referer header validation
 - Referer header tells which URL initiated the request
- Custom HTTP Headers
 - Different headers to browser-initiated and site-initiated requests

Questions ?