IT-Security

Chapter 5: Application Security

Part 1

- Motivation
- OWASP
- Broken Access Control
- SQL injection, cross-site scripting (XSS)
- Broken Authentication
-

Prof. Dr. Reiner Hüttl, TH Rosenheim, © 2023, 15.03.2023

What do we want to learn?

- What are the most common vulnerabilities of (web) applications?
- How do attacks with SQL injection and cross site scripting work?
- What measures can I take against attacks?
- How can I integrate security into software development from the very beginning?

Motivation

- Common errors
  - an application is planned, developed and tested without considering security
  - many security vulnerabilities are due to poor, careless and insecure programming
  - security technologies are built in (as an alibi) without considering the real threats
  - security is added later as an extension

Example: Apple‘s SSL bug: goto fail;

Faulty method:

```
SSLVerifySignedServerKeyExchange(SSLContext *ctx, bool isRsa, SSLBuffer signedParams,
                                 uint8_t *signature, UInt16 signatureLen)
{
```

faulty code snippet:

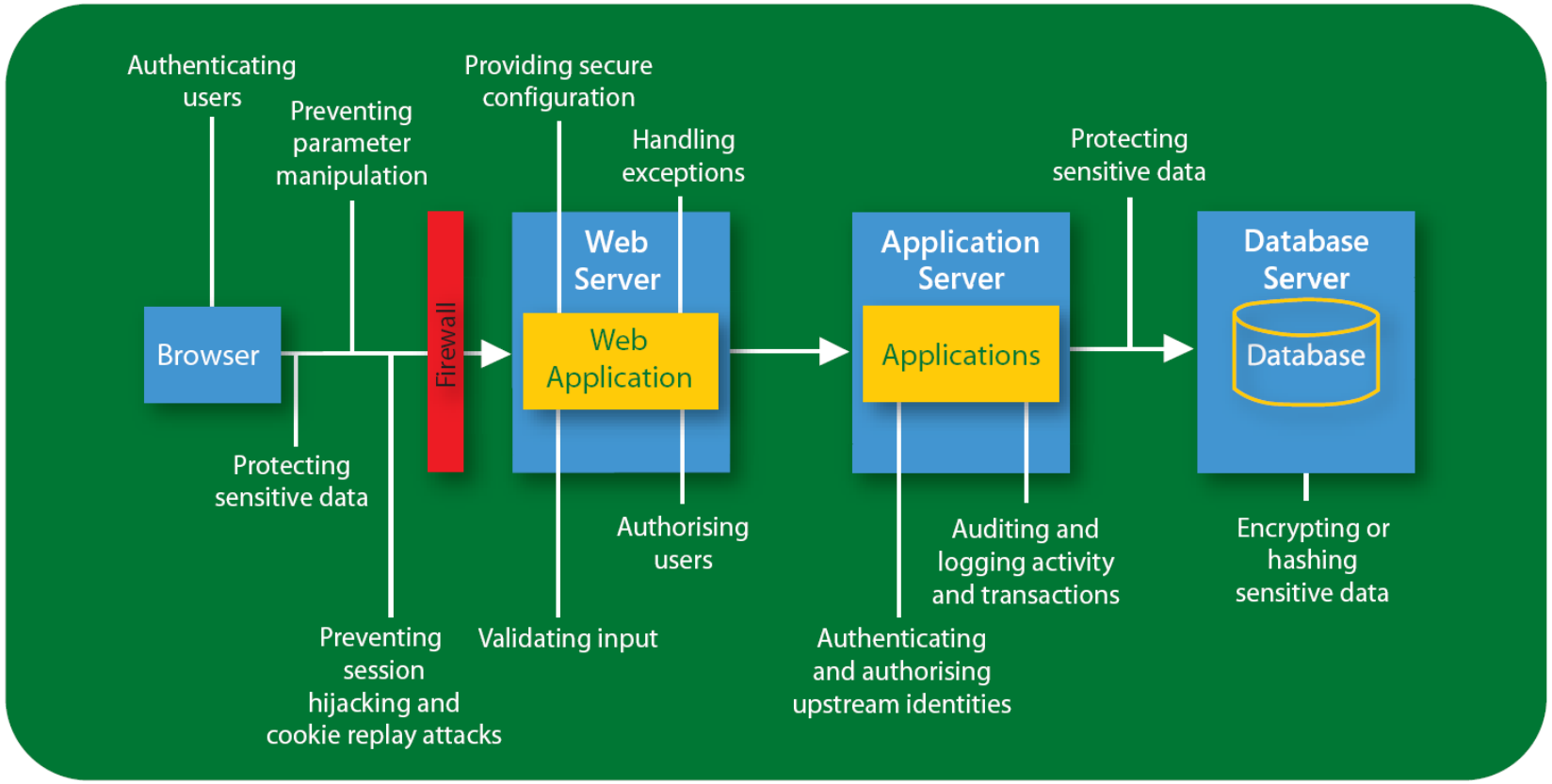
```
    if ((err = ReadyHash(&SSLHashSHA1, &hashCtx)) != 0)
        goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &clientRandom)) != 0)
        goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
        goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
        goto fail;
        goto fail;
    if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
        goto fail;

        err = sslRawVerify(ctx,
                           ctx->peerPubKey,
                           dataToSign,                  /* plaintext */
                           dataToSignLen,               /* plaintext length */
                           signature,
                           signatureLen);
        if(err) {
                sslErrorLog("SSLDecodeSignedServerKeyExchange: sslRawVerify "
                    "returned %d\n", (int)err);
                goto fail;
        }

fail:
    SSLFreeBuffer(&signedHashes);
    SSLFreeBuffer(&hashCtx);
    return err;
```

always **goto fail;**

never called

Always successful?
Not what you would expect!

Top Web Application Security Issues



From „Developer Highway Code (The drive für safer coding)“, Microsoft
http://download.microsoft.com/documents/uk/msdn/security/The%20Developer%20Highway%20Code.pdf

Top Ten Vulnerabilities in Web Applications

List from the Open Web Application Security Project (OWASP)

https://owasp.org/www-project-top-ten/

OWASP TOP 10

2017	2021
A01:2017-Injection	A03:2021-Injection
A02:2017-Broken Authentication	A07:2021-Identification and Authentication Failures
A03:2017-Sensitive Data Exposure	A02:2021-Cryptographic Failures
A04:2017-XML External Entities (XXE)	A05:2021-Security Misconfiguration
A05:2017-Broken Access Control	A01:2021-Broken Access Control
A06:2017-Security Misconfiguration	A05:2021-Security Misconfiguration
A07:2017-Cross-Site Scripting (XSS)	A03:2021-Injection
A08:2017-Insecure Deserialization	A08:2021-Software and Data Integrity Failures
A09:2017-Using Components with Known Vulnerabilities	A06:2021-Vulnerable and Outdated Components
A10:2017-Insufficient Logging & Monitoring	A09:2021-Security Logging and Monitoring Failures*
	(New) A04:2021-Insecure Design
	(New) A10:2021-Server-Side Request Forgery (SSRF)*

* From the Survey

OWASP Nr. 1: Broken Access Control

- Problem
  - Access control enforces a policy such that users cannot act outside of their intended permissions.
  - Failures typically lead to unauthorized information disclosure, modification, or destruction of all data
- Typical access control vulnerabilities
  - Metadata manipulation (such as JWT)
  - APIs with missing access controls for POST, PUT, DELETE
  - **Elevation of Privileges**
    - Vulnerabilities that allow an attacker to extends his privileges
    - **Horizontal**: access to resources of other users of the same privilege level
    - **Vertical**: access to resources of a higher privilege level (e.g., admin)

More access control vulnerabilities

- **Cross-Origin Access**
  - problem: Microservices and REST APIs often need cross-domain accesses
  - measure: **SOP Same Origin Policy**
  - **CORS (Cross Origin Resource Sharing):** Access for JavaScript to other origins controlled via http-response header
- **Direct object reference**
  - Access to objects via modification of URL (e.g., change invoice ID).
  - Measures: perform access control before granting access,
    no direct object references in URL, indirect object references are more secure.
- **Path Traversal / Force browsing**
  - Direct access to files of the web server
  - Cause: insufficient validation
  - Measures: Indirect access to filename, whitelist validation, normalization.

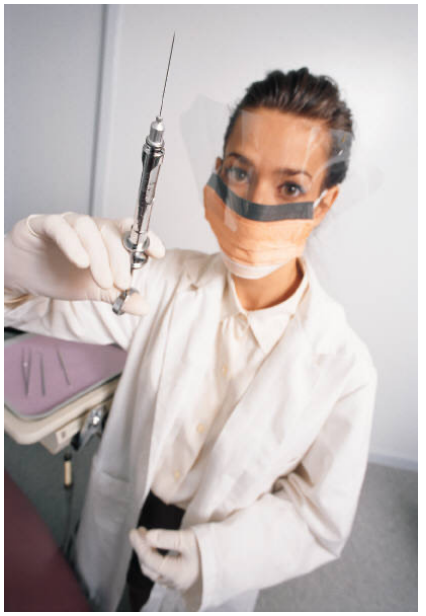
How to prevent Broken Access Control?

- The main measure is to **enforce access control** before delivering any data or function
- Design your application according to **secure access control patterns** (least privilege, need to know, deny by default, …)
- There are many measures to secure access control
  - minimize CORS usage
  - disable web server directory listing
  - rate limit APIs to protect against attack tools
  - log access control failures and alert admins
  - perform unit and integration tests for access control

OWASP Nr. 3: Injection

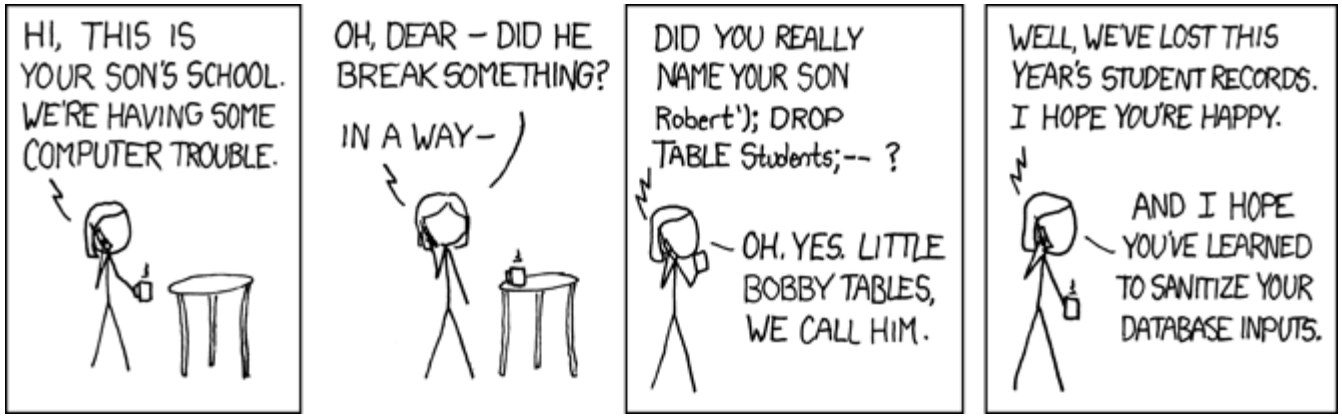
- An Application uses code interpreted at runtime
  - interpreted code is dynamically created or modified at runtime
  - user input flows directly into created code
  - examples: SQL, Shell, LDAP, XML, ...
- Protective measures against attacks
  - **Input Sanitization**: escape or remove the metacharacters
  - **Blacklisting** : only unwanted characters in input are escaped or removed
  - **Whitelisting :** only input with allowed characters will be passed through
  - use frameworks that take over protection
    e.g. LdapQueryBuilder in Spring-LDAP
  - use **Web Application Firewalls (WAF)**

SQL Injection



- Problem
  - Dynamically created database queries are manipulated by user input
- Possible damage
  - disclosure of confidential data
  - malicious modification and deletion of data
  - injecting foreign code to damage systems
  - bypassing password protection of web applications
- Occurs "only" in applications that work with SQL-based databases

Examples for SQL Injection



http://xkcd.com/327/

SELECT * FROM Usr WHERE UserName = ' **john' –** ' AND Password= ''

SELECT * FROM Usr WHERE UserName = ' **john' OR 'a' = 'b'** AND Password= ''

SELECT * FROM Customer WHERE CustId = **1; DELETE FROM Customer**

SELECT Id, Title, Abstract FROM News
WHERE Category= **1 UNION SELECT 1, UsrName, Passwd FROM Usr**

SQL Injection: provoke error messages to get internal knowledge of the database

http://www.stock.example/fund.asp?id=1+**OR+qwe=1**

Error message:
Invalid Column name qwe

SELECT * FROM news WHERE id = **1 HAVING 1=1**

Error message:
Attribute news.id must be GROUPED or used in an aggregate function

Solutions for SQL Injection (1)

- Use secure APIs
  - prevent direct access to SQL interpreter
  - offer parameterized interfaces
- Prepared statements (parameterized queries)
  - perform automatic escaping of metacharacters
  - additionally have better performance for DB queries
- Example for a prepared statement with JDBC (Java)

```
PreparedStatement ps = conn.prepareStatement("UPDATE news SET
title=? WHERE id = ?;
…
ps.setString(1, title);
ps.setInt(2, id);
int rowCount = ps.executeUpdate( );
```

Solutions for SQL Injection (2)

- Administrative measures
  - minimal rights for SQL users
  - no system accounts for database users
  - install a Web Application Firewall
  - white list input validation
- Neutralize SQL metacharacters ("data washing")
  - Important: identify all metacharacters and deactivate them
  - Examples
    - double single quotes
    - double backslash
    - encapsulate resulting strings in new quotes
    - apply functions that validate numeric input
- Example: OWASP Enterprise Security API (http://www.owasp.org/index.php/ESAPI)
  - API with databank encoder for some databases in different programming languages

Similar problems

- **Shell command injection (**OS command injection**)**
  - OS commands are called from a web application with dynamic parameters
  - allows an attacker to execute arbitrary OS commands on the server and to compromise applications and all its data and part of the hosting infrastructure

  - Example in Perl

```
$username = $form{“username”};
print ‘finger $username’;
```

Call: finger qwe; rm –rf /

  - Solutions:
    - try to get along without shell commands
    - call external programs directly (system ,exec)
    - programming the functionality or use secure APIs (e.g. mail)
    - handle metacharacters
    - prevent user input in command arguments

OWASP Nr. 3: Cross-Site-Scripting (XSS), since 2021 classified as injection problem

- Problem
  - HTML constructs injected by an attacker are passed to other users' browsers via a web application
  - JavaScript is automatically executed in the browser
  - user input to a web page is passed unfiltered as a query string and may contain malicious tags
  - XSS is a metacharacter and an output problem
- Possible consequences:
  - running malicious script in a client's browser
  - stealing users' cookies and breaking authentication
- Occurs especially in web applications where users provide the content
  - web frontends for e-mail systems and newsgroups
  - web-based discussion forums
  - social networks
  - can be combined with "social engineering

Types of XSS

- Classic classification (2005)
  - **Persistent (Stored) XSS**: the application stores unchecked user input
  - **Reflected XSS**: unchecked user input ends up directly in the HTML output again
  - **DOM Based XSS**: source of data is in the DOM, data ends up in the DOM again
- Modern classification (2012)

Data Persistence \ Where untrusted data is used	XSS	Server	Client
	Stored	Stored Server XSS	Stored Client XSS
	Reflected	Reflected Server XSS	Reflected Client XSS

- ❑ **DOM-Based XSS is a subset of Client XSS (where the data source is from the client only)**
- ❑ **Stored vs. Reflected only affects the likelihood of successful attack, not nature of vulnerability or defense**

https://owasp.org/www-community/Types_of_Cross-Site_Scripting

Measures against XSS

- Server XSS Defense
  - perform context-sensitive output encoding on the server
  - HTML encoding of untrusted data before inserting it into HTML output (HTML body, attributes, JavaScript, CSS, URL).
  - But pure **encoding** is usually not enough, because data between <script> tags, in event handlers, in CSS or in a URL can still contain XSS attacks
    - therefore, use security encoding libraries (e.g. Microsoft Anti-Cross Site Scripting Library, OWASP Java Encoder Project)
    - use frameworks that mask XSS by design (e.g. Ruby on Rails, React JS)
    - separate untrusted data from active browser content
- Client XSS Defense
  - use secure JavaScript APIs

OWASP Top 7:
Identification and Authenication Failures

- **Broken Authentication**
  - application functions related to authentication and session management are often implemented incorrectly
  - this allows attackers to compromise passwords or session tokens
  - this allows them to assume the identity of other users temporarily or permanently
- **Session management** is the basis for authentication and access control in stateful applications
- **Session-Hijacking**
  - an attacker has access to a victim's session ID and thus bypasses authentication

Measures against gegen Session-Hijacking

- **Keep session ID secret (primary measure !!)**
- **use a secure transmission channel (encrypt, e.g. https)**

- limit lifetime of session (**time out**)
- generate a new random session ID when user logs in
- do not use persistent sessions
- do not allow parallel sessions of a user (no concurrent sessions)
- use **Session Binding**: Bind session ID to client IP address or browser fingerprint
- encrypt content of authentication cookies
- set cookie attributes (secure, HttpOnly, domain, expires)
- use an Anti-CSRF token (**Synchronizer token** provides protection against replay attacks)
- include a logout option in your application

Other threats to web applications

- **Clickjacking**
  - is when an attacker uses multiple transparent or opaque layers to trick a user into clicking an invisible action
  - Countermeasure: **Frame Busting** prevents framing (set X-Frame-Options: DENY/ SAMEORIGIN)
  - **CSP Content Security Policy:** declare in HTTP response header which dynamic resources, such as JavaScript, CSS, the browser is allowed to load
    e.g. do not allow JavaScript inline but only in files, content only type text/json
    https://content-security-policy.com

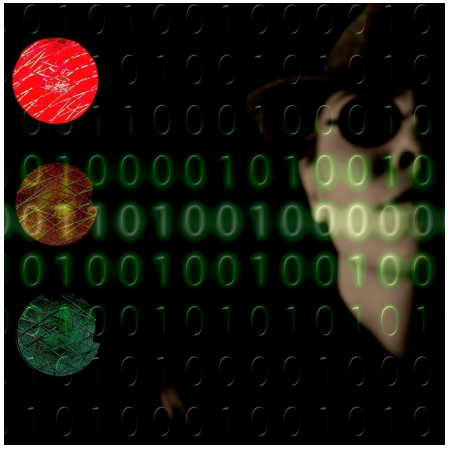
- **Replay-Attacken**
  - replay request by an attacker or accidentally by a user
  - Measure: Anti-Replay Token, Anti-CSRF Token

more threats to web applications

- **Remote Code Execution (RCE)**
  - attackers can run code on the target machines via a vulnerability
  - e.g. command injection, path traversal and execution of files, exploitation of vulnerabilities in the network or IT systems
  - RCE is absolute worst case!

  - Measures: Defense in Depth, Updating all Components and libraries

DOS Denial of Service



- DOS cannot be prevented
- An **emergency plan** enables you to react appropriate in the event of a DoS and reduces the damage
- There are different variants of DOS: Application crash, OS crash, CPU, memory or resource overload.
- Measures to reduce vulnerability
  - write solid code
  - perform testing and performance analysis with profiler
  - perform monitoring to detect changes of application behavior
  - do not trust any data over the network
  - allow complex operations only if you are sure that a trusted client is sitting at the other end
  - request resources as late as possible and release them as early as possible
  - validate requests before sending error messages and implement a controlled error handling
  - disconnect clients without authentication when attacked