FH JOANNEUM
University of Applied Sciences

Threats and Mitigations

For example:
CSRF/XSRF, XSS, CORS, Session Hijacking, Cookie Poisoning, URL/FormField Tampering, SQL/Code Injection

John Feiner

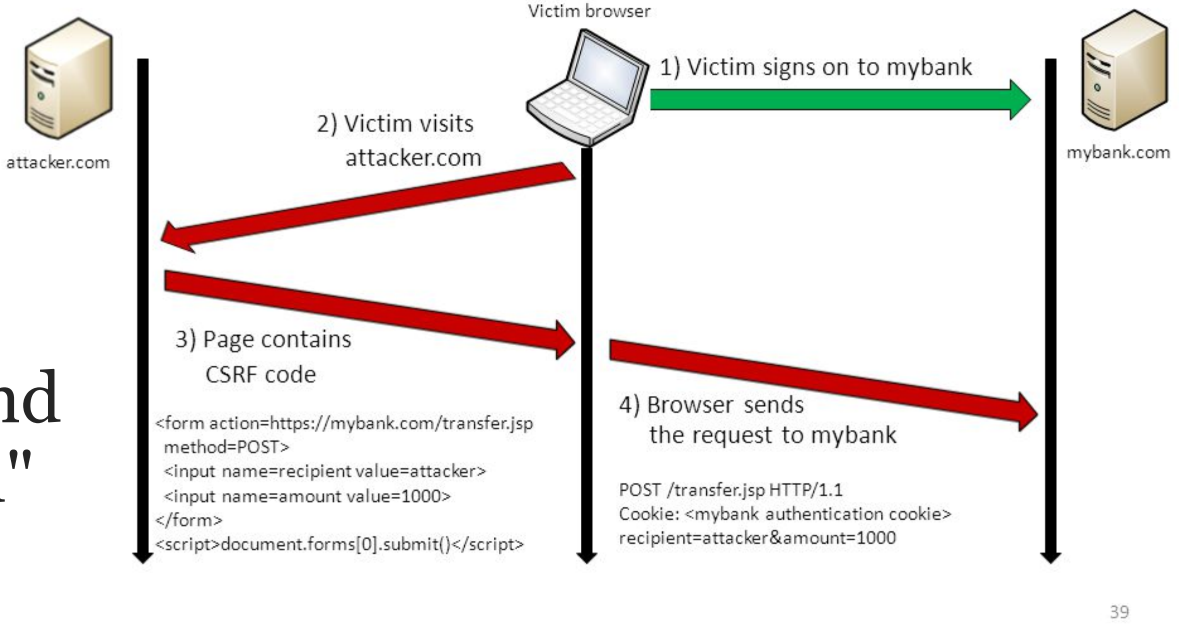
Sec: XSRF

?

Sec: XSRF = CSRF

Cross-Site Request Forgery (XSRF/ CSRF/'sea-surf')

malicious website send command to "trusted" user


Victim browser
attacker.com
mybank.com
1) Victim signs on to mybank
2) Victim visits attacker.com
3) Page contains CSRF code
4) Browser sends the request to mybank
39

```
<form action=https://mybank.com/transfer.jsp
 method=POST>
 <input name=recipient value=attacker>
 <input name=amount value=1000>
</form>
<script>document.forms[0].submit()</script>
```

```
POST /transfer.jsp HTTP/1.1
Cookie: <mybank authentication cookie>
recipient=attacker&amount=1000
```

https://images.app.goo.gl/JeLFn9RifviCoRRaA

Sec: XSRF/CSRF - Mitigation

Prevent browser from sending "wrong" cookies

- user same site cookie attribute

  https://tools.ietf.org/html/draft-ietf-httpbis-rfc6265bis-02#section-5.3.7

CSRF Tokens

- Generated on server, secret, unpredictable, optional: new token for each request

Note: CORS will NOT prevent CSRF

Sec: XSS

?

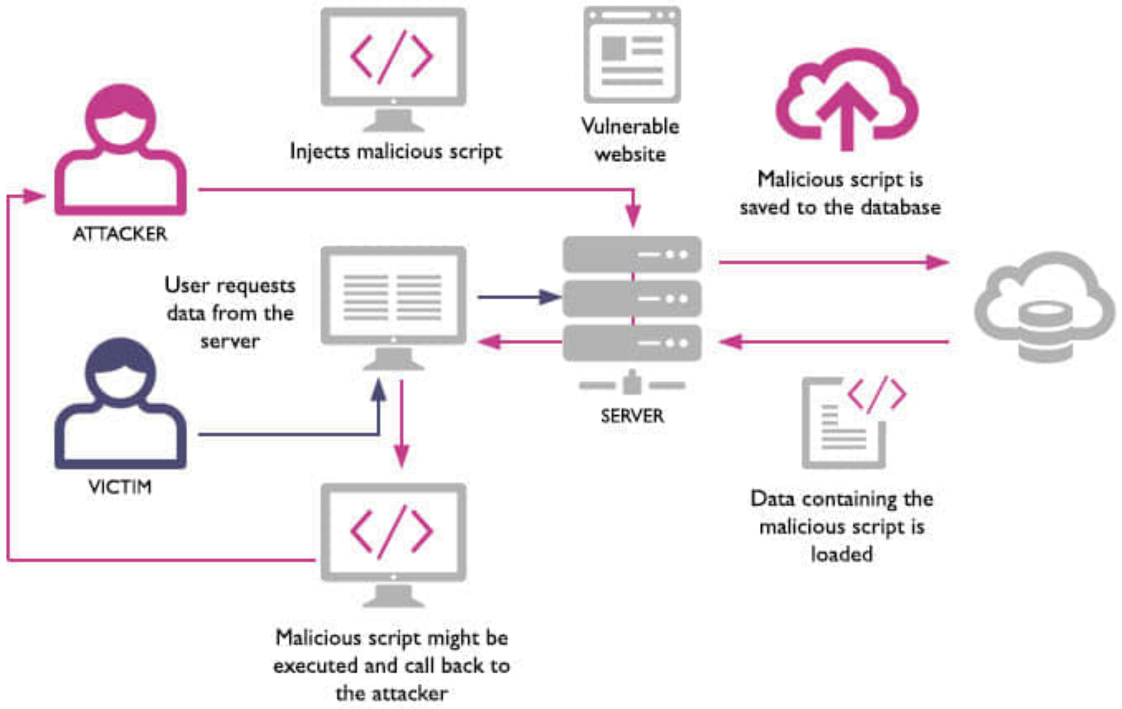
Sec: XSS

Cross-site scripting

Inject malicious script in trusted web app

For example,

website does not validate input


ATTACKER
Injects malicious script
Vulnerable website
Malicious script is saved to the database
User requests data from the server
SERVER
VICTIM
Data containing the malicious script is loaded
Malicious script might be executed and call back to the attacker

https://images.app.goo.gl/zmdwt2faTLqTKqUy9

Sec: XSS - Mitigation

Client data = untrusted data
- it could be code (JS)

Never(!) input untrusted data
- (as tag, as style, as ...)

Input validation:
- whitelists
- escape special characters

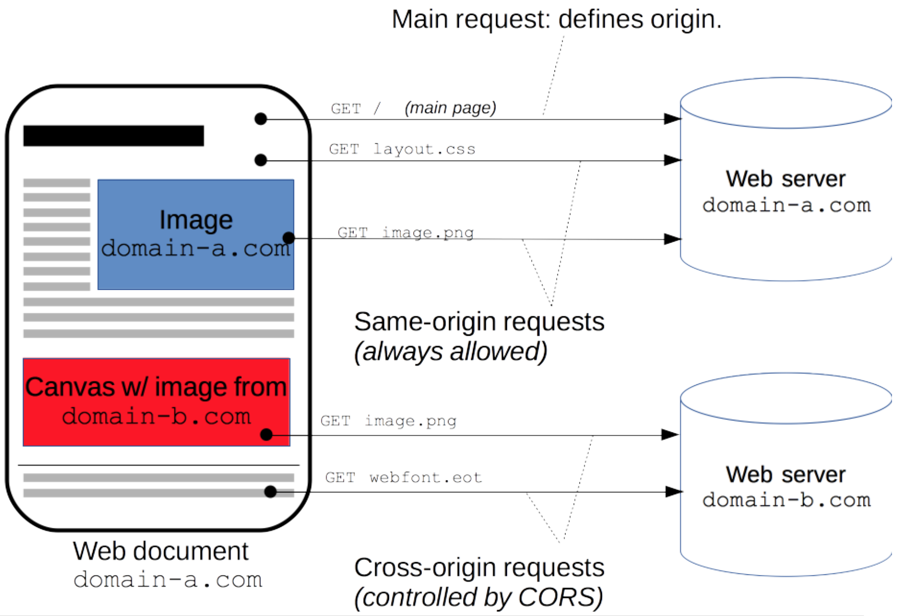
Sec: CORS

?

Sec: CORS

Cross-Origin Resource Sharing

Configure server to add HTTP header to tell browser to allow access to resources (despite coming from different origin)



Sec: CORS - Mitigation

Take care when relaxing SOP (Same Origin Policy):

Never allow "from any domain"

Double-check CORS configurations:

Configuration is on the external server. We intentionally allow access to resources outside given (sub)domain. Limit trusted 3rd party with strict whitelists, check pattern matching of domains.

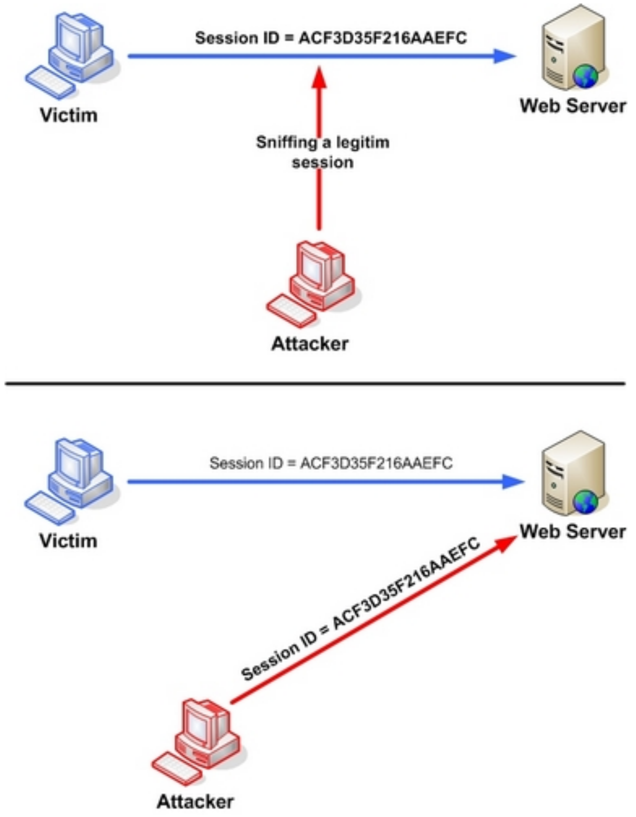
Sec: Session Hijacking

?

Sec: Session Hijacking

Session hijacking / cookie hijacking
    uses exploited session key(s)

For example:
    steal cookie (packet sniffing),
    session fixation,
    cross-site scripting,...


Session ID = ACF3D35F216AAEFC
Victim
Web Server
Sniffing a legitim session
Attacker
Session ID = ACF3D35F216AAEFC
Victim
Web Server
Session ID = ACF3D35F216AAEFC
Attacker

https://images.app.goo.gl/jJxdiLB6nGoBRSux6

Sec: Session Hijacking - Mitigation

Prevent sniffing
- (e.g. MitM Man in the Middle, Man in the Browser)
- Encryption (TLS), Fingerprint requests

Session keys:
- Long keys, random keys (no chance to guess),
- Short life-time of keys

Prevent XSS:
- Validate data, escape data, user HTTPOnly cookies
- Content security policy (client whitelist for resources)

Sec: Cookie Poisoning

?

Sec: Cookie Poisoning

Attacker

- modifies session cookie
- then impersonate a valid client

Alternatives:

- client side
- man-in-the-middle

See also: Session hijacking, session fixation

https://www.acunetix.com/blog/web-security-zone/what-is-cookie-poisoning/

Sec: Cookie Poisoning - Mitigation

HSTS

- HTTP Strict Transport Security
- Browser refuses any HTTP connection (e.g. SSL Stripping) for two years

Sec: Cookie Poisoning - Mitigation

HSTS - Example

Server sends to client
(in header of first https response)

```
Strict-Transport-Security: max-age=31536000; includeSubDomains; preload
```

Sec: URL Tampering

?

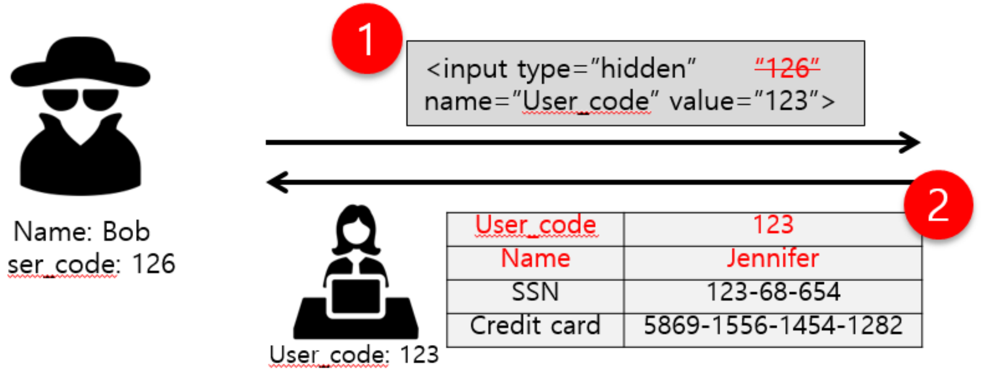
Sec: URL Tampering

Attacker

modifies url parameters

For example:

modify ids in query strings


1
<input type="hidden" ~~"126"~~ name="User_code" value="123">
2
Name: Bob
ser_code: 126
User_code: 123

User_code	123
Name	Jennifer
SSN	123-68-654
Credit card	5869-1556-1454-1282

https://medium.com/@chawdamrunal/what-is-parameter-tampering-5b1beb12c5ba

Sec: URL Tampering - Mitigation

POST data (or in Headers, or in Cookies)

Validate input data (types, allowed chars)

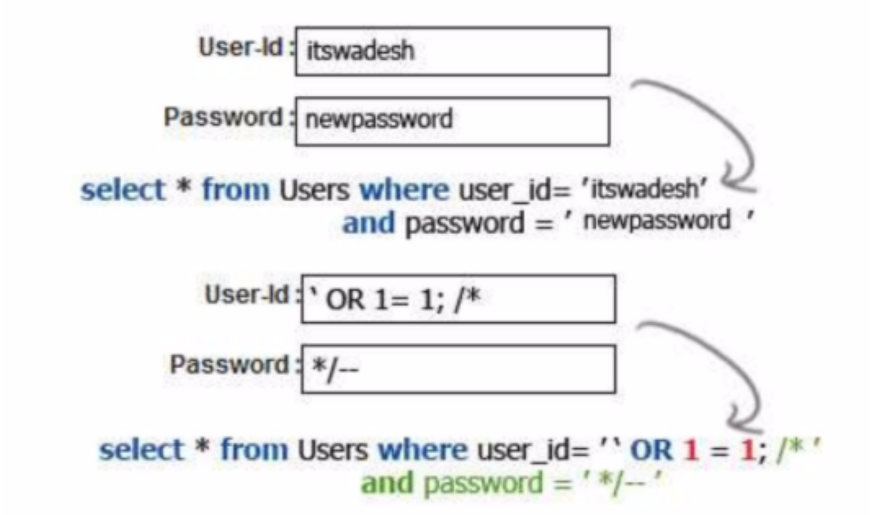
Sec: SQL Injection

?

Sec: SQL Injection

Inject commands in SQL queries

See: prepared statements



```
select * from Users where user_id= 'itswadesh'
    and password = ' newpassword '

select * from Users where user_id= '' OR 1 = 1; /* '
    and password = ' */-- '
```

Sec: SQL Injection - Mitigation

Database:

Use prepared statement only!

Others:

Escape data

Whitelist data

Use stored procedures

Sec: SQL Injection - Mitigation

Prepared statement example:

```
// This should REALLY be validated too
String custname = request.getParameter("customerName");
// Perform input validation to detect attacks
String query = "SELECT account_balance FROM user_data WHERE user_name = ? ";
PreparedStatement pstmt = connection.prepareStatement( query );
pstmt.setString( 1, custname);
ResultSet results = pstmt.executeQuery( );
```

https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html

Sec: Code Injection

?

Sec: Code Injection

User code interpreted by web application / web server

```php
// A dangerous way to use the eval() function
// in PHP
$myvar = "varname";
$x = $_GET['arg'];
eval("\$myvar = \$x;");
```

Sec: Code Injection - Mitigation

Do NOT allow untrusted data in

- XPath, JavaScript code, external (OS) commands, ...

Others = "Input Validation":

- Escape data
- Whitelist data

OWASP

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

OWASP = Open Web Application Security Project ... Still relevant?

https://cheatsheetseries.owasp.org

OWASP Top 10

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

1. **Injection**. Injection flaws, such as SQL, NoSQL, OS, and LDAP injection, occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization.
2. **Broken Authentication**. Application functions related to authentication and session management are often implemented incorrectly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users' identities temporarily or permanently.
3. **Sensitive Data Exposure**. Many web applications and APIs do not properly protect sensitive data, such as financial, healthcare, and PII. Attackers may steal or modify such weakly protected data to conduct credit card fraud, identity theft, or other crimes. Sensitive data may be compromised without extra protection, such as encryption at rest or in transit, and requires special precautions when exchanged with the browser.
4. **XML External Entities (XXE)**. Many older or poorly configured XML processors evaluate external entity references within XML documents. External entities can be used to disclose internal files using the file URI handler, internal file shares, internal port scanning, remote code execution, and denial of service attacks.
5. **Broken Access Control**. Restrictions on what authenticated users are allowed to do are often not properly enforced. Attackers can exploit these flaws to access unauthorized functionality and/or data, such as access other users' accounts, view sensitive files, modify other users' data, change access rights, etc.

OWASP Top 10

6. **Security Misconfiguration**. Security misconfiguration is the most commonly seen issue. This is commonly a result of insecure default configurations, incomplete or ad hoc configurations, open cloud storage, misconfigured HTTP headers, and verbose error messages containing sensitive information. Not only must all operating systems, frameworks, libraries, and applications be securely configured, but they must be patched/upgraded in a timely fashion.
7. **Cross-Site Scripting XSS**. XSS flaws occur whenever an application includes untrusted data in a new web page without proper validation or escaping, or updates an existing web page with user-supplied data using a browser API that can create HTML or JavaScript. XSS allows attackers to execute scripts in the victim’s browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites.
8. **Insecure Deserialization**. Insecure deserialization often leads to remote code execution. Even if deserialization flaws do not result in remote code execution, they can be used to perform attacks, including replay attacks, injection attacks, and privilege escalation attacks.
9. **Using Components with Known Vulnerabilities**. Components, such as libraries, frameworks, and other software modules, run with the same privileges as the application. If a vulnerable component is exploited, such an attack can facilitate serious data loss or server takeover. Applications and APIs using components with known vulnerabilities may undermine application defenses and enable various attacks and impacts.
10. **Insufficient Logging & Monitoring**. Insufficient logging and monitoring, coupled with missing or ineffective integration with incident response, allows attackers to further attack systems, maintain persistence, pivot to more systems, and tamper, extract, or destroy data. Most breach studies show time to detect a breach is over 200 days, typically detected by external parties rather than internal processes or monitoring.