Securing Web Applications from Top Ten Security Threats

OWASP Top Ten (2017 Edition)
(Open Web Application Security Project)

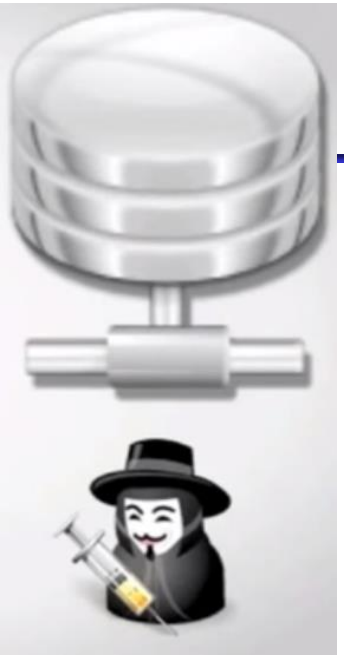
1. Injection
2. Broker Authentication
3. Sensitive Data Exposure
4. XML External Entities (XXE)
5. Broken Access Control
6. Security Misconfiguration
7. Cross-Site Script (XSS)
8. Insecure Deserialization
9. Using Components with Known Vulnerabilities
10. Insufficient Logging and Monitoring

More details @
https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf

OWASP Top 10 Application Security Risks – 2017

A1: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.
A2: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.
A3:Sensitive Data Exposure	Many web applications and APIs do not properly protect sensitive data, such as financial, and healthcare data. 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.
A4: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 port scanning, remote code execution, and denial of service attacks.
A5: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.
A6:Security Misconfiguration	Security misconfiguration is the most commonly seen issue. This is commonly a result of insecure default configurations, incomplete or ad hoc configurations, 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 and upgraded in a timely fashion.
A7: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.
A8: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.
A9: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.
A10: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.

1. Injection

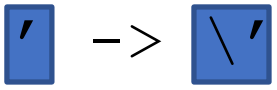


- Executing code provided (injected) by attacker

  e.g., SQL injection

username: admin
password: unknown' or '1'='1

- Solutions:
  - validate user input
  - escape values (use escape functions)
  - use parameterized queries (SQL)
  - enforce least privilege when accessing a DB, OS etc.



2. Broken Authentication

Hacker uses flaws in the authentication or session management functions to **impersonate** other users:

- **Scenario #1:** Session timeout isn't set properly. Instead of selecting "logout" the user simply closes the browser tab and walks away. Attacker uses the same browser an hour later, and that browser is still authenticated.
- **Scenario #2:** Session hijacking by stealing session id (e.g., using eavesdropping if not https) then hacker uses this authenticated session id to connect to the application without needing to enter user name and password.
- **Scenario #3:** Hacker gains access to the system's password database. User passwords are not properly hashed, exposing every users' password to the attacker.
- **Scenario #4:** Brute force password guessing, more likely using tools that generate random passwords.

2. Broken Authentication - Solutions

- Session IDs should **timeout**. User sessions or authentication tokens should get properly invalidated during logout.
- User authentication credentials should be **protected when stored** using hashing or encryption.
- Session IDs should not be exposed in the URL, **use cookies for storing session ID**
- Session IDs should be regenerated after successful login.
- Passwords, session IDs, and other credentials should not be sent over unencrypted connections (**use https** for login)
- Enforce **minimum passwords length and complexity** makes a password difficult to guess
- Enforce **account disabling** after an established number of invalid login attempts (e.g., three attempts is common).

3. Sensitive Data Exposure

Storing and transmitting sensitive data insecurely

- Failure to identify all sensitive data
- Failure to identify all the places that this sensitive data gets stored
  - Databases, files, directories, log files, backups, etc.
- Failure to identify all the places that this sensitive data is sent
  - On the web, to backend databases, to business partners, internal communications
- Failure to properly protect this data in every location

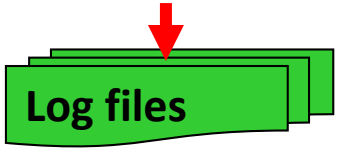
Typical Impact

- Attackers access or modify confidential or private information
  - e.g, credit cards, health care records, financial data (yours or your customers)
- Attackers extract secrets to use in additional attacks
- Company embarrassment, customer dissatisfaction, and loss of trust
- Expense of cleaning up the incident, such as forensics, sending apology letters, reissuing thousands of credit cards, providing identity theft insurance
- Business gets sued and/or fined

7

3. Sensitive Data Exposure – Example

1 Victim enters credit card number in form

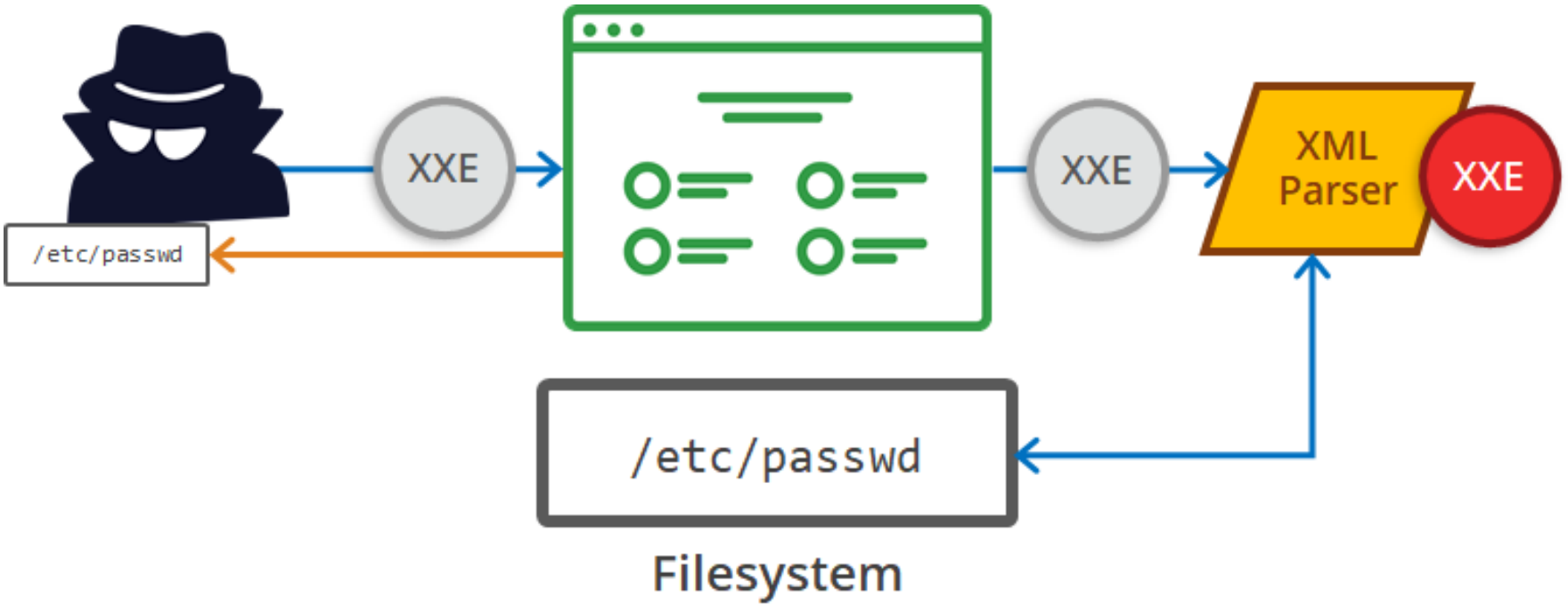


2 Error handler logs credit card details because merchant gateway is unavailable

3 Logs are accessible to all members of IT staff for debugging purposes

4 Malicious insider steals credit card numbers

4. XML External Entities (XXE)



- The application accepts XML documents directly with malicious XML data, which is then parsed by an XML processor that has document type definitions (DTDs) enabled.

Scenario: An attacker attempts a denial-of-service attack by including a potentially endless file:

<!ENTITY xxe SYSTEM "file:///dev/random" >]>

XXE Example

Request

```
POST http://example.com/xml HTTP/1.1

<!DOCTYPE foo [
  <!ELEMENT foo ANY>
  <!ENTITY bar SYSTEM
  "file:///etc/lsb-release">
]>
<foo>
  &bar;
</foo>
```

Response

```
HTTP/1.0 200 OK

DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=16.04
DISTRIB_CODENAME=xenial
DISTRIB_DESCRIPTION="Ubuntu 16.04 LTS"
```

Scenario #1: The attacker attempts to extract data from the server:

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

XXE Example

Request

```
POST http://example.com/xml HTTP/1.1

<!DOCTYPE foo [
  <!ELEMENT foo ANY>
  <!ENTITY bar SYSTEM
  "http://192.168.0.1/secret.txt">
]>
<foo>
  &bar;
</foo>
```

Response

```
HTTP/1.0 200 OK

Hello, I'm a file on the local network (behin
d the firewall)
```

Scenario #2: An attacker probes the server's private network:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
  <!DOCTYPE foo [
  <!ELEMENT foo ANY >
<!ENTITY xxe SYSTEM "https://192.168.1.1/private" >]>
  <foo>&xxe;</foo>
```

5. Broken Access Control

Insecure Direct Object References

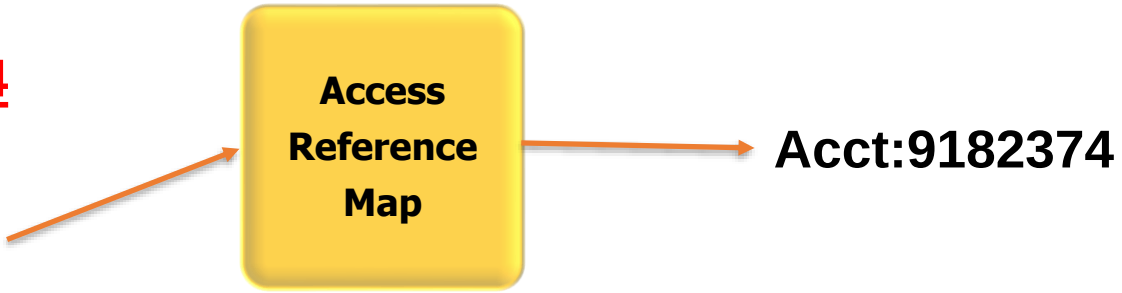
- Attacker manipulates the URL or form values to get unauthorized access
  - to objects (data in a database, objects in memory etc.):
    `http://shop.com/cart?id=`**`413246`** (your cart)
    `http://shop.com/cart?id=`**`123456`** (someone else's cart ?)
  - to files:
    `http://s.ch/?page=`**`home`** -> **`home`**
    `http://s.ch/?page=`**`/etc/passwd`** -> **`/etc/passwd`**
- Solution:
  - avoid exposing IDs, keys, filenames to users if possible
  - validate input, accept only correct values
  - verify authorization to all accessed objects (files, data etc.)

5. Broken Access Control

- Eliminate the direct object reference
  - Replace them with a temporary mapping value such as a random mapping

Instead of :
http://app?id=9182374

Use:
http://app?id=7d3J93



- Validate the direct object reference
  - Verify the user is allowed to access the target object
  - Verify the requested mode of access is allowed to the target object (e.g., read, write, delete)

5. Broken Access Control

Missing Function Level Access Control

- Often Web applications verify function level access rights before making that functionality visible in the UI. However, applications need to **perform the same access control checks on the server when each function is accessed**.
- "Hidden" URLs that don't require further authorization

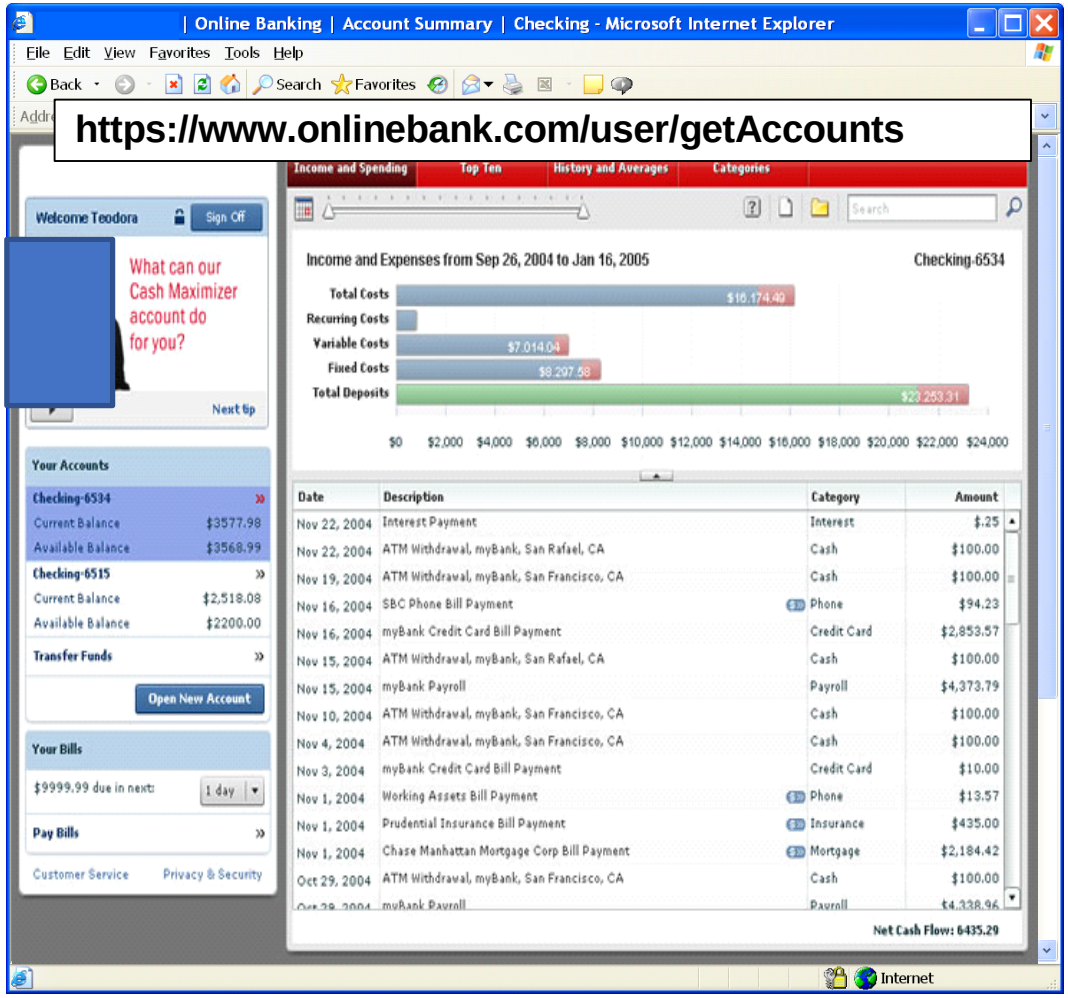
  `http://site.com/`**`admin/adduser?name=x&pwd=x`**

  (even if `http://site.com/`**`admin/`** requires authorization)

- Solution
  - Add missing authorization ☺
  - Don't rely on **security by obscurity** – it will not work!

5. Broken Access Control

Missing Function Level Access Control Illustrated



- Attacker notices the URL indicates his role

  /user/getAccounts

- He modifies it to another directory (role)

  /admin/getAccounts, or

  /manager/getAccounts

- Attacker views more accounts than just their own

6 – Security Misconfiguration

- Hackers can take advantage of poor server configuration (e.g., default accounts, unpatched flaws, unprotected files and directories) to gain unauthorized access to application functionality or data
- Security misconfiguration can happen at any level of an application stack, including the platform, web server, application server, database, framework, and custom code.

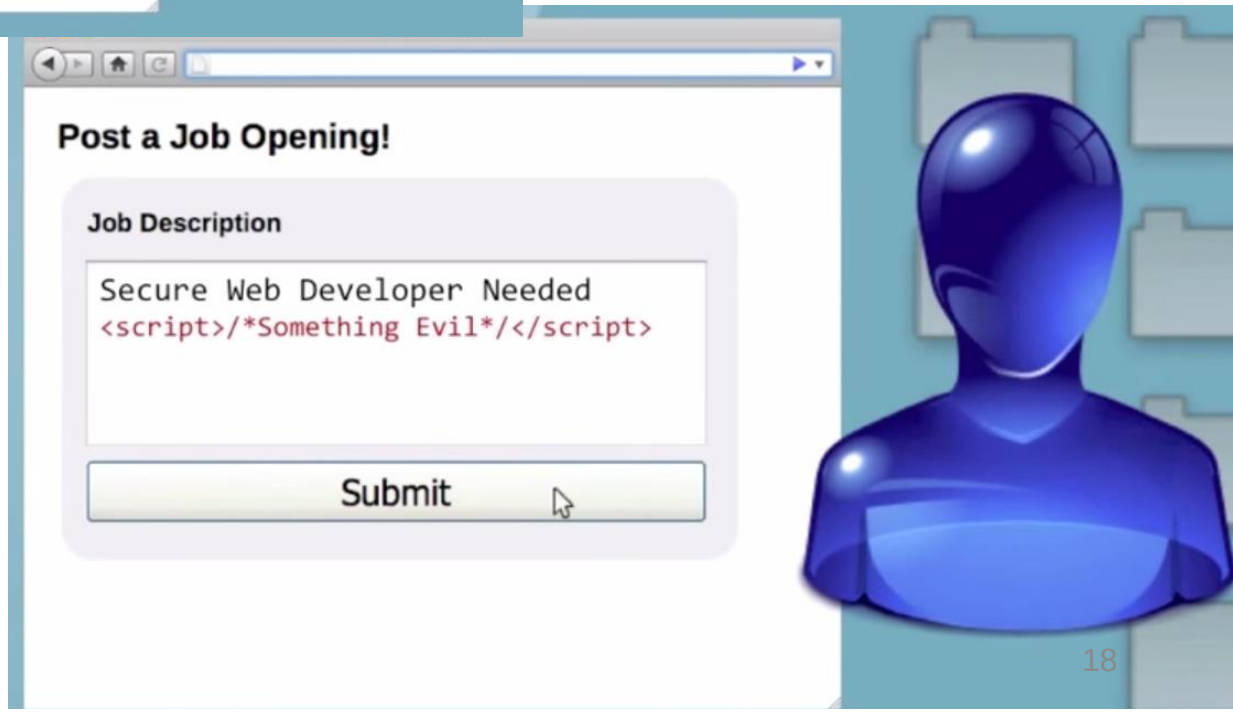
Solution

- Verify your system's configuration by scanning to find misconfiguration or missing patches
- Secure configuration by "**hardening**" the servers:
  - Disable unnecessary packages, accounts, processes & services.
  - Disable default accounts
  - Patch OS, Web server, and Web applications
  - Run Web server user a regular (non-privileged) user

7. Cross-site scripting (XSS)

- Cross-site scripting (XSS) vulnerability
  - An application takes user input and sends it to a Web server without validation or encoding
  - Attacker can execute JavaScript code in the victim's browser
  - To hijack user sessions, deface web sites etc.
- **Solution**:
  - Validate user input, encode HTML output


Mozilla Firefox
New Job Posting!
Static Content
Job Description
Secure Web Developers Needed!
User Supplied Content
Apply Now
Save Job


Post a Job Opening!
Job Description
Secure Web Developer Needed
<script>/*Something Evil*/</script>
Submit

Visitors will get the evil script


Mozilla Firefox
New Job Posting!
Job Description
Secure web develo
lol!
Apply Now
ATTACKERS CAN USE JAVASCRIPT TO....
STEAL YOUR SESSION ID:
document.cookie
REWRITE ANY PART OF THE PAGE


https://www.yourbanksite.com
YourBankSite
Please Log in
User Name:
Password:
LOGIN
ATTACKERS CAN USE JAVASCRIPT TO....
OVERLAY THE LOGIN SCREEN WITH THEIR OWN, ALLOWING ATTACKS TO HARVEST USERNAMES AND PASSWORDS

8. Insecure Deserialization

- Applications will be vulnerable if they deserialize hostile or tampered objects supplied by an attacker:
  - Insecure Deserialization is about abusing trust developers have in objects that are often not considered as dangerous as user input

- This can result in two primary types of attacks:
  - Data tampering attacks: e.g., object content is changed by the attacker to ***elevate their privileges***.
  - Ar
  - e.g.

```
{"orderLines" : "(function dos() { while(true); })()"}
```

- **Solution:**
  - Implementing integrity checks such as digital signatures on any serialized objects to prevent data tampering.
  - Enforcing strict type constraints during deserialization (i.e., validate the datatype of object properties)

Insecure Deserialization Example

An attacker could inject a SQL Injection payload directly into the serialized object

```
{"Username":"'or'1'='1","TimeSpent":"00:43:01","Score":"1445"}
```

Deserialization of...

Username	' or '1='1
TimeSpent	00:43:01
Score	1445

```
"SELECT score FROM highscore WHERE user = '" + gamestate.username + "';"
```

```
"SELECT score FROM highscore WHERE user = '' or '1='1';"
```

9. Using Components with Known Vulnerabilities

Vulnerable Components Are Common

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

Widespread

- Virtually every application has these issues because most development teams don't focus on ensuring their components/libraries are up to date

Typical Impact

- Full range of weaknesses is possible, including injection, broken access control, XSS ...
- The impact could range from minimal to complete host takeover and data compromise

10. Insufficient Logging & Monitoring

- Insufficient logging, detection, monitoring and active response to attacks
  - Auditable events, such as logins, failed logins, and high-value transactions are not logged.
  - Warnings and errors generate no, inadequate, or unclear log messages.
  - Logs of applications and APIs are not monitored for suspicious activity.
  - Logs are only stored locally and not consolidated and analyzed
  - Appropriate alerting thresholds and response escalation processes are not in place or effective.
- Penetration testing and scans by do not trigger alerts.
  - The application is unable to detect, escalate, or alert for active attacks in real time or near real time.

Solution

- Establish continuous application security testing and monitoring

Insufficient Logging & Monitoring Examples

- If the attacker has installed a crypto miner on the server, the CPU load for will increase drastically

=> By constantly monitoring the CPU usage on the server, such activities can be detected and appropriate action taken

- If the attacker starts downloading and steal a lot of data from the server, this will show up as an unusual amount of outgoing network traffic

=> By monitoring the network traffic, it is possible to detect such data extraction and thereafter prevent it

- In really however, most breaches are first discovered after 200 days... this makes it clear that many are not effectively using logging and monitoring
  - Insufficient Logging and Monitoring can facilitate malicious activities, delay breach detection and incident response and makes digital forensics harder to know what the attacker has done and to recover

=> If a breach is detected earlier, it can be stopped before the attacker managers to escalate it, greatly minimizing the damage.

Web Application Penetration Testing

Tools to scan Web App for vulnerabilities:

- Portswigger Burp Suite

  https://portswigger.net/

- OWASP Zed Attack Proxy (ZAP)
  https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project

Vulnerable Apps to play with

- https://github.com/OWASP/NodeGoat
- https://github.com/bkimminich/juice-shop

Summary

- Understand threats and typical attacks
- Validate, validate, validate (!) user input before using it
  - Do not trust the client input
- Avoid the misconception of **security by obscurity**
- Protect sensitive data in Storage and in Transit
- Read and follow recommendations for your development platform
- Use web vulnerability scanning tools
  https://www.owasp.org/index.php/Category:Vulnerability_Scanning_Tools
- Harden the Web server and development platform configuration