Chapter 1 Fundamentals

- Vulnerability is a quality defect based on bad coding practices
- Exploit is a piece of code that make use of a vulnerability
- Identification is the process of presenting an identity to a system
- Authentication is the process of actually proving who you are (password)
- Authorization is what the user is allowed to do. (role)
- Weakest link in IT other than human is Web Application

Hackers profile

- Script kiddies
    - All about tools, reputation and wreaking havoc
- Hackivists (Loosely organized groups with political motivations)
    - Dos, Defacements, Information Theft
- Organized Criminal Gangs (Money)
- Nation States
    - Intelligence and certain objectives
    - Targeted, big budget and almost unlimited resources

Attacker

- Attacker can choose weakest point
- Attacker can probe for unknown vulnerabilities
- Attacker can strike at will
- Attacker can play dirty

Defender

- Defender must defend all points
- Defender can only defend against known attacks
- Defender must be constantly vigilant
- Defender must play by the rules

Business Impact of being hacked

- Compliance issues (Legal and regulatory)
- Data theft, deletion or modification
- Loss of revenues
- Legal problems
- Damage to credibility
- Loss of customer trust

Notes

- Encryption only protects from eavesdropping by third parties
- Firewall does not protect the web application. It only blocks traffic on other “non-web ports”

- Nessus and other vulnerability scanners only recognize threat signatures for certain, well-defined configurations
- IDS will only recognize threat signatures for certain, well-defined configurations
- Configuring an application firewall correctly is at least as difficult as programming the application security in the first place
- 10% of security is achieved by Security Features
- 90% of security is achieved by Securing Features (Secure Coding)
- Security is only achieved by combining both Security and Securing Features

Security Features

- HTTPS-Encryption
- Firewall

Secure Features

- Hard filters that validate user input to prevent XSS
- Hard sanitization of output to prevent XSS
- Correct usage of prepared statements to prevent SQLi

Security Test Approaches

- Only a Source Code Review and Test can identify more than 50% of the security vulnerabilities in software.

Approach	Knowledge of the auditors	Coverage degree
Blackbox Pentest	• No detailed knowledge about the target system • Identical to a hacker attack	• Less than 10% of all vulnerabilities are usually found.
Glassbox (Partial Source Code Assessment)	• Knowledge about the target system and/or • Administrative access to the target system, and thereby access to logs and internal system information • Knowledge about the target architecture • Partial analysis of the source code	• Up to 25% of all vulnerabilities can be found.
Security Source Code Review	• Manual review of the source code and verification of the vulnerabilities found on a test system	• Almost all (95%) vulnerabilities can be found, to which attack vectors currently exist.

Chapter 2 Input Validation & Output Sanitization

User Input

- Never trust user input
- User input is everything that is provided by any external system which is not under the sole control of the application.
- User do not care if the input is valid

    - For example entering phone number without area code
- User may make error
    - Entering 0 instead of O
- User might want to trick/hack you
    - User enters a negative item count in a web-shop
- Worst Case scenario of faulty input validation
    - Injection attacks
        - SQLi
        - OS command injection
        - XML injection
    - Cross-Site Scripting (XSS)

Application Output

- Never trust application output
- Output refers to everything that is written to a file, stream or any other location with the intention to be interpreted in a separate step
- Sanitizing is modifying the output in a way that user supplied input (any data outside of our control sphere) is not interpreted in an undesired way.
- Application might return dangerous output
    - Disclose information
    - Attack the client (XSS)
- Worst Case Scenario
    - Cross-Site Scripting (XSS)
    - Information Disclosure
    - Various Other Attacks
    - XML injections
    - JSON injections

Secure Application

- Valid input in our application (Input Validation)
- Return clean output to the user (Output Sanitization)

Input Validation

- Types of input validation
    - Regular Expressions
    - Type Checks
    - Whitelists
    - Blacklists (Never ever use them)
- Validate as soon as possible
    - The longer the application works with the unvalidated data, the higher the risk of exploitation
- Validate as strict as possible
- Validate as much as possible

- Validate on the server side always
    - If user supplied data is processed or saved by the application
- Validate On the client side
    - If user supplied data is further processed by Javascript (DOM based XSS)

Output Sanitization

- Types of output sanitization
    - Filters that modifies the output before it's written to the user
        - String replace functions
        - String remove functions
    - Escape HTML Characters
- When do we NOT have to sanitize output
    - If the data we are going to output comes from a trusted source
    - If proper input validation is in place
- When do we have to sanitize output
    - If user input is directly written to output (and no proper input validation was performed)
    - If we output data from any other untrusted sources
- Sanitize output as late as possible
    - Just before the data is send to the client
- Sanitize output as strict as possible
- Sanitize on the Server side ALWAYS
    - Where the final output for the user is created
- Sanitize on the Client side
    - If user supplied data is further processed by Javascript (DOM based XSS)

Chapter 3 Cross-Site Scripting (XSS)

Cross-Site Scripting

- Execution of malicious javascript in the users' browser
- Attack is executed in the browser of the victim, never on the server
- 3 Types of XSS
    - Non-persistent (Reflected) XSS
        - Malicious code exists only in GET or POST request
        - Malicious code never stored on the attacked server
    - Persistent (Stored) XSS
        - Attacker stores malicious code on the vulnerable server
    - DOM-based XSS
        - Malicious code will be created "on the fly" in the browser
- Use/Harms of XSS
    - Website defacement

    - Arbitrary redirection
    - Phishing attacks
    - Credential theft
    - Session hijacking
    - Worms
    - Trojans
    - Drive-by downloads
- Most common vulnerability ever
- Output sanitization via HTML-Encoding does not always work
    - E.g. `out.println("<iframe SRC="+path+">")`, only strong input validation via regex works
- Input validation via type check (+ boundary check)
- Common payloads

```
<body background="javascript:alert('I run in Background');">

<iframe src="javascript:alert('Hello');“/>

<input type=image src="&#74;avascript:alert('Have You')">

<img src='iamnothere.gif' onError=”alert('EventHandler')">

<frameset onLoad="alert('EventHandler')">

<layer name="extern" src="http://evil.com/test.html">
```

Summary for Reflected XSS

- Payload is never stored on the server
    - It is “stored” in the request to the server
    - It is reflected back to the user in the response
- Payload can always be found in server response
- Very easy to detect - even with automated tools

Fix for Reflection XSS

- Perform strict input validation AND/OR strict output sanitization on server
- Actual solution depends heavily on the situation
- Output sanitization is enough for most of the cases

Output Sanitizer projects

- OWASP Java HTML Sanitizer project
- HTML Purifier
- Kses
- htmlLawed
- Blueprint
- Google Caja

Summary for Persistent XSS

- Payload is stored on the server
- Payload can always be found in server response
- Very easy to detect - even with automated tools

Fix for Persistent XSS

- Perform strict input validation OR strict output sanitization on server
- Use well tested Anti XSS Frameworks

DOM-Based XSS

- Javascript payload is not within request to server
- Javascript payload is not visible in response of server
- Javascript payload gets created “on the fly” in the browser
- Client side problem created by DOM manipulation of user input
  - DOM might reverse escaping done on server
  - In worst case, server will never see user input (remember “#”)
- Very hard to detect
  - Automated tools have a hard time detecting DOM based XSS because payload is not ready to run in server response
  - Source code audit of Javascript files is necessary to find everything

Fix for DOM-based XSS

- Know Javascript
- Know the dangerous functions
  - innerhtml()
  - eval()
  - val()
  - html()
  - text()
- Know how these functions behave when used sequentially
- Perform strict client side validation or client side output sanitization of any user input that is written to DOM
  - Only applies here (else should rely on server-side validation)
- Read the DOM based XSS wiki
- Take extra cautions when dealing with Javascript Frameworks such as JQuery - they might behave unexpectedly
  - element.add()
  - element.append()
  - element.html()

Vulnerability	Risk associated to vulnerability	Knowledge required to identify	Ease of Exploitation	How hard to fix?
XSS Reflected	Medium	Automated Tools	Easy	Very Easy
XSS Persistent	Medium	Basic	Easy	Very Easy
XSS DOM based	Medium	Advanced	Medium	Medium

Chapter 4 SQL Injection

Types of SQL injections

- Verbose SQL injections
  - Attacker gets verbose output
  - Might be able to read out entire DB with a few queries (Results in list form)

```
<Input>
user = “admin”;
password = "' or 1=1;-- -" ;

<Code>
SELECT * FROM users WHERE username = ‘”+user+ “' and password = ‘”+password+”‘”
```

  -
- Blind SQL injection
  - Output to SQL Manipulation is Binary (true or false)
  - Attacker can also read out entire DB -> might take a couple of million requests

```
'; if condition  waitfor delay '0:0:5' –
'; union select if( condition , benchmark (100000, sha1('test')),
'false'),1,1,1,1;
```

  - Input correct and wrong query and see if anything on the site changes
  - Can be exploited using SQLMAP

SQL Exploitation

1. Verify the script is vulnerable

- Trigger an error condition
  - `category=animal‘x“x<d`
- Insert a query that is always true

    - `category='animal' or 1=1;-- -`
- Insert a query that is always false
    - `1.category='animal' and 1=2;-- -`
- Recheck if the true condition works
    - `category='animal' or 1=1;-- -`

2. Execute arbitrary SELECT queries
    - `category=animal'; SELECT @@version;-- -`
3. Join different select queries using UNION SELECT (same number of columns)
    - `SELECT id, name FROM images UNION SELECT id, price FROM product`
4. Learn about the database structure
    - Information Schema
5. Read the user table with user and password

Preventing SQL Injection

- JDBC PREPARED STATEMENTS (parameterized queries)
    - Solution to all SQLi problems

```
PreparedStatement prepStmt = connection.prepareStatement("select username from
Users where username = ? and password = ?; ");
prepStmt.setString(1, user);
prepStmt.setString(2, password);
ResultSet rs = prepStmt.executeQuery();
```

- Use frameworks: JOOQ, Hibernate (replace parameters via setXXX(name,/index,value))
- If not possible then:
    - Whitelist columns/table names

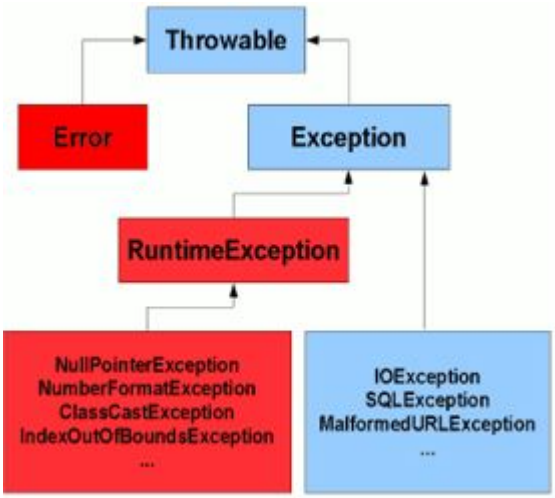
```
List<String> colsList = Arrays.asList("username", "email");
List<String> opsList = Arrays.asList("=", "!=");

if(colsList.contains(col) && opsList.contains(op)){
PreparedStatement prepStmt =    connection.prepareStatement("select " + col +
" from    Users where " + col + " " + op + " ?");
prepStmt.setString(1, search);
ResultSet rs = prepStmt.executeQuery();
}
```

    - Use stored procedures
    - Do plausibility checks (ONLY int, string length, character set)
    - Escape all input (meta characters)
    - Enclose user input by quotes (Especially table names)

Database Setup - Best Practices

- Access to the database is required to be authenticated. No default passwords must be used for this authentication
- Login with least privileges on the database
- Direct database access by client applications should be only possible for admin. The communication must be cryptographically secured

Chapter 5 Exception Handling and Logging



Checked Vs Uncheck Exception

- Checked Exception
  - Belong to parent class Exception
    - Try/Catch/Finally
    - Throwing the same exception (or a superclass) in the method signature
  - Occur when there is error in code
  - Identified by compiler and can be handled programmatically
- Unchecked Exception
  - Belong to parent class RuntimeException
  - Error arises during program execution
  - Cannot be handled programmtically, might be handled in the code
  - Most common is NullPointerException
- Both types of exceptions may pose information leaks/threats

Exception Name	Threat/Leak Description
FileNotFoundException	Disclose file system structure
SQlException	Disclose database structure
BindException	Disclose open ports when untrusted client can choose server port
OutofMemoryError	Denial of Service
StackOverflowError	Denial of Service

Secure way of coding

- Do not catch generic Exception like IOException, instead be specific like FileNotFoundException
- Do not print stacktrace
- Put the return statement outside of the finally clause
- Never catch NullPointerException
    - Program bugs indicated by Runtime exceptions need to be fixed
- Log exceptions

Best practices for Exceptions

- Throw Exception relevant to the Interface
- Reason for exception should be specific
- If a method throws an exception received from another method, encapsulate it in a locally generated Exception child class
- Balance catch blocks
- Create Custom Error Pages
    - Using Struct
    - Using java servlets (In web.xml)
    - Using JSP (For HTTP 404 or HTTP 500 errors)
- Make use of finally clause
- Throw early but catch late
- Take care of exception
- Configure/provide exception handlers(s) in web.xml
- Use generic messages with no specifications
- Do not rely only on framework's error handling mechanism

Logging

- Java from JDK 1.4 onwards provides a logging framework in the package java.util.logging
- Logging levels in Log4j
    - Debug
    - Info
    - Warn
    - Error
    - Fatal
    - OFF(Highest rank and intended to turn off logging in java)
- Logging levels in Java.util.logging
    - Severe
    - Warning
    - Info
    - Config
    - Fine

    - Finer
    - Finest

Logging best practices

- Log Debug message inside isdebugenabled() block
- Make use of good java logging frameworks like java.util.logging or log4j
    - For Log4j, the logging levels can be changed without restarting the application
- Do not log sensitive information in production environment
- Select appropriate logging level for every logging message
- Specify the format of the java logging
- Log messages consistently and the messages must be informative
- Use code level prefix when logging so that it is clear which part of code is printing log message

Chapter 6A Securing Against XSS

Regular Expression

- Java package java.util.regex
- Involves pattern matching
- Check the pattern here
- Wildcard characters and predefined classes

```
* - Match 0 or more occurrences
? - Match 0 or 1 occurrence
+ - Match 1 or more occurrences
{x} - Match exactly x times
{x,y} – Match between x(inclusive) to y(inclusive) occurrences
[123x] - Match a single instance of any character in the brackets
[0-9] - Shorthand for [0123456789], often used with [A-Z]
[^0-9] - Match a single character except those specified in brackets
.  Any character
\d A digit: [0-9]
\D A non-digit: [^0-9]
\s A whitespace character
\S A non-whitespace character: [^\s]
\w A word character: [a-zA-Z_0-9]
\W A non-word character: [^\w]
[ ] - Match a space

Logical Operators
XY : X followed by Y
X|Y:  Either X or Y
(X) : X, as a capturing group••Example:
([13]x)|([ab]3)  pattern will match 1x,3x,a3,b3
```

- Usage

```
String email = "xyz@hotmail.com";
//Set the email pattern string
Pattern p = Pattern.compile(".+@.+\\.[a-z]+");
//Match the given string with the pattern
Matcher m = p.matcher(email);

//check whether match is found
boolean matchFound = m.matches();
if (matchFound)
 System.out.println("Valid Email Id.");
else
 System.out.println("Invalid Email Id.");
  }
}

OR

email.matches(regex)


To check if target string is digit type:\\d+ OR [0-9]+
To check if target string is just composed of alphabets:[a-zA-Z]+
To check if target string is alphanumeric type:[a-zA-Z0-9]+


Allow - and _ before @ (no need to escape using \\)
^([0-9a-zA-Z-_]{1,}@[a-zA-Z]{1,}\\.edu)$
```

Chapter 7 Session Management

Typical Session Management Vulnerabilities

- Unprotected Session Cookies
- Session ID in URL
- Session ID doesn't change after login/logout
- Cleartext authentication
- Administrative login reachable by anyone (inter/intranet)
- Inappropriate Sessions Timeouts

Storage and Transmission via Cookie - HttpOnly Cookie Flag

- The problem of stealing cookies can be fixed by using Http-Only cookies
- Http-Only flag prevents javascript from accessing the cookies
- XSS still remains

Storage and Transmission via Cookie - Secure Cookie Flag

- Secure flag is set - cookie is only transmitted via HTTPS

- Protect your cookie on open wifi

Session ID in URL

- Attacker can steal session ID of user
    - By reading logfiles on server/ proxy/ local browser history (GET parameters are saved)
    - Because victim posted URL to
        - Messageboard
        - facebook
        - any public sites

Session ID doesn't change after login/logout

- Steal once - Reuse as often as wanted
    - Once attacker steal the session once, he can reuse as often as he wants.

Cleartext authentication

- Attacker with access to network traffic can read passwords
    - Wireshark, Burp

Administrative login reachable by anyone (inter/intranet)

- Makes administrative login vulnerable to brute force attacks
    - Large password lists are freely available
    - Various tools automate the process

Inappropriate Session Timeouts

- Idle time before a session is auto invalidated
- Depends on application needs
- Prevent other people from accessing your account with computer is left idle and user not in front o the screen
- More sensitive apps should have lesser timeout set. I.e. bank apps timeout can be set at 10min
- Ideal is 15minutes

Chapter 8 Cross-Site Request Forgery (CSRF)

A Cross Site Request Forgery (CSRF) attack forces a logged-on victim's browser to send a request to a vulnerable web application, which then performs the chosen action on behalf of the victim. The malicious code is often not on the attacked site. This is why it is called Cross site

Unlike cross-site scripting (XSS), which exploits the trust a user has for a particular site, CSRF exploits the trust that a site has in a user's browser.

Real life attack scenarios for CSRF

- Any administrative backend with critical operations
- Any change password function (If old one is not required)
- Any change email function

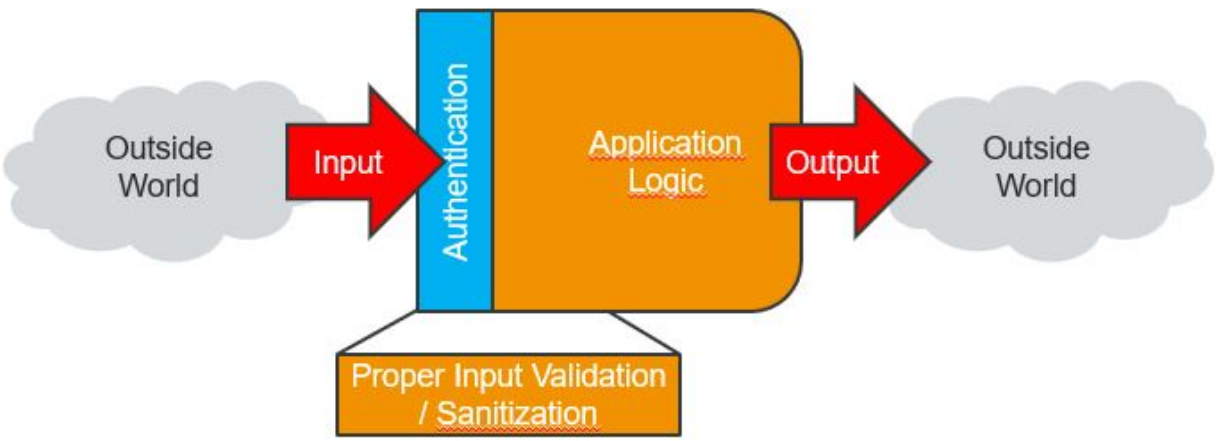
- Any other privileged write action in the context of the victim
- Perform any operation of the attacked website as this user

Protecting against CSRF

- Using a CSRF token for every POST request (Hidden HTML form field)
    - The session cookie is identical to the CSRF token
    - `<input type="hidden" name="_csrf" value=>`
- If the CSRF token is stored in cookie, then the web application will still be vulnerable to CSRF, because all cookies will be submitted with every request
- If there is XSS vulnerability on the website, then there is vulnerability to CSRF bypass, because attacker can read out any CSRF token using XSS
- Using Captcha
    - Request the user to do some specific actions before being able to send a POST request
- CSRF token can be a variant of your session ID
    - MD5 of your session ID then use it as CSRF token

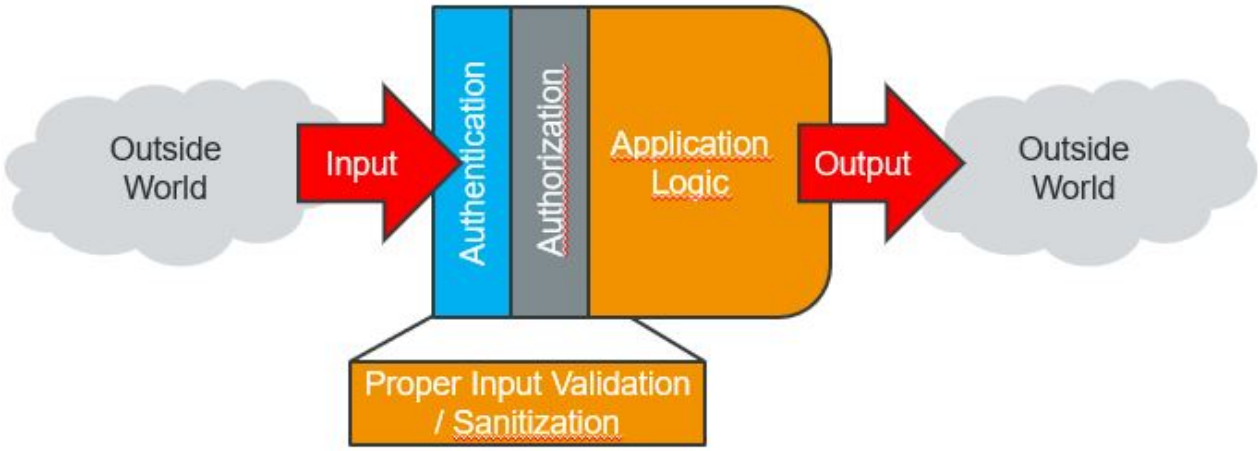
Chapter 9 Authentication and Authorization

Prevent Authentication problems

- Implement a dedicated Authentication layer (For example in j2EE, we can implement an Authentication Filter)
- 

Prevent Authorization problems

- Implement a dedicated Authorization layer

- 

Best Practices for Authentication / Authorization

- Authenticate over a secure channel (HTTPS)
  - Clear text communication will be sniffed
- Use proper session management
  - Rely on existing best-practice implementations
  - Avoid custom implementations
- Use generic error messages in case of problems
- Keep Session ID fresh
  - Create a new session with every login/logout
  - Explicitly invalidate old sessions
- Keep Session ID random
  - Make sure is really random
  - Use established implementations for session management
- Keep sessions in a “secured” session cookie
  - Use HTTP flags for cookies for extra protection against XSS
  - Use Secure flag for cookies if you use HTTPS
- Don’t forget to
  - Provide logout feature
  - Provide related functions (change password, email)
- For critical use cases / operations
  - Use multi-factor authentication
  - Re-authenticate during high value transactions / critical operations
- Outsource your authentication using proven techniques, e.g. SAML
- Use the right authentication method which match your security requirements
- Don’t just only authenticate at the “gate”
- Use given authentication / authorization features of your framework
  - Use them properly
  - Otherwise implement using best practice design patterns

Password Management

- Use a salted hash
- Append the salt to the password and hash it
- Save the salt and the hash in the user's database record
- Java - Generate a secure password hash

```
import org.apache.shiro.crypto.hash.SimpleHash;

private static final int ITERATIONS = 1024;
private static final int RAND_BITS = 256;
private static final int RADIX = Character.MAX_RADIX;

public String hashIt(String password){
    SecureRandom random = new SecureRandom();
    String salt = new BigInteger(RAND_BITS, random).toString(RADIX);
    SimpleHash hash = new SimpleHash("SHA-512", password, salt, ITERATIONS);
    return hash.toHex();
}
```

- Never store password in clear
- Never store password as plain hash
- Use "Salt/Hash/Iterate" approach
- Use bcrypt() or argon2()

Application passwords

- Never store in plaintext
- Store only in protected config files

Email / Password Change

- Attacker might get access to your unlocked PC
- Password might be changed via XSS and CSRD automatically
- Use CSRF Token
- Use CAPTCHA for extra security
- Password Reset
    - Never send new password in clear via email
    - Should be based on email
    - Only use random one time link for password reset
- Change Passwords/Emails
    - Current password should always be required
    - Don't forget the CSRF token for protection
    - Use a captcha if you are paranoid

Practical

Prepared Statement

```
Class.forName("com.mysql.jdbc.Driver");
String url="jdbc:mysql://localhost/assignment?user=root&password=root";
Connection conn= DriverManager.getConnection(url);

String sql="select * from test where name=? and number=?";
PreparedStatement pstmt=conn.prepareStatement(sql);
pstmt.setString(1,"test");
pstmt.setInt(2,56);

ResultSet rs= pstmt.executeQuery();
// rs may be empty but never null
// first row is at index position 1, not 0



String sql2="Insert into test(name,number) values(?,?)";
PreparedStatement pstmt2=conn.prepareStatement(sql2);
pstmt2.setString(1,"test");
pstmt2.setInt(2,56);

int rec= pstmt2.executeUpdate();

pstmt.close();
conn.close() will automatically close everything
```

Stored Procedures

```
String simpleProc = "{ call testing(?) }";//Routine name with corresponding argument

CallableStatement cs = conn.prepareCall(simpleProc);
cs.setString(1,name);
cs.execute();
ResultSet rs=cs.getResultSet();
```

Session Management

```
HttpSession persists for 30min of inactivity

HttpSession session=request.getSession(true) //get new session
HttpSession session=request.getSession(false) //get pre-existing session
session.invalidate();
session.setMaxInactiveInterval(60); in session (seconds)

request.getSession().getId()

if (request.getSession() != null) {
```

```
                request.getSession().invalidate();
            }

 session.removeAttribute("name");
 session.setAttribute("name","cs");
 String name = (string)session.getAttribute("name");
 ObjectType value =  (ObjectType)session.getAttribute("attributeName");
```

Others

```
response.sendRedirect("error.jsp");
```

- If there is no Session, then `request.getAttribute()` will return null
- Print to web browser `out.println(member.getUsername());`

```
request.setAttribute("login", "success");
RequestDispatcher rd = request.getRequestDispatcher("index.jsp");
rd.forward(request, response);
request.getAttribute("login");
```

- `ArrayList<movieSeat> seats= new ArrayList<movieSeat>();`

```
<form name="form">
    <input type="text" id="user" name="user">
        </br></br>
    <input type="text" id="pass" name="pass">
    <input type="submit" name="submit" value="submit" onclick="hack()"></form>

<script>
    function hack(){
        alert(form.user.value+", "+ document.form.pass.value)
    }
</script>

XssImage = new Image();
XssImage.src=SOME_URL = will perform a post
```

```
<%@ page import="org.apache.logging.log4j.Logger" %>
<%@ page import="org.apache.logging.log4j.LogManager" %>

This code when added to the JSP creates an instance of the Logger:
<%!  Logger logger = LogManager.getRootLogger(); %>

Try this code in your JSP page:
logger.error("Error JSP");
```

```
<img src= http://localhost:8080/WebGoat/attack?transferFunds=4000 &
Screen=280&menu=900 width="1" height="1"/>

width and height 1 to make it invisible
```

```
Javascript regex
     if (txtpass.length > 6);
     if ( ( txtpass.match(/[a-z]/) ) && ( txtpass.match(/[A-Z]/) ) );
     if (txtpass.match(/\d+/));
     if ( txtpass.match(/.[!,@,#,$,%,^,&,*,?,_,~,-,(,)]/) );
```

```
((?=.*[0-9])(?=.*[a-z])(?=.*[A-Z])(?!.*[ ])(?=.*[!@#$%^&*()])).{8,16}

(?=.*[0-9]) for minimum 1 digit forward search
(?!.*[ ]) for negative search of space


OR
((?=.*[0-9]+)(?=.*[a-z]+)(?=.*[A-Z]+)(?!.*[ ]+)(?=.*[!@#$%^&*()]+)).{8,100}
```

```
org.apache.commons.lang3.StringEscapeUtils

StringEscapeUtils.escapeHtml4(search);
```

```
// similar behavior as an HTTP redirect
window.location.replace("http://stackoverflow.com");

// similar behavior as clicking on a link
window.location.href = "http://stackoverflow.com";

window.open('https://www.google.com', '_blank');
```

- use concat() to combine 2 or more columns into 1 columns