

Break 'em and Build 'em Web

SecAppDev 2015

Ken van Wyk, @KRvW

Leuven, Belgium

23-27 February 2015

KRvW Associates, LLC

Ken van Wyk, ken@krvw.com, @KRvW

Copyright© 2015 KRvW Associates, LLC



Part I - Break 'em!

Module flow

Description of the flaw and how it is exploited

Exercise to attack the flaw (for most)

We'll let you try to figure each exercise out yourself

Then instructor will demonstrate the attack

We'll also briefly discuss mitigations, but will come back to those in 2nd half of class

The tools we'll use

OWASP tools (freely available)

Firefox web browser

- With FoxyProxy plug-in

WebScarab -- a web application testing proxy

- ZAP is also installed in our VM

WebGoat -- a simple web application containing numerous flaws and exercises to exploit them

- Runs on (included) Apache Tomcat J2EE server

Setting up your virtual machine

Install VirtualBox on your system from the USB or download provided

You will need administrative privileges to install it if it isn't already there

From the File menu, *import* the appliance prepared for this class

You may need to adjust the memory allocated for the VM (default is 2 Gb)

You may need to tweak network settings and/or graphics hardware settings — like 3D and 2D acceleration

Setting up WebGoat

We'll boot from the provided Virtual Machine Class software pre-installed, but run from command line

- First cd into `~/Desktop/WebGoat-next`

To compile and run, type -

- `mvn clean tomcat:run-war`

Launch Firefox and point to server from bookmark

- `http://localhost:8080/WebGoat/attack`

At this point, WebGoat is running, but you'll still need a testing proxy to perform some attacks

Next, set up WebScarab

Run WebScarab

Default listener runs on TCP port 8008

Ensure listener is running within WebScarab

Configure proxy

Use FoxyProxy in Firefox and select WebScarab

- This configures browser to proxy traffic on TCP/8008 on 127.0.0.1 (localhost)

Connect once again to <http://localhost/WebGoat/attack>

WebGoat tips

Report card shows overall progress

Don't be afraid to use the "hints" button

Show cookies and parameters can also help

Show java also helpful

None of these are typical on real apps...

Learn how to use it

Fabulous learning tool

Familiarizing Goat and Scarab

WebGoat

Do “Web Basics”
exercise

Try Hints and other
buttons

Look at report card

WebScarab

Turn on intercepts

- Requests
- Responses

Explore and experiment

- Parsed vs. raw view

Try editing a request

- Modify parameters
- Add/omit parameters

A word of warning on ethics

You will see, learn, and perform real attacks against a web application today.

You may only do this on applications where you are authorized (like today's class).

Violating this is a breach of law in most countries.

Never cross that ethical “line in the sand”!

OWASP Top-10 (2013)

A1 - Injection

A2 - Broken authentication and session management

A3 - Cross-site scripting

A4 - Insecure direct object reference

A5 - Security misconfiguration

A6 - Sensitive data exposure

A7 - Missing function level access control

A8- Cross site request forgery (CSRF)

A9 - Using components with known vulnerabilities

A10 - Unvalidated redirects and forwards

#1 Injection flaws

Occurs when
“poisonous” data causes
software to misbehave

Most common is SQL
injection

Attacker taints input data
with SQL statement

SQL passes to SQL
interpreter and runs

Data “jumps” from data
context to SQL context

Consider the following
input to an HTML form

Form field fills in a
variable called
“CreditCardNum”

Attacker enters

- ‘
- ‘ --
- ‘ or 1=1 --

What happens next?

SQL basics

Attacker should understand SQL query syntax

Integer comparisons

- “...or 1=1...”

String comparisons

- “...or ‘1’=‘1’...”

SQL injection attacks can become complex

Attacker with in-depth knowledge of SQL can have a “field day”

SQL string injection exercise

String SQL Injection - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Address <http://localhost/WebGoat/attack?Screen=58&menu=1200>

Logout ?

String SQL Injection

OWASP WebGoat V5.2

◀ Hints ▶ Show Params Show Cookies Lesson Plan Show Java Solution

Introduction
General
Access Control Flaws
AJAX Security
Authentication Flaws
Buffer Overflows
Code Quality
Concurrency
Cross-Site Scripting (XSS)
Denial of Service
Improper Error Handling
Injection Flaws

[Command Injection](#)
[Blind SQL Injection](#)
[Numeric SQL Injection](#)
[Log Spoofing](#)
[XPATH Injection](#)
[LAB: SQL Injection](#)
 Stage 1: String SQL Injection
 Stage 2: Parameterized Query #1
 Stage 3: Numeric SQL Injection
 Stage 4: Parameterized Query #2
[String SQL Injection](#)
[Database Backdoors](#)
Insecure Communication
Insecure Configuration
Insecure Storage

Solution Videos [Restart this Lesson](#)

SQL injection attacks represent a serious threat to any database-driven site. The methods behind an attack are easy to learn and the damage caused can range from considerable to complete system compromise. Despite these risks, an incredible number of systems on the internet are susceptible to this form of attack.

Not only is it a threat easily instigated, it is also a threat that, with a little common-sense and forethought, can easily be prevented.

It is always good practice to sanitize all input data, especially data that will be used in OS command, scripts, and database queries, even if the threat of SQL injection has been prevented in some other manner.

General Goal(s):

The form below allows a user to view their credit card numbers. Try to inject an SQL string that results in all the credit card numbers being displayed. Try the user name of 'Smith'.

Enter your last name:

```
SELECT * FROM user_data WHERE last_name = 'Your Name'
```

No results matched. Try Again.

OWASP Foundation | Project WebGoat | Report Bug

Local intranet

SQL integer injection exercise

Numeric SQL Injection - Microsoft Internet Explorer

Address: <http://localhost/WebGoat/attack?Screen=67&menu=1200>

Logout ?

Numeric SQL Injection

OWASP WebGoat V5.2

Hints Show Params Show Cookies Lesson Plan Show Java Solution

- Introduction
- General
- Access Control Flaws
- AJAX Security
- Authentication Flaws
- Buffer Overflows
- Code Quality
- Concurrency
- Cross-Site Scripting (XSS)
- Denial of Service
- Improper Error Handling
- Injection Flaws
 - [Command Injection](#)
 - [Blind SQL Injection](#)
 - [Numeric SQL Injection](#)
 - [Log Spoofing](#)
 - [XPath Injection](#)
 - [LAB: SQL Injection](#)
 - Stage 1: String SQL Injection
 - Stage 2: Parameterized Query #1
 - Stage 3: Numeric SQL Injection
 - Stage 4: Parameterized Query #2
 - [String SQL Injection](#)
 - [Database Backdoors](#)
- Insecure Communication
- Insecure Configuration
- Insecure Storage

Solution Videos

Restart this Lesson

SQL injection attacks represent a serious threat to any database-driven site. The methods behind an attack are easy to learn and the damage caused can range from considerable to complete system compromise. Despite these risks, an incredible number of systems on the internet are susceptible to this form of attack.

Not only is it a threat easily instigated, it is also a threat that, with a little common-sense and forethought, can easily be prevented.

It is always good practice to sanitize all input data, especially data that will be used in OS command, scripts, and database queries, even if the threat of SQL injection has been prevented in some other manner.

General Goal(s):

The form below allows a user to view weather data. Try to inject an SQL string that results in all the weather data being displayed.

Select your local weather station:

```
SELECT * FROM weather_data WHERE station = [station]
```

OWASP Foundation | Project WebGoat | Report Bug

Done Local intranet

Injection issues and remediation

Passing unchecked data to any interpreter is dangerous

Filtering out dangerous data alone can be problematic

SQL injection remediation

Use static strings

Parse for provably safe input

- Not a good idea

Parameterized queries

- Via PreparedStatement

Stored procedures

- Safe, but SQL engine dependent

What about input validation?

If we're using PreparedStatement, do we have to worry about input validation?

Of course!

Consider other data payloads, like XSS

Other injection dangers

SQL injection is common but others exist

XML

LDAP

Command shell

Comma delimited files

Log files

Context is everything

Must be shielded from presentation layer

Input validation will set you free

Positive validation is vital

Examples – How NOT to...

```
//Make connection to DB
Connection connection = DriverManager.getConnection(DataURL,
LOGIN, PASSWORD);

String Username = request.getParameter("USER"); // From HTTP
request
String Password = request.getParameter("PASSWORD"); // same

int iUserID = -1;
String sLoggedInUser = "";

String sel = "SELECT User_id, Username FROM USERS WHERE Username
= '" +Username + "' AND Password = '" + Password + "'";

Statement selectStatement = connection.createStatement ();
ResultSet resultSet = selectStatement.executeQuery(sel);
```

Examples – PreparedStatement

```
String firstname = req.getParameter("firstname");  
String lastname = req.getParameter("lastname");
```

```
String query = "SELECT id, firstname, lastname  
FROM authors WHERE forename = ? and surname = ?";
```

```
PreparedStatement pstmt =  
connection.prepareStatement( query );
```

```
pstmt.setString( 1, firstname );
```

```
pstmt.setString( 2, lastname );
```

```
try
```

```
{
```

```
    ResultSet results = pstmt.execute( );
```

```
}
```

#2 Broken authentication and session management (was #3)

HTTP has no inherent session management

And only rudimentary authentication

Every developer has to invent (or reuse) one

Mistakes are common

Credentials transmitted unencrypted

Stored unsafely

Passed in GET (vs. POST)

Session cookies revealed or guessable

Authentication design patterns

Mechanisms

Basic/digest

Forms

Certificate

Forms (most often)

Username/password

HTTP parameters

Password hashed

Sent to auth service

Validate or not

Authentication services

Centralized

Federated

Role based

Data stores

- XML, LDAP, SQL, text file

Certificates

Strong, but rarely used

Basic authentication exercise

Basic Authentication - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Address <http://localhost/WebGoat/attack?Screen=259&menu=500&Restart=259>

Logout ?

Basic Authentication

OWASP WebGoat V5.2

◀ Hints ▶ Show Params Show Cookies Lesson Plan Show Java Solution

- Introduction
- General
- Access Control Flaws
- AJAX Security
- Authentication Flaws
 - [Password Strength](#)
 - [Forgot Password](#)
 - [Basic Authentication](#)
 - [Multi Level Login 1](#)
 - [Multi Level Login 2](#)
- Buffer Overflows
- Code Quality
- Concurrency
- Cross-Site Scripting (XSS)
- Denial of Service
- Improper Error Handling
- Injection Flaws
- Insecure Communication
- Insecure Configuration
- Insecure Storage
- Parameter Tampering
- Session Management Flaws
- Web Services
- Admin Functions
- Challenge

Solution Videos

[Restart this Lesson](#)

Basic Authentication is used to protect server side resources. The web server will send a 401 authentication request with the response for the requested resource. The client side browser will then prompt the user for a user name and password using a browser supplied dialog box. The browser will base64 encode the user name and password and send those credentials back to the web server. The web server will then validate the credentials and return the requested resource if the credentials are correct. These credentials are automatically resent for each page protected with this mechanism without requiring the user to enter their credentials again.

General Goal(s):

For this lesson, your goal is to understand Basic Authentication and answer the questions below.

What is the name of the authentication header:

What is the decoded value of the authentication header:

OWASP Foundation | Project WebGoat | Report Bug

Done Local intranet

Spoofing auth cookie exercise

The screenshot shows a Microsoft Internet Explorer browser window with the title "Spoof an Authentication Cookie - Microsoft Internet Explorer". The address bar contains the URL "http://localhost/WebGoat/attack?Screen=261&menu=1700". The page content includes a navigation bar with "Hints", "Show Params", "Show Cookies", "Lesson Plan", "Show Java", and "Solution" buttons. A sidebar on the left lists various security topics, with "Spoof an Authentication Cookie" highlighted. The main content area features a "Sign In" section with a "Login" button and a "Restart this Lesson" link. The Aspect Security logo is visible in the bottom right corner of the page content.

Session management basics

Web contains no inherent session management

Unique ID assigned to each session on server

ID passed to browser and returned in each GET/POST

JSESSIONID for J2EE

Once authenticated, session token is as powerful as valid username/password

Must be rigorously protected

Confidential

Random

Unpredictable

Unforgeable

A word about setting cookies

*Set-Cookie: name=VALUE; domain=DOMAIN_NAME;
expires=DATE; path=/PATH/; secure; httponly*

Set via HTTP headers

Only name field is required

Secure attribute instructs client to SSL encrypt

RFC 2965 still allows the client significant leeway

No guarantee for confidentiality, but still a good practice

Httponly attribute prevents scripts from accessing cookie (e.g., Javascript in XSS attacks)

Session management pitfalls

Exposing session token

Session fixation

Custom tokens

Not resetting session token

Session hijacking and replay

CSRF susceptible

Session fixation

Session Fixation - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Address <http://localhost/WebGoat/attack?Screen=16&menu=1700> Go Links

Logout ?

OWASP WebGoat V5.2 < Hints > Show Params Show Cookies Lesson Plan Show Java Solution

Introduction
General
Access Control Flaws
AJAX Security
Authentication Flaws
Buffer Overflows
Code Quality
Concurrency
Cross-Site Scripting (XSS)
Denial of Service
Improper Error Handling
Injection Flaws
Insecure Communication
Insecure Configuration
Insecure Storage
Parameter Tampering
Session Management Flaws
[Hijack a Session](#)
[Spoof an Authentication Cookie](#)
[Session Fixation](#)
Web Services
Admin Functions
Challenge

Solution Videos **Restart this Lesson**

STAGE 1: You are Hacker Joe and you want to steal the session from Jane. Send a prepared email to the victim which looks like an official email from the bank. A template message is prepared below, you will need to add a Session ID (SID) in the link inside the email. Alter the link to include a SID.

You are: Hacker Joe

Mail To: jane.plane@owasp.org
Mail From: admin@webgoatfinancial.com
Title:

Created by: Reto Lippuner, Marcel Wirth

OWASP Foundation | Project WebGoat | Report Bug

Local intranet

#3 Cross site scripting (“XSS”) (Was #2)

Can occur whenever a user can enter data into a web app

Consider all the ways a user can get data to the app

When data is represented in browser, “data” can be dangerous

Consider this user input

```
<script>  
alert(document.cookie)  
</script>
```

Where can it happen?

ANY data input

Two forms of XSS

Stored XSS

Reflected XSS

Stored XSS

Attacker inputs script data on web app

Forums, “Contact Us” pages are prime examples

All data input must be considered

Victim accidentally views data

Forum message, user profile, database field

Can be years later

Malicious payload lies patiently in wait

Victim can be anywhere

Stored XSS exercise

Stored XSS Attacks - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Address <http://localhost/WebGoat/attack?Screen=50&menu=900>

Logout ?

Stored XSS Attacks

OWASP WebGoat V5.2

◀ Hints ▶ Show Params Show Cookies Lesson Plan Show Java Solution

- Introduction
- General
- Access Control Flaws
- AJAX Security
- Authentication Flaws
- Buffer Overflows
- Code Quality
- Concurrency
- Cross-Site Scripting (XSS)
 - Phishing with XSS
 - LAB: Cross Site Scripting
 - Stage 1: stored XSS
 - Stage 2: Block stored XSS using Input Validation
 - Stage 3: stored XSS Revisited
 - Stage 4: Block stored XSS using Output Encoding
 - Stage 5: Reflected XSS
 - Stage 6: Block Reflected XSS
 - Stored XSS Attacks
- Cross Site Request Forgery (CSRF)
- Reflected XSS Attacks
- HTTPOnly Test
- Cross Site Tracing (XST) Attacks
- Denial of Service
- Improper Error Handling
- Injection Flaws

Solution Videos

Restart this Lesson

It is always a good practice to scrub all input, especially those inputs that will later be used as parameters to OS commands, scripts, and database queries. It is particularly important for content that will be permanently stored somewhere in the application. Users should not be able to create message content that could cause another user to load an undesirable page or undesirable content when the user's message is retrieved.

Title:

Message:

Submit

Message List

ASPECT SECURITY
Application Security Specialists

OWASP Foundation | Project WebGoat | Report Bug

Local intranet

Reflected XSS

Attacker inserts script data into web app

App immediately “reflects” data back

Search engines prime example

“String not found”

Generally combined with other delivery mechanisms

HTML formatted spam most likely

Image tags containing search string as HTML parameter

- Consider width=0 height=0
IMG SRC

Reflected XSS exercise

Reflected XSS Attacks - Microsoft Internet Explorer

Address: http://localhost/WebGoat/attack?Screen=49&menu=900

OWASP WebGoat V5.2

Reflected XSS Attacks

Logout ?

Hints Show Params Show Cookies Lesson Plan Show Java Solution

Introduction
General
Access Control Flaws
AJAX Security
Authentication Flaws
Buffer Overflows
Code Quality
Concurrency
Cross-Site Scripting (XSS)
Phishing with XSS
LAB: Cross Site Scripting
Stage 1: stored XSS
Stage 2: Block stored XSS using Input Validation
Stage 3: stored XSS Revisited
Stage 4: Block stored XSS using Output Encoding
Stage 5: Reflected XSS
Stage 6: Block Reflected XSS
Stored XSS Attacks
Cross Site Request Forgery (CSRF)
Reflected XSS Attacks
HTTPOnly Test
Cross Site Tracing (XST) Attacks
Denial of Service
Improper Error Handling
Injection Flaws

Solution Videos Restart this Lesson

For this exercise, your mission is to come up with some input containing a script. You have to try to get this page to reflect that input back to your browser, which will execute the script and do something bad.

Shopping Cart

Shopping Cart Items -- To Buy Now	Price	Quantity	Total
Studio RTA - Laptop/Reading Cart with Tilting Surface - Cherry	69.99	1	\$0.0
Dynex - Traditional Notebook Case	27.99	1	\$0.0
Hewlett-Packard - Pavilion Notebook with Intel Centrino	1599.99	1	\$0.0
3 - Year Performance Service Plan \$1000 and Over	299.99	1	\$0.0

The total charged to your credit card: \$0.0

Enter your credit card number:

Enter your three digit access code:

Done Local intranet

XSS issues

Why is this #3?

Input validation and
output escaping
problems are pervasive
Focus on functional spec
Eradicating it entirely
from an app is tough
work

Why is it such a big
deal?

Highly powerful attack
Anything the user can
do, the attacker can do
Take over session
Install malware
Copy/steal sensitive data

XSS remediation

Multi-tiered approach

Input validation

Output encoding
 (“escaping”)

But how?

It’s not so simple

Blocking “<>”,
 “<script>”, etc. can lead
 to disaster

Strive for positive
 input validation, not
 negative

Prove something is safe

Beware of
 internationalization

Every single input

Database import, XML
 data, the list goes on and
 on

Code

Regular expression
processors

Positive validation

Coding guidelines

Safe code patterns

Common libraries and
frameworks

Centrally maintainable

Code reviews should
verify conformance

Consider tools with
custom rule sets

Negative validation
models must be
justified

Often no easier to write

Presentation layer input validation

Client-side (Javascript)
input validation

Trivially bypassed

Not a suitable security
control by itself

Good for usability

App server validation

XML config files

Regular expression
processing to verify
fields

Positive validation

Instant feedback to user

Examples - Javascript

```
// XSS filter code. takes out coding characters and returns
the rest
function emitSpclChr(nameStrng){
    for(j=0;j<nameStrng.length;j++){
        thisChar = nameStrng.charAt(j);
        if(thisChar=="<" || thisChar==">" ||
thisChar=="?" || thisChar=="*" || thisChar=="(" ||
thisChar=="")){

nameStrng=nameStrng.replace(thisChar, "");
            j=j-1;
        }
    }
    return (nameStrng);
}
//end XSS
```

Examples - Javascript

```
<SCRIPT>
regex1=/^[a-z]{3}$/;
regex2=/^[0-9]{3}$/;
regex3=/^[a-zA-Z0-9 ]*$/;
regex4=/^(one|two|three|four|five|six|seven|eight|nine)$/;
regex5=/^\d{5}$/;
regex6=/^\d{5}(-\d{4})?$/;
regex7=/^[2-9]\d{2}-?\d{3}-?\d{4}$/;
function validate() {
msg='JavaScript found form errors'; err=0;
if (!regex1.test(document.form.field1.value)) {err+=1; msg+='\n bad field1';}
if (!regex2.test(document.form.field2.value)) {err+=1; msg+='\n bad field2';}
if (!regex3.test(document.form.field3.value)) {err+=1; msg+='\n bad field3';}
if (!regex4.test(document.form.field4.value)) {err+=1; msg+='\n bad field4';}
if (!regex5.test(document.form.field5.value)) {err+=1; msg+='\n bad field5';}
if (!regex6.test(document.form.field6.value)) {err+=1; msg+='\n bad field6';}
if (!regex7.test(document.form.field7.value)) {err+=1; msg+='\n bad field7';}
if ( err > 0 ) alert(msg);
else document.form.submit();
}
</SCRIPT>
```

Examples – A bit better

```
protected final static String ALPHA_NUMERIC =
    "[a-zA-Z0-9\\s\\.\\-]+$";
// we only want case insensitive letters and numbers
public boolean validate(HttpServletRequest request,
String parameterName) {
boolean result = false;
Pattern pattern = null;
parameterValue = request.getParameter(parameterName);
if(parameterValue != null) {
pattern = Pattern.compile(ALPHA_NUMERIC);
result = pattern.matcher(parameterValue).matches();
}return result;
} else
{ // take alternate action }
```


Output encoding

Necessary for safely outputting untrusted data

Context is vital to understand

HTML

Javascript

CSS

etc

Encoding scheme needs to match context of output stream

Build/acquire an output encoding library

Different data types

Examples – HTML escape

Context

```
<body> UNTRUSTED DATA HERE </body>
```

```
<div> UNTRUSTED DATA HERE </div>
```

```
    other normal HTML elements
```

```
String safe =
```

```
ESAPI.encoder().encodeForHTML(request.getParameter("input"));
```

Examples – HTML attributes

Context

```
<div attr = UNTRUSTED DATA > content </div>
```

```
<div attr = 'UNTRUSTED SINGLE QUOTED DATA'> content </div>
```

```
<div attr = "UNTRUSTED DOUBLE QUOTED DATA"> content </div>
```

```
String safe =  
ESAPI.encoder().encodeForHTMLAttribute  
(request.getParameter("input"));
```

#4 Insecure direct object reference

Architectural flaw in application

Giving user access to a real world object is dangerous

Absolutely will be tampered

Results can have major impact

Examples include

Files

User credentials

Payment information

Sensitive application data or functions

Object reference exercise

Bypass a Path Based Access Control Scheme - Microsoft Internet Explorer

Address: http://localhost/WebGoat/attack?Screen=17&menu=200

Logout ?

Bypass a Path Based Access Control Scheme

OWASP WebGoat V5.2

Hints Show Params Show Cookies Lesson Plan Show Java Solution

Restart this Lesson

Solution Videos

The 'guest' user has access to all the files in the lesson_plans directory. Try to break the access control mechanism and access a resource that is not in the listed directory. After selecting a file to view, WebGoat will report if access to the file was granted. An interesting file to try and obtain might be a file like tomcat/conf/tomcat-users.xml

Current Directory is: C:\Documents and Settings\Instructor\Desktop\WebGoat-5.2\tomcat\webapps\WebGoat\lesson_plans

Choose the file to view:

- AccessControlMatrix.html
- BackDoors.html
- BasicAuthentication.html
- BlindSqlInjection.html
- BufferOverflow.html
- ChallengeScreen.html
- ClientSideFiltering.html
- ClientSideValidation.html
- CommandInjection.html
- ConcurrencyCart.html
- CrossSiteScripting.html
- CSRF.html
- DangerousEval.html
- DBCrossSiteScripting.html
- DBSQLInjection.html

View File

Viewing file: C:\Documents and Settings\Instructor\Desktop\WebGoat-5.2

Local intranet

Shopping cart direct object

Exploit Hidden Fields - Microsoft Internet Explorer

Address: http://localhost/WebGoat/attack?Screen=53&menu=1600

Logout ?

Exploit Hidden Fields

OWASP WebGoat V5.2

Navigation: < Hints > Show Params Show Cookies Lesson Plan Show Java Solution

Restart this Lesson

Solution Videos

Try to purchase the HDTV for less than the purchase price, if you have not done so already.

Shopping Cart

Shopping Cart Items -- To Buy Now	Price:	Quantity:	Total
56 inch HDTV (model KTV-551)	2999.99	<input type="text" value="1"/>	\$2999.99

The total charged to your credit card: \$2999.99

ASPECT SECURITY
Application Security Specialists

OWASP Foundation | Project WebGoat | Report Bug

Local intranet

Object reference issues

Map objects in server code

Many web apps use presentation layer security to “hide” sensitive functions

This approach is doomed to failure

Strive for a positive input validation whenever possible

Map exposed names to system objects on the server

Discard all others

OS-layer data access control and compartmentalization also highly useful

#5 Security misconfiguration (was 6)

Weakness in underlying components

Server, OS, framework, etc.

Can be just as damaging as a direct application weakness

Attackers don't care where a weakness is

Can be easier for an attacker to find

General, not specific to your app

Many are published

Can be easier to defend against also

IDS signatures, firewall rules

Defenses

Rigorous infrastructure testing

Penetration testing works well for this

Keep up with published reports

IT Security should be watching for these

Find the holes before the attacker does

Testbeds as well as production

Many products available to assist here

#6 Sensitive data exposure

Business software
routinely processes
sensitive data

Payment information

Customer information

Proprietary data

Application management
data

Potential exposures
abound

Failure to encrypt in
transit

Failure to encrypt stored
data

Poor crypto choices

Safe crypto usage

Crypto is a powerful tool for protecting data, but it is commonly misused in unsafe ways

Problems abound

- Key management

- Poorly chosen keys

- Inadequate algorithms

Remember “encoding” is not the same as “encrypting”

Encoding exercise

Encoding Basics - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Back Forward Stop Home Search Favorites Refresh Mail Print People

Address <http://localhost/WebGoat/attack?Screen=228&menu=1500> Go Links

Logout ?

OWASP WebGoat V5.2 < Hints > Show Params Show Cookies Lesson Plan Show Java Solution

Introduction
General
Access Control Flaws
AJAX Security
Authentication Flaws
Buffer Overflows
Code Quality
Concurrency
Cross-Site Scripting (XSS)
Denial of Service
Improper Error Handling
Injection Flaws
Insecure Communication
Insecure Configuration
Insecure Storage
[Encoding Basics](#)
Parameter Tampering
[Session Management Flaws](#)
Web Services
Admin Functions
Challenge

Solution Videos [Restart this Lesson](#)

This lesson will familiarize the user with different encoding schemes.

Enter a string:

Enter a password (optional):

Description	Encoded	Decoded
Base64 encoding is a simple reversible encoding used to encode bytes into ASCII characters. Useful for making bytes into a printable string, but provides no security.		
Entity encoding uses special sequences like & for special		

javascript:; Local intranet

Crypto issues

Sensitive data must be protected in transit and at rest

Protection should be proportional to the value of the data

Some tips

- Store keys in safe place

- Use strong keys that are not easily guessed

Use strong algorithms
Avoid re-using keys

Pretty basic, so why are so many mistakes made?

Insecure transport layer

This is the “in transit”
portion of insecure
crypto

Key management is
biggest problem

Exchanging keys
securely is where many
mistakes made

Information in URL
field is subject to
disclosure

Insecure comms issues

Issues are similar to other crypto issues

Key management is the big issue in crypto

Mutual authentication is highly advisable

SSL certificates on both sides

Not always feasible

Consider Wi-Fi model

#7 Missing function level access control

Many web apps lack even the most rudimentary access control

if authenticated then...is
NOT access control

Attackers are often times able to navigate to sensitive data/functions

Potential exposures abound

Non-privileged user accesses privileged functions or data

Data leakage across administrative boundaries

Access to URLs via “forced browsing”

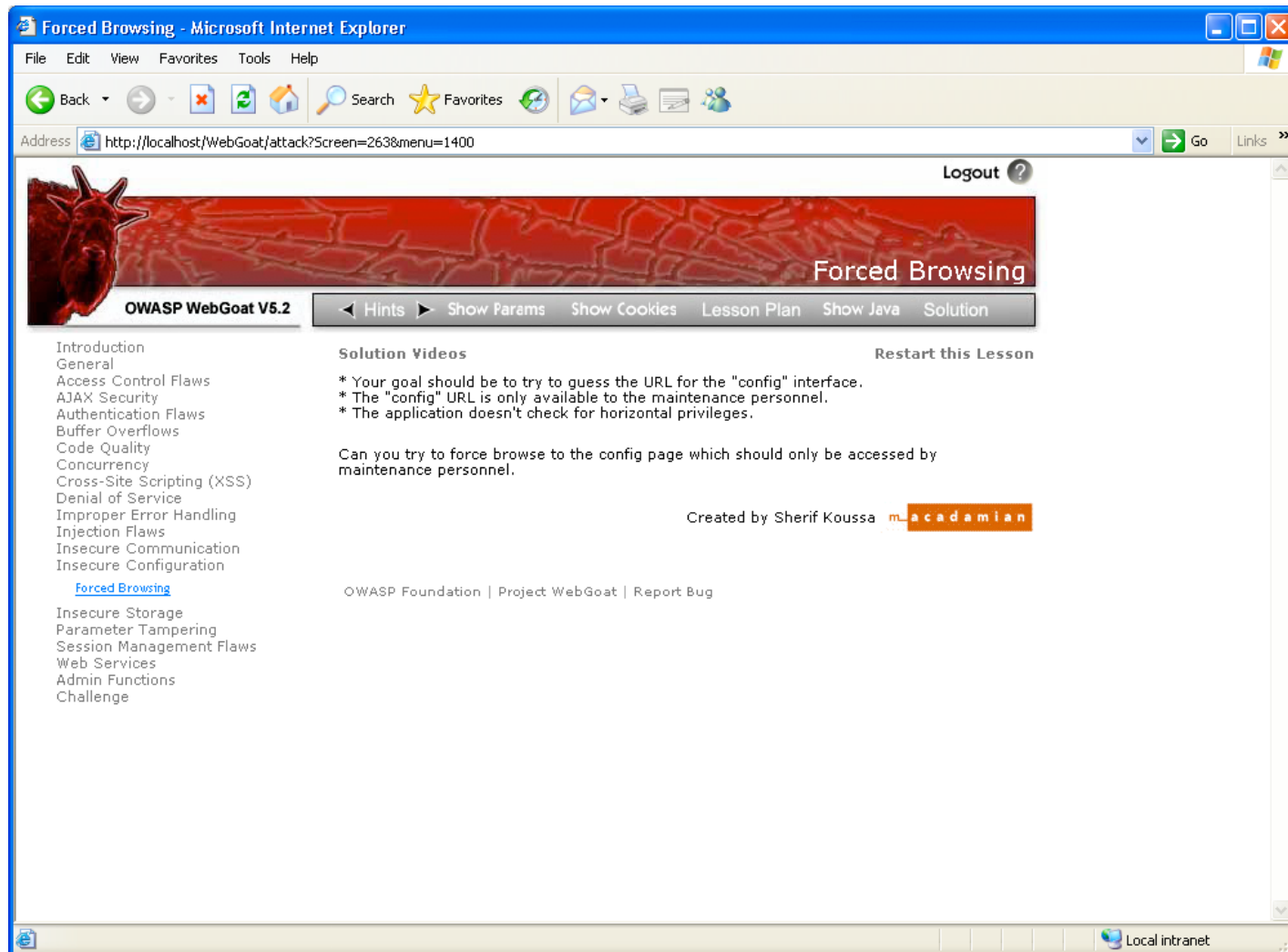
Access to URLs is most basic presentation layer control

Attackers only need a browser to guess URLs

Admin functions commonly “hidden” this way

“Forced browsing” attacks are pervasive and easy to automate

URL access exercise



URL access issues

Expect attackers to “spider” through your application’s folder/function tree

Expect attackers to experiment with HTML parameters via GET and POST

Presentation layer security is not sufficient

J2EE and .NET are a big help here

Access control fundamentals

Question every action

Is the user allowed to
access this

- File
- Function
- Data
- Etc.

By role or by user

Complexity issues

Maintainability issues

Creeping exceptions

Role-based access control

Must be planned
carefully

Clear definitions of

Users

Objects

Functions

Roles

Privileges

Plan for growth

Even when done well,
exceptions will happen

Access control matrix

Assets → Roles ↓	Admin Pages	Tax & Plan	Bill Pay	Public	Account Use	Account Admin
Administrators	X					
Owners				X	X	X
Guests				X		
Users				X	X	
Planners		X		X	X	X
Payers			X	X	X	X

Store information with ROLES and you've got a capabilities or permissions model

Store information with AS and you've got Access Control (ACL)

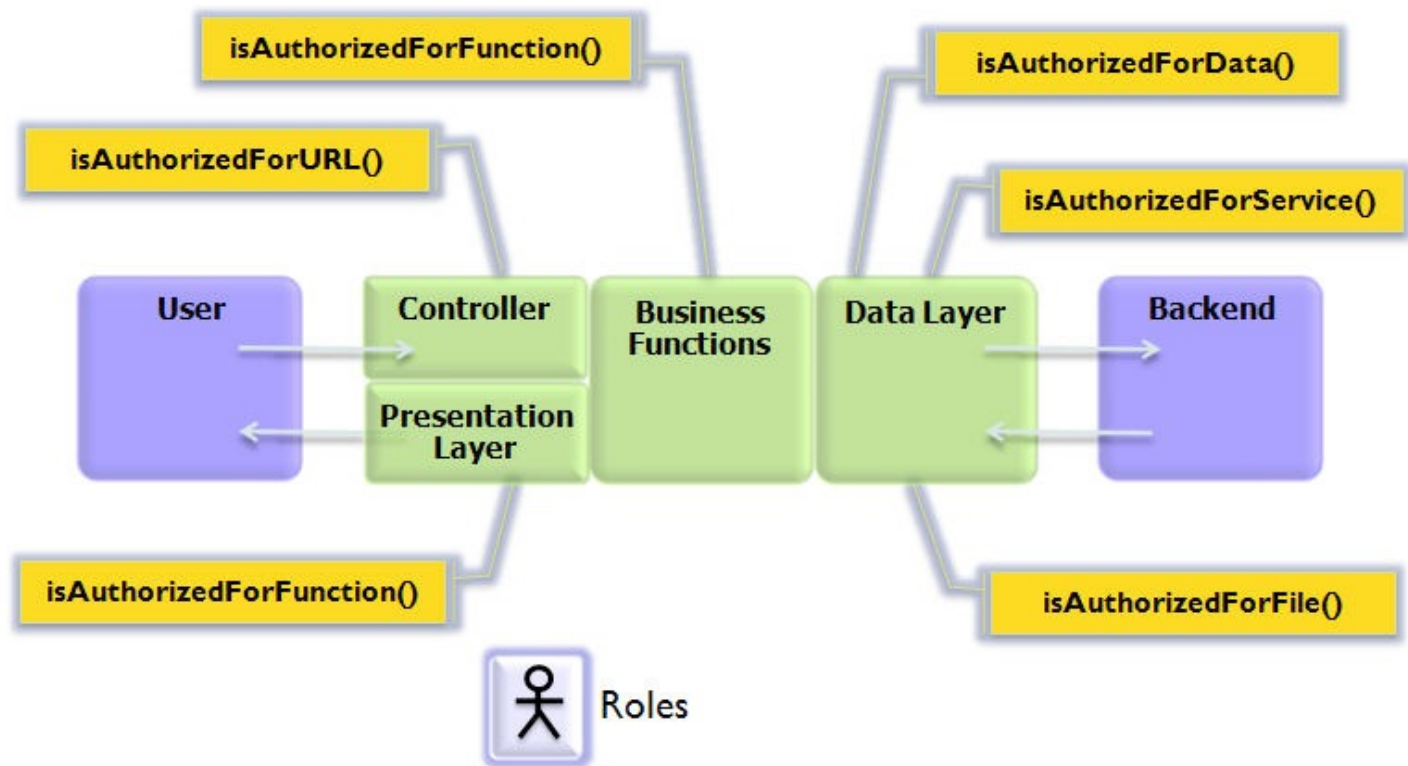
OWASP's ESAPI

OWASP Top Ten Coverage

OWASP Top Ten	OWASP ESAPI
A1. Cross Site Scripting (XSS)	Validator, Encoder
A2. Injection Flaws	Encoder
A3. Malicious File Execution	HTTPUtilities (upload)
A4. Insecure Direct Object Reference	AccessReferenceMap
A5. Cross Site Request Forgery (CSRF)	User (csrftoken)
A6. Leakage and Improper Error Handling	EnterpriseSecurityException, HTTPUtils
A7. Broken Authentication and Sessions	Authenticator, User, HTTPUtils
A8. Insecure Cryptographic Storage	Encryptor
A9. Insecure Communications	HTTPUtilities (secure cookie, channel)
A10. Failure to Restrict URL Access	AccessController

ESAPI access control

Enforcing Access Control



ESAPI access control

In the presentation layer:

```
<% if ( ESAPI.accessController().isAuthorizedForFunction( ADMIN_FUNCTION ) )
{ %>
  <a href="/doAdminFunction">ADMIN</a>
  <% } else { %>
  <a href="/doNormalFunction">NORMAL</a>
  <% } %>
```

In the business logic layer:

```
try {
    ESAPI.accessController().assertAuthorizedForFunction( BUSINESS_FUNCTION );
    // execute BUSINESS_FUNCTION
} catch (AccessControlException ace) {
    ... attack in progress
}
```

#8 Cross site request forgery (CSRF)

Relatively new, but
potentially disastrous

Attacker sends an
image request to victim

During an active session
on vulnerable app

Request may include
malicious parameters

Response may include
session cookie

Consider if the image
request arrived via
spam email

Emailer renders the
HTML and retrieves all
“images”

Occurs while web
browser is open and
logged into popular
banking site

CSRF exercise

Cross Site Request Forgery (CSRF) - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Address <http://localhost/WebGoat/attack?Screen=9&menu=900&Restart=9>

Logout ?

Cross Site Request Forgery (CSRF)

OWASP WebGoat V5.2

◀ Hints ▶ Show Params Show Cookies Lesson Plan Show Java Solution

Introduction
General
Access Control Flaws
AJAX Security
Authentication Flaws
Buffer Overflows
Code Quality
Concurrency
Cross-Site Scripting (XSS)
 Phishing with XSS
 LAB: Cross Site Scripting
 Stage 1: stored XSS
 Stage 2: Block stored XSS using Input Validation
 Stage 3: stored XSS Revisited
 Stage 4: Block stored XSS using Output Encoding
 Stage 5: Reflected XSS
 Stage 6: Block Reflected XSS
 Stored XSS Attacks
 Cross Site Request Forgery (CSRF)
 Reflected XSS Attacks
 HTTPOnly Test
 Cross Site Tracing (XST) Attacks
Denial of Service
Improper Error Handling
Injection Flaws

Solution Videos

Restart this Lesson

Your goal is to send an email to a newsgroup that contains an image whose URL is pointing to a malicious request. Try to include a 1x1 pixel image that includes a URL. The URL should point to the CSRF lesson with an extra parameter "transferFunds=4000". You can copy the shortcut from the left hand menu by right clicking on the left hand menu and choosing copy shortcut. Whoever receives this email and happens to be authenticated at that time will have his funds transferred. When you think the attack is successful, refresh the page and you will find the green check on the left hand side menu.

Title:

Message:

Submit

Message List

test

Created by Sherif Koussa [m_l_academian](#)

Local intranet

CSRF issues

What's the big deal?

`` can be used to hide commands other than images

Session cookies often have long timeout periods

Can redirect commands elsewhere on local network

Consider

`http://www.example.com/admin/doSomething.ctl?username=admin&password=admin`

Email delivery mechanism common

Further reading

www.owasp.org

CSRF remediation

OWASP says, “Applications must ensure that they are not relying on credentials or tokens that are automatically submitted by browsers. The only solution is to use a custom token that the browser will not ‘remember’ and then automatically include with a CSRF attack.”

This requires a lot of new coding

Very few existing web apps are protected

Phishers beginning to actively use this technique

CSRF Guard (from OWASP)

One solution set is freely available

Take a look at CSRF Guard

<http://www.owasp.org/index.php/>

Category:OWASP_CSRFGuard_Project

Uses a randomized token sent in a hidden HTML parameter – NOT auto by browser

Also look at CSRF Tester

<http://www.owasp.org/index.php/>

Category:OWASP_CSRFTester_Project

#9 Using components with known vulnerabilities

Application ingredient lists often include weak components

Older versions with published vulns

Fundamentally weak components

Applications often “advertise” their weaknesses

Server headers

Stack traces when exceptions not handled correctly

Developers using weak code

According to OWASP, the following two components were downloaded 22 million times in 2011

- Apache CXF
- Authentication Bypass
- Spring Remote Code Execution

See OWASP Top-10 2013 list for details

Remediations

The most important factor is vigilance

Keep up to date with component weaknesses and patches

Inventory of deployed components and versions

- Include all dependencies

Establish and enforce policies

Can't avoid vulnerable component

Remove the weak functions

- Remember to update when using new version

Wrappers to disable unused or weak functions

#10 Unvalidated Redirects and Forwards

Pages that take users to other URLs can be duped

Users think site is trustworthy

Comes from your domain

foo.com/redir.php?url=www.evil.com

Unchecked, can be used to send users to malicious sites

Malware launchpads

Target-rich environment for phishers

Am I vulnerable?

Review code for
redirects or forwards

If target URL is a
parameter, ensure
positive validation

Spider through site and
look for redirect
responses

Response code 300-307
(esp 302)

Fuzz test redirectors if
code isn't available

Better still

Avoid using redirects and forwards entirely

If you must, don't rely on user parameters

If parameters are essential, don't rely on what the user inputs

Positive input validation

ESAPI has a method for checking

`sendRedirect()`

OWASP 10 lessons

Key principles

Positive validation

Access control through entire app architecture

Session management

Protecting sensitive data at rest and in transit

Mutual authentication

Error handling

Logging

Defensive programming

Part II - Fix 'em!

WebGoat Dev Labs



Lab agenda

We'll do three hands-on labs

XSS remediation

SQL injection prevention

Role-based access control

Some background

Let's explore the WebGoat architecture a bit first

All source code is in our Eclipse project

- We'll edit source in Eclipse and use command line to re-build
`mvn clean tomcat:run-war`

Instructor and student versions

- Suggest refraining from looking at instructor code until after each lab

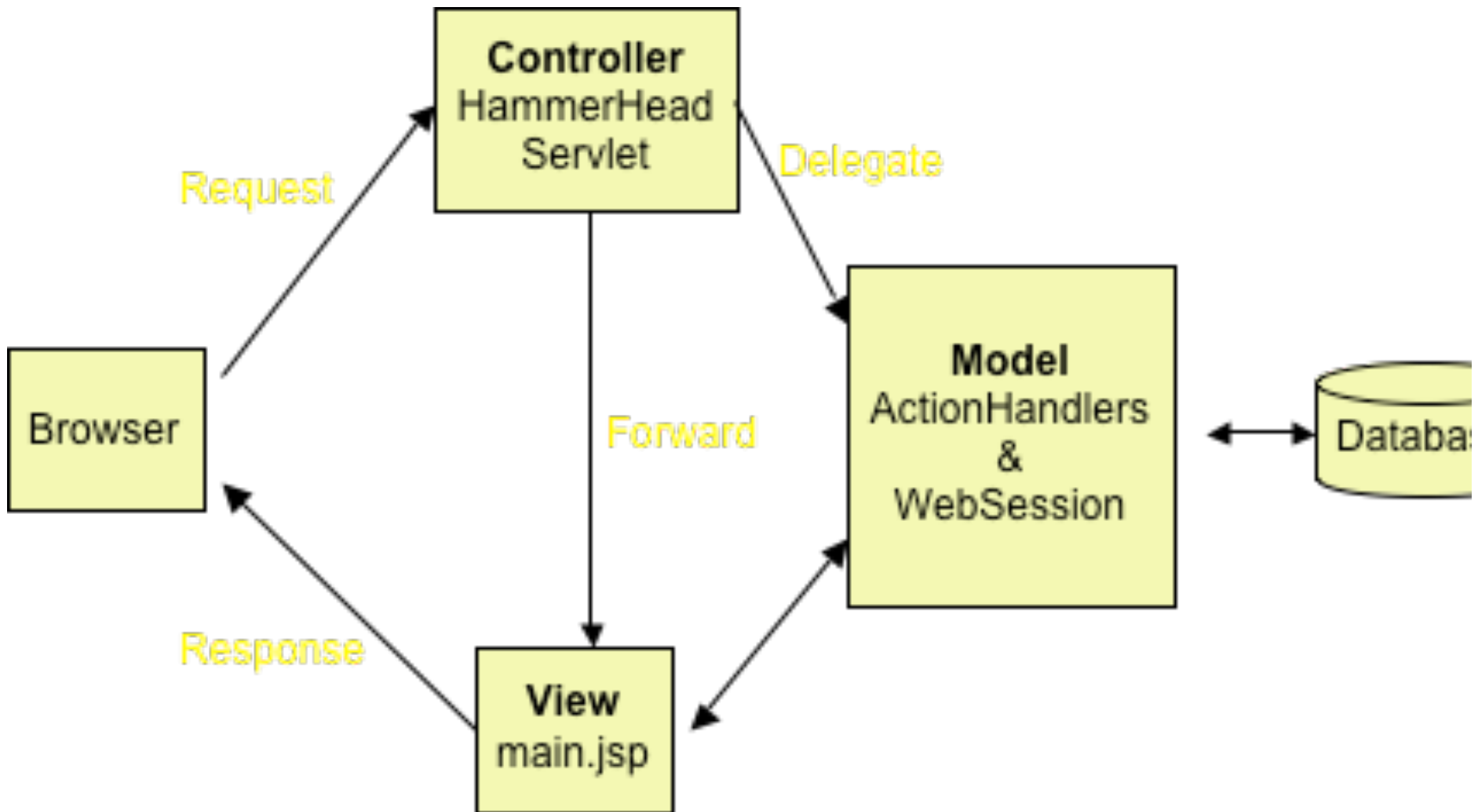
WebGoat architecture overview

All labs use a custom Action Handler that is invoked from the main WebGoat servlet, HammerHead.java

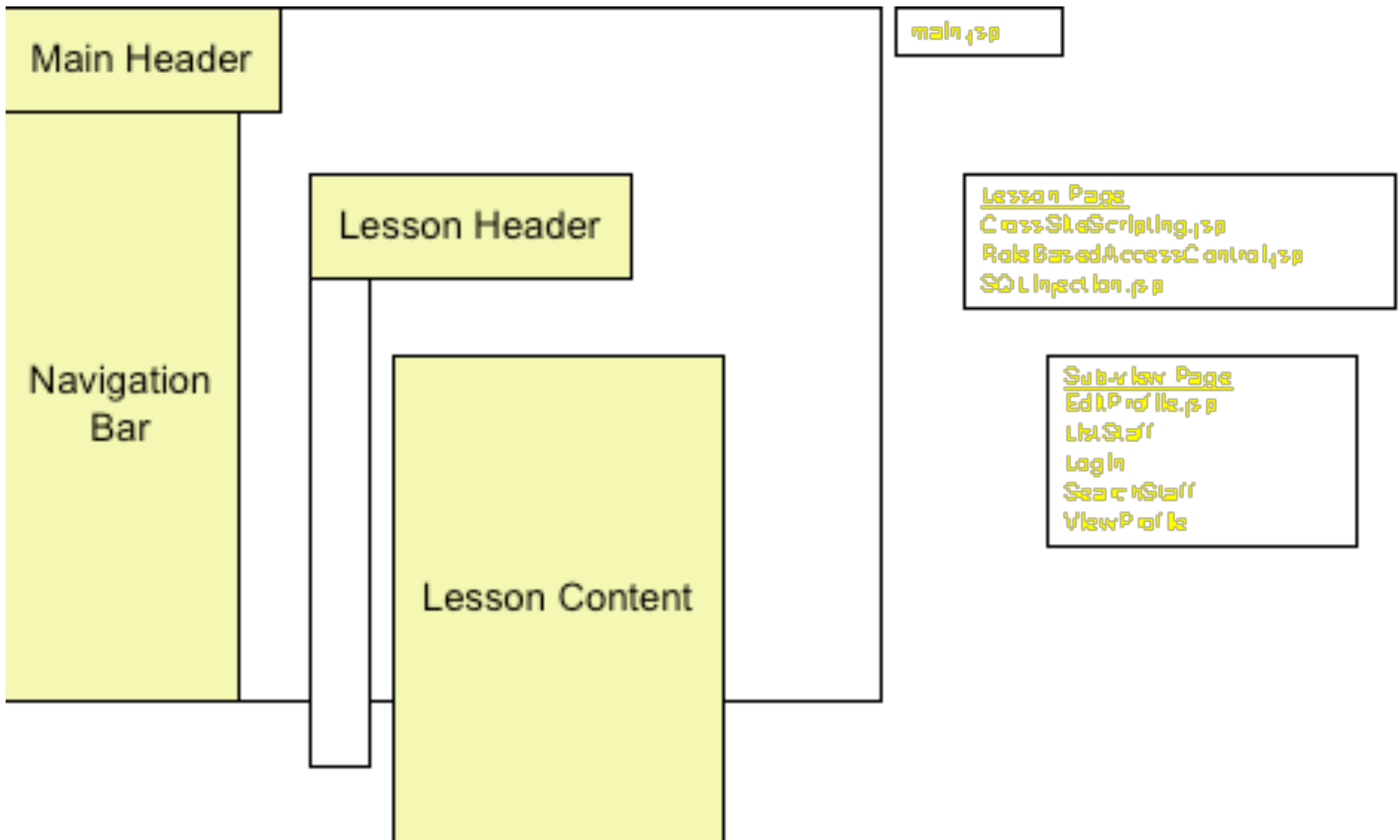
The handler will execute their business logic, load the data into the WebSession object, and then turn control over to the view component (JSP)

The WebGoat presentation only allows for a lesson to write into the Lesson Content portion of each page

WebGoat architecture



WebGoat page layout



Code layout

Each lab's action handlers are in a folder with same name

RoleBasedAccessControl lab is in

- org.owasp.webgoat.lessons.RoleBasedAccessControl

Various java classes for each lab function

JSP layout

All the JSPs are in
WebContent/Lessons/

Hint: only one lab requires modifying any JSPs

Backups are provided

Each lab class has a `_BACKUP` class
Contains original source for the class
Useful if things go badly wrong...

Let's take a look in Eclipse

Access control policy

- Overall Policy

Assets Roles	Search	List Staff	View Profile	Edit Profile	Create / Delete Profile
Employee	X	X (Self Only)	X	X (Portions)	
Manager	X	X	X		
HR	X	X	X	X (Others Only)	X
Admin	X	X	X	X	X

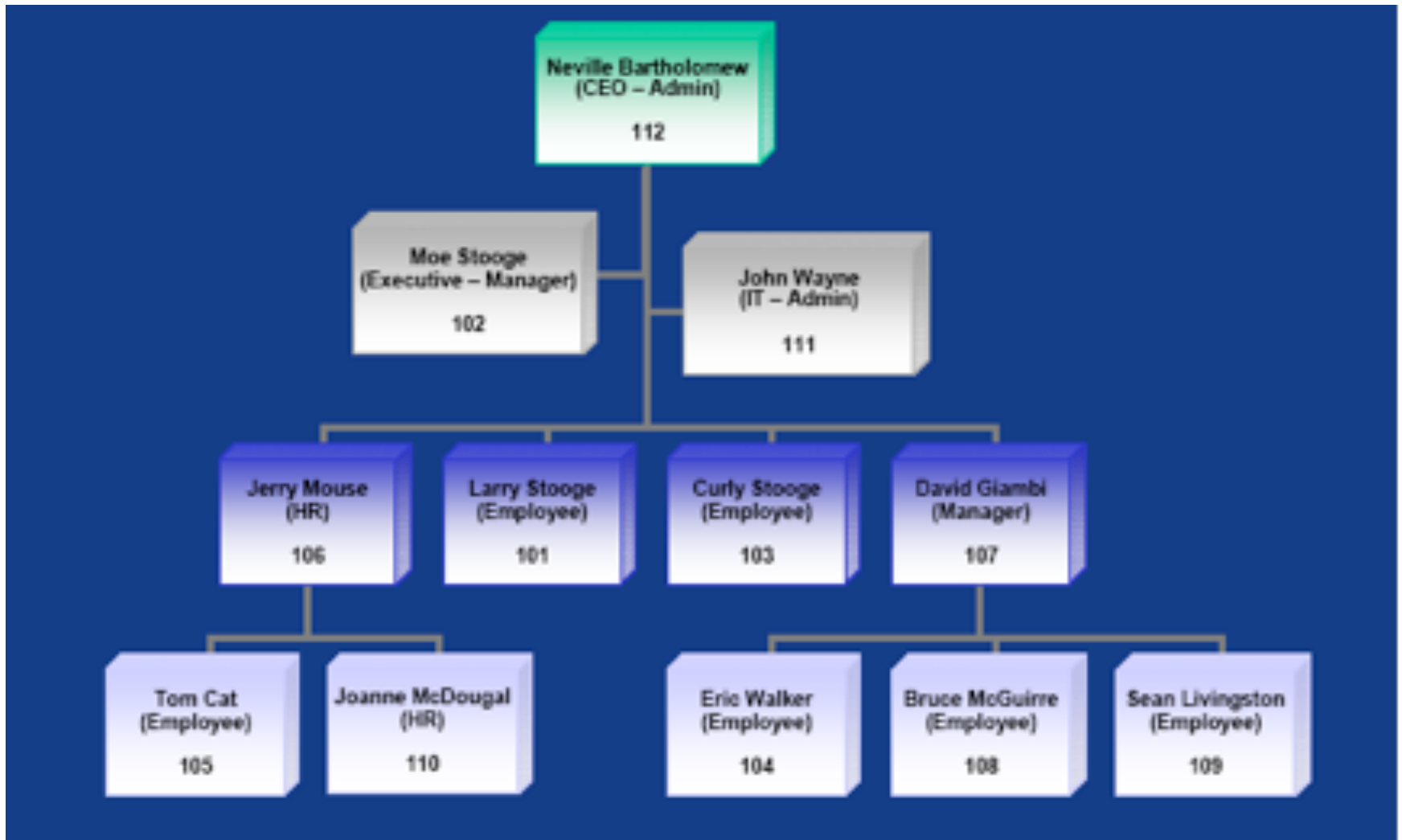
- Data Access Policy

- Employees can see their data
- Employees can edit portions of their data
- Managers can see their data and their employees' data
- HR can see and edit all employees. HR cannot edit their data

Database schema

- Employee
 - userid INT NOT NULL PRIMARY KEY
 - first_name VARCHAR(20)
 - last_name VARCHAR(20)
 - ssn VARCHAR(12)
 - password VARCHAR(10)
 - title VARCHAR(20)
 - phone VARCHAR(13)
 - address1 VARCHAR(80)
 - address2 VARCHAR(80)
 - manager INT
 - start_date CHAR(8)
 - salary INT
 - ccn VARCHAR(30)
 - ccn_limit INT
 - disciplined_date CHAR(8)
 - disciplined_notes VARCHAR(60)
 - personal_description VARCHAR(60)
- Roles
 - userid INT NOT NULL
 - role VARCHAR(10) NOT NULL
 - PRIMARY KEY (userid, role)
- Ownership
 - employer_id INT NOT NULL
 - employee_id INT NOT NULL
 - PRIMARY KEY (employee_id, employer_id)

Org chart for Goat Hills Financial



Lab 1: Cross-Site Scripting

LAB: Cross Site Scripting - Windows Internet Explorer

http://localhost/WebGoat/attack?Screen=338&menu=900

File Edit View Favorites Tools Help

LAB: Cross Site Scripting

Logout ?

LAB: Cross Site Scripting

OWASP WebGoat V5.2

Hints Show Params Show Cookies Lesson Plan Show Java Solution

Introduction
General
Access Control Flaws
AJAX Security
Authentication Flaws
Buffer Overflows
Code Quality
Concurrency
Cross-Site Scripting (XSS)
Phishing with XSS
LAB: Cross Site Scripting
Stage 1: Stored XSS
Stage 2: Block Stored XSS
Stage 3: Stored XSS
Stage 4: Block Stored XSS
Stage 5: Reflected XSS
Stage 6: Block Reflected XSS
Stored XSS Attacks
Cross Site Request Forgery (CSRF)
Reflected XSS Attacks
HTTPOnly Test
Cross Site Tracing (XST) Attacks
Denial of Service
Improper Error Handling
Injection Flaws
Insecure Communication
Insecure Configuration

Solution Videos Stage 1: Execute a Stored Cross Site Scripting (XSS) Restart this Lesson attack.
As 'Tom', execute a Stored XSS attack against the Street field on the Edit Profile page. Verify that 'Jerry' is affected by the attack.
The passwords for the accounts are the prenames.

Goat Hills Financial
Human Resources

Please Login

Larry Stogge (employee) [v]
Password []
Login

Local intranet 100%

Lab overview

Six stages

Stored XSS attack

Positive input validation using regex

Stored XSS attack redux

Output encoding

Reflected XSS attack

Positive input validation using regex

Stage 1

Login as “Tom”

Plant and execute a stored XSS attack on the Street field of the Edit Profile page

Verify “Jerry” is affected

Hint: All passwords are the users’ first names in lowercase

Note to self: don’t use first name as password

Stage 2

Block the XSS input using positive input validation

Hints

Start by looking in UpdateProfile action handler

- See request.getParameter calls in parseEmployeeProfile

Java.util.regex is your friend

Try it, then we'll step through the solution

Stage 3

Login as “David” and view “Bruce’s” profile

There’s an XSS attack already in Bruce’s data

Think that’ll get caught by the input validator?

Stage 4

Since it's too late for input validation, fix this one using output encoding

Hints

Look at output in JSP

htmlEncoder class in org.owasp.webgoat.util

Stage 5

Login as “Larry”

Use the Search Staff page to construct a reflected XSS attack

How could Larry attack another employee?

Stage 6

Use positive input validation to block this reflected XSS vulnerability

Hints

Same issues exist here re parsers and regex

Look through FindProfile to find where the name parameter is being input

Review checklist

Things to consider when reviewing software

Input validation on everything

- Centralized
- Easily maintained
- Regex-based

Consistently applied

Lab 2: SQL Injection

LAB: SQL Injection - Windows Internet Explorer

http://localhost/WebGoat/attack?Screen=54&menu=1200

File Edit View Favorites Tools Help

LAB: SQL Injection

Logout ?

OWASP WebGoat V5.2

Hints Show Params Show Cookies Lesson Plan Show Java Solution

Introduction
General
Access Control Flaws
AJAX Security
Authentication Flaws
Buffer Overflows
Code Quality
Concurrency
Cross-Site Scripting (XSS)
Denial of Service
Improper Error Handling
Injection Flaws
Command Injection
Blind SQL Injection
Numeric SQL Injection
Log Spoofing
XPath Injection
LAB: SQL Injection
Stage 1: String SQL Injection
Stage 2: Parameterized Query #1
Stage 3: Numeric SQL Injection
Stage 4: Parameterized Query #2
String SQL Injection
Database Backdoors
Insecure Communication
Insecure Configuration
Insecure Storage
Parameter Tampering
Session Management Flaws

Solution Videos Stage 1: Use String SQL Injection to bypass authentication. Use SQL injection to log in as the boss ('Neville') without using the correct password. Verify that Neville's profile can be viewed and that all functions are available (including Search, Create, and Delete). Restart this Lesson

Goat Hills Financial
Human Resources

Please Login

Larry Stooge (employee) Password Login

Done Local intranet 100%

Lab overview

Four stages

Use SQL injection to login as “Neville” without a correct password

Block SQL injection using a parameterized query

As “Larry,” use SQL injection to view “Neville’s” profile

Block SQL injection

Stage 1

Use a SQL string injection attack to login as the boss, “Neville”

WebScarab might be handy

Validate that all functions available to Neville are accessible

Stage 2

Look in Login handler

Alter the back-end SQL call

Change from concatenated string to parameterized query

- PreparedStatement is your friend

Stage 3

Login as “Larry”

Execute a numeric SQL injection in the View
function

Stage 4

This time it's in the ViewProfile action handler

Again, use a parameterized query to prevent the SQL injection from working

Review checklist

Look through all SQL connections

Must not ever be mutable

No user-supplied data can affect the intent

Static strings are OK

Lab 3: Access control

LAB: Role Based Access Control - Windows Internet Explorer

http://localhost/WebGoat/attack?Screen=37&menu=200

File Edit View Favorites Tools Help

LAB: Role Based Access Control

Logout ?

OWASP WebGoat V5.2

◀ Hints ▶ Show Params Show Cookies Lesson Plan Show Java Solution

Introduction
General
Access Control Flaws
[Using an Access Control Matrix](#)
[Bypass a Path Based Access Control Scheme](#)
[LAB: Role Based Access Control](#)
[Stage 1: Bypass Business Layer Access Control](#)
[Stage 2: Add Business Layer Access Control](#)
[Stage 3: Bypass Data Layer Access Control](#)
[Stage 4: Add Data Layer Access Control](#)
[Remote Admin Access](#)
AJAX Security
Authentication Flaws
Buffer Overflows
Code Quality
Concurrency
Cross-Site Scripting (XSS)
Denial of Service
Improper Error Handling
Injection Flaws
Insecure Communication
Insecure Configuration
Insecure Storage
Parameter Tampering
Session Management Flaws
Web Services
Admin Functions
Challenge

Solution Videos Stage 1: Bypass Presentational Layer Access Control. Restart this Lesson
As regular employee 'Tom', exploit weak access control to use the Delete function from the Staff List page. Verify that Tom's profile can be deleted. The password for a user is always his prename.

Goat Hills Financial
Human Resources

Please Login

Larry Stogge (employee) ▼
Password
Login

Done Local intranet 100%

Lab overview

Four stages

Bypass business layer access control

Add access control using RBAC

Bypass data layer access control

Add access control using RBAC

Stage 1

Login as “Tom”

Bypass access control in the Delete function in the Staff List page

Delete Tom’s profile

Stage 2

Look in the `handleRequest` method of the `RoleBasedAccessControl` handler

How is the action protecting for authorized access?

Look at `isAuthorized` method (using Eclipse)

Failures should throw `UnauthorizedException()`

Stage 3

Login as “Tom”

Exploit weak access control to View another employee’s profile

Stage 4

Implement data layer access control to block access to other users' profiles

Can build control programmatically or via better SQL

You can use the following method

`isAuthorizedForEmployee(s, userId, subjectUserID)`

Be sure to throw `UnauthorizedException` on failure

Review checklist

Look for RBAC structure (or other AC)

Look for consistent application of AC architecture

Focus review around most sensitive functions and data

Kenneth R. van Wyk
KRvW Associates, LLC

Ken@KRvW.com
<http://www.KRvW.com>
@KRvW

