

Cybersecurity Internship Report

Prepared by: Oriji Kelvin Promise

Slide 1: First we start by doing a spider scan with ZAP

The screenshot shows the ZAP interface during a spider scan. The top menu bar includes File, Edit, View, Analyse, Report, Tools, Import, Online, and Help. The main window has tabs for Standard Mode, Sites, Contexts, and Default Context. The Spider tab is selected, showing a progress bar at 100% completion for a scan of 'http://localhost:8081/WebGoat'. The results table lists 21 URLs found, with 14 added to the scope. The table columns are Processed, Method, URI, and Flags. Most URLs are marked as 'Out of Scope'.

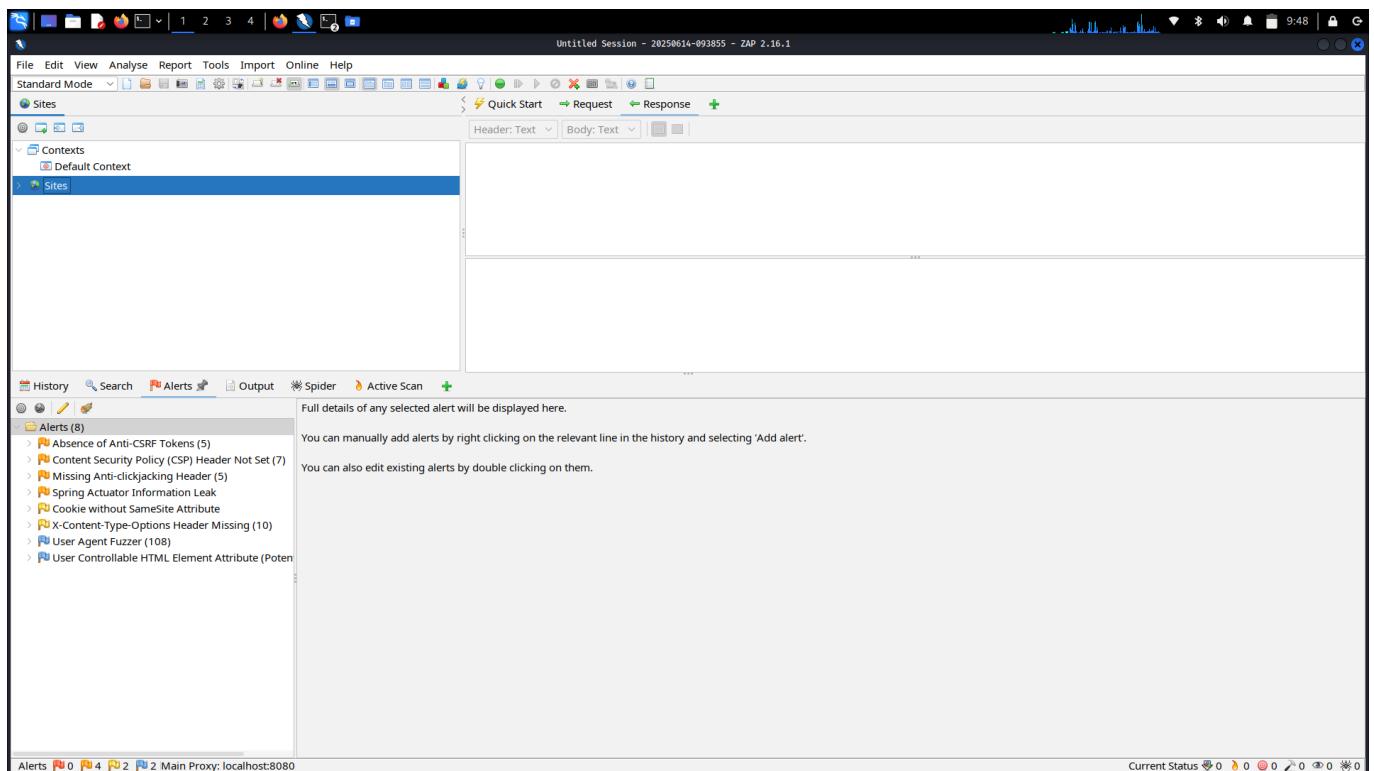
Processed	Method	URI	Flags
✓	GET	http://localhost:8081/WebGoat/	
✓	GET	http://localhost:8081/WebGoat/login	
✓	GET	http://localhost:8081/WebGoat/start.mvc	
✓	GET	http://localhost:8081/WebGoat/registration	
✓	GET	http://localhost:8081/WebGoat/css/img/favicon.ico	
✓	GET	http://localhost:8081/WebGoat/css/main.css	
✓	GET	http://localhost:8081/WebGoat/plugins/bootstrap/css/bootstrap.min.css	
✓	GET	http://localhost:8081/WebGoat/css/font-awesome.min.css	
✓	GET	http://localhost:8081/WebGoat/css/animate.css	
✓	POST	http://localhost:8081/WebGoat/login	
✗	GET	http://daneden.me/animate	Out of Scope
✗	GET	https://github.com/nickpettit/glide	Out of Scope
✗	GET	http://getbootstrap.com/	Out of Scope
✗	GET	https://github.com/twbs/bootstrap/blob/master/LICENSE	Out of Scope
✗	GET	http://fontawesome.io/	Out of Scope
✗	GET	http://fontawesome.io/license	Out of Scope
✗	POST	http://localhost:8081/WebGoat/register.mvc	Out of Scope
✗	GET	http://localhost:8081/WebGoat/login?error	Out of Scope

Slide 2: Next we do an active scan

The screenshot shows the ZAP interface with a network scan in progress. The left sidebar displays contexts and sites, with 'Sites' selected. The main pane shows a list of captured requests from the scan. The bottom navigation bar includes tabs for History, Search, Alerts, Output, Spider, Active Scan, and Export.

ID	Req. Timestamp	Resp. Timestamp	Method	URL	Code	Reason	RTT	Size Resp. Header	Size Resp. Body
2,233	6/14/25, 9:46:11 AM	6/14/25, 9:46:11 AM	GET	http://localhost:8081/WebGoat/plugins	200		5 ms	139 bytes	1,929 bytes
2,235	6/14/25, 9:46:11 AM	6/14/25, 9:46:11 AM	GET	http://localhost:8081/WebGoat/plugins/bootstrap/css	200		29 ms	139 bytes	1,929 bytes
2,237	6/14/25, 9:46:11 AM	6/14/25, 9:46:11 AM	GET	http://localhost:8081/WebGoat/plugins/bootstrap	200		39 ms	139 bytes	1,929 bytes
2,239	6/14/25, 9:46:11 AM	6/14/25, 9:46:11 AM	GET	http://localhost:8081/WebGoat/start.mvc	200		30 ms	139 bytes	1,929 bytes
2,241	6/14/25, 9:46:11 AM	6/14/25, 9:46:11 AM	GET	http://localhost:8081/WebGoat/plugins	200		11 ms	139 bytes	1,929 bytes
2,243	6/14/25, 9:46:11 AM	6/14/25, 9:46:11 AM	GET	http://localhost:8081/WebGoat/plugins/bootstrap/css	200		7 ms	139 bytes	1,929 bytes
2,245	6/14/25, 9:46:11 AM	6/14/25, 9:46:11 AM	GET	http://localhost:8081/WebGoat/plugins/bootstrap	200		36 ms	139 bytes	1,929 bytes
2,247	6/14/25, 9:46:11 AM	6/14/25, 9:46:11 AM	GET	http://localhost:8081/WebGoat/start.mvc	200		16 ms	139 bytes	1,929 bytes
2,249	6/14/25, 9:46:11 AM	6/14/25, 9:46:11 AM	GET	http://localhost:8081/WebGoat/plugins/bootstrap/css	200		11 ms	139 bytes	1,929 bytes
2,251	6/14/25, 9:46:11 AM	6/14/25, 9:46:11 AM	GET	http://localhost:8081/WebGoat/plugins/bootstrap	200		12 ms	139 bytes	1,929 bytes
2,253	6/14/25, 9:46:11 AM	6/14/25, 9:46:11 AM	GET	http://localhost:8081/WebGoat/start.mvc	200		8 ms	139 bytes	1,929 bytes
2,255	6/14/25, 9:46:11 AM	6/14/25, 9:46:11 AM	GET	http://localhost:8081/WebGoat/plugins/bootstrap/css	200		8 ms	139 bytes	1,929 bytes
2,257	6/14/25, 9:46:11 AM	6/14/25, 9:46:11 AM	GET	http://localhost:8081/WebGoat/start.mvc	200		23 ms	139 bytes	1,929 bytes
2,259	6/14/25, 9:46:11 AM	6/14/25, 9:46:11 AM	GET	http://localhost:8081/WebGoat/plugins/bootstrap	200		17 ms	139 bytes	1,929 bytes
2,261	6/14/25, 9:46:11 AM	6/14/25, 9:46:11 AM	GET	http://localhost:8081/WebGoat/start.mvc	200		4 ms	139 bytes	1,929 bytes
2,263	6/14/25, 9:46:11 AM	6/14/25, 9:46:11 AM	GET	http://localhost:8081/WebGoat/start.mvc	200		7 ms	139 bytes	1,929 bytes
2,265	6/14/25, 9:46:11 AM	6/14/25, 9:46:11 AM	GET	http://localhost:8081/WebGoat/start.mvc	200		9 ms	139 bytes	1,929 bytes
2,267	6/14/25, 9:46:11 AM	6/14/25, 9:46:11 AM	GET	http://localhost:8081/WebGoat/start.mvc	200		5 ms	139 bytes	1,929 bytes

Slide 3: This is the alerts tab



Slide 4: [Missing Image Placeholder]

The screenshot shows the ZAP 2.16.1 interface. The top menu bar includes File, Edit, View, Analyse, Report, Tools, Import, Online, and Help. The main window has tabs for Standard Mode, Sites, Contexts, and Requests. The Requests tab is active, showing a captured HTTP request for 'Default Context'. The response header is displayed as:

```
HTTP/1.1 200
Content-Type: text/html; charset=UTF-8
Content-Language: en-US
Date: Sat, 14 Jun 2025 08:45:27 GMT
Content-Length: 1929
```

The response body contains the following HTML code:

```
</header>
<section class="main-content-wrapper">
<section id="main-content">

<br/><br/>
<form action="/WebGoat/login" method="POST" style="width: 200px;">
```

The left sidebar shows the 'Alerts' section with 8 items, including:

- Absence of Anti-CSRF Tokens (5)
- Content Security Policy (CSP) Header Not Set (7)
- Missing Anti-clickjacking Header (5)
- Spring Actuator Information Leak
- Cookie without SameSite Attribute
- X-Content-Type-Options Header Missing (10)
- User Agent Fuzzer (108)
- User Controllable HTML Element Attribute (Poten

The 'Absence of Anti-CSRF Tokens' alert is expanded, showing details such as URL (http://localhost:8081/WebGoat), Risk (Medium), Confidence (Low), Attack (No Anti-CSRF tokens were found in a HTML submission form.), Evidence (<form action="/WebGoat/login" method="POST" style="width: 200px;">), CWE ID (352), WASC ID (9), Source (Passive (10202 - Absence of Anti-CSRF Tokens)), Input Vector (No Anti-CSRF token [anticsrf, CSRFToken, __RequestVerificationToken, csrfmiddlewaretoken, authenticity_token, OWASP_CSRFTOKEN, anoncsrf, csrf_token, _csrf, __csrfSecret, __csrf_magic, CSRF, _token, _csrf_token, _csrfToken] was found in the following HTML form: [Form 1: "exampleInputEmail1" "exampleInputPassword1"].), and Solution (Phase: Architecture and Design). A note at the bottom of the alert panel states: 'Use a suitable library or framework that does not allow this weakness to occur or provide a construct that makes this weakness easier to avoid.'

Slide 5: We try to exploit vulnerabilities on WebGoat using SQL injection

localhost:8080/WebGoat/start.mvc?username=kelvin#lesson/SqlInjection.lesson/1

SQL Injection (mitigation)

Cross Site Scripting

Cross Site Scripting (stored)

Cross Site Scripting (mitigation)

Path traversal

(A5) Security Misconfiguration >

(A6) Vuln & Outdated Components >

(A7) Identity & Auth Failure >

(A8) Software & Data Integrity >

(A9) Security Logging Failures >

(A10) Server-side Request Forgery >

Client side >

Challenges >

What is SQL?

SQL is a standardized (ANSI in 1986, ISO in 1987) programming language which is used for managing relational databases and performing various operations on the data in them.

A database is a collection of data. The data is organized into rows, columns and tables, and indexed to make finding relevant information more efficient.

Example SQL table containing employee data; the name of the table is 'employees':

Employees Table
userid first_name last_name department salary auth_tan
32147 Paulina Travers Accounting \$46.000 P45JSI
89762 Tobi Barnett Development \$77.000 TA9LL1
96134 Bob Franco Marketing \$83.700 LO9S2V
34477 Abraham Holman Development \$50.000 UU2ALK
37648 John Smith Marketing \$64.350 3SL99A

A company saves the following employee information in their databases: a unique employee number ('userid'), last name, first name, department, salary and a transaction authentication number ('auth_tan'). Each of these pieces of information is stored in a separate column and each row represents one employee of the company.

SQL queries can be used to modify a database table and its index structures and add, update and delete rows of data.

There are three main categories of SQL commands:

- Data Manipulation Language (DML)
- Data Definition Language (DDL)
- Data Control Language (DCL)

Each of these command types can be used by attackers to compromise the confidentiality, integrity, and/or availability of a system. Proceed with the lesson to learn more about the SQL command types and how they relate to protections goals.

If you are still struggling with SQL and need more information or practice, you can visit <http://www.sqlcourse.com/> for free and interactive online training.

It is your turn!

Look at the example table. Try to retrieve the department of the employee Bob Franco. Note that you have been granted full administrator privileges in this assignment and can access all data without authentication.

SQL query

```
SELECT department FROM employees WHERE first_name='Bob'
```

Submit

You have succeeded!

```
SELECT department FROM employees WHERE first_name='Bob'
```

DEPARTMENT

Marketing

Slide 6: Evidence of SQL Injection Exploitation - Injected payload and bypassed login.

Kali Linux

WebGoat

localhost:8080/WebGoat/start.mvc?username=kelvin#lesson/SqlInjection.lesson/2

Introduction

General

(A1) Broken Access Control

(A2) Cryptographic Failures

(A3) Injection

SQL Injection (intro)

SQL Injection (advanced)

SQL Injection (mitigation)

Cross Site Scripting

Cross Site Scripting (stored)

Cross Site Scripting (mitigation)

Path Traversal

(A5) Security Misconfiguration

(A6) Vuln & Outdated Components

(A7) Identity & Auth Failure

(A8) Software & Data Integrity

(A9) Security Logging Failures

(A10) Server-side Request Forgery

Client side

Challenges

Show hints

Reset lesson

1 2 3 4 5 6 7 8 9 10 11 12 13

Data Manipulation Language (DML)

As implied by the name, data manipulation language deals with the manipulation of data. Many of the most common SQL statements, including SELECT, INSERT, UPDATE, and DELETE, may be categorized as DML statements. DML statements may be used for requesting records (SELECT), adding records (INSERT), deleting records (DELETE), and modifying existing records (UPDATE).

If an attacker succeeds in "injecting" DML statements into a SQL database, he can violate the confidentiality (using SELECT statements), integrity (using UPDATE statements), and availability (using DELETE or UPDATE statements) of a system.

- DML commands are used for storing, retrieving, modifying, and deleting data.
- SELECT - retrieve data from a database
- INSERT - insert data into a database
- UPDATE - updates existing data within a database
- DELETE - delete records from a database

Example:

- Retrieve data:

```
SELECT phone  
FROM employees  
WHERE user_id = 96134;
```

This statement retrieves the phone number of the employee who has the user_id 96134.

It is your turn!

Try to change the department of Tobi Barnett to 'Sales'. Note that you have been granted full administrator privileges in this assignment and can access all data without authentication.

SQL query

UPDATE employees SET DEPARTMENT='Sales' WHERE first_name='Tobi'

Submit

Congratulations. You have successfully completed the assignment.
UPDATE employees SET DEPARTMENT='Sales' WHERE first_name='Tobi'
USERID FIRST_NAME LAST_NAME DEPARTMENT SALARY AUTH_TAN

USERID	FIRST_NAME	LAST_NAME	DEPARTMENT	SALARY	AUTH_TAN
89762	Tobi	Barnett	Sales	77000	TA9LL1

Slide 7: Further SQL Injection - Update operations on user credentials.

The screenshot shows a Kali Linux desktop environment with a Firefox browser window open to the 'WebGoat' application at `localhost:8080/WebGoat/start.mvc?username=kelvin#lesson/SqliInjection.lesson/3`. The browser title bar says 'WebGoat'. The main content area is titled 'SQL Injection (intro)'. On the left, there's a sidebar with a navigation tree. The current section is 'SQL Injection (intro)', which is highlighted in blue. Other sections include 'SQL Injection (advanced)', 'SQL Injection (mitigation)', 'Cross Site Scripting', 'Cross Site Scripting (stored)', 'Cross Site Scripting (mitigation)', 'Path traversal', 'A5 Security Misconfiguration', 'A6 Vuln & Outdated Components', 'A7 Identity & Auth Failure', 'A8 Software & Data Integrity', 'A9 Security Logging Failures', and 'A10 Server-side Request Forgery'. Below the sidebar, there's a breadcrumb trail: 'Introduction > General > (A1) Broken Access Control > (A2) Cryptographic Failures > (A3) Injection > SQL Injection (intro)'. The main content area has a heading 'Data Definition Language (DDL)'. It defines DDL as commands for defining data structures and lists objects like tables, indexes, views, relationships, triggers, and more. It notes that successful injection can violate integrity and availability. A bullet list details DDL commands: CREATE (for creating objects), ALTER (for altering database structure), and DROP (for deleting objects). An example shows the creation of a 'employees' table with columns for userid, first_name, last_name, department, salary, and auth_tan. A note states that this statement creates the 'employees' table mentioned on page 2. Below this, a message says 'Now try to modify the schema by adding the column "phone" (varchar(20)) to the table "employees". :'. A text input field labeled 'SQL query' contains the command `ALTER TABLE employees ADD COLUMN phone varchar(20)`. A 'Submit' button is below the input field. A success message in a green box says 'Congratulations. You have successfully completed the assignment.' followed by the executed SQL command.

Slide 8: Demonstration of SQL DDL Injection - Altering database structure.

The screenshot shows a Kali Linux desktop with a Firefox browser open to the 'WebGoat' application at `localhost:8080/WebGoat/start.mvc?username=kelvin#lesson/Sqllnjection.lesson/4`. The browser title bar says 'WebGoat'. The main content area is titled 'SQL Injection (intro)' and discusses Data Control Language (DCL). It includes a list of bullet points about DCL commands (GRANT, REVOKE), a note about attackers injecting DCL commands, and a SQL query input field containing:

```
✓
SQL
query   GRANT ALL ON grant_rights TO unauthorized_user
Submit
```

Below the input field, a message says: 'Congratulations. You have successfully completed the assignment.'

Slide 9: Evidence of SQL DCL Injection - Granting privileges.

The screenshot shows a Firefox browser window on a Kali Linux desktop. The title bar says 'WebGoat'. The address bar shows 'localhost:8080/WebGoat/start.mvc?username=kelvin#lesson/SqlInjection.lesson/8'. The page content is titled 'SQL Injection (intro)'. On the left, there's a sidebar with a navigation tree. The main area has a heading 'Try It! String SQL injection' and a code editor containing a SQL query. Below it is a table of user data. At the bottom, there's an explanation of the exploit.

SQL Injection (intro)

Introduction >
General >
(A1) Broken Access Control >
(A2) Cryptographic Failures >
(A3) Injection >
SQL Injection (intro) >
SQL Injection (advanced) >
SQL Injection (mitigation) >
Cross Site Scripting >
Cross Site Scripting (stored) >
Cross Site Scripting (mitigation) >
Path traversal >
(A5) Security Misconfiguration >
(A6) Vuln & Outdated Components >
(A7) Identity & Auth Failure >
(A8) Software & Data Integrity >
(A9) Security Logging Failures >
(A10) Server-side Request Forgery >
Client side >
Challenges >

Try It! String SQL injection

The query in the code builds a dynamic query as seen in the previous example. The query is built by concatenating strings making it susceptible to String SQL injection:

```
"SELECT * FROM user_data WHERE first_name = 'John' AND last_name = '" + lastName + "'";
```

Try using the form below to retrieve all the users from the users table. You should not need to know any specific user name to get the complete list.

SELECT * FROM user_data WHERE first_name = 'John' AND last_name = ' or '1' = '1' > [Get Account Info]

You have succeeded:
USERID, FIRST_NAME, LAST_NAME, CC_NUMBER, CC_TYPE, COOKIE, LOGIN_COUNT,
101, Joe, Snow, 987654321, VISA, ,
101, Joe, Snow, 2234200065411, MC, ,
102, John, Smith, 2435600002222, MC, ,
102, John, Smith, 43522090902222, AMEX, ,
103, Jane, Plane, 123456789, MC, ,
103, Jane, Plane, 333496703333, AMEX, ,
10312, Jolly, Hershey, 176896789, MC, ,
10312, Jolly, Hershey, 333300003333, AMEX, ,
10323, Grumpy, youaretheweakestlink, 673834489, MC, ,
10323, Grumpy, youaretheweakestlink, 33413003333, AMEX, ,
15603, Peter, Sand, 123609789, MC, ,
15603, Peter, Sand, 338893453333, AMEX, ,
15613, Joseph, Something, 33843453333, AMEX, ,
15837, Chaos, Monkey, 3249386533, CM, ,
19204, Mr, Goat, 33812953533, VISA, ,
Your query was: SELECT * FROM user_data WHERE first_name = 'John' and last_name = " or '1' = '1'
Explanation: This injection works, because 'or '1' = '1' always evaluates to true (The string ending literal for '1' is closed by the query itself, so you should not inject it). So the injected query basically looks like this: SELECT * FROM user_data WHERE (first_name = 'John' and last_name = '') or (TRUE), which will always evaluate to true, no matter what came before it.

Slide 10: Confirming SQL SELECT * Injection - Extracting full records.

The screenshot shows a Firefox browser window on a Kali Linux desktop. The URL is `localhost:8080/WebGoat/start.mvc?username=kelvin#lesson/SqlInjection.lesson/9`. The page title is "SQL Injection (Intro)". On the left, there's a sidebar with various security categories like "Introduction", "General", "Broken Access Control", etc. The main content area has a heading "Try It! Numeric SQL injection". It contains a query box with the following code:

```
"SELECT * FROM user_data WHERE login_count = " + Login_Count + " AND userId = " + User_ID;
```

Below the code, it says "Using the two Input Fields below, try to retrieve all the data from the users table." and "Warning: Only one of these fields is susceptible to SQL Injection. You need to find out which, to successfully retrieve all the data." There are two input fields: "Login_Count" with value "0" and "User_Id" with value "0 or 1=1". A button labeled "Get Account Info" is present. The "Get Account Info" button is highlighted with a red border. Below the input fields, a message says "You have succeeded:" followed by a long list of user records. At the bottom, it says "Your query was: SELECT * From user_data WHERE Login_Count = 0 and userId= 0 or 1=1".

Slide 11: Modifying salary via SQL Injection chaining.

The screenshot shows a Kali Linux desktop environment with a Firefox browser open to the 'WebGoat' challenge at `localhost:8080/WebGoat/start.mvc?username=kelvin#lesson/SqliInjection.lesson/10`. The browser title bar says 'WebGoat'. The sidebar on the left contains a navigation menu with the following items:

- SQL Injection (mitigation)
- Cross Site Scripting
- Cross Site Scripting (stored)
- Cross Site Scripting (mitigation)
- Path traversal
- (A5) Security Misconfiguration >
- (A6) Vuln & Outdated Components >
- (A7) Identity & Auth Failure >
- (A8) Software & Data Integrity >
- (A9) Security Logging Failures >
- (A10) Server-side Request Forgery >
- Client side >
- Challenges >

The main content area displays the 'Compromising confidentiality with String SQL injection' lesson. It includes a note about SQL injection being a common vulnerability and how it can be exploited. A section titled 'What is String SQL injection?' provides a brief explanation. Below this, a 'It is your turn!' section asks the user to exploit the system to view all employee data. The user is prompted to enter their 'Employee Name' (set to '`' or 1=1 --`') and 'Authentication TAN' (set to '3SL99A'). A button labeled 'Get department' is present. A success message at the bottom states: 'You have succeeded! You successfully compromised the confidentiality of data by viewing internal information that you should not have access to. Well done!'. A table below shows the database results:

USERID	FIRST_NAME	LAST_NAME	DEPARTMENT	SALARY	AUTH_TAN	PHONE
32147	Paulina	Travers	Accounting	46000	P45JSI	null
34477	Abraham	Holman	Development	50000	UU2ALK	null
37648	John	Smith	Marketing	64350	3SL99A	null
89762	Tobi	Barnett	Sales	77000	TA9LL1	null
96134	Bob	Franco	Marketing	83700	LO9S2V	null

Slide 12: SQL Injection combined with multiple statements.

The screenshot shows a Firefox browser window with the address bar at `localhost:8080/WebGoat/start.mvc?username=kelvin#lesson/SqliInjection.lesson/11`. The left sidebar contains a navigation tree with categories like 'Injection', 'Broken Access Control', 'Cryptographic Failures', 'Injection (intro)', 'Injection (advanced)', etc. A breadcrumb trail at the top right shows 'Kali Linux' → 'WebGoat' → 'SqliInjection.lesson/11'. The main content area has a heading 'Compromising Integrity with Query chaining'. It includes a note about SQL query chaining and a success message: 'Well done! Now you are earning the most money. And at the same time you successfully compromised the integrity of data by changing your salary!'. Below this is a table of employee data:

USERID	FIRST_NAME	LAST_NAME	DEPARTMENT	SALARY	AUTH_TAN	PHONE
37648	John	Smith	Marketing	999999	3SL99A	null
96134	Bob	Franco	Marketing	83700	L09S2V	null
89762	Tobi	Barnett	Sales	77000	TA9LL1	null
34477	Abraham	Holman	Development	50000	UU2ALK	null
32147	Paulina	Travers	Accounting	46000	P45JSI	null

Below the table is a form with fields: 'Employee Name:' (containing '99 where userid = 37648 --'), 'Authentication TAN:' (containing 'TAN'), and a 'Get department' button. The page also includes a note: 'Remember: Your name is John Smith and your current TAN is 3SL99A.'

Slide 13: Completing all exploitation steps for SQL Injection in WebGoat.

Kali Linux

WebGoat

localhost:8080/WebGoat/start.mvc?username=kelvin#lesson/SqliInjection.lesson/12

Kali Linux Kali Tools Kali Docs Kali Forums Kali NetHunter Exploit-DB OffSec Foundations of Cybersecurity Cybersecurity Certifica... Courses | EC-Council L... cc - ObriZum Snapcraft - Snaps are ...

SQL Injection (intro)

Introduction
General
(A1) Broken Access Control
(A2) Cryptographic Failures
(A3) Injection
SQL Injection (intro)
SQL Injection (advanced)
SQL Injection (mitigation)
Cross Site Scripting
Cross Site Scripting (stored)
Cross Site Scripting (mitigation)
Path traversal
(A5) Security Misconfiguration
(A6) Vuln & Outdated Components
(A7) Identity & Auth Failure
(A8) Software & Data Integrity
(A9) Security Logging Failures
(A10) Server-side Request Forgery
Client side
Challenges

Show hints Reset lesson

1 2 3 4 5 6 7 8 9 10 11 12 13

Compromising Availability

After successfully compromising confidentiality and integrity in the previous lessons, we are now going to compromise the third element of the CIA triad: **availability**. There are many different ways to violate availability. If an account is deleted or its password gets changed, the actual owner cannot access this account anymore. Attackers could also try to delete parts of the database, or even drop the whole database, in order to make the data inaccessible. Revoking the access rights of admins or other users is yet another way to compromise availability; this would prevent these users from accessing either specific parts of the database or even the entire database as a whole.

It is your turn!

Now you are the top earner in your company. But do you see that? There seems to be a **access_log** table, where all your actions have been logged to! Better go and *delete it* completely before anyone notices.

Action contains: %';drop table access_log;

Search logs

Success! You successfully deleted the access_log table and that way compromised the availability of the data.

Slide 14: Next we try to exploit vulnerabilities using Cross Site Scripting (XSS).

The screenshot shows a Kali Linux desktop environment with a Firefox browser window open. The URL in the address bar is `localhost:8080/WebGoat/start.mvc?username=kelvin#lesson/CrossSiteScripting.lesson/1`. The page title is "Cross Site Scripting". On the left, there is a sidebar with a navigation menu for the WebGoat application, including sections like Introduction, General, A1 Broken Access Control, A2 Cryptographic Failures, A3 Injection, SQL Injection (intro), SQL Injection (advanced), SQL Injection (mitigation), Cross Site Scripting, Cross Site Scripting (stored), Cross Site Scripting (mitigation), Path traversal, A5 Security Misconfiguration, A6 Vuln & Outdated Components, A7 Identity & Auth Failure, A8 Software & Data Integrity, A9 Security Logging Failures, and A10 Server-side Request Forgery. The "Cross Site Scripting" section is currently selected. The main content area displays the "What is XSS?" section, which defines XSS as a vulnerability that combines the allowance of HTML/script tags as input that renders into a browser without encoding or sanitization. It also states that XSS is the most prevalent and pernicious web application security issue. Below this, there is a "Quick examples:" section with two bullet points: "From the JavaScript console in the developer tools of the browser (Chrome, Firefox)" and "Any data field returned to the client is potentially injectable". Under the first point, there is a code snippet:

```
alert("XSS Test");
alert(document.cookie);
```

 Under the second point, there is another code snippet:

```
<script>alert("XSS Test")</script>
```

. At the bottom of the content area, there is a "Try It! Using Chrome or Firefox" section with instructions: "Open a second tab and use the same URL as this page you are currently on (or any URL within this instance of WebGoat). On the second tab, open the JavaScript console in the developer tools and type: `alert(document.cookie)`. The cookies should be the same on each tab." There is a checkbox labeled "The cookies are the same on each tab" followed by a "Submit" button. Below the "Submit" button, a message says "Congratulations. You have successfully completed the assignment."

Kali Linux Kali Tools Kali Docs Kali Forums Kali NetHunter Exploit-DB Google Hacking DB OffSec Foundations of Cybersecurity Cybersecurity Certifications Courses | EC-Council L... cc - Obirizum Snapcraft - Snaps are ...

localhost:8080/WebGoat/start.mvc?username=kelvinHlesson/CrossSiteScripting.lesson/6

Cross Site Scripting (stored)
Cross Site Scripting (mitigation)
Path traversal
(A5) Security Misconfiguration >
(A6) Vuln & Outdated Components >
(A7) Identity & Auth Failure >
(A8) Software & Data Integrity >
(A9) Security Logging Failures >
(A10) Server-side Request Forgery >
Client side
Challenges >

It is always a good practice to validate all input on the server side. XSS can occur when unvalidated user input gets used in an HTTP response. In a reflected XSS attack, an attacker can craft a URL with the attack script and post it to another website, email it, or otherwise get a victim to click on it.

An easy way to find out if a field is vulnerable to an XSS attack is to use the `alert()` or `console.log()` methods. Use one of them to find out which field is vulnerable.

Shopping Cart			
Shopping Cart Items -- To Buy Now	Price	Quantity	Total
Studio RTA - Laptop/Reading Cart with Tilting Surface - Cherry	69.99	1	\$0.00
Dynex - Traditional Notebook Case	27.99	1	\$0.00
Hewlett-Packard - Pavilion Notebook with Intel Centrino	1599.99	1	\$0.00
3 - Year Performance Service Plan \$1000 and Over	299.99	1	\$0.00

Enter your credit card number:

Enter your three digit access code:

Purchase

Congratulations, but alerts are not very impressive are they? Let's continue to the next assignment.
 Thank you for shopping at WebGoat.
 Your support is appreciated

We have charged credit card:4128 3214 0002 1999>

 \$1997.96

Search HTML

```
</table>
<br>
<table width="90%" cellspacing="0" cellpadding="2" align="center">
<tbody>
<tr>
<td>Enter your credit card number:</td>
<td>
<input name="field1" value="4128 3214 0002 1999" type="TEXT">
</td>
</tr>
</tbody>
</table>
```

Filter Styles Show .cls Layout Computed Changes Compatibility

Pseudo-elements This Element Grid Box Model

Element :: { input, button, select, textarea bootstrap.min.css:7 margin: 0; border: 2px solid black; }

Screenshot of a Kali Linux desktop environment showing a Firefox browser window. The browser is displaying a WebGoat lesson titled "Cross Site Scripting".

The URL in the address bar is `localhost:8080/WebGoat/start.mvc?username=kelvin#lesson/CrossSiteScripting.lesson/9`.

The sidebar on the left shows a navigation tree for the WebGoat application, including categories like Introduction, General, and various attack types (A1 to A10). The "Cross Site Scripting" link under A3 Injection is highlighted.

The main content area displays the "Identify potential for DOM-Based XSS" section. It contains a form with a text input field containing "start.mvc#test/" and a submit button. Below the form, a message says "Correct! Now, see if you can send in an exploit to that route in the next assignment.".

At the bottom of the page, there is a detailed screenshot of a browser developer tools' element inspector. The element selected is a `<div>` with the class "lesson-page-wrapper". The right-hand panel shows the CSS properties for this element, including:

```

element :: {
    inline
}
+ :: {
    bootstrap.min.css:7
    -webkit-box-sizing: border-box;
    box-sizing: border-box;
}

```

The "margin" and "border" properties are both set to 0.

Screenshot of a Kali Linux desktop environment showing a Firefox browser window. The browser is displaying the 'Cross Site Scripting' lesson from the WebGoat application at <http://127.0.0.1:8080/WebGoat/start.mvc#lesson/CrossSiteScripting.lesson/10>.

The left sidebar shows a navigation tree for the WebGoat application, including categories like Introduction, General, and various attack types (A1 to A10). The 'Cross Site Scripting' section is currently selected.

The main content area displays the 'Try It! DOM-Based XSS' challenge. It includes a note about 'blind' attacks, a search bar, and a navigation menu with items 1 through 12. The challenge itself asks to enter a random number from the console into a text input field. The user has entered '-2127538668' and clicked 'Submit'. A message below the input field says 'Correct, I hope you did not cheat, using the console!'.

At the bottom, a developer tools Network tab shows the following requests:

- GET http://127.0.0.1:8880/WebGoat/service/lessonmenu.mvc
- phoneHome invoked
- POST http://127.0.0.1:8880/WebGoat/CrossSiteScripting/phone-home.xss
- phone home said {"lessonCompleted":true,"feedback":"Congratulations. You have successfully completed the assignment.","feedbackArgs":null,"output":"phoneHome Response is -2127538668","outputArgs":null,"assignment":"DOMCrossSiteScripting","attemptWasMade":true}
- GET http://127.0.0.1:8880/WebGoat/service/lessonmenu.mvc
- GET http://127.0.0.1:8880/WebGoat/service/lessonmenu.mvc
- GET http://127.0.0.1:8880/WebGoat/service/lessonmenu.mvc

The screenshot shows a Kali Linux desktop environment with a browser window open to a XSS attack assignment on WebGoat. The browser title is "WebGoat". The URL in the address bar is `127.0.0.1:8080/WebGoat/start.mvc#lesson/CrossSiteScripting.lesson/11`. The page content is a multiple-choice quiz about XSS attacks, with the correct answers highlighted in blue.

1. Is Google's XSS filter effective?

- Solution 2: Yes, because Google has got an algorithm that blocks malicious code.
- Solution 3: No, because the script that is executed will break through the defense algorithm of the browser.
- Solution 4: No, because the browser trusts the website if it is acknowledged trusted, then the browser does not know that the script is malicious.

2. When do XSS attacks occur?

- Solution 1: When malicious scripts are injected into a website's server-side code.
- Solution 2: When a user submits sensitive information without encryption.
- Solution 3: When a website fails to validate or sanitize user input, allowing malicious scripts to be executed in a user's browser.
- Solution 4: When a website uses outdated SSL/TLS protocols.

3. What are Stored XSS attacks?

- Solution 1: The script is permanently stored on the server and the victim gets the malicious script when requesting information from the server.
- Solution 2: The script stores itself on the computer of the victim and executes locally the malicious code.
- Solution 3: The script stores a virus on the computer of the victim. The attacker can perform various actions now.
- Solution 4: The script is stored in the browser and sends information to the attacker.

4. What are Reflected XSS attacks?

- Solution 1: Reflected attacks reflect malicious code from the database to the web server and then reflect it back to the user.
- Solution 2: They reflect the injected script off the web server. That occurs when input sent to the web server is part of the response.
- Solution 3: Reflected attacks reflect from the firewall off to the database where the user requests information from.
- Solution 4: Reflected XSS is an attack where the injected script is reflected off the database and web server to the user.

5. Is JavaScript the only way to perform XSS attacks?

- Solution 1: Yes, you can only make use of tags through JavaScript.
- Solution 2: Yes, otherwise you cannot steal cookies.
- Solution 3: No, there is ECMAScript too.
- Solution 4: No, there are many other ways. Like HTML, Flash or any other type of code that the browser executes.

Congratulations. You have successfully completed the assignment.

Slide 19: Mitigation Steps for Common Web Vulnerabilities

To protect web applications from common vulnerabilities like SQL Injection, XSS, and CSRF:

■ SQL Injection (SQLi):

- Use prepared statements (parameterized queries) instead of dynamic SQL.
- Sanitize and validate all user inputs.
- Apply the principle of least privilege to database accounts.

■■ Cross Site Scripting (XSS):

- Escape output depending on the context (HTML, JavaScript, URL, etc.).
- Use Content Security Policy (CSP) headers to reduce script execution.
- Sanitize inputs to remove or encode dangerous characters.

■ Cross Site Request Forgery (CSRF):

- Implement anti-CSRF tokens on forms.
- Check the HTTP Referer or Origin header on sensitive requests.
- Use secure cookies and set the SameSite attribute.

■ Regular vulnerability scanning and security reviews are crucial for maintaining web application security.

Thank You