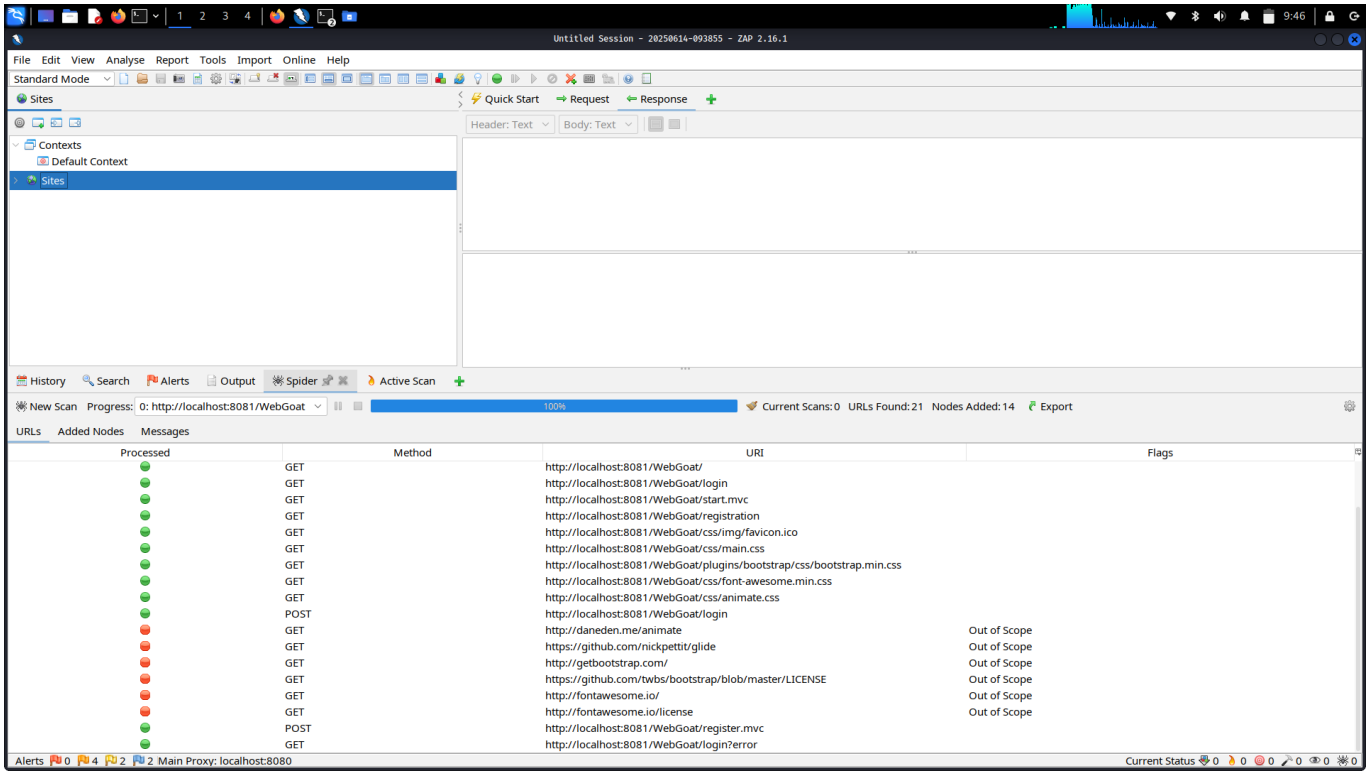
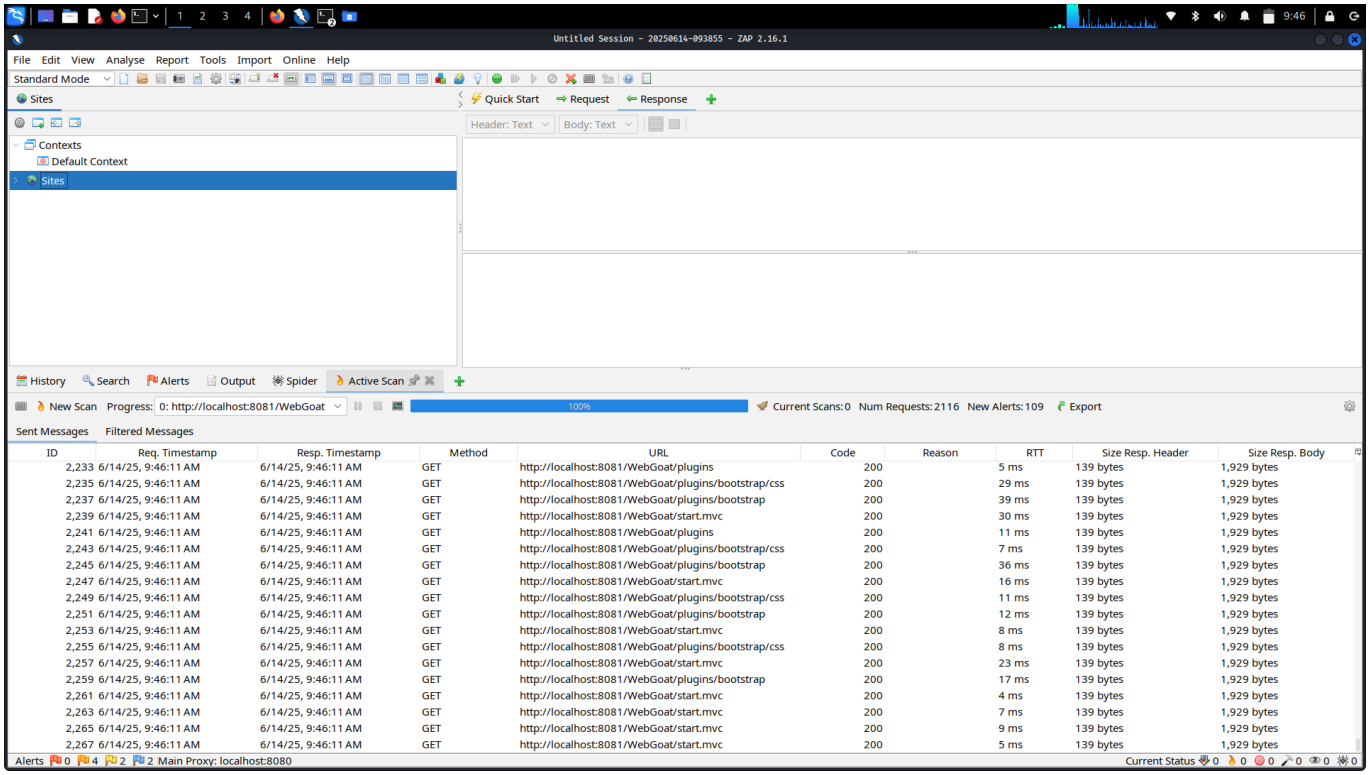
Cybersecurity Internship Report

Prepared by: Oriji Kelvin Promise

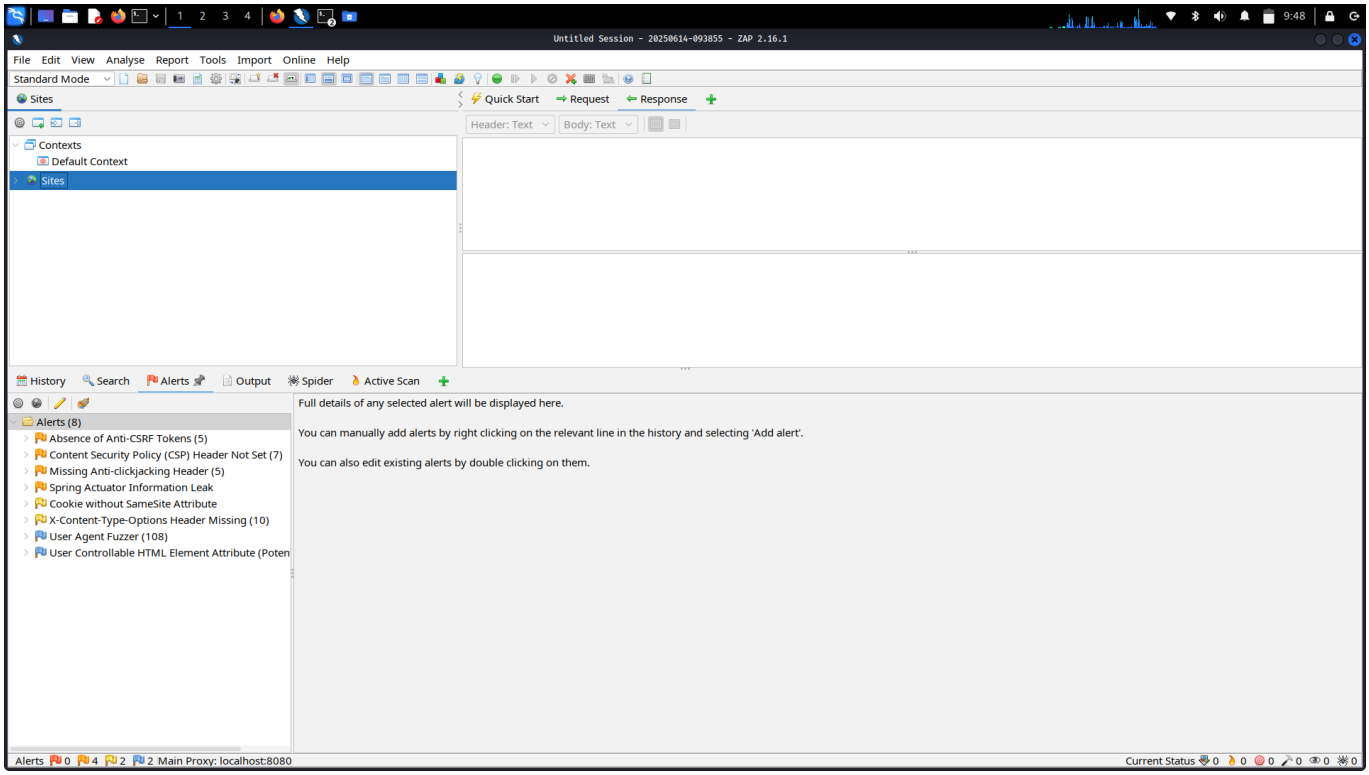
Slide 1: First we start by doing a spider scan with ZAP



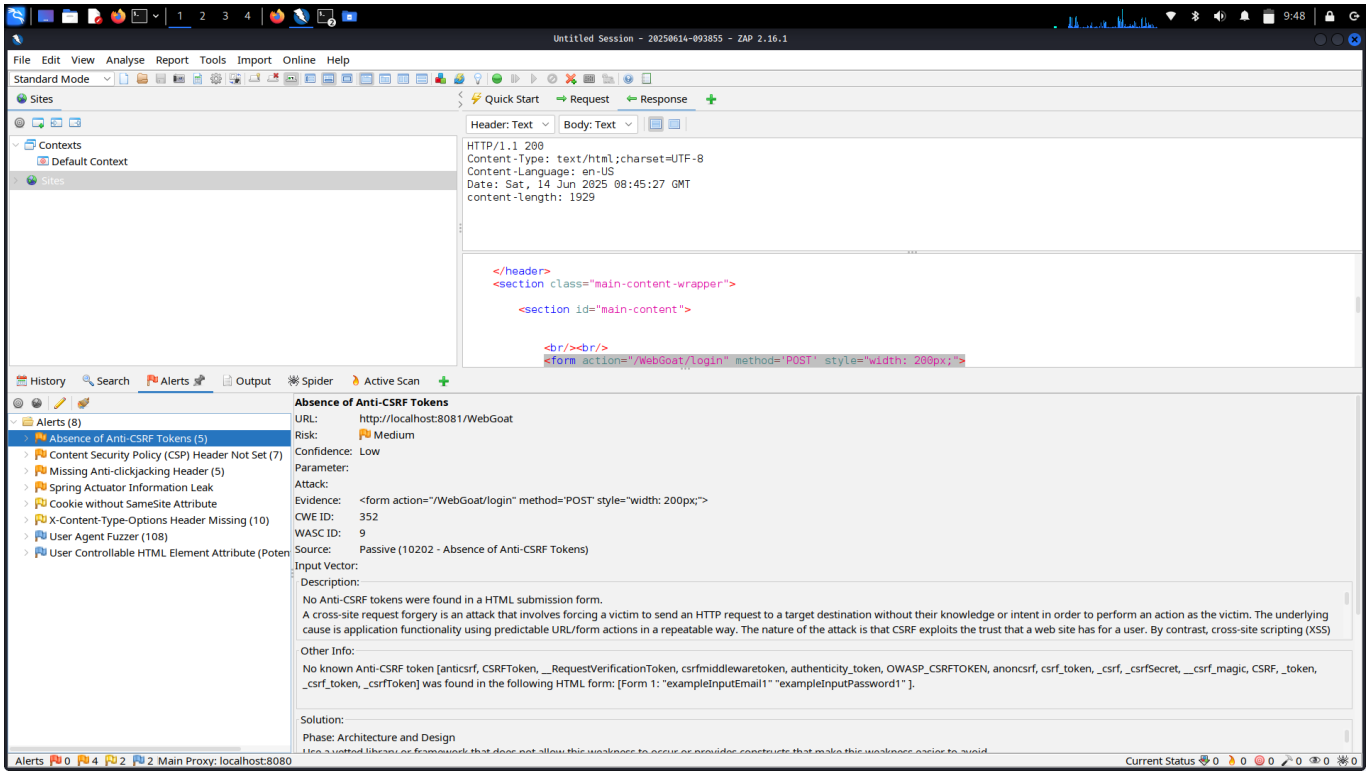
Slide 2: Next we do an active scan



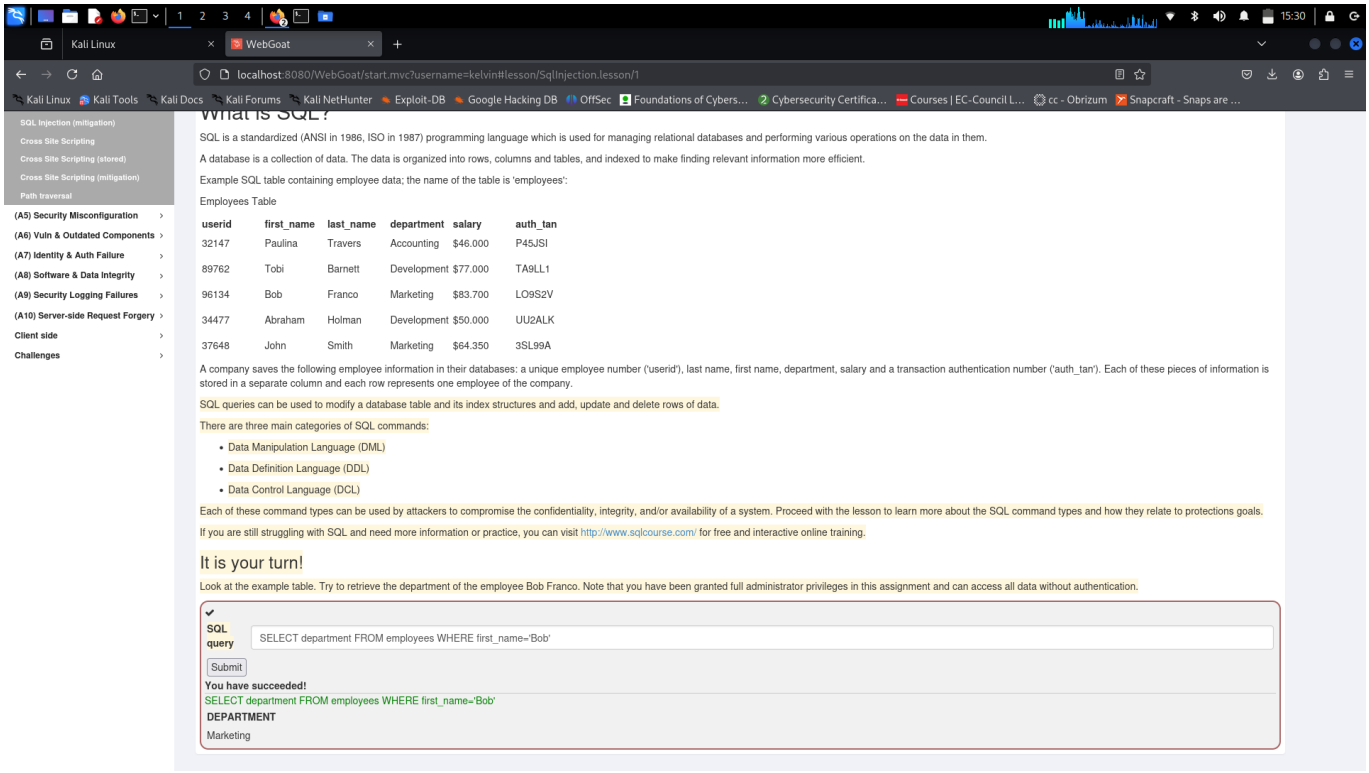
Slide 3: This is the alerts tab



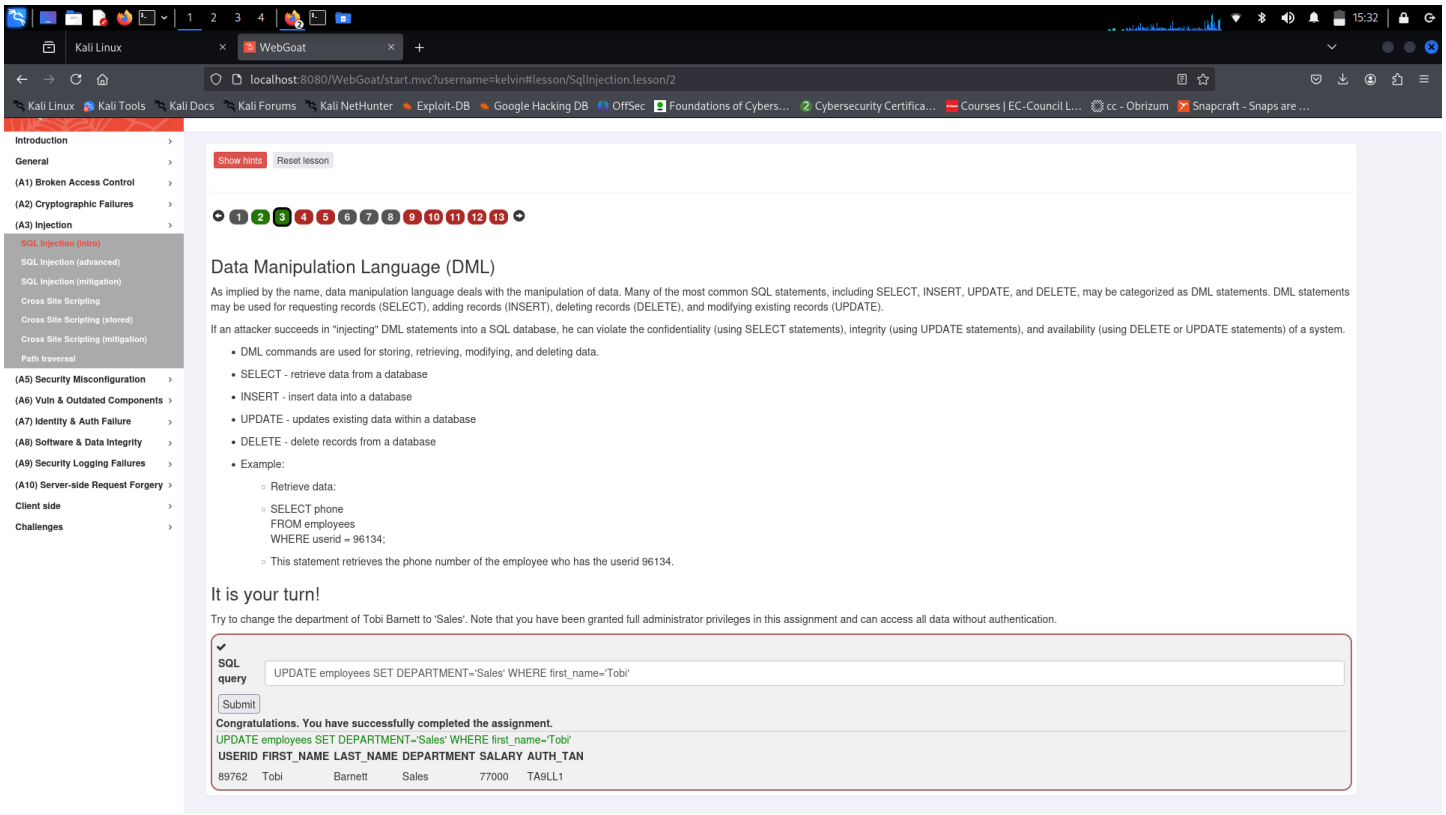
Slide 4: [Missing Image Placeholder]



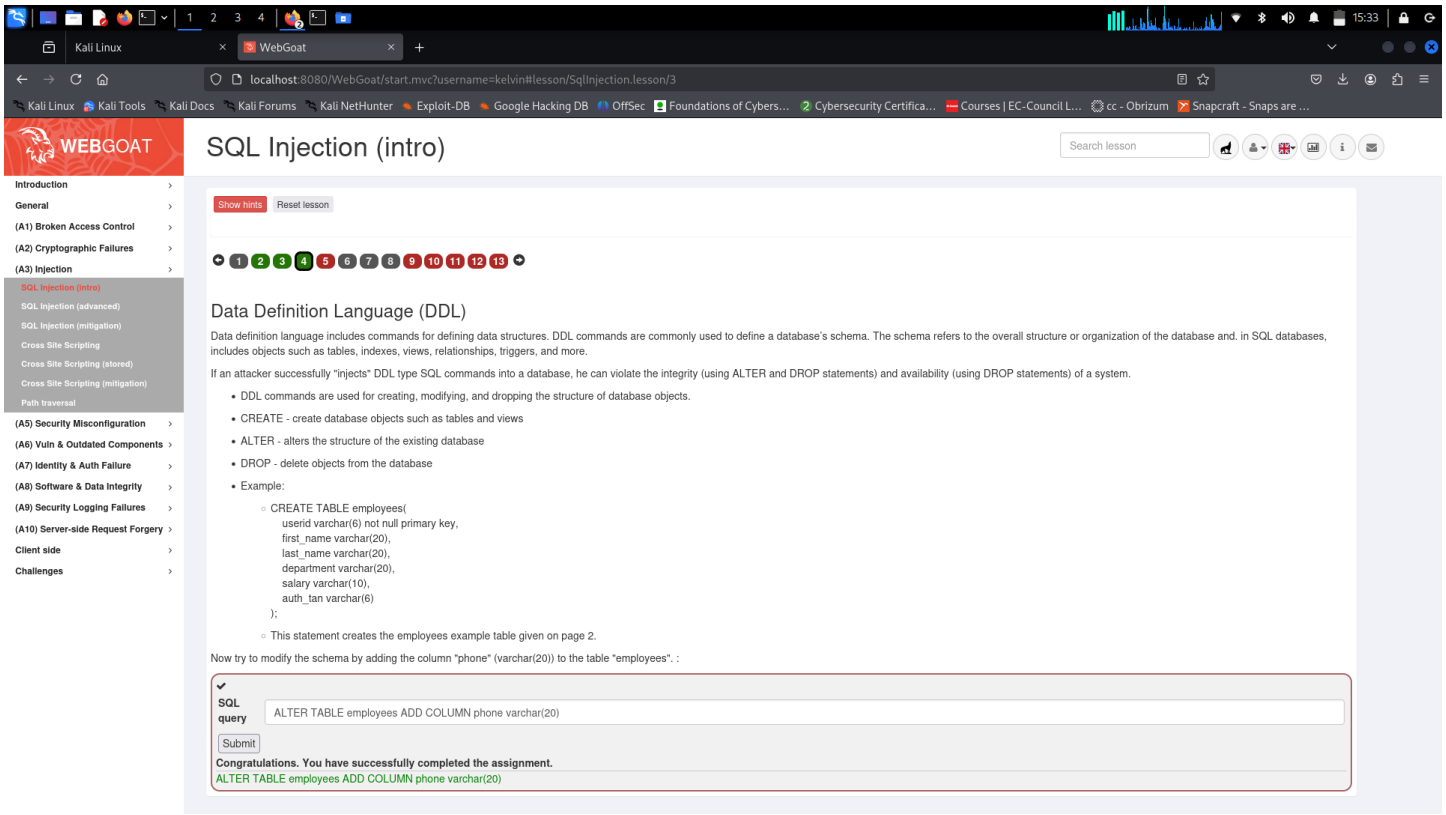
Slide 5: We try to exploit vulnerabilities on WebGoat using SQL injection



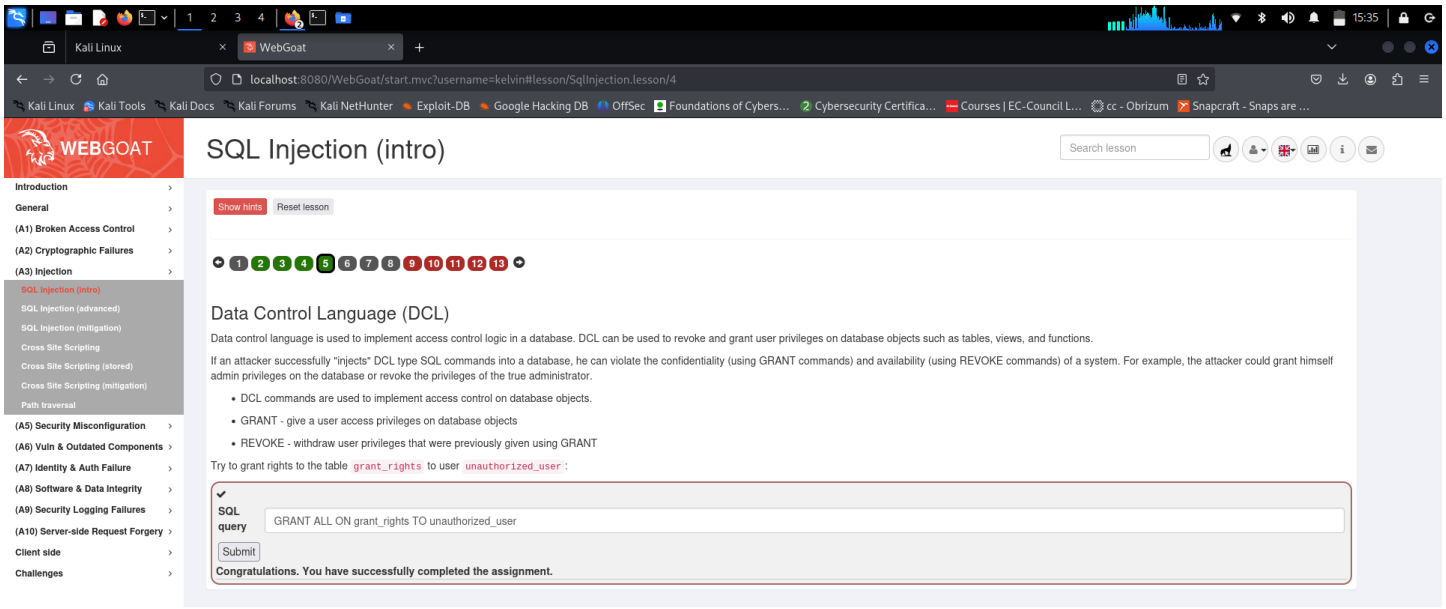
Slide 6: Evidence of SQL Injection Exploitation - Injected payload and bypassed login.



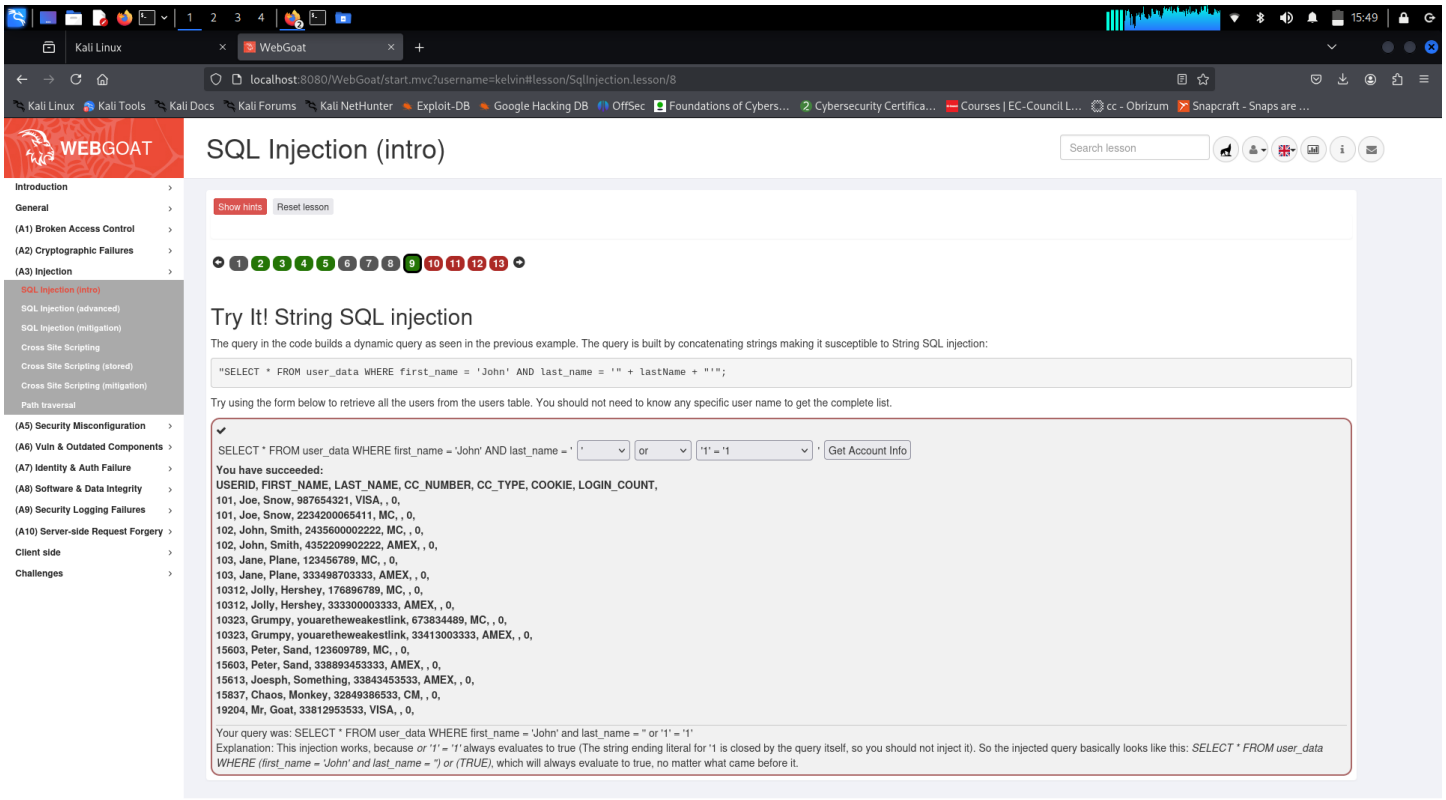
Slide 7: Further SQL Injection - Update operations on user credentials.



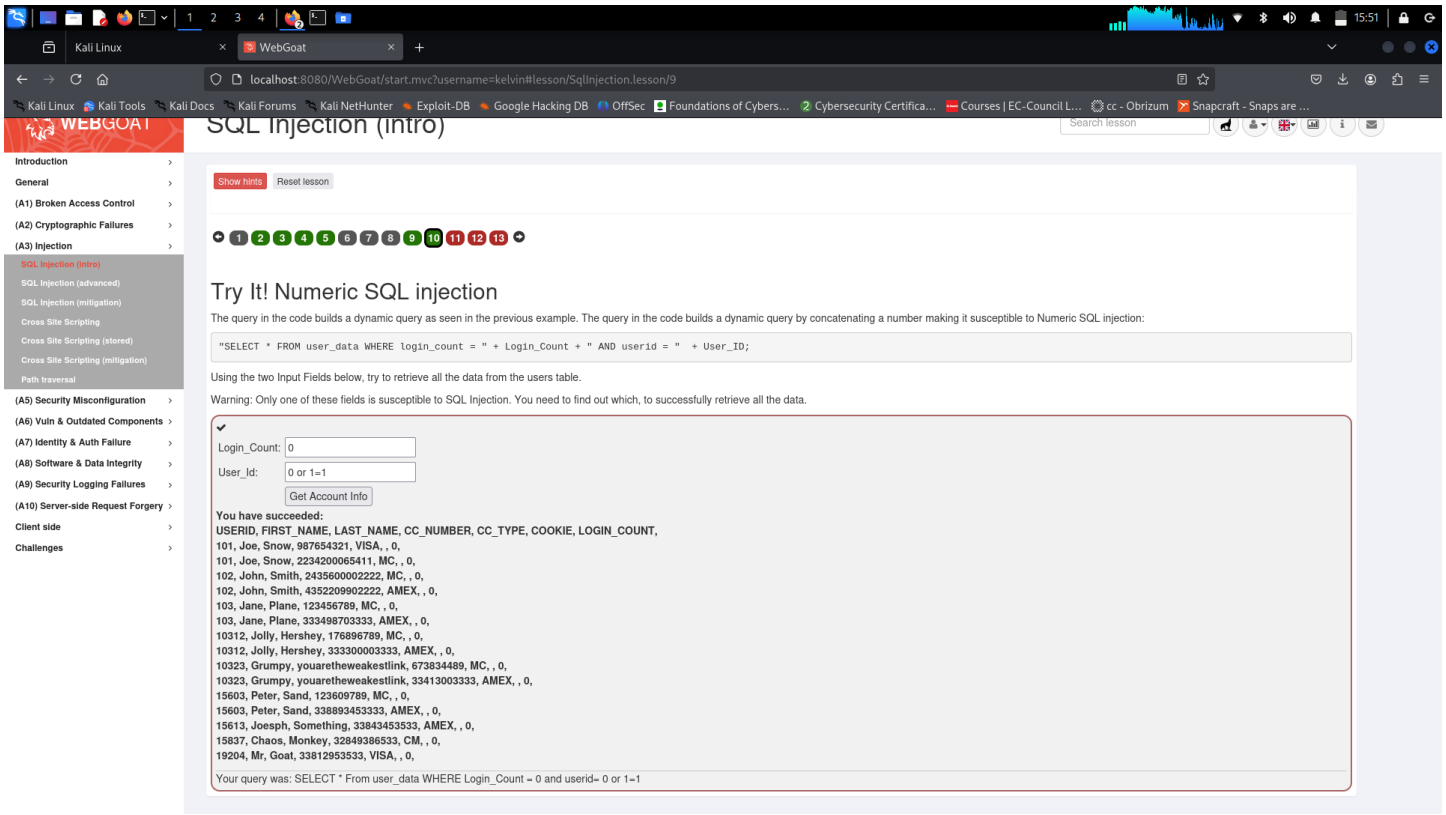
Slide 8: Demonstration of SQL DDL Injection - Altering database structure.



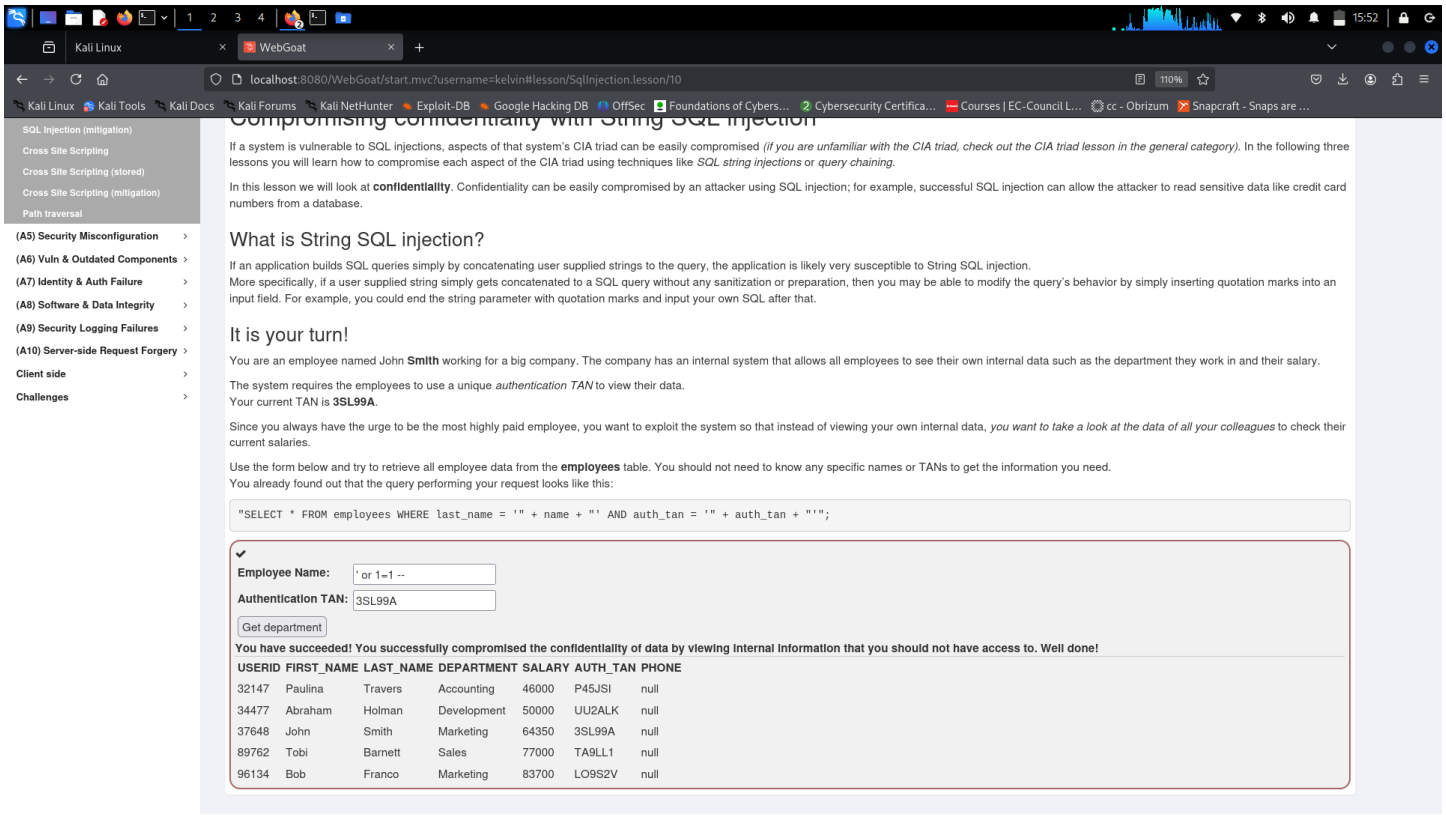
Slide 9: Evidence of SQL DCL Injection - Granting privileges.



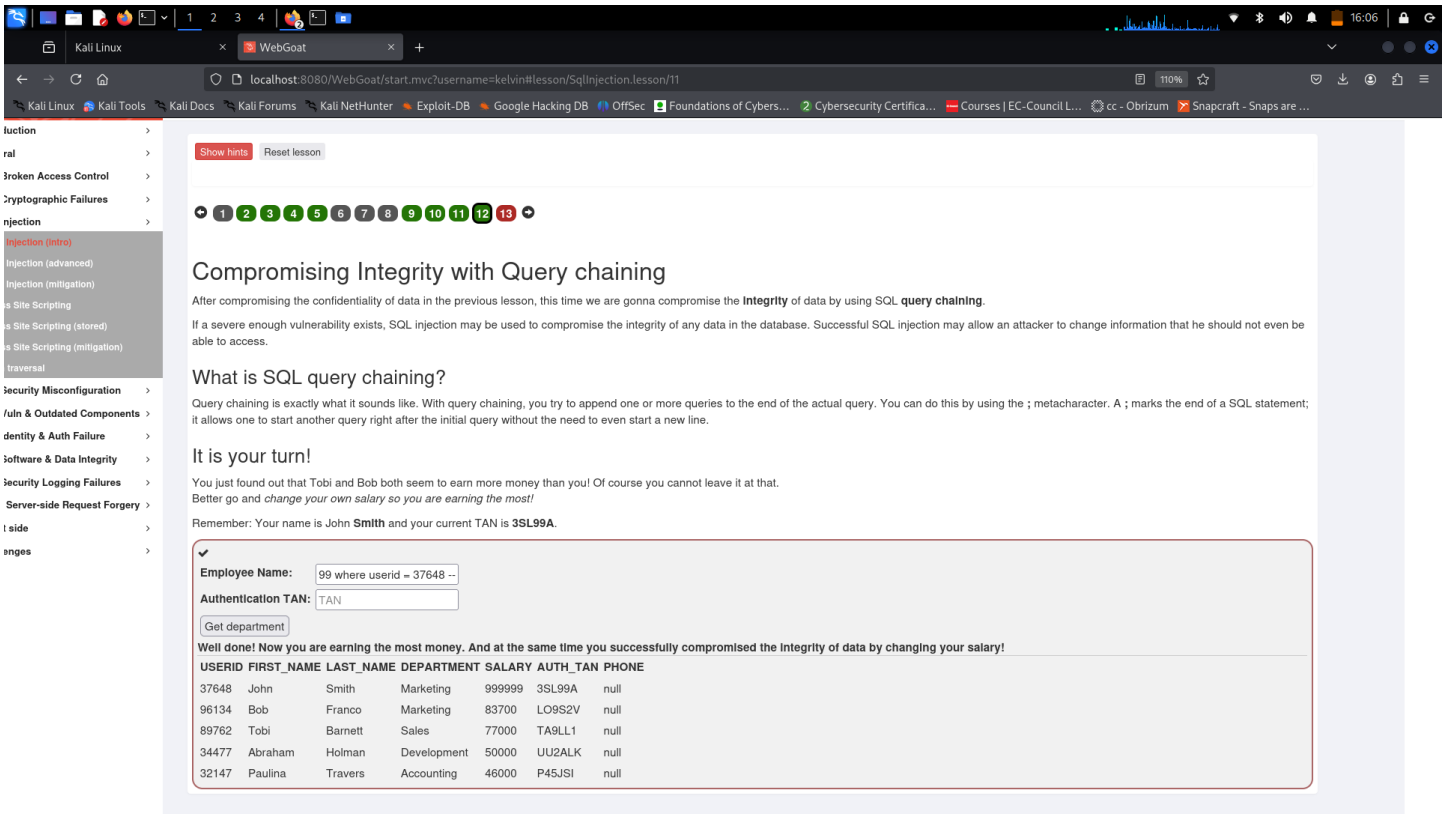
Slide 10: Confirming SQL SELECT * Injection - Extracting full records.



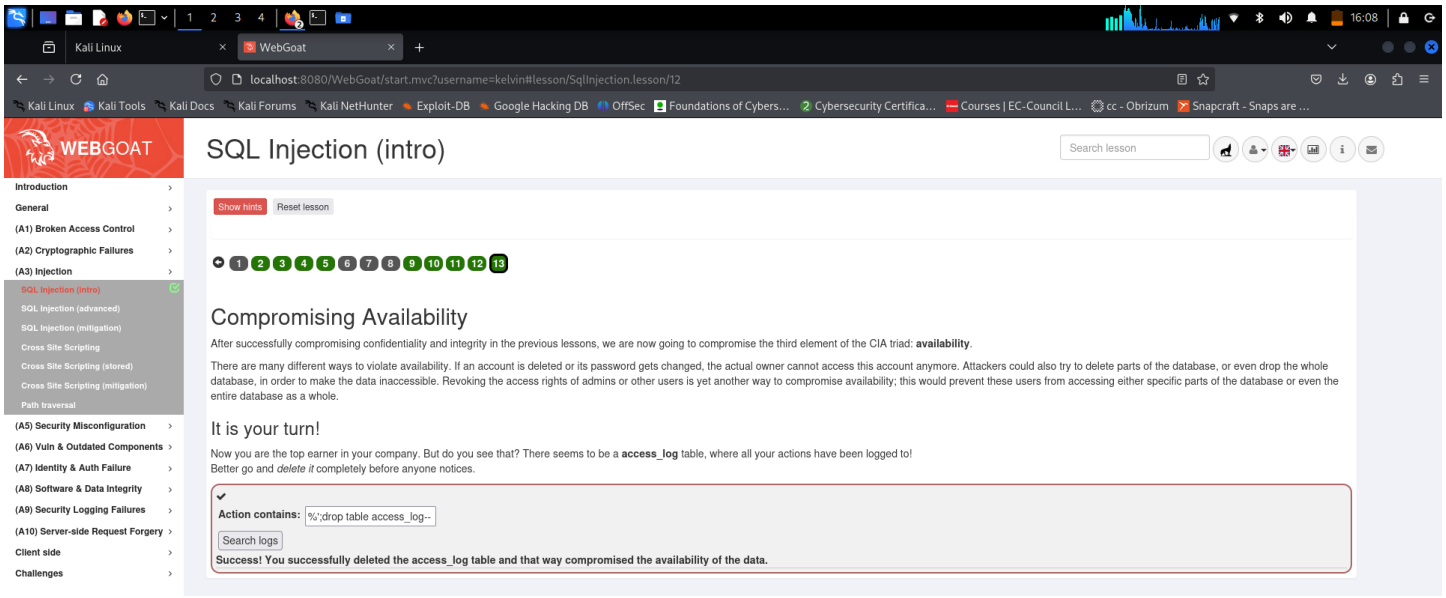
Slide 11: Modifying salary via SQL Injection chaining.



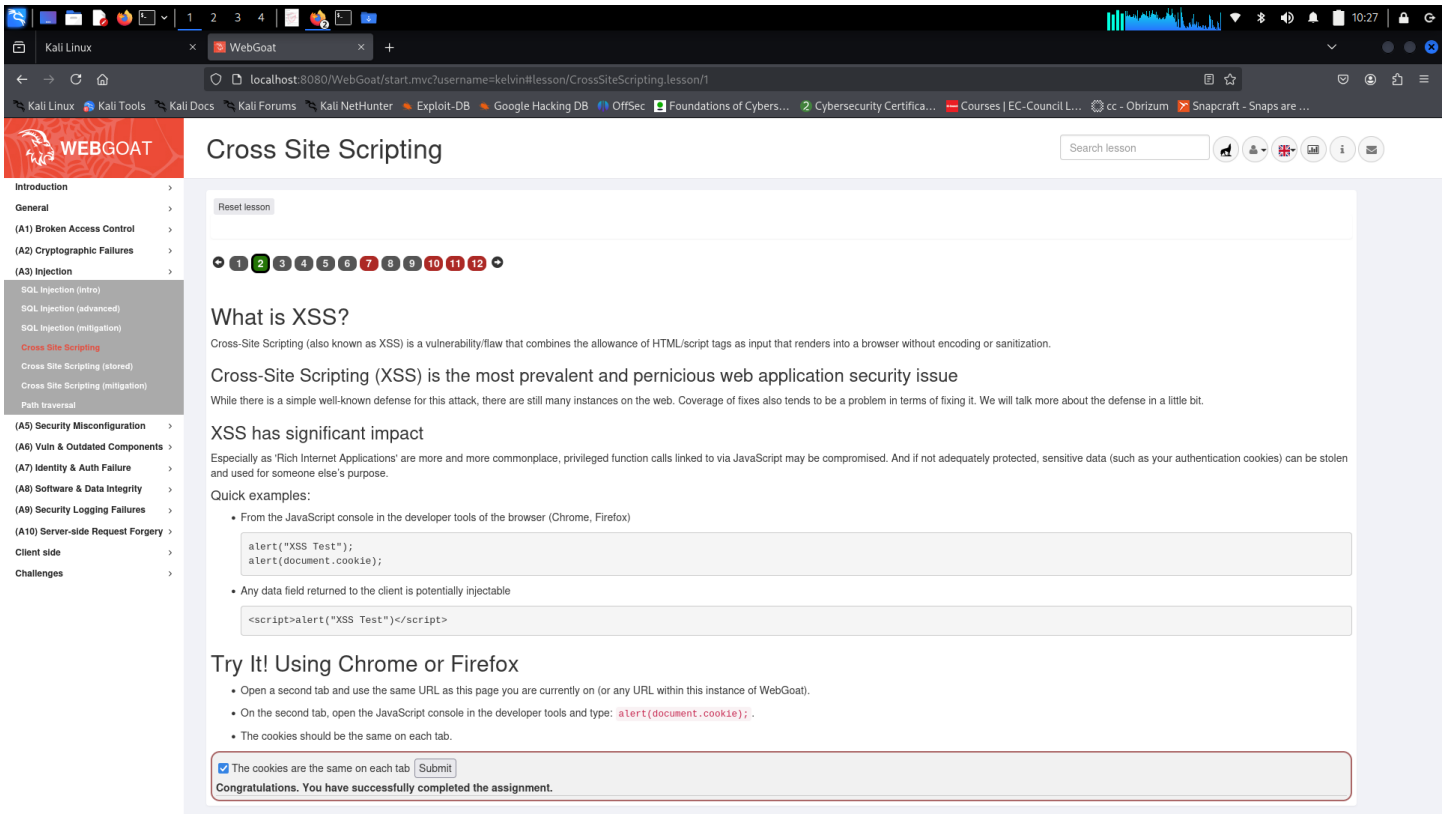
Slide 12: SQL Injection combined with multiple statements.

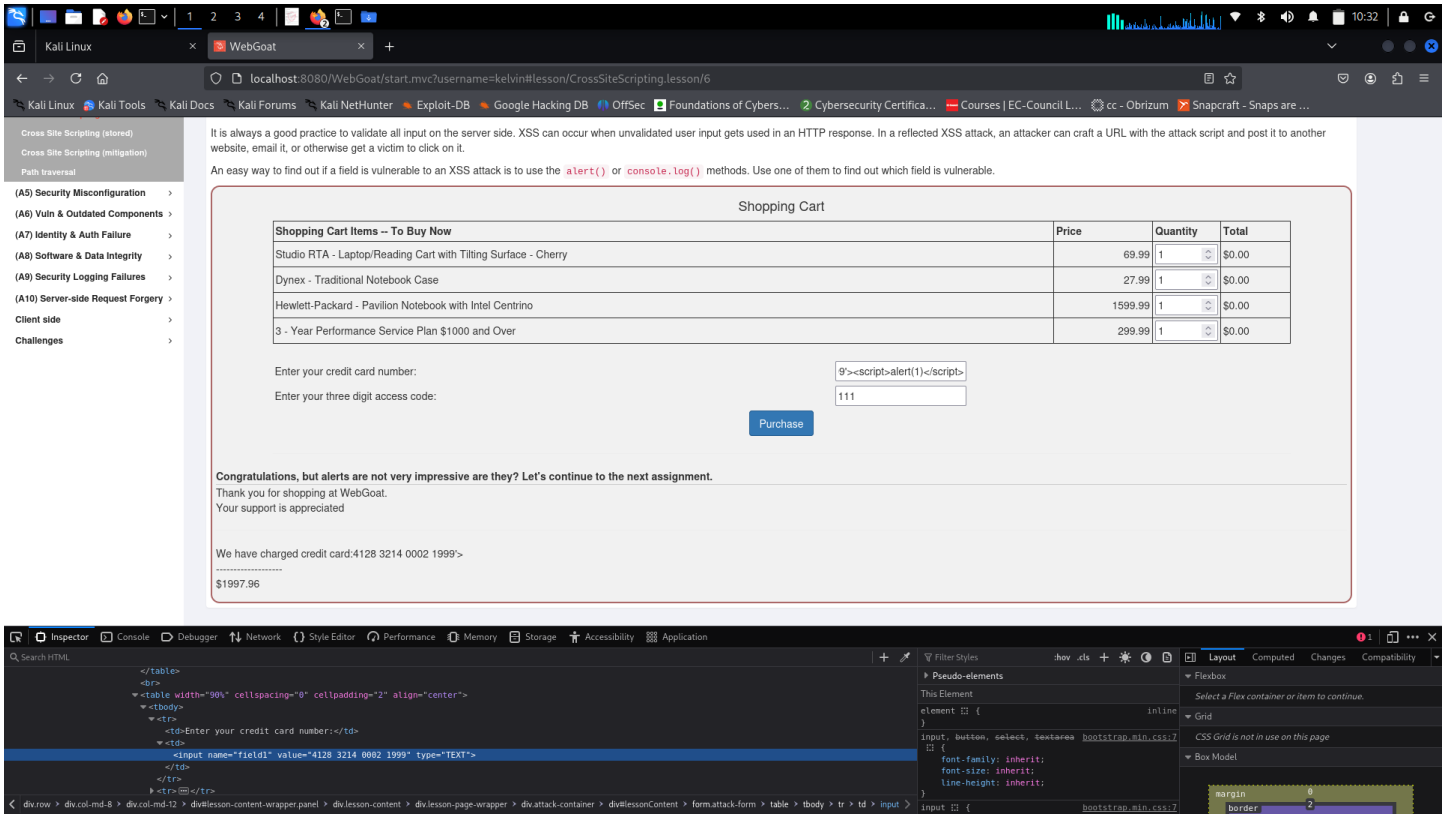


Slide 13: Completing all exploitation steps for SQL Injection in WebGoat.

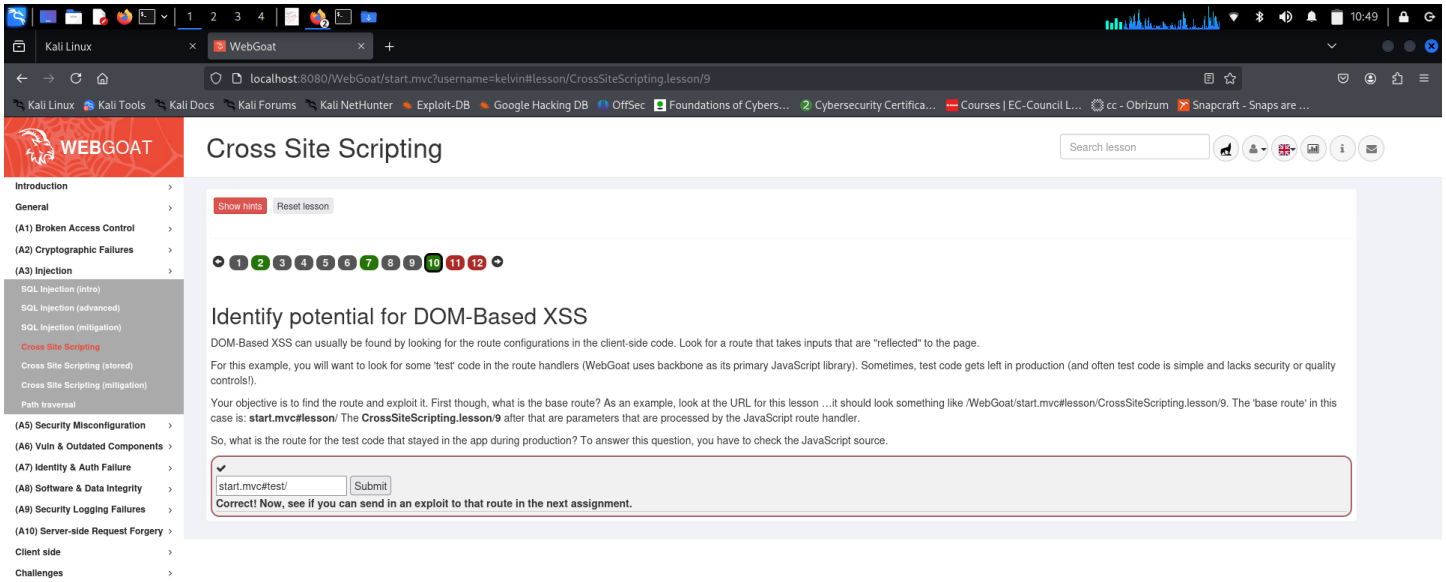


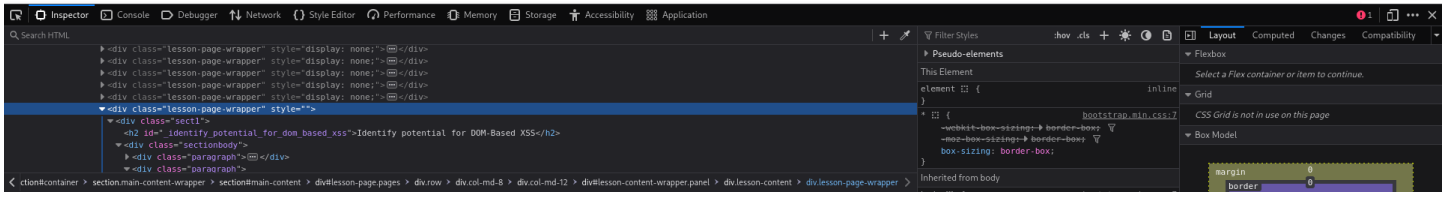
Slide 14: Next we try to exploit vulnerabilities using Cross Site Scripting (XSS).

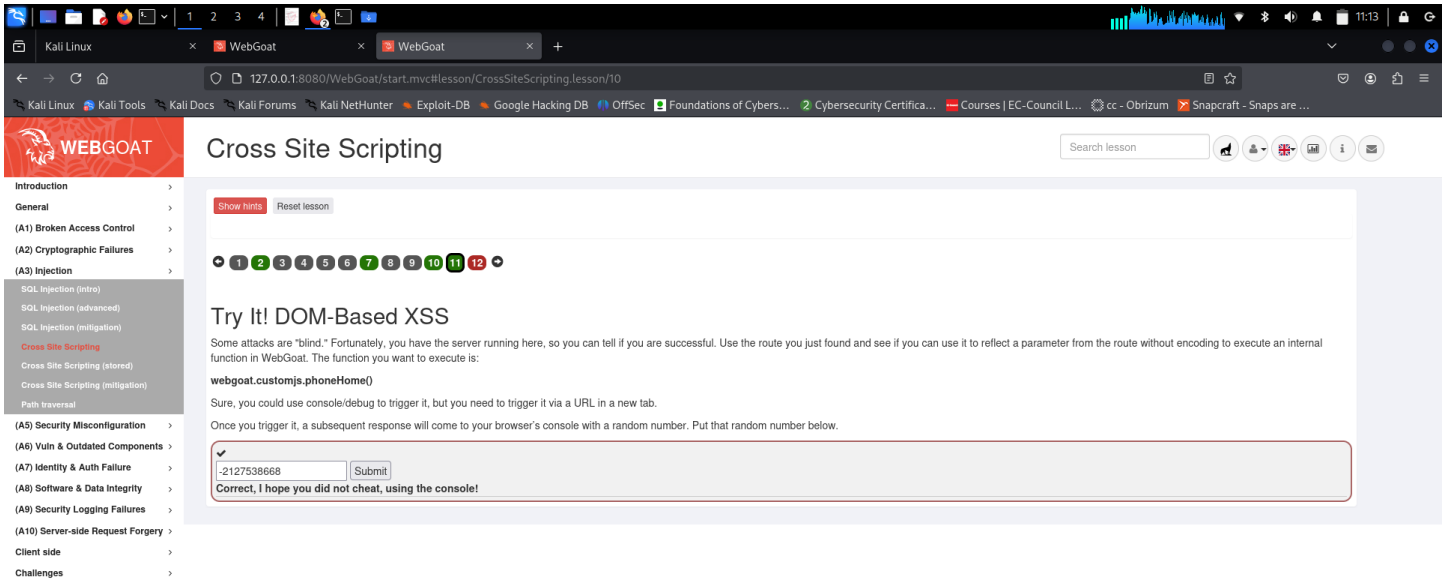


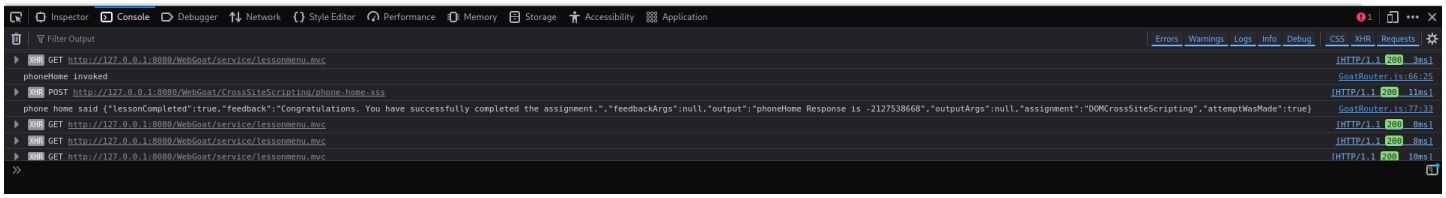

Kali Linux
WebGoat
localhost:8080/WebGoat/start.mvc?username=kelvin#lesson/CrossSiteScripting.lesson/6
Cross Site Scripting (stored)
Cross Site Scripting (mitigation)
Path traversal
(A5) Security Misconfiguration
(A6) Vuln & Outdated Components
(A7) Identity & Auth Failure
(A8) Software & Data Integrity
(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:
9'><script>alert(1)</script>
Enter your three digit access code:
111
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

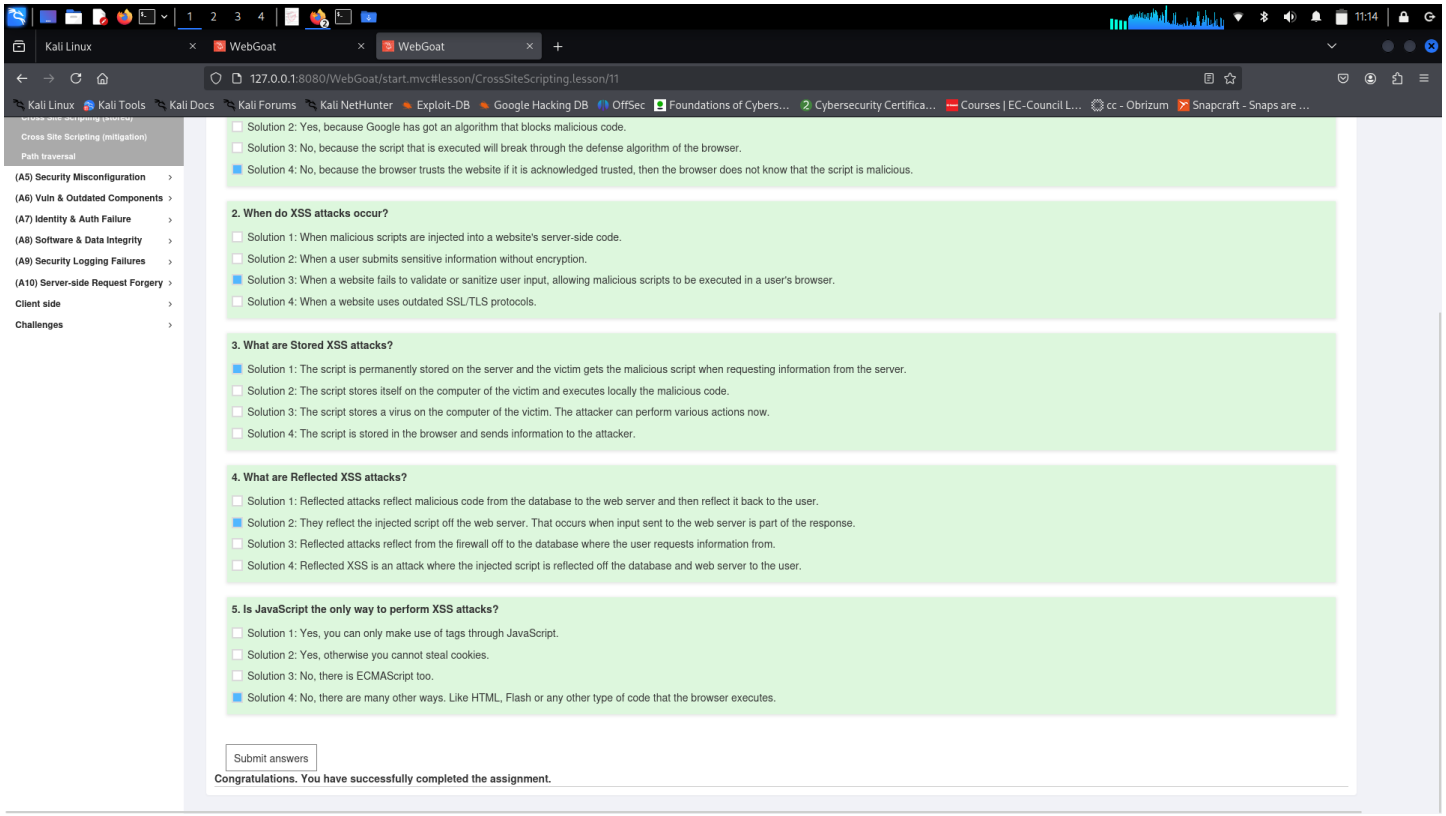
Inspector
Console
Debugger
Network
Style Editor
Performance
Memory
Storage
Accessibility
Application
<table width="90%" cellspacing="0" cellpadding="2" align="center">
<td>Enter your credit card number:</td>
<input name="field1" value="4128 3214 0002 1999" type="TEXT">
Layout
Computed
Changes
Compatibility
Flexbox
Grid
Box Model


Kali Linux
WebGoat
localhost:8080/WebGoat/start.mvc?username=kelvin#lesson/CrossSiteScripting.lesson/9
WEBGOAT
Cross Site Scripting
Search lesson
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
Identify potential for DOM-Based XSS
DOM-Based XSS can usually be found by looking for the route configurations in the client-side code. Look for a route that takes inputs that are "reflected" to the page.
For this example, you will want to look for some 'test' code in the route handlers (WebGoat uses backbone as its primary JavaScript library). Sometimes, test code gets left in production (and often test code is simple and lacks security or quality controls!).
Your objective is to find the route and exploit it. First though, what is the base route? As an example, look at the URL for this lesson ...it should look something like /WebGoat/start.mvc#lesson/CrossSiteScripting.lesson/9. The 'base route' in this case is: start.mvc#lesson/ The CrossSiteScripting.lesson/9 after that are parameters that are processed by the JavaScript route handler.
So, what is the route for the test code that stayed in the app during production? To answer this question, you have to check the JavaScript source.
start.mvc#test/
Submit
Correct! Now, see if you can send in an exploit to that route in the next assignment.


Inspector
Console
Debugger
Network
Style Editor
Performance
Memory
Storage
Accessibility
Application
Search HTML
Filter Styles
Pseudo-elements
This Element
Inherited from body
Layout
Computed
Changes
Compatibility
Flexbox
Grid
Box Model


Kali Linux
WebGoat
WebGoat
127.0.0.1:8080/WebGoat/start.mvc#lesson/CrossSiteScripting.lesson/10
WEBGOAT
Cross Site Scripting
Search lesson
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
Try It! DOM-Based XSS
Some attacks are "blind." Fortunately, you have the server running here, so you can tell if you are successful. Use the route you just found and see if you can use it to reflect a parameter from the route without encoding to execute an internal function in WebGoat. The function you want to execute is:
webgoat.customjs.phoneHome()
Sure, you could use console/debug to trigger it, but you need to trigger it via a URL in a new tab.
Once you trigger it, a subsequent response will come to your browser's console with a random number. Put that random number below.
-2127538668
Submit
Correct, I hope you did not cheat, using the console!


Inspector
Console
Debugger
Network
Style Editor
Performance
Memory
Storage
Accessibility
Application
Filter Output
GET http://127.0.0.1:8080/WebGoat/service/lessonmenu.mvc
phoneHome invoked
POST http://127.0.0.1:8080/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://127.0.0.1:8080/WebGoat/service/lessonmenu.mvc
GET http://127.0.0.1:8080/WebGoat/service/lessonmenu.mvc
GET http://127.0.0.1:8080/WebGoat/service/lessonmenu.mvc


Kali Linux
WebGoat
WebGoat
127.0.0.1:8080/WebGoat/start.mvc#lesson/CrossSiteScripting.lesson/11
Kali Linux Kali Tools Kali Docs Kali Forums Kali NetHunter Exploit-DB Google Hacking DB OffSec Foundations of Cybers... Cybersecurity Certifica... Courses | EC-Council L... cc - Obrizum Snapcraft - Snaps are ...
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
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.
Submit answers
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 securi

Thank You