Lab 2

Web Security Threats and Defenses

Review Question

Compare roughly OWASP Top 10 Attacks to SANS Institute Top 25 ?

Hands-on Lab: Understanding OWASP Top Ten Attack

This lab use WebGoat to illustrate typical security flaws within web applications.

1. Installing WebGoat

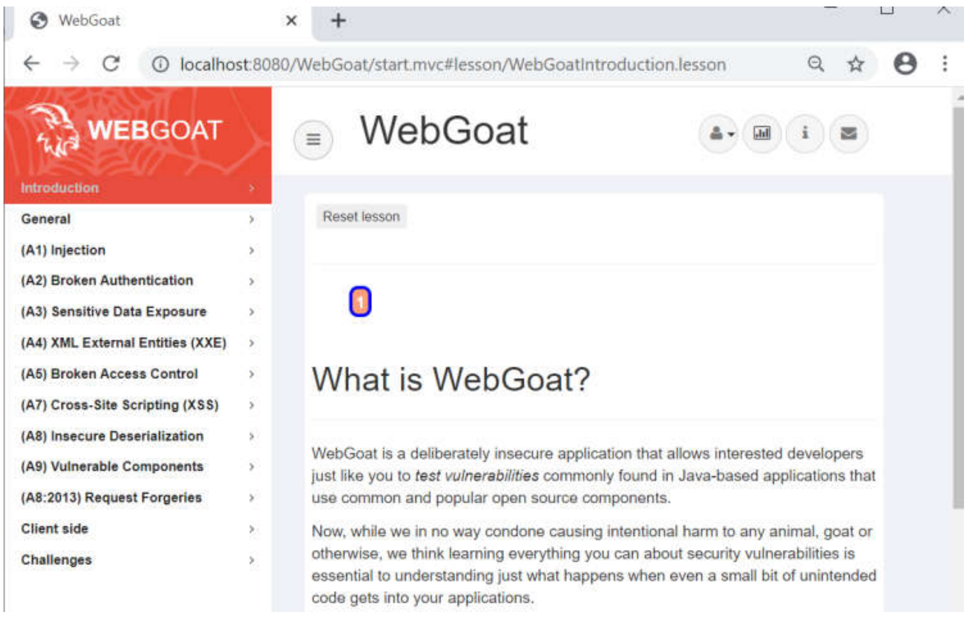
WebGoat is an insecure web application maintained by OWASP designed to teach web application security lessons. The application aims to provide a realistic teaching environment, providing users with hints and code to further explain the lesson.

There are several ways to install WebGoat, but the easy way is to start it as a docker container. We have already installed it I Lab 1, here is a recall of the Docker command.

```
docker run -p 8080:8080 -p 9090:9090 -e TZ=Europe/Amsterdam webgoat/goatandwolf:v8.1.0
```

- WebGoat comes with WebWolf which plays the role of the attacker. It helps you while solving some of the assignments and challenges within WebGoat
- It is recommended to disconnect your computer from the Internet. Running WebGoat would make your computer vulnerable to the threats.

- Open a browser and go to http://localhost:8080/WebGoat
- Create a user-id and choose a password.
- Login!



WebGoat
localhost:8080/WebGoat/start.mvc#lesson/WebGoatIntroduction.lesson
WEBGOAT
Introduction
General
(A1) Injection
(A2) Broken Authentication
(A3) Sensitive Data Exposure
(A4) XML External Entities (XXE)
(A5) Broken Access Control
(A7) Cross-Site Scripting (XSS)
(A8) Insecure Deserialization
(A9) Vulnerable Components
(A8:2013) Request Forgeries
Client side
Challenges
WebGoat
Reset lesson
1

What is WebGoat?

WebGoat is a deliberately insecure application that allows interested developers just like you to *test vulnerabilities* commonly found in Java-based applications that use common and popular open source components.

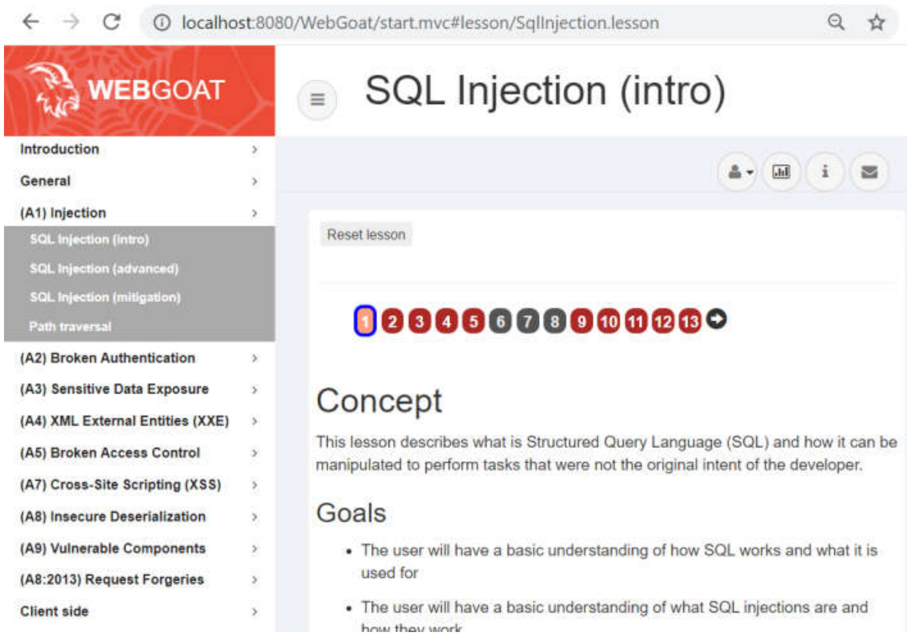
Now, while we in no way condone causing intentional harm to any animal, goat or otherwise, we think learning everything you can about security vulnerabilities is essential to understanding just what happens when even a small bit of unintended code gets into your applications.

2. Injection Flaws: SQL Injection

SQL injection is a particularly widespread and dangerous form of injection. To exploit an SQL injection flaw, the attacker must find a parameter that the web application passes through to a database. By carefully embedding malicious SQL commands into the content of the parameter, the attacker can trick the web application into forwarding a malicious query to the database. These attacks are not difficult to attempt, and more tools are emerging that scan for these flaws. The consequences are particularly damaging, as an attacker can obtain, corrupt, or destroy database contents.

The solution for these exercises can found here : https://github.com/WebGoat/WebGoat/wiki/(Almost)-Fully-Documented-Solution-(en)#sql-injection-introduction

• Choose Injection Flaws and SQL Injection from the menu as shown in the figure below.


localhost:8080/WebGoat/start.mvc#lesson/SqlInjection.lesson
WEBGOAT
Introduction
General
(A1) Injection
SQL Injection (intro)
SQL Injection (advanced)
SQL Injection (mitigation)
Path traversal
(A2) Broken Authentication
(A3) Sensitive Data Exposure
(A4) XML External Entities (XXE)
(A5) Broken Access Control
(A7) Cross-Site Scripting (XSS)
(A8) Insecure Deserialization
(A9) Vulnerable Components
(A8:2013) Request Forgeries
Client side
SQL Injection (intro)
Reset lesson
1 2 3 4 5 6 7 8 9 10 11 12 13

Concept

This lesson describes what is Structured Query Language (SQL) and how it can be manipulated to perform tasks that were not the original intent of the developer.

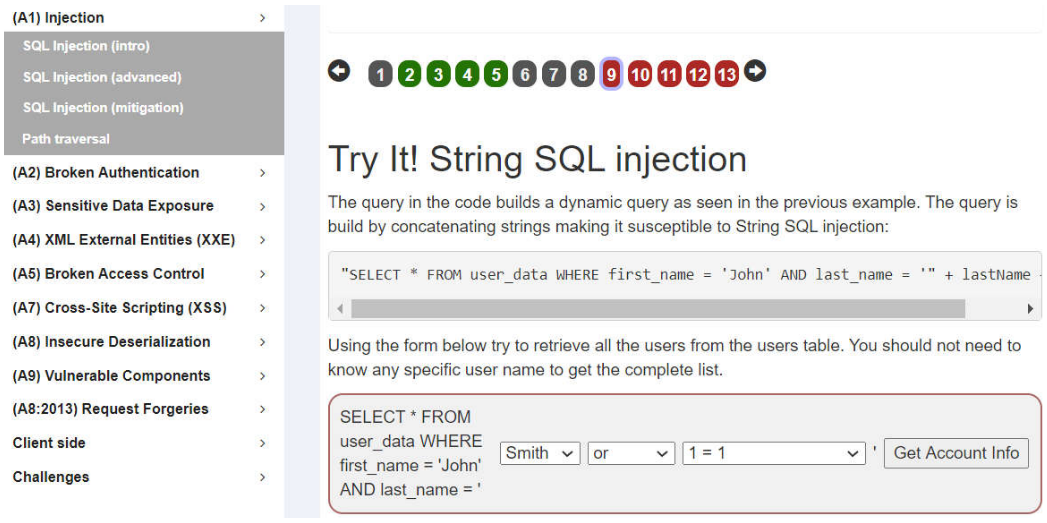
Goals

- The user will have a basic understanding of how SQL works and what it is used for
- The user will have a basic understanding of what SQL injections are and how they work

Follow the steps on the screen until page 9.

2.1 String SQL Injection

On page 9 try to get the complete list of the users.


(A1) Injection
SQL Injection (intro)
SQL Injection (advanced)
SQL Injection (mitigation)
Path traversal
(A2) Broken Authentication
(A3) Sensitive Data Exposure
(A4) XML External Entities (XXE)
(A5) Broken Access Control
(A7) Cross-Site Scripting (XSS)
(A8) Insecure Deserialization
(A9) Vulnerable Components
(A8:2013) Request Forgeries
Client side
Challenges
1 2 3 4 5 6 7 8 9 10 11 12 13

Try It! String SQL injection

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

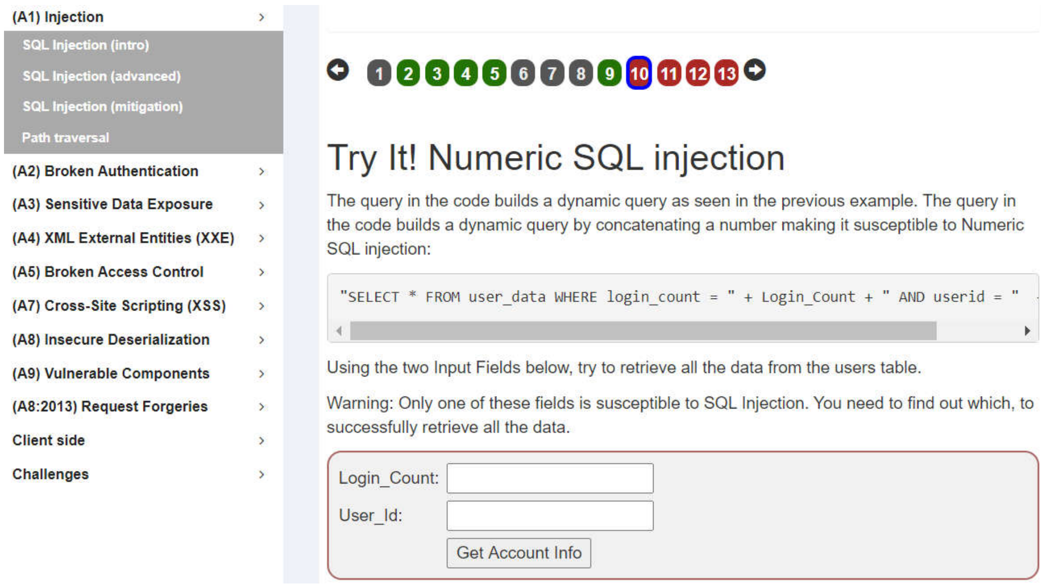
```
"SELECT * FROM user_data WHERE first_name = 'John' AND last_name = '" + lastName
```

Using the form below try 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 = ' Smith or 1 = 1 ' Get Account Info

2.2 Numeric SQL Injection

On page 10 try to get the complete list of the credit card numbers.



(A1) Injection
SQL Injection (intro)
SQL Injection (advanced)
SQL Injection (mitigation)
Path traversal
(A2) Broken Authentication
(A3) Sensitive Data Exposure
(A4) XML External Entities (XXE)
(A5) Broken Access Control
(A7) Cross-Site Scripting (XSS)
(A8) Insecure Deserialization
(A9) Vulnerable Components
(A8:2013) Request Forgeries
Client side
Challenges

Try It! Numeric SQL injection

The query in the code builds a dynamic query as seen in the previous example. The query in the code builds a dynamic query by concatenating a number making it susceptible to Numeric SQL injection:

```
"SELECT * FROM user_data WHERE login_count = " + Login_Count + " AND userid = "
```

Using the two Input Fields below, try to retrieve all the data from the users table.

Warning: Only one of these fields is susceptible to SQL Injection. You need to find out which, to successfully retrieve all the data.

Login_Count:	
User_Id:	
	Get Account Info

2.3 Compromising confidentiality with String SQL injection

If a system is vulnerable to SQL injections, aspects of that system's CIA triad can be easily compromised (if you are unfamiliar with the CIA triad, check out the CIA triad lesson in the general category). In the following three lessons you will learn how to compromise each aspect of the CIA triad using techniques like SQL string *injections* or *query chaining*.

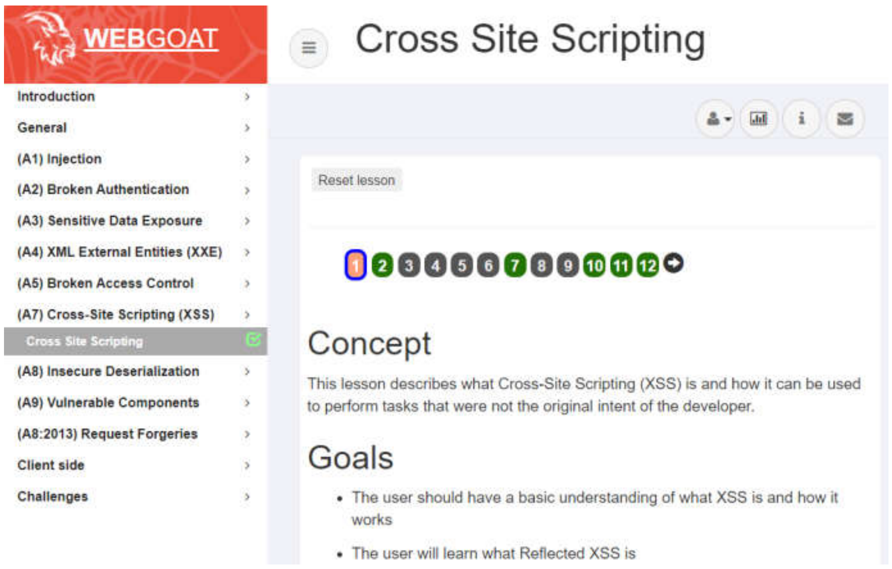
On page 11 try to try to retrieve all employee data from the employees table.

3 Cross-Site Scripting (XSS)

Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into otherwise benign and trusted web sites. XSS attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user. Flaws that allow these attacks to succeed are quite widespread and occur anywhere a web application uses input from a user within the output it generates without validating or encoding it.

Choose Cross-Site Scripting from the menu as shown in the Figure below.

The solution for these exercises can found here : https://github.com/WebGoat/WebGoat/wiki/(Almost)-Fully-Documented-Solution-(en)#a7-cross-site-scripting-xss


WEBGOAT
Introduction
General
(A1) Injection
(A2) Broken Authentication
(A3) Sensitive Data Exposure
(A4) XML External Entities (XXE)
(A5) Broken Access Control
(A7) Cross-Site Scripting (XSS)
Cross Site Scripting
(A8) Insecure Deserialization
(A9) Vulnerable Components
(A8:2013) Request Forgeries
Client side
Challenges
Reset lesson

Cross Site Scripting

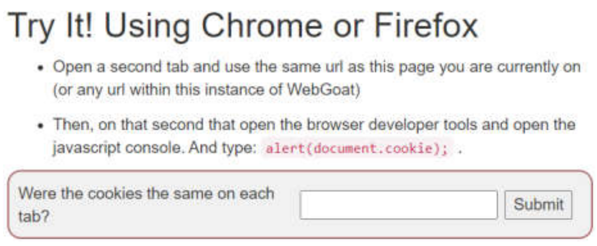
Concept

This lesson describes what Cross-Site Scripting (XSS) is and how it can be used to perform tasks that were not the original intent of the developer.

Goals

- The user should have a basic understanding of what XSS is and how it works
- The user will learn what Reflected XSS is

1. Follow the steps on the screen until page 2. Do the exercise on page 2, shown in the following Figure.

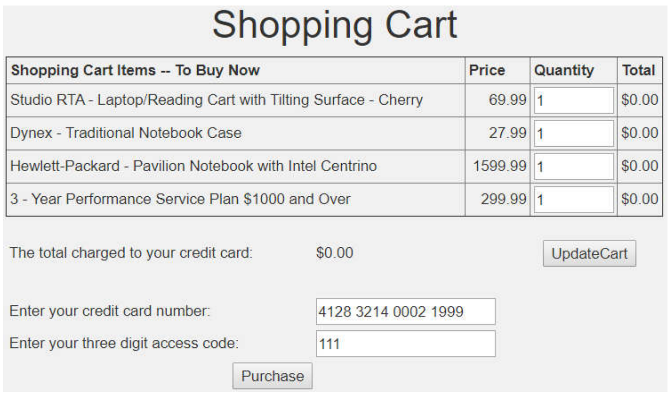


Try It! Using Chrome or Firefox

- Open a second tab and use the same url as this page you are currently on (or any url within this instance of WebGoat)
- Then, on that second that open the browser developer tools and open the javascript console. And type: `alert(document.cookie);` .

Were the cookies the same on each tab? [ ] Submit

2. Follow the steps on the screen until page 7. Do the exercise on page 7, shown in the following Figure.



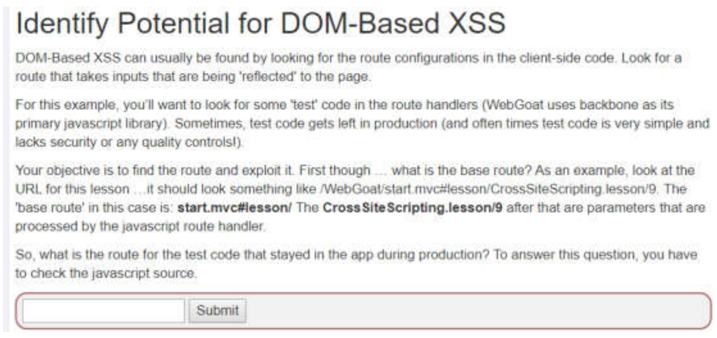
Shopping Cart

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

The total charged to your credit card: $0.00 UpdateCart

Enter your credit card number: 4128 3214 0002 1999
Enter your three digit access code: 111
Purchase

3. Follow the steps on the screen until page 10. Do the exercise on page 10, shown in the following Figure.



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 being 'reflected' to the page.

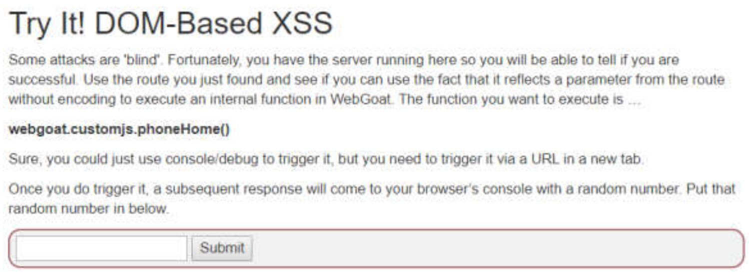
For this example, you'll want to look for some 'test' code in the route handlers (WebGoat uses backbone as its primary javascript library). Sometimes, test code gets left in production (and often times test code is very simple and lacks security or any quality controls!).

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.

[ ] Submit

4. Follow the steps on the screen until page 11. Do the exercise on page 11, shown in the following Figure.



Try It! DOM-Based XSS

Some attacks are 'blind'. Fortunately, you have the server running here so you will be able to tell if you are successful. Use the route you just found and see if you can use the fact that it reflects 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 just use console/debug to trigger it, but you need to trigger it via a URL in a new tab.

Once you do trigger it, a subsequent response will come to your browser's console with a random number. Put that random number in below.

[ ] Submit

5. Take the quiz on page 12.

3 Cross site Request Forgeries

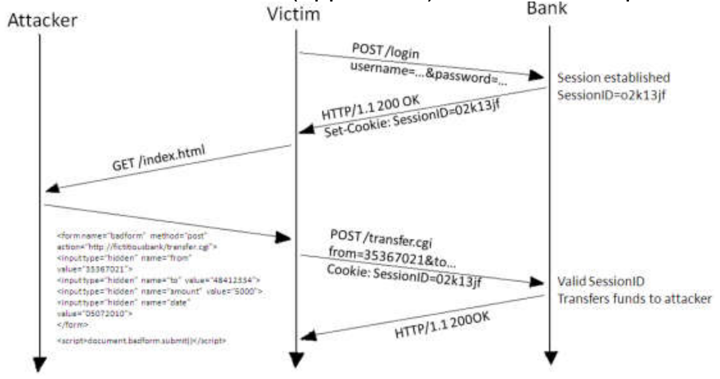
Cross-site request forgery, also known as **one-click attack** or **session riding** and abbreviated as CSRF (sometimes pronounced sea-surf) or XSRF, is a type of malicious exploit of a website where unauthorized commands are transmitted from a user that the website trusts. Unlike cross-site scripting

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

A cross-site request forgery is a 'confused deputy' attack against a web browser. CSRF commonly has the following characteristics:

- It involves sites that rely on a user's identity.
- It exploits the site's trust in that identity.
- It tricks the user's browser into sending HTTP requests to a target site.
- It involves HTTP requests that have side effects.

An active session of the user should be present on the browser on which the victim is clicking the button. It is required to make this attack successful. The native behavior of the browser is that it attaches the cookie of the domain (application) to which the requests are being made.



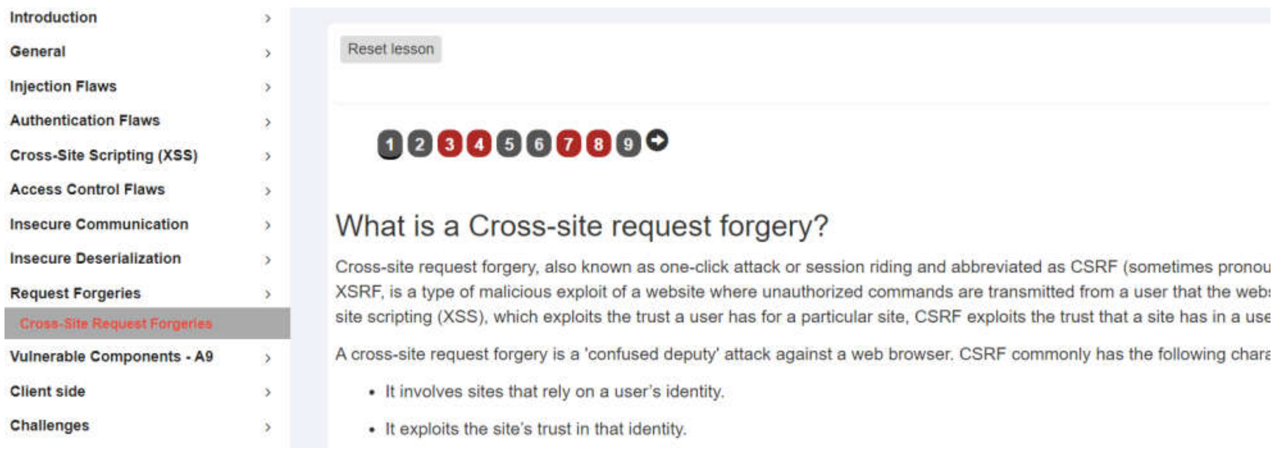
```
<form name="badform" method="post"
action="http://fictitiousbank/transfer.cgi">
<inputtype="hidden" name="from"
value="35367021">
<inputtype="hidden" name="to" value="48412334">
<inputtype="hidden" name="amount" value="5000">
<inputtype="hidden" name="date"
value="05072010">
</form>
<script>document.badform.submit()</script>
```

Source: www.opensourceforu.efytimes.com

The above figure demonstrates a CSRF attack in which the attacker lures the victim to transfer funds to attacker's account. It is achieved by crafting a request that contains the parameters: '**from**' (account from which funds will be debited) and '**to**' (account to which funds will be credited). The fact that the victim has a valid session results in a successful execution of the transfer action proving that the banking application is vulnerable to CSRF.

Back to the WebGoat application, Choose Cross-Site Request forgeries from the menu as shown in the Figure below.

The solution for these exercises can be found here :
https://github.com/WebGoat/WebGoat/wiki/(Almost)-Fully-Documented-Solution-(en)#a7-cross-site-scripting-xss



What is a Cross-site request forgery?

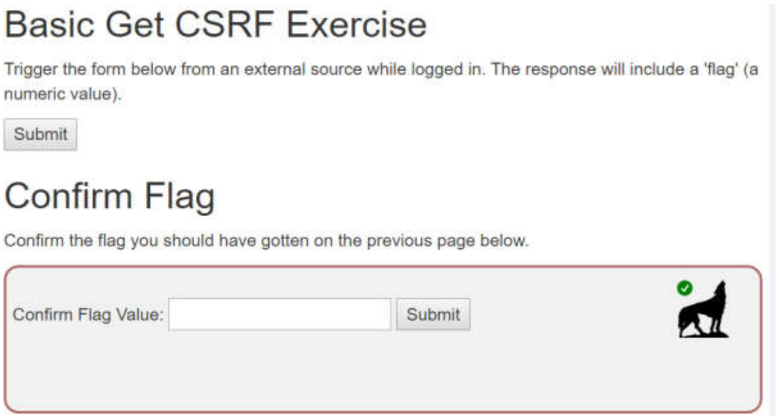
Cross-site request forgery, also known as one-click attack or session riding and abbreviated as CSRF (sometimes pronou
XSRF, is a type of malicious exploit of a website where unauthorized commands are transmitted from a user that the web
site scripting (XSS), which exploits the trust a user has for a particular site, CSRF exploits the trust that a site has in a use

A cross-site request forgery is a 'confused deputy' attack against a web browser. CSRF commonly has the following chara

- It involves sites that rely on a user's identity.
- It exploits the site's trust in that identity.

1.1 Basic Get CSRF Exercise

Follow the steps on the screen until page 3. Do the exercise on page 3, shown in the following Figure.



Basic Get CSRF Exercise

Trigger the form below from an external source while logged in. The response will include a 'flag' (a numeric value).

Submit

Confirm Flag

Confirm the flag you should have gotten on the previous page below.

Confirm Flag Value: Submit

The form has hidden inputs.

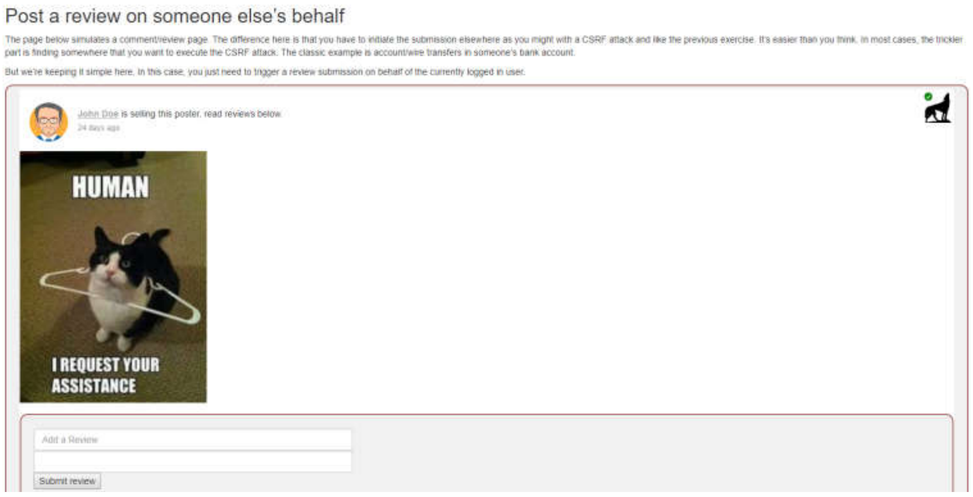
- You will need to use an external page and/or script to trigger it.
- Try creating a local page or one that is uploaded and points to this form as its action.
- The trigger can be manual or scripted to happen automatically

Save the following code in a file **csrf.html** and open it in a web browser.

```
<form name="attack" action="http://host:port/WebGoat/csrf/basic-get-flag" method="POST">
     <input type="hidden" name='csrf' value='true'>
</form>
<script>document.attack.submit();</script>
```

1.2 Post a review on someone else's behalf

Follow the steps on the screen until page 4. Do the exercise on page 4, shown in the following Figure.



Post a review on someone else's behalf

The page below simulates a comment/review page. The difference here is that you have to initiate the submission elsewhere as you might with a CSRF attack and like the previous exercise. It's easier than you think. In most cases, the trickier part is finding somewhere that you want to execute the CSRF attack. The classic example is account/wire transfers in someone's bank account.

But we're keeping it simple here. In this case, you just need to trigger a review submission on behalf of the currently logged in user.

John Doe is selling this poster, read reviews below.

24 days ago

Answer to the Review question

Although the CWE/25 and OWASP Top 10 are different, they share many of the same vulnerabilities. Here is a list of the OWASP Top 10 entries for 2017 and their corresponding CWEs. The Common Weakness Enumeration (CWE) is a list of software security vulnerabilities found all throughout the software development industry. It's a community-driven project maintained by MITRE, a non-profit research and development group.

OWASP Top 10	SANS CWE 25
A1: Injection	• CWE-78: Improper Neutralization of Special Elements Used in an OS Command ('OS Command Injection') • CWE-89: SQL Injection • CWE-94: Code Injection • CWE-434: Unrestricted Upload of File with Dangerous Type • CWE-494: Download of Code Without Integrity Check • CWE-829: Inclusion of Functionality from Untrusted Control Sphere
A2: Broken Authentication	• CWE-306: Missing Authentication for Critical Function • CWE-307: Improper Restriction of Excessive Authentication Attempts • CWE-798: Use of Hard-coded Credentials • CWE-807: Reliance on Untrusted Inputs in a Security Decision • CWE-862: Missing Authorization • CWE-863: Incorrect Authorization
A3: Sensitive Data Exposure	• CWE-311: Missing Encryption of Sensitive Data • CWE-319: Cleartext Transmission of Sensitive Information
A4: XML External Entities	• None
A5: Broken Access Control	• CWE-73: External Control of File Name or Path • CWE-285: Improper Authorization
A6: Security Misconfiguration	• CWE-250: Execution with Unnecessary Privileges • CWE-676: Use of Potentially Dangerous Function • CWE-732: Incorrect Permission Assignment for Critical Resource
A7: Cross-Site Scripting (XSS)	• CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-Site Scripting')
A8: Insecure Deserialization	• CWE-134: Use of Externally-Controlled Format String
A9: Using Components with Known Vulnerabilities	• CWE-190: Integer Overflow or Wraparound • CWE-327: Use of a Broken or Risky Cryptographic Algorithm • CWE-759: Use of a One-way Hash Without a Salt
A10: Insufficient Logging and Monitoring	• None