Filtering and blocking web incursions with AWS WAF

**SPL-TF-200-SIAWAF-1 - Version 1.2.9**

© 2023 Amazon Web Services, Inc. or its affiliates. All rights reserved. This work may not be reproduced or redistributed, in whole or in part, without prior written permission from Amazon Web Services, Inc. Commercial copying, lending, or selling is prohibited. All trademarks are the property of their owners.

Note: Do not include any personal, identifying, or confidential information into the lab environment. Information entered may be visible to others.

Corrections, feedback, or other questions? Contact us at [*AWS Training and Certification*](https://support.aws.amazon.com/#/contacts/aws-training).

**Lab overview**

Welcome to the AWS Web Application Firewall Challenge Lab! AWS WAF is a web application firewall that helps protect your web applications or APIs against common web exploits and bots that may affect availability, compromise security, or consume excessive resources. AWS WAF gives you control over how traffic reaches your applications by enabling you to create security rules that control bot traffic and block common attack patterns, such as SQL injection (SQLi) and cross-site scripting (XSS). You can also customize rules that filter out specific traffic patterns.

In this lab, you are a security engineer working for a company that just launched an online juice store. You have been asked to look for and mitigate SQLi and XSS vulnerabilities in their web application.

Unlike traditional labs which included detailed, step-by-step instructions, this Challenge Lab provides you with objectives and only minimal direction. You will need to apply your knowledge of security and common web exploits to complete the tasks.

In the first set of tasks, you are presented with an insecure web application and prompted to use SQL injection and cross-site scripting attacks to exploit it. If you get stuck, expand the *Hint* and *Solution* menus for for assistance with your attacks.

In the second set of tasks, you deploy and configure AWS WAF with a series of managed and custom rules designed to mitigate the attacks you previously launched.

OBJECTIVES

By the end of this lab, you will be able to do the following:

* Deploy simple SQLi and XSS attacks to compromise a web application.
* Create a web access control list (web ACL) in AWS WAF and associate it with an Application Load Balancer.
* Apply AWS managed rule groups to a web ACL.
* Apply custom rules to a web ACL.

TECHNICAL KNOWLEDGE PREREQUISITES

Advanced knowledge of SQL, as well as a solid understanding of network security, common web protocols, and the RESTful framework is required for this lab. You should also be comfortable working in a Windows Server environment using Remote Desktop.

**System requirements**

This lab requires:

* Google Chrome

DURATION

This lab requires approximately *60* minutes to complete.

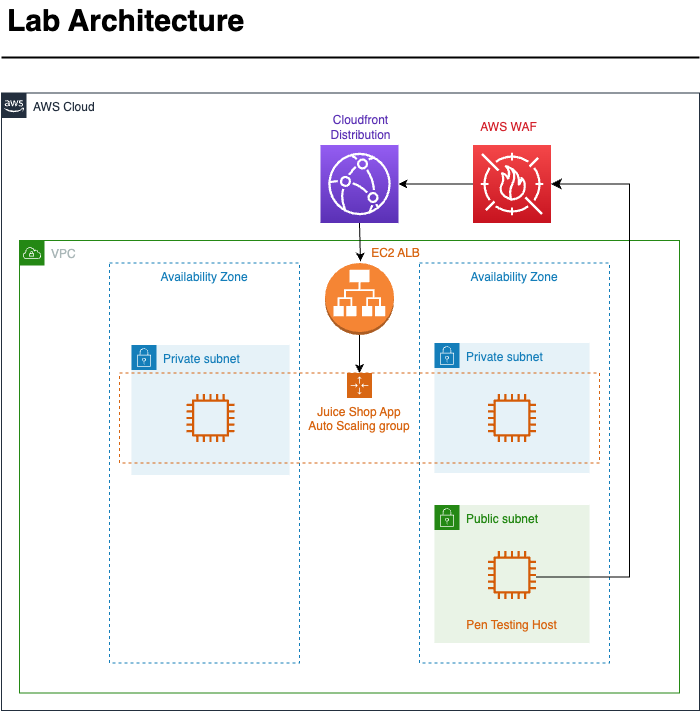
ICON KEY

Various icons are used throughout this lab to call attention to different types of instructions and notes. The following list explains the purpose for each icon:

* **Command:** A command that you must run.
* **Expected output:** A sample output that you can use to verify the output of a command or edited file.
* **Note:** A hint, tip, or important guidance.
* **Learn more:** Where to find more information.
* **Caution:** Information of special interest or importance (not so important to cause problems with the equipment or data if you miss it, but it could result in the need to repeat certain steps).
* **WARNING:** An action that is irreversible and could potentially impact the failure of a command or process (including warnings about configurations that cannot be changed after they are made).
* **Consider:** A moment to pause to consider how you might apply a concept in your own environment or to initiate a conversation about the topic at hand.
* **Security:** An opportunity to incorporate security best practices.
* **Hint:** A hint to a question or challenge.
* **Answer:** An answer to a question or challenge.

LAB ENVIRONMENT

The following diagram shows the basic architecture of the lab environment:



*Image description: The preceding diagram depicts a data flow. Traffic goes from an EC2 instance, called Pen Testing Host, through AWS WAF and then to a Cloudfront distribution that sits in front of an Application Load Balancer. The ALB forwards requests to an Autoscaling Group.*

The following list details the most important resources in the diagram:

* A *VPC* with two *private subnets* and one *public subnet* spread across two Availability Zones.
* An internal-facing *Application Load Balancer* sitting in front of an Autoscaling Group with two nodes.
* A *Cloudfront* distribution fronting the Application Load Balancer.
* Traffic passes through *AWS WAF* before reaching the *Cloudfront* distribution.
* An *EC2* instance called PenTestingHost located in the public subnet.

**Start lab**

1. To launch the lab, at the top of the page, choose **Start lab**.

 You must wait for the provisioned AWS services to be ready before you can continue.

1. To open the lab, choose **Open Console**.

You are automatically signed in to the AWS Management Console in a new web browser tab.

**Do not change the Region unless instructed.**

COMMON SIGN-IN ERRORS

**Error: You must first sign out**



If you see the message, **You must first log out before logging into a different AWS account:**

* Choose the **click here** link.
* Close your **Amazon Web Services Sign In** web browser tab and return to your initial lab page.
* Choose **Open Console** again.

**Error: Choosing Start Lab has no effect**

In some cases, certain pop-up or script blocker web browser extensions might prevent the **Start Lab** button from working as intended. If you experience an issue starting the lab:

* Add the lab domain name to your pop-up or script blocker’s allow list or turn it off.
* Refresh the page and try again.

**Task 1: Configure OWSAP Zap and Firefox**

In this task, you use *Fleet Manager* connect to a Windows Server *EC2* instance. Once connected to the instance, you configure *OWASP Zed Attack Proxy (ZAP)* and *Firefox*. *OWASP ZAP Proxy* is an open source penetration testing tool maintained by the Open Worldwide Application Security Project (OWASP). As a “man-in-the-middle” proxy, ZAP intercepts all communications between a tester’s browser and a target web app. This allows pen testers to inspect, modify, and manipulate requests and responses to uncover vulnerabilities.

1. In a panel to the left of these instructions, you will find a series of URLs. Copy the URL labeled **PenetrationTestingHost** and paste it into a new browser tab.

You are brought to the *Fleet Manager - Remote Desktop* connection page.

**WARNING:** Please make sure that you are using the **Chrome** as the Internet browser because for Fleet Manager RDP, only **Chrome** browser supports bidirectional copying and pasting between RDP sessions and your local machine.

**Note:** If you are unable to use **RDP** with **Fleet Manager**, you can also [connect to your windows instance using a Remote Desktop client](https://labs.skillbuilder.aws/sa/lab/arn%3Aaws%3Alearningcontent%3Aus-east-1%3A470679935125%3Ablueprintversion%2FSPL-TF-200-SIAWAF-1%3A1.2.9-96842914/en-US#connect_rdp).

1. For **Authentication type** choose **Key pair**.
2. From the panel to the left of these instructions, choose  **Download PEM**.
3. Save the file to a directory of your choice.
4. For **Key pair content**, choose **Browse your local machine to select the key pair file**.
5. Select the **Browse** button to find and upload the **PEM** key from your local machine.
6. Select the **Connect** button to launch an RDP session with the *PenTestingHost*.

**Caution:** *Fleet Manager* RDP connections have a maximum session duration of 60 minutes. When that duration is reached, Fleet Manager ends the session. If you run into any issues while interacting with your instance via Fleet Manager, open the **Actions** drop-down menu, and then select **Renew session** to restart your session.

1. Choose the icon with four arrows  to expand the window containing your RDP session.

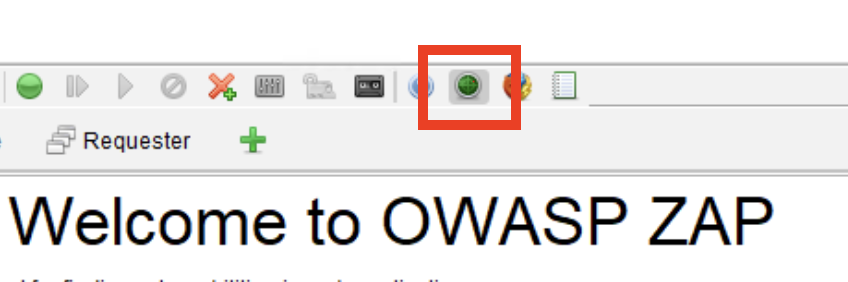
**Note:** You will be prompted with a Networks pop-up window asking: **Do you want to allow your PC to be discoverable by other PCs and devices on this network?** Choose **No**.

1. Choose the desktop shortcut labeled **OWASP ZAP 2.12.0** to open the *ZAP* application.

The *ZAP* logging terminal appears, followed by the application splash screen. Wait for the configuration process to complete and do not close any windows that appear on the screen.

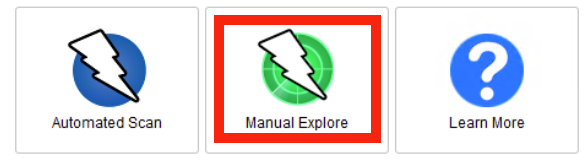
**Learn more:** Refer to [OWASP Zed Attack Proxy (ZAP)](https://www.zaproxy.org/) for more information about *OWASP ZAP*.

1. A popup window appears asking if you want to **persist the ZAP Session**. Select **Yes, I want to persist this session with name based on the current timestamp** and then choose the **Start** button.
2. Choose the **maximize** button at the top of the screen to expand the *ZAP Console* window.
3. Look for and choose the **dark green circle** on the toolbar. This is the **Disable the ZAP HUD** button.



*Image description: The preceding image shows a dark green button on the toolbar.*

1. Choose the **Manual Explore** button in the center of the screen.

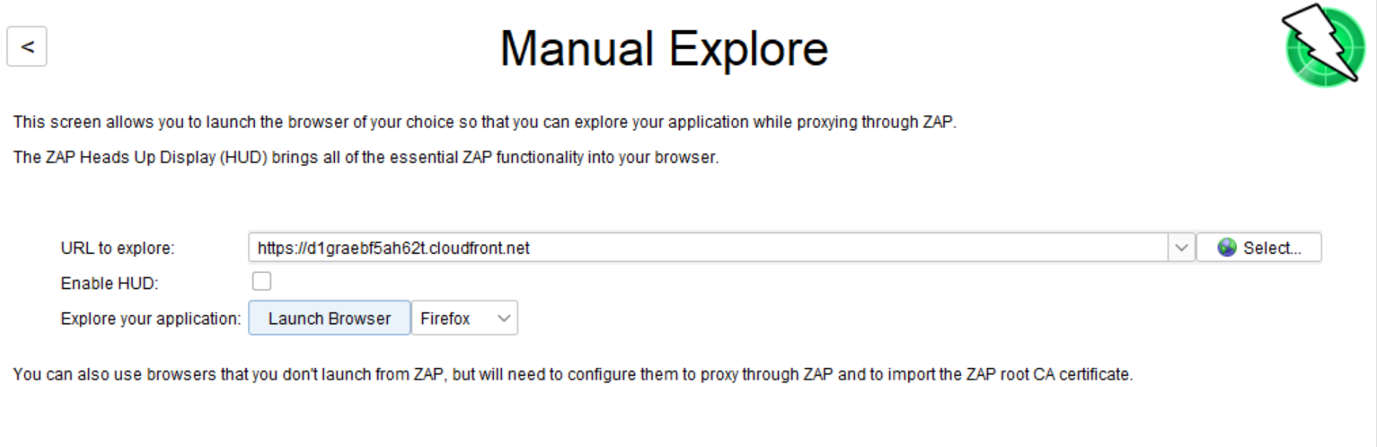


*Image description: The preceding image shows the Manual Explore button in the ZAP UI.*

You are brought to the *Manual Explore* page.

1. Configure the following options:

* From the panel to the left of these instructions, copy the URL labeled **JuiceShopURL** and paste it into the **URL to explore** field in ZAP.
* Leave the **Enable HUD** checkbox **unselected**.
* Open the dropdown menu next to **Explore your application** and select **Firefox**.
* Choose the **Launch Browser** button.



*Image description: The preceding image shows the URL for the UnprotectedJuiceShop, the HUD enabled, and Firefox selected as the browser.*

**Note:** *ZAP* automatically configures *Firefox* to proxy through *ZAP*. This eliminates the need to worry about about certificate validation warnings for sites using HTTPS.

*Firefox* opens and connects to the JuiceShop application.

**Congratulations!** You have successfully configured *OWASP ZAP* and *Firefox* for penetration testing. In the next task, you will conduct reconnaissance about your target.

**Task 2: Reconnaissance**

In this task, you conduct reconnaissance in the hope that it will yield useful information that will aid in your attack. Reconnaissance testing can help you determine which form fields are vulnerable to SQL injection and XSS exploits. You can also find error messages that may reveal details about the database schema or software version being used.

TASK 2.1: GENERATE HTTP REQUESTS

In the following steps, you interact with the Juice Shop application. The HTTP requests generated by your interactions are intercepted and logged by the *ZAP Console*. In subsequent steps, you will modify and replay these requests to exploit the application.

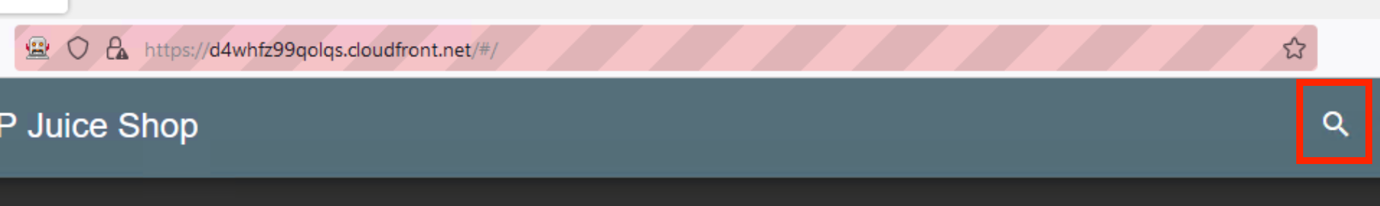
1. Complete the following actions in the web interface:

* Use the search bar to search for

apple juice

.

* Navigate to the registration page **(https://[JuiceShopDomain].cloudfront.net/#/register)** and create a new user.
* Navigate to the login page **(https://[JuiceShopDomain].cloudfront.net/#/login)** and log in as your new user.



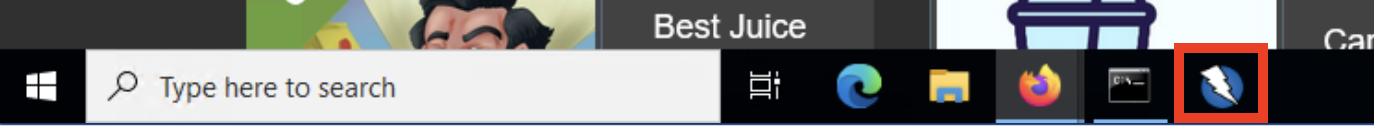
*Image description: The preceding image shows where the search bar is located.*

**Note:** In some cases, you may need to wait a moment for your newly created user to become active. If you are unable to log in, wait 30 seconds and then try again.

TASK 2.2: SQL INJECTION

The API requests that were sent to the application when you searched for apple juice, created a new user, and then logged in as that user were intercepted by *ZAP*. In the following steps, you use the *ZAP Console* to find these requests, modify them, and see if you can exploit the application.

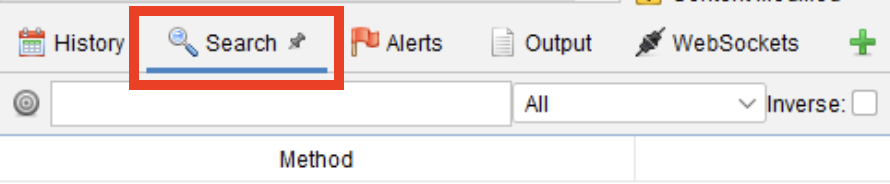
1. In the taskbar at the bottom of the Windows interface, choose the **blue OWASP ZAP icon** to return to the ZAP Console.



*Image description: The preceding image shows Windows Taskbar icon for OWASP ZAP.*

Now let’s use the *ZAP Console* to see if any of the input fields are vulnerable to SQL injection.

1. Choose the **Search** button in the panel at the bottom of the ZAP Console.



*Image description: The preceding image shows Search button in the ZAP Console.*

In the following steps, you attempt to force the Juice Shop application to display an error message that provides you with information about the application and database.

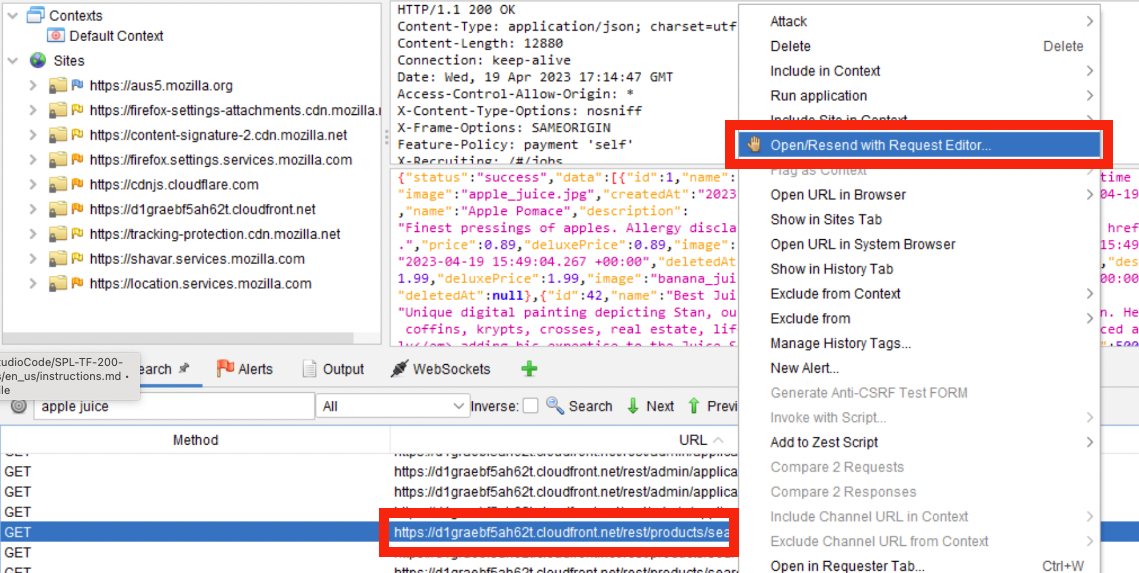
**Note:** Error-based injections take advantage of information contained in error messages to learn about the types of information stored in a database and its structure.

1. **Command:** Enter

apple juice

 in the **Search** field then press **ENTER**.

1. Among the search results, look for and select a **GET** request with the following URL: **https://[JuiceShopDomain].cloudfront.net/rest/products/search?q=**. Open the context menu (right-click) and select **Open/Resend with Request Editor**.



*Image description: The preceding image shows a URL ending with /rest/products/search?q=.*

The *Manual Request Editor* opens. It has two tabs - *Request* and *Response*, each of which is divided into two panels. Headers are displayed in the upper panel and the message body is displayed in the bottom panel. This window enables you to modify and replay HTTP requests. Start by testing to see if you can exploit the search field by passing a boolean attack in the query string.

**Note:** HTTP query strings are the part of the URL that follows a question mark (?) and are used to send additional data to a web server as part of an HTTP request. When you submitted

apple juice

 in the search bar, **apple%20juice** (the URL-encoded equivalent of apple juice) was sent to the API as a query string.

1. With the **Request** tab selected, place your cursor in the text area and update the URL in the first line of the header so that it ends with

/search?q=' or 1=1 --

 and then choose the **Send** button.

**Note:** An SQL boolean attacks is a type of SQL injection that manipulates queries using true/false conditions. In this case, the string

' or 1=1 --

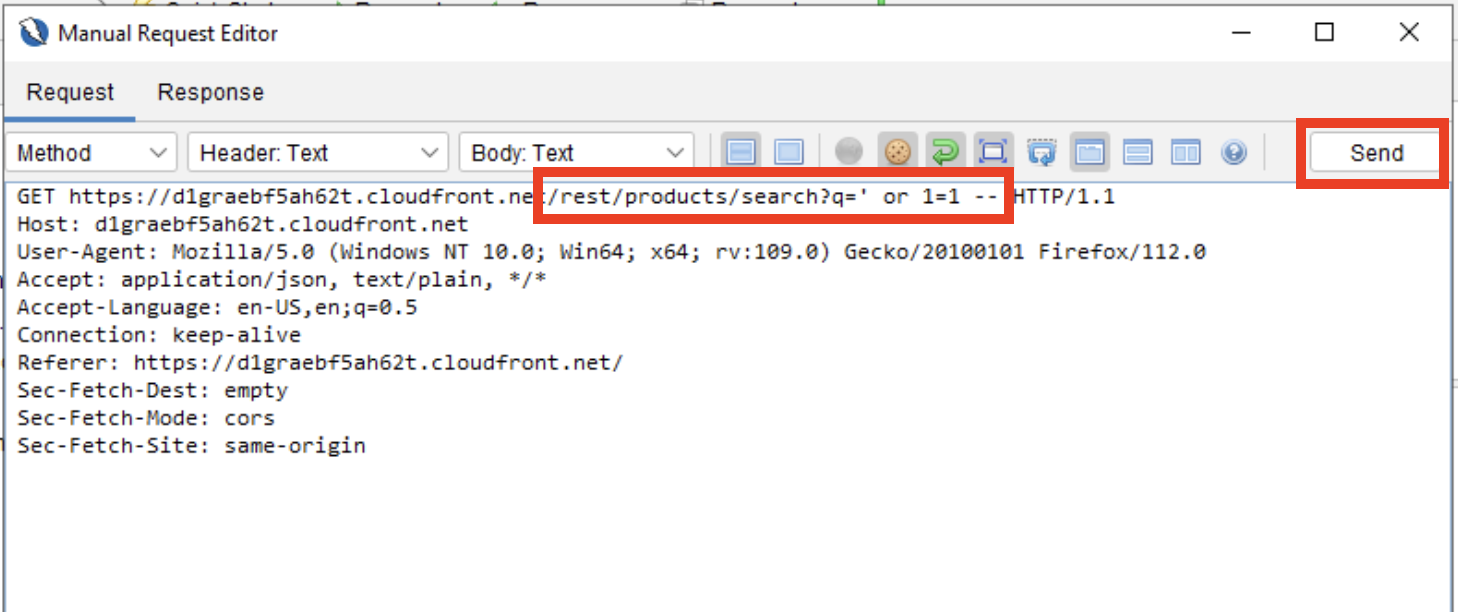
 alters the query so that it always return true, due to the

1=1

 condition. The double hyphen

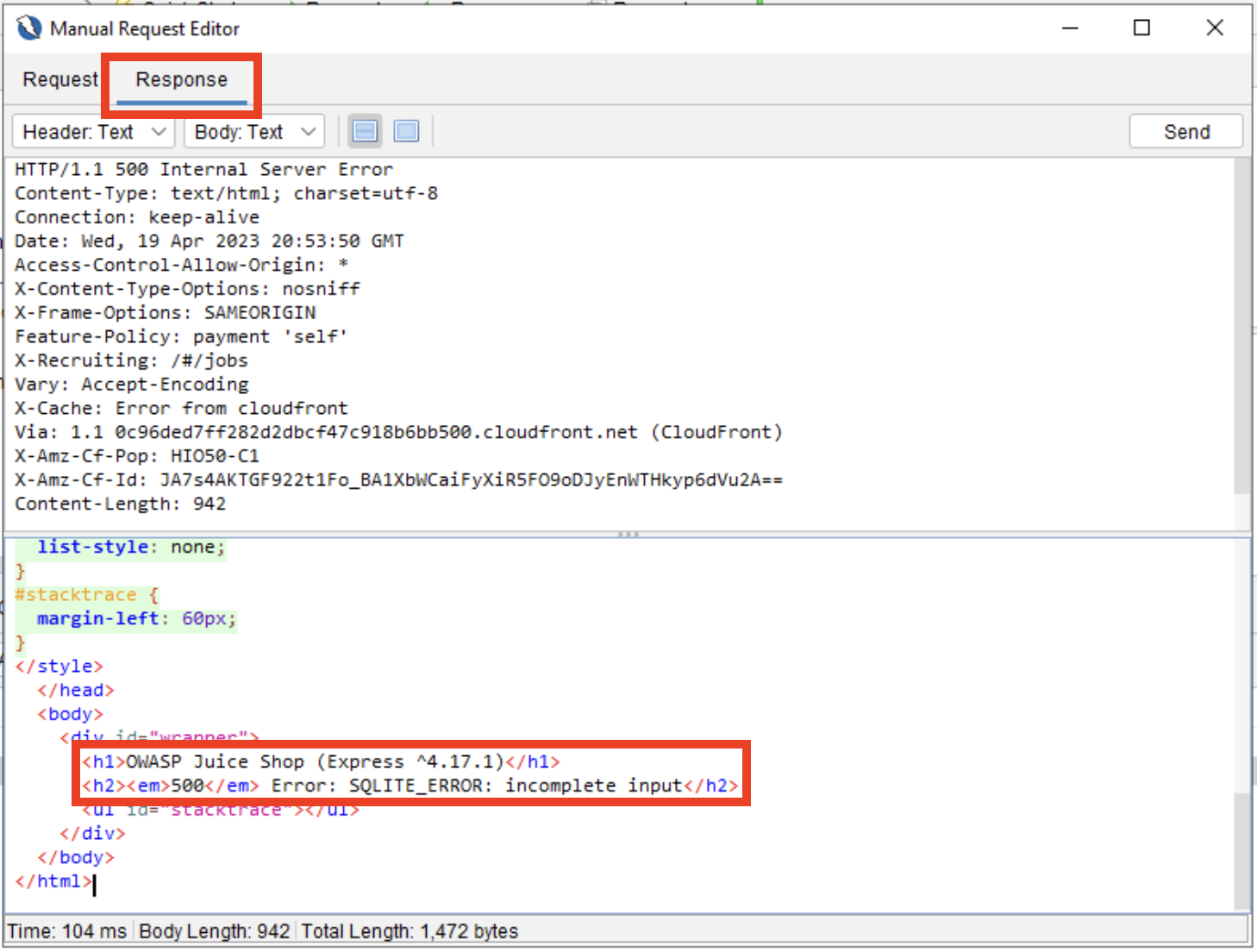
--

 comments out the rest of the query, preventing any further conditions from being applied.



*Image description: The preceding image shows the updated URL and the Send button.*

The **Response** tab opens, displaying an unhandled exception. The message body is displayed in the lower panel. Scroll to the bottom of the panel to view following message:



*Image description: The preceding image shows the an unhandled error in the HTTP response.*

The response has provided you with exactly the information you were looking for and highlights the importance of proper error handling. Based on the response, you now know that the application uses an *SQLite* database.

**Note:** Knowing the database type (*MySQL, SQLite, Oracle,* etc.) is crucial for crafting an effective SQL injection payload. Different database systems have different vulnerabilities and methods for gaining access. Reconnaissance helps ensure you develop an injection that is tailored to the specific configuration of the target database.

**Congratulations!** You have successfully completed your reconnaissance and confirmed that the application is using an *SQLite* database. You also demonstrated that it is susceptible to SQL injection.

**Task 3: Persisted XSS attack**

Now that we’ve discovered that the site is using *SQLite* and vulnerable to SQL injection, let’s see if it’s also susceptible to cross-site scripting attacks. In the following steps, you use a persisted XSS attack to embed a script in the list of registered users.

**Note:** A persisted XSS attack, also known as a stored XSS attack, occurs when an attacker injects malicious scripts into a web application’s database or other permanent storage. This injected code is then served to users when they access the affected web pages, causing the malicious script to execute within their browsers. The attacker can use this script to steal sensitive information, manipulate web content, or perform other unauthorized actions on behalf of the user.

1. Close the **Manual Request Editor** window and return to the search bar.
2. **Command:** In the **Search bar** enter

api/Users

 and press **ENTER**.

1. From the search results, choose the **POST** request whose URL ends with **/api/Users**, open the **Context menu** (right-click) and select **Open/Resend with Request Editor**.

The request opens in the *Manual Request Editor*.

**Note:** This is the request that was sent to the server when you created a new user in the application interface. It is a *POST* request sent to the *Users* endpoint and includes a variety of standard headers including a User-Agent, the Content-Type, and a cookie. The message body should look similar to the following:

{"email":"myuser@example.com","password":"123456","passwordRepeat":"123456","securityQuestion":{"id":1,"question":"Your eldest siblings middle name?","createdAt":"2023-04-19T15:48:55.057Z","updatedAt":"2023-04-19T15:48:55.057Z"},"securityAnswer":"123456"}

**Consider:** It’s possible that not all of the data included in this message body is required to create a new user. Let’s try to create a new user using only the *email*, *password*, and *passwordRepeat* keys and insert the XSS payload into the *email* key-value pair.

1. **Command:** Place your cursor inside of the **message body panel** and replace the existing message with the following text:

{"email":"<iframe src=\"javascript:alert(`xss`)\">","password":"123456","passwordRepeat":"123456"}

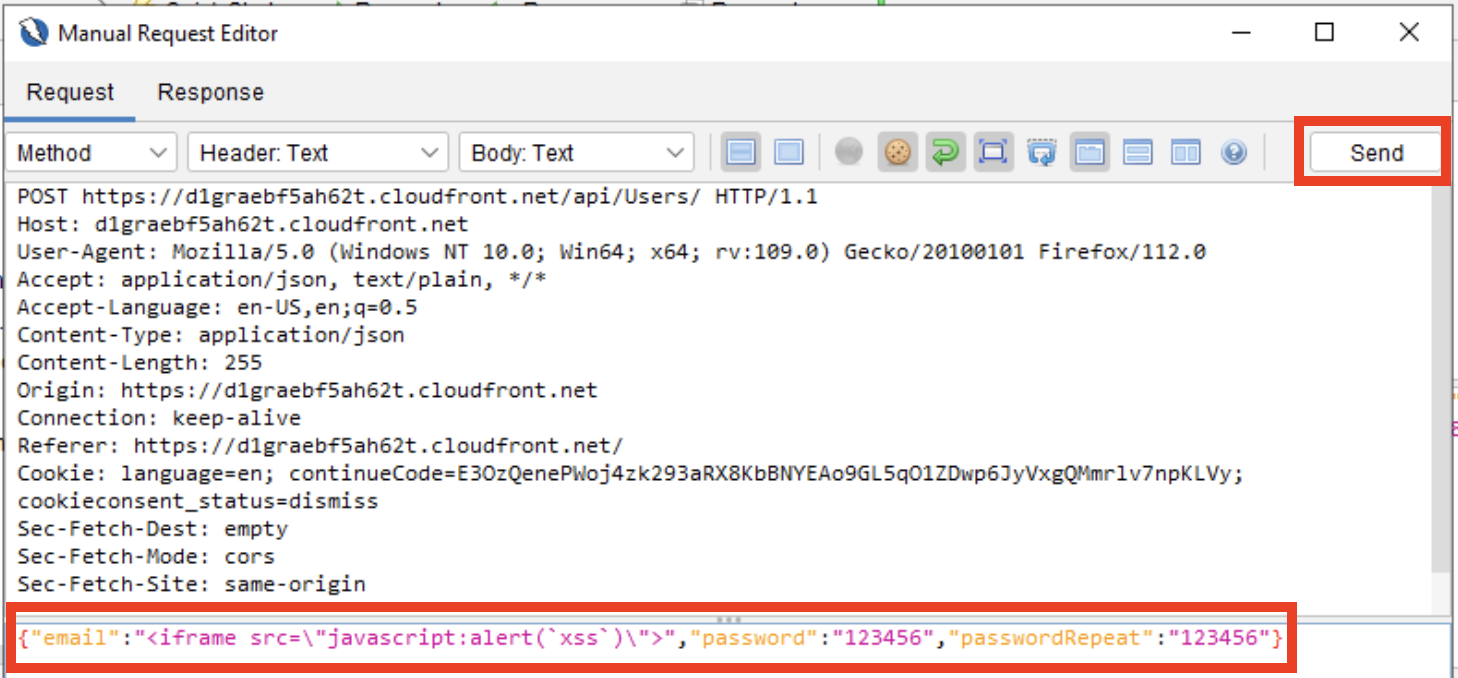
The following list explains the components of the script:

* *iframe*: Creates an iframe element
* *src=*: Sets the iframe’s source (src) attribute to a JavaScript code snippet
* *javascript:alert(*

*xss*

*)*: Triggers an alert with the message “xss”

1. Choose the **Send** button to replay the request with your modified payload.



*Image description: The preceding image shows a modified HTTP request containing an XSS attack in the payload.*

**Expected output:**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\* This is OUTPUT ONLY. \*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

{"status":"success","data":{"username":"","role":"customer","deluxeToken":"","lastLoginIp":"0.0.0.0","profileImage":"/assets/public/images/uploads/default.svg","isActive":true,"id":22,"email":"","updatedAt":"2023-04-19T21:28:08.976Z","createdAt":"2023-04-19T21:28:08.976Z","deletedAt":null}}

Great work! The *Response* tab opens and displays a message stating that your user was created. Your malicious script has now been written to the database. In the future, if another someone views this user’s profile, the script will launch inside of their browser.

**Note:** The JSON response includes additional fields such as *deluxeToken*, and *role* that you did not include in your request. This suggests that the *Users* table in the database includes attributes that are not visible in the website UI. These may come in handy in subsequent tasks.

**Congratulations!** You successfully launched an XSS attack against the website.

**Task 4: Challenge - Bypass the login page**

This is the first challenge task in this lab. Use what you have learned about *ZAP*, *SQL injection attacks*, and the *Juice Shop database* to complete the task. If you need assistance, expand the *Hint* menus to reveal additional information. If you are unable to complete the task, expand the *Solution* menu and follow the instructions inside it to achieve your objective. Do not proceed to the next task without first completing this challenge.

OBJECTIVE

Your objective in this challenge is to bypass the *Juice Shop* login page.

CHALLENGE SUB-TASKS

* Navigate to **https://[JuiceShopDomain].cloudfront.net/#/login**.
* Find an exploit that enables you to log into the website as an authenticated user.

**Hint 1**

**Hint 2**

**Hint 3**

**Solution**

**Congratulations!** You have successfully completed the first challenge and bypassed the login page.

**Task 5: Challenge - Create a new admin user**

This is the second challenge task in this lab. Just like the previous challenge, if you need assistance, you can expand the *Hint* and *Solution* menus. Do not proceed to the next task without first completing this challenge.

OBJECTIVE

Your objective in this challenge is to exploit the user registration page and create a new user with administrative privileges.

CHALLENGE SUB-TASKS

* Use the *ZAP Console* to find an exploit that enables you to create a user with administrative privileges.

**Hint 1**

**Hint 2**

**Solution**

**Congratulations!** You have successfully completed the first challenge and bypassed the login page.

**Task 6: Creating a Web ACL in AWS WAF**

Now that you’ve confirmed that the application is vulnerable to SQLi and XSS attacks, it’s time to defend it. In this task, you create a web access control list (web ACL) in *AWS WAF*. A web ACL is a web application firewall that lets you monitor the HTTP and HTTPS requests to AWS resources.

1. In the top-right corner of the screen, choose the icon with four arrows  to minimize the window containing your RDP session.
2. At the top of the AWS Management Console, in the search bar, search for and choose

WAF & Shield

.

1. Choose the **Create web ACL** button.
2. Before entering any configuration details, under **Resource type**, select  **CloudFront distributions**.

The page should refresh.

1. In the panel at the top of the screen, enter the following configuration:

* **Name**:

JuiceShopACL

* **Description**:

Managed Rule sets and custom rules for JuiceShop

* **CloudWatch metric name**:

JuiceShopACL

* **Resource type**: **Amazon CloudFront distributions**

1. Scroll to the bottom of the screen and in the *Associated AWS resources* panel, choose **Add AWS resources**.

The *Add AWS resources* popup appears.

1. Select the **checkbox** next to your *Cloudfront distribution* and then choose **Add**.
2. Choose the **Next** button.

You are brought to the *Add rules and rule groups* page.

1. In the **Rules** panel, open the **Add rules** dropdown menu and select **Add managed rule groups**.

Start by adding managed rule groups to your ACL. Managed rule groups are collections of predefined, ready-to-use rules created and maintained by AWS and AWS Marketplace sellers.

1. At the top of the list of managed rule groups, select the  **AWS managed rule groups** dropdown menu.
2. Scroll down the page to the **Core rule set** and choose the **Add to web ACL** toggle button next to it.

**Note:** The *Core rule set* contains rules that are generally applicable to web applications. This group provides protection against exploitation of a wide range of vulnerabilities, including those described in *OWASP* publications.

1. Continue down the list and choose the toggle button next to **SQL database**.

**Note:** The *SQL database* rule group contains rules to block request patterns associated with exploitation of SQL databases, including SQL injection attacks.

1. Scroll to the bottom of the page and select **Add rules**.

The managed rule groups you added are displayed at the top of the screen.

**Note:** Examine the panel immediately below the rules. Your ACL is currently consuming 900 out of a possible 5000 web ACL capacity units (WCUs). Applying ACLs requires compute resources, which are measured in WCUs. AWS WAF calculates capacity differently for each rule type, to reflect each rule’s relative cost. Simple rules that cost little to run use fewer WCUs than more complex rules that use more processing power. For example, a size constraint rule statement uses fewer WCUs than a statement that inspects requests using a regex pattern set.

1. Scroll to the bottom of the page and choose the **Next** button.

You are brought to the **Set rule priority** page. This page allows you to change the order in which rules are applied.

1. Leave the default priority unchanged and choose the **Next** button.
2. Do not change any of the metrics settings, scroll to the bottom of the page, and choose the **Next** button.
3. Take a moment to review your ACL and then choose the **Cerate web ACL** button at the bottom of the page.

**Congratulations!** You’ve successfully created your first web ACL. In the next task, you will see it in action.

**Task 7: Testing your web ACL**

In this task, you replay the attacks you launched at the earlier in this lab, to verify that your newly created web ACL mitigates them.

1. From the panel to the left of these instructions, copy the URL labeled **PenetrationTestingHost** and paste it into a new browser tab to return to your **Fleet Manager** session.
2. If your existing session has ended, choose the **Try again** button and use the PEM key you previously downloaded to reconnect to the **PenetrationTestingHost**.
3. Choose the icon with four arrows  to expand the window containing your RDP session.

Let’s start by checking to see if the ACL will prevent users from exploiting the search bar with a boolean attack.

1. Open the *ZAP Console* and select the **Search** tab.
2. **Command:** Enter

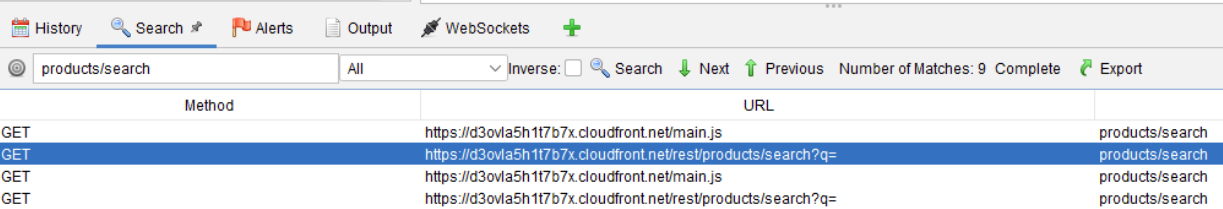
products/search

 in the **Search** bar and then select the **Search** button.

1. Locate and choose the **GET** request in the search results whose URL ends with

/rest/products/search?q=

. Open the context menu (right-click) and select **Open/Resend with Request Editor**.



*Image description: The preceding image shows an HTTP request to /rest/products/search?q= in the ZAP search window.*

The Manual Request Editor opens.

1. **Command:** Update the end of the URL in the first line of the request to

/rest/products/search?q=' or 1=1 --

 and then choose the **Send** button.

**Expected output:**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\* This is OUTPUT ONLY. \*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<HTML><HEAD><META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=iso-8859-1">

<TITLE>ERROR: The request could not be satisfied</TITLE>

</HEAD><BODY>

<H1>403 ERROR</H1>

<H2>The request could not be satisfied.</H2>

<HR noshade size="1px">

Request blocked.

We can't connect to the server for this app or website at this time. There might be too much traffic or a configuration error. Try again later, or contact the app or website owner.

<BR clear="all">

If you provide content to customers through CloudFront, you can find steps to troubleshoot and help prevent this error by reviewing the CloudFront documentation.

<BR clear="all">

<HR noshade size="1px">

<PRE>

Generated by cloudfront (CloudFront)

Request ID: HgKOwHjr5vLod9HGr-JvPt5OCg6nJz4nqGUi-V36\_GO3gdJxXBZSVg==

</PRE>

<ADDRESS>

</ADDRESS>

</BODY></HTML>

Excellent! So far, you ACL is working as expected. Now let’s see if your ACL prevents attackers from exploiting the login page.

1. Close the **Manual Request Editor**.
2. **Command:** Return to the **Searches** tab and enter

rest/User/login

 in the **Search bar**.

1. Select any of the matching results, open the context menu (right-click), and choose **Open/Resend with Request Editor**.
2. **Command:** Choose the **Request** tab and replace the payload with the following text:

{"email":"' or 1=1 --","password":"123"}

1. Choose the **Send** button.

**Expected output:**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\* This is OUTPUT ONLY. \*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<HTML><HEAD><META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=iso-8859-1">

<TITLE>ERROR: The request could not be satisfied</TITLE>

</HEAD><BODY>

<H1>403 ERROR</H1>

<H2>The request could not be satisfied.</H2>

<HR noshade size="1px">

Request blocked.

We can't connect to the server for this app or website at this time. There might be too much traffic or a configuration error. Try again later, or contact the app or website owner.

<BR clear="all">

If you provide content to customers through CloudFront, you can find steps to troubleshoot and help prevent this error by reviewing the CloudFront documentation.

<BR clear="all">

<HR noshade size="1px">

<PRE>

Generated by cloudfront (CloudFront)

Request ID: cUx-NuvsiSYbbVZiLxWztDAcDeq4bpP9Vi3YW4\_f76lOH2881evgOA==

</PRE>

<ADDRESS>

</ADDRESS>

</BODY></HTML>

Once again, your managed rule groups have successfully defended the Juice Shop application. Now let’s see how they handle an XSS attack.

1. **Command:** Return to the **Search** tab and enter

api/Users

 in the **Search bar**.

1. Locate the **POST** request that created the **xss@eample.com** user. Select it, open the context menu (right-click), and choose **Open/Resend with Request Editor**.
2. **Command:** Choose the **Request** tab and replace the payload with the following text:

{"email":"<iframe src=\"javascript:alert(`xss`)\">","password":"123456","passwordRepeat":"123456"}

1. Choose the **Send** button.

**Expected output:**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\* This is OUTPUT ONLY. \*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<HTML><HEAD><META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=iso-8859-1">

<TITLE>ERROR: The request could not be satisfied</TITLE>

</HEAD><BODY>

<H1>403 ERROR</H1>

<H2>The request could not be satisfied.</H2>

<HR noshade size="1px">

Request blocked.

We can't connect to the server for this app or website at this time. There might be too much traffic or a configuration error. Try again later, or contact the app or website owner.

<BR clear="all">

If you provide content to customers through CloudFront, you can find steps to troubleshoot and help prevent this error by reviewing the CloudFront documentation.

<BR clear="all">

<HR noshade size="1px">

<PRE>

Generated by cloudfront (CloudFront)

Request ID: YdWwzs5h6YvEelmPmwYxuibdcOVA0CJX7IpgIz60C1nm7WrIKQWQng==

</PRE>

<ADDRESS>

</ADDRESS>

</BODY></HTML>

The request was successfully blocked. So far, so good. Now, let’s use the same HTTP request and attempt to create a new user with administrative privileges.

1. Return to the **Request** tab.
2. **Command:** Replace the payload with the following text:

{"email":"newadmin@example.com","password":"123456","passwordRepeat":"123456","role":"admin"}

1. Choose the **Send** button.

**Expected output:**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\* This is OUTPUT ONLY. \*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

{"status":"success","data":{"username":"","deluxeToken":"","lastLoginIp":"0.0.0.0","profileImage":"/assets/public/images/uploads/defaultAdmin.png","isActive":true,"id":24,"email":"newadmin@example.com","role":"admin","updatedAt":"2023-04-18T19:18:02.884Z","createdAt":"2023-04-18T19:18:02.884Z","deletedAt":null}}

Uh-oh! A new user with **“role”:“admin”** was created. This is not surprising, as the message body did not include anything obviously malicious. One can easily imagine many legitimate situations in which an API request could contain a string like **“role”:“admin”**. In this specific case, however, you have identified this as a critical vulnerability and will need to create a custom WAF rule to block such requests.

**Congratulations!** Well done! You created a web ACL that blocks common SQLi and XSS attacks. Unfortunately, the Juice Shop application remains vulnerable to one of the attacks you discovered. In the next task, you create a custom rule to address this.

**Task 8: Working with the AWS WAF rule builder**

The managed rule groups you applied in the previous task blocked most of your attacks, but one of them got through. In this task, you use the AWS WAF rule builder to create a custom WAF rule that designed to mitigate the attacks not prevented by the managed rule groups.

1. In the top-right corner of the screen, choose the icon with four arrows  to minimize the window containing your RDP session.
2. At the top of the AWS Management Console, in the search bar, search for and choose

WAF & Shield

.

1. Choose the **JuiceShopACL** link.

The *JuiceShopACL* page appears, displaying a chart of requests inspected by the web ACL.

1. Open the **Rules** tab.
2. Choose the **Add rules** button and then select **Add my own rules and rule groups** from the dropdown menu.

**Note:** Custom rules enable you to build your own logic for processing requests and provide a higher degree of flexibility and control. Rules can inspect many aspects of a request, including its headers, query strings, URI path, and body. Then, actions are applied to the request according to the conditions specified in the rule.

In the the following steps, you configure a custom rule that prevents the

api/Users

 endpoint from accepting requests containing **“role”:“admin”** in the message body.

**Security:** Give careful thought to the statements included in your rules. Rules that are too broad may result in false positives. On the other hand, granular rules - particularly those that involve JSON parsing - consume significantly more WCUs.

1. In the **Rule type** panel, select **Rule builder**.
2. Scroll down the page to the **Name** field and enter

CustomSQLiRuleForJuiceShop

.

1. Open the **If a request** dropdown menu and select **matches all the statements (AND)**.

**Note:** Multiple statements can be chained together using AND, OR, and NOT operators.

1. In the **Statement 1** panel, enter the following configuration:

* **Negate statement results**: Leave checkbox unselected
* **Inspect**: **URI path**
* **Match type**: **Contains string**
* **String to match**:

api/Users

* **Text transformation**: **None**

**Note:** This statement should prevent the search bar from being exploited. It blocks all requests that include a query string that start with an apostrophe (').

1. In the **Statement 2** panel, enter the following configuration:

* **Negate statement results**: Leave checkbox unselected
* **Inspect**: **Body**
* **Content type**: **Plain text**
* **Match type**: **Contains string**
* **String to match**:

"role":"admin"

* **Text transformation**: **None**
* **Oversize handling**: **Continue - Inspect the contents that are within the size limitations according to the rule inspection criteria**

1. In the **Action** panel, select **Block**.

**Note:** This statement should prevent the user registration page from being exploited. It blocks all requests whose URI includes the string

api/Users

 and whose message body includes **“role”:“admin”**. Note that this rule is only applied to requests passing through **Cloudfront**. Site administrators will still be able to create new admin users by connecting directly with the Juice Shop server.

Now that your rule has been configured, take a moment to confirm that it is valid.

1. Scroll to the top of the page and choose the **Validate** button above the **Name** field.

A banner appears confirming that the rule is valid.

1. At the bottom of the page, choose the **Add rule** button.

**Congratulations!** You have successfully created a custom rule. Now it’s time to test it.

**Task 9: Test the custom rule**

In this task, you return to the *OWASP ZAP Console* and again attempt to exploit the search and user registration functionality in the Juice Shop application.

1. Retrieve the URL labeled **PenetrationTestingHost** from the panel to the left of the instructions and paste it into a new browser tab.
2. If your previous RDP session has ended, choose the **Try again** button and use the PEM key to reconnect to the **PenetrationTestingHost**.
3. Choose the icon with four arrows  to expand the window containing your RDP session.
4. **Command:** Return to the **Search** tab and enter

api/Users

 in the **Search bar**.

1. Select any of the matching **POST** requests, open the context menu (right-click), and choose **Open/Resend with Request Editor**.
2. Choose the **Request** tab and replace the payload with the following text:

{"email":"newadmin2@example.com","password":"123456","passwordRepeat":"123456","role":"admin"}

1. Choose the **Send** button to send the malicious payload again.

**Expected output:**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\* This is OUTPUT ONLY. \*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<HTML><HEAD><META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=iso-8859-1">

<TITLE>ERROR: The request could not be satisfied</TITLE>

</HEAD><BODY>

<H1>403 ERROR</H1>

<H2>The request could not be satisfied.</H2>

<HR noshade size="1px">

Request blocked.

We can't connect to the server for this app or website at this time. There might be too much traffic or a configuration error. Try again later, or contact the app or website owner.

<BR clear="all">

If you provide content to customers through CloudFront, you can find steps to troubleshoot and help prevent this error by reviewing the CloudFront documentation.

<BR clear="all">

<HR noshade size="1px">

<PRE>

Generated by cloudfront (CloudFront)

Request ID: BWqQgCAH\_8UUbEnDXWl2lwVq6zJ8ejzwBgouZ-79SNepchHkntX-Xw==

</PRE>

<ADDRESS>

</ADDRESS>

</BODY></HTML>

**Congratulations!** Your custom rule blocked the attack. Of course, it’s possible that the application remains vulnerable to other attacks and it would be wise to enable logging so that you can gather additional information about traffic passing through your web ACL. These logs can be sent to an Amazon CloudWatch Logs log group, an Amazon Simple Storage Service (Amazon S3) bucket, or an Amazon Kinesis Data Firehose.

**Conclusion**

You have successfully done the following:

* Deployed simple SQLi and XSS attacks to compromise a web application.
* Created a web access control list (web ACL) in AWS WAF and associated it with an Application Load Balancer.
* Applied AWS managed rule groups to a web ACL.
* Applied custom rules to a web ACL.

**End lab**

Follow these steps to close the console and end your lab.

1. Return to the **AWS Management Console**.
2. At the upper-right corner of the page, choose **AWSLabsUser**, and then choose **Sign out**.
3. Choose **End lab** and then confirm that you want to end your lab.

**Additional Resources**

* For more information about how to use AWS WAF, see [AWS WAF Documentation](https://docs.aws.amazon.com/waf/latest/developerguide/waf-chapter.html).
* To learn how to integrate AWS WAF into a comprehensive web application security enforcement policy, see [Defense in depth using AWS Managed Rules for AWS WAF (part 1)](https://aws.amazon.com/blogs/security/defense-in-depth-using-aws-managed-rules-for-aws-waf-part-1/) and [Deploying defense in depth using AWS Managed Rules for AWS WAF (part 2)](https://aws.amazon.com/blogs/security/deploying-defense-in-depth-using-aws-managed-rules-for-aws-waf-part-2/)

**Connect to your Windows instance using RDP**

* To the left of the instructions you are currently reading, choose **Download PEM**.
* Save the file to the directory of your choice.
* Open the [Amazon EC2 console](https://console.aws.amazon.com/ec2/)
* In the navigation pane, select **Instances**. Select the **PenTestingHost** Instance and then choose **Connect**.
* On the **Connect to instance** page, choose the **RDP client** tab, and then choose **Get password**.
* Choose **Browse** and navigate to the private key (**.pem**) file you created earlier. Select the file and choose **Open** to copy the entire contents of the file to this window.
* Choose **Decrypt Password**. The console displays the default administrator password for the instance under **Password**, replacing the **Get password** link shown previously. Save the password in a safe place. This password is required to connect to the instance.

FOR YOUR RDP CLIENT, USE THE FOLLOWING DETAILS TO CONNECT:

* **PenTestingHost IP:** Copy and paste the Public IP from the EC2 console.
* **Username:** Enter

**Administrator**

.

* **Password:** Copy and paste the **password** that you saved previously.

[Return to the instructions](https://labs.skillbuilder.aws/sa/lab/arn%3Aaws%3Alearningcontent%3Aus-east-1%3A470679935125%3Ablueprintversion%2FSPL-TF-200-SIAWAF-1%3A1.2.9-96842914/en-US#bastion)

For more information about AWS Training and Certification, see [*https://aws.amazon.com/training/*](https://aws.amazon.com/training/).

*Your feedback is welcome and appreciated.*  
If you would like to share any feedback, suggestions, or corrections, please provide the details in our [*AWS Training and Certification Contact Form*](https://support.aws.amazon.com/#/contacts/aws-training).