Protecting workloads on AWS from the Instance to the Edge

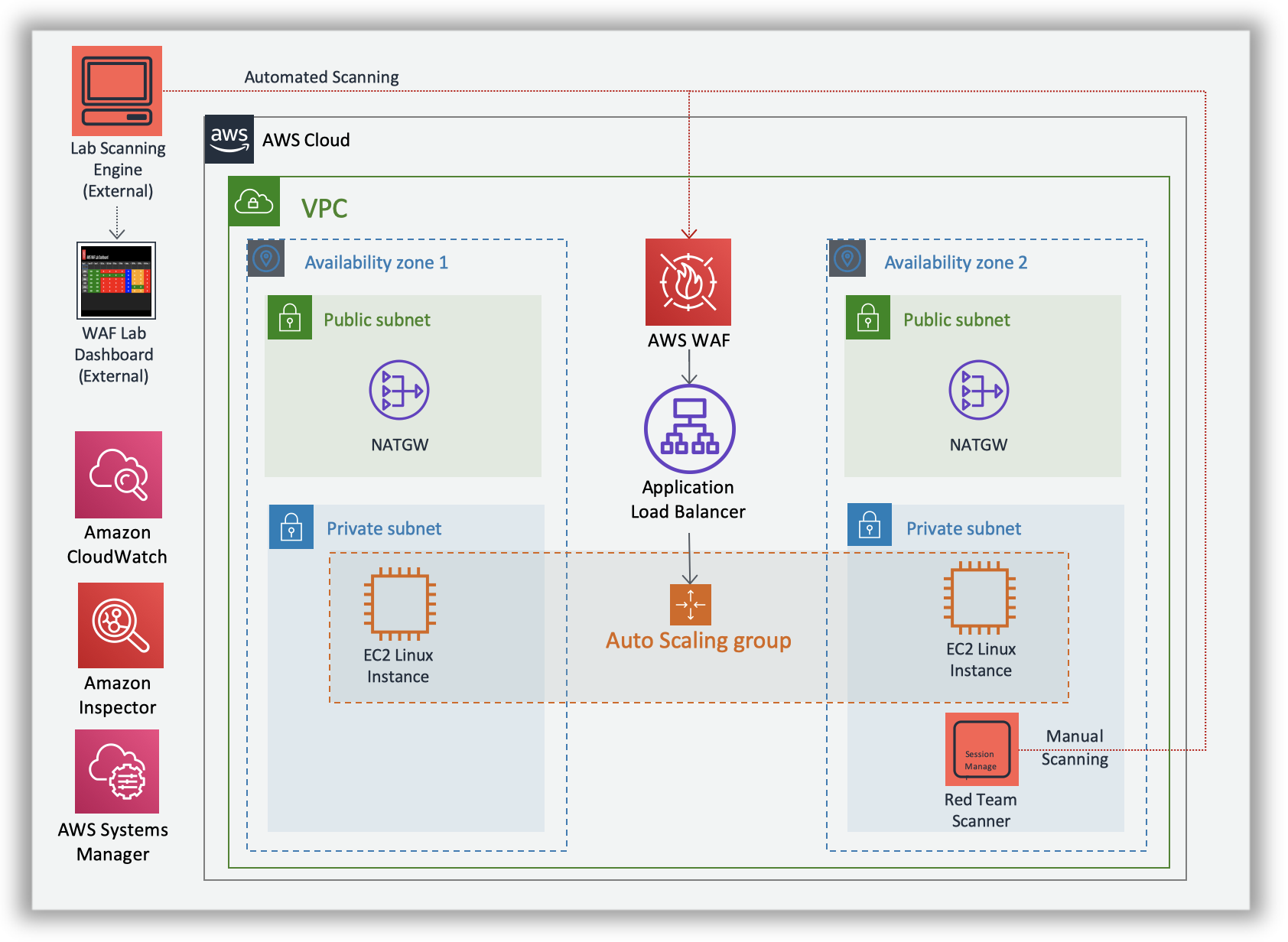
Welcome to the AWS Protecting Workloads Workshop!

In this workshop, you will build an environment consisting of two Amazon Linux web servers behind an application load balancer. The web servers will be running a PHP web site that contains several vulnerabilities.

## **Scenario**

Welcome to Widgets LLC! You have just joined the team and your first task is to enhance security for the company website. The site runs on Linux, PHP and Apache and uses an EC2 an autoscaling group behind an Application Load Balancer (ALB). After an initial architecture assessment you have found multiple vulnerabilities and configuration issues. The dev team is swamped and will not be able to remediate code level issues for several weeks. Your mission in this workshop round is to build an effective set of controls that mitigate common attack vectors against web applications, and provide you with the monitoring capabilities needed to react to emerging threats when they occur.

## **Workshop Architecture**



## **Environment setup**

To setup the workshop environment, launch the CloudFormation stack below

**US East 1 (N. Virginia)**

[Deploy in us-east-1](https://console.aws.amazon.com/cloudformation/home?region=us-east-1#/stacks/new?stackName=pww&templateURL=https://s3.amazonaws.com/protecting-workloads-workshop/public/artifacts/pww-workshop-env-build.yml)

1. Click ***Next*** on the Specify Template section.
2. Click ***Next***
3. Click Next on the ***Configure stack options*** section.
4. Finally, acknowledge that the template will create IAM roles under Capabilities and click **Create**.

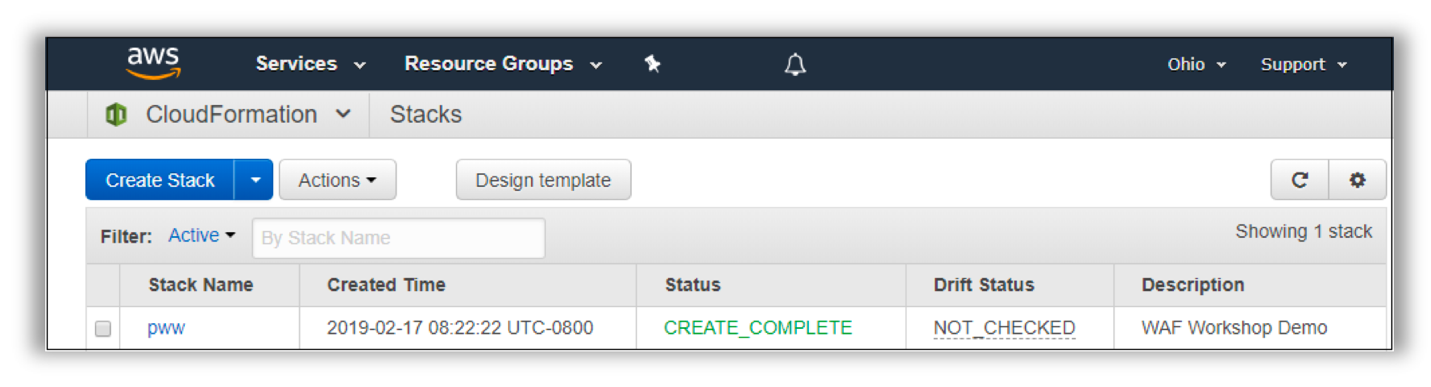
# Mitigating Common Web Application Attack Vectors Using AWS WAF - Assess Phase

In the previous Build Phase, you built a CloudFormation stack that contains a PHP website on Amazon EC2 instances behind an application load balancer. You are now going to assess the posture of the site and then add an AWS WAF Web ACL to your site. In this section you will do the following tasks:

1. Identify the stack you built
2. Look up the output values for your environment and test access
3. Use your Red Team Host to test for website vulnerabilities

## **Identify the stack that you built**

1. Go to the CloudFormation console in the same AWS region in which you created the stack in the Build Phase. You should see a list of stacks similar to the figure below. Locate the stack you created. In this documentation, the name of the stack is pww. Copy this stack name into a scratch file on your workstation in case you need it later.



## **Look up the Stack Outputs**

1. Go to the stack outputs and look for the website URL stored in the **albEndpoint** output value. Test access to the site by right clicking and opening in a new tab. Note the URL for your site as this will be used throughout this workshop round.
2. While still in stack outputs, right click the link in **RedTeamHostSession** and open in new tab. This will launch an AWS Systems Manager Session Manager to the host you will use to perform manual scans against your site URL.

**AWS Systems Manager Session Manager**

Session Manager is a fully managed AWS Systems Manager capability that lets you manage your Amazon EC2 instances through an interactive one-click browser-based shell or through the AWS CLI. Session Manager provides secure and auditable instance management without the need to open inbound ports, maintain bastion hosts, or manage SSH keys.

**Attention**

Please ensure you are **using the improved AWS WAF console and API experience** for this workshop.

## **Website Scanning Environment and Tools**

In order to test your AWS WAF ruleset, this lab has been configured with two scanning capabilities; a Red Team Host where you can invoke manual scanning and an automated scanner which runs from outside your lab environment.

The scanner performs 10 basic tests designed to help simulate and mitigate common web attack vectors.

1. Canary GET - Should not be blocked
2. Canary POST - Should not be blocked
3. SQL Injection (SQLi) in Query String
4. SQL Injection (SQLi) in Cookie
5. Cross Site Scripting (XSS) in Query String
6. Cross Site Scripting (XSS) in Body
7. Inclusion in Modules
8. Cross Site Request Forgery (CSRF) Token Missing
9. Cross Site Request Forgery (CSRF) Token Invalid
10. Path Traversal

Workshop Architecture

**Note about Tests**

These basic tests are designed to provide common examples you can use to test AWS WAF functionality. You should perform thorough analysis and testing when implementing rules into your production environments.

### **Website Scanning Environment and Tools - Manual Scanning**

Once you have started a Session Manager connection to your Red Team Host, the scanner script can be invoked by typing the following command:

runscanner

The scanner script will run each of the tests above and report back the following information:

* **Request**: The HTTP request command used.
* **Test Name**: The name of the test from list above.
* **Result**: The HTTP status code returned.

The logic in the scanner script color codes the response as follows:

* **Green**: 403 - Forbidden (Except for canary GET and POST tests.)
* **Red**: 200 - OK
* **Blue**: 404 - Not Found
* **Yellow**: 500 - Internal Server Error

**About Scanner Tests and Colors**

The color coding of the tests is provided to help to quickly assess the behavior of your WAF rules against their intended behavior. The goal is to achieve green color responses for all the tests. The purpose of the canary GET and POST requests are to ensure you have not unintentionally blocked legitimate traffic to your test site. These two tests should always return a 200 - OK response.

What are the results of running the scanner script? Were the simulated malicious requests blocked? As you can see by running the script there are several vulnerabilities that need to be addressed. In the remediate phase you will configure an AWS WAF Web ACL to block these requests. When AWS WAF blocks a web request based on the conditions that you specify, it returns HTTP status code 403 (Forbidden). For a full view of the request and response information, you can paste the **Request** command directly into the console and add the --debug argument.

**Note about Testing Tool**

The scanner script uses an open source [HTTP client called httpie](https://httpie.org/). HTTPie—aitch-tee-tee-pie—is a command line HTTP client with an intuitive UI, JSON support, syntax highlighting, wget-like downloads, plugins, and more.

# Mitigating Common Web Application Attack Vectors Using AWS WAF - Remediate Phase

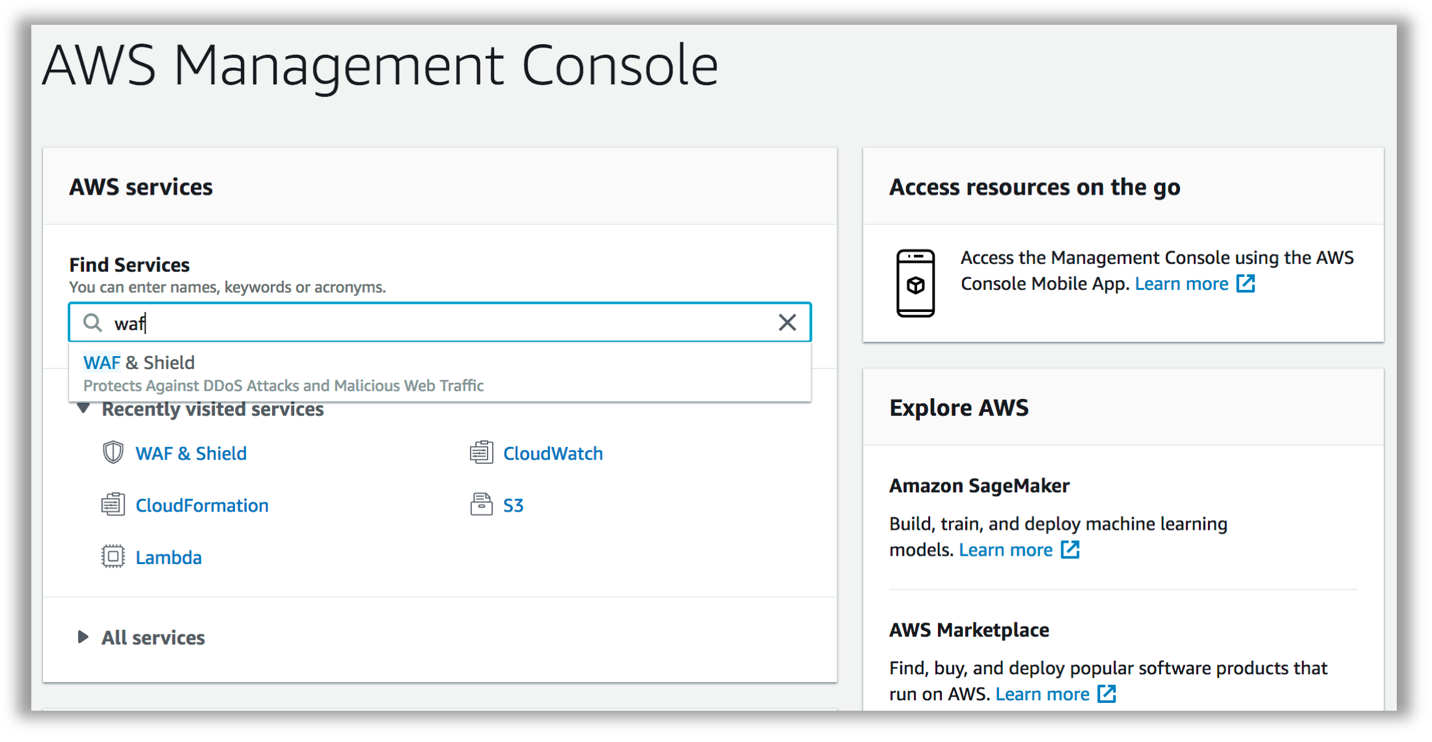
In the previous Build Phase, you identified several vulnerabilities in your web application. You are now going to design and implement an AWS WAF ruleset to help mitigate these vulnerabilities. In this section you will do the following tasks:

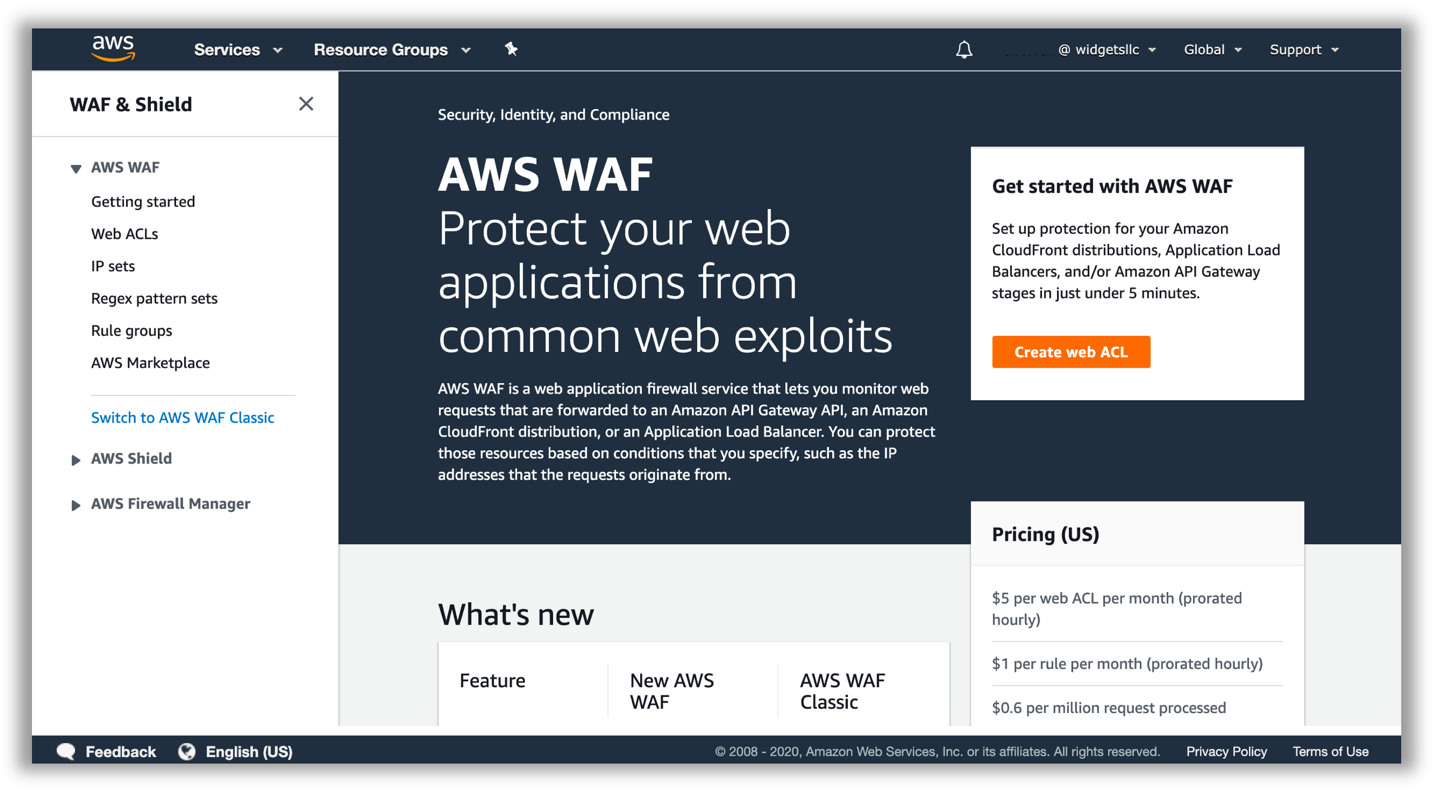
1. Identify the WAF ACL for your site
2. AWS WAF Rule design and considerations
3. Console Walkthrough - Creating a Rule
4. WAF Rule Creation and Solutions

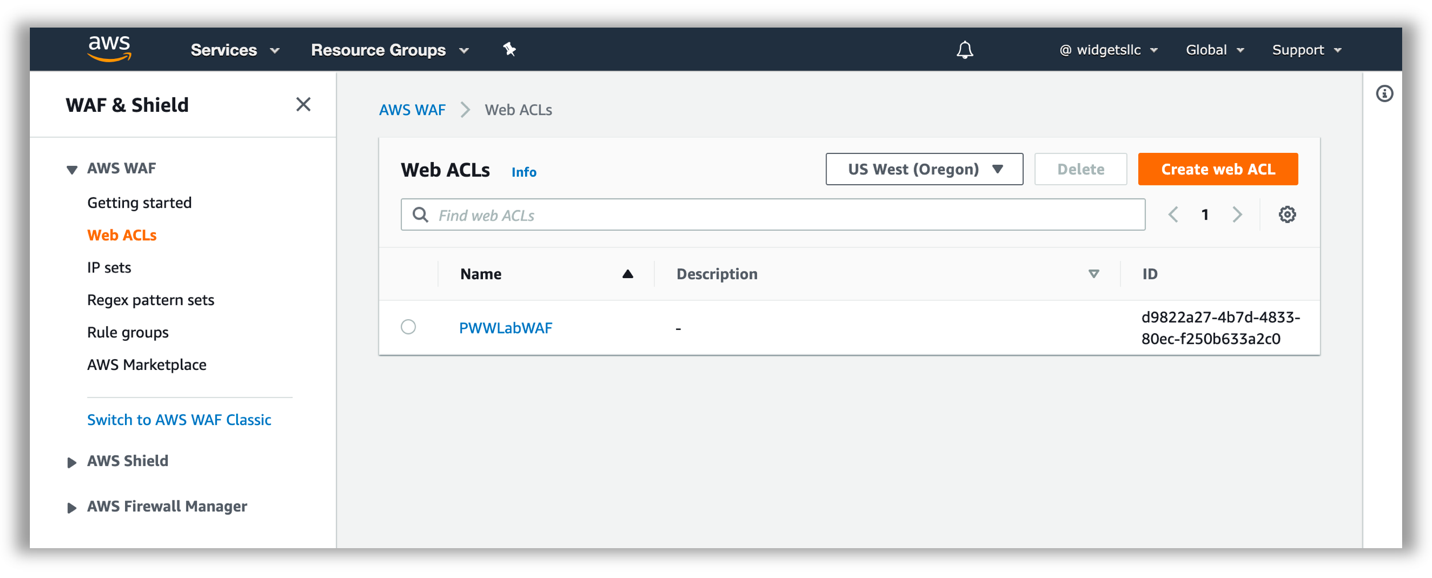
**Attention**

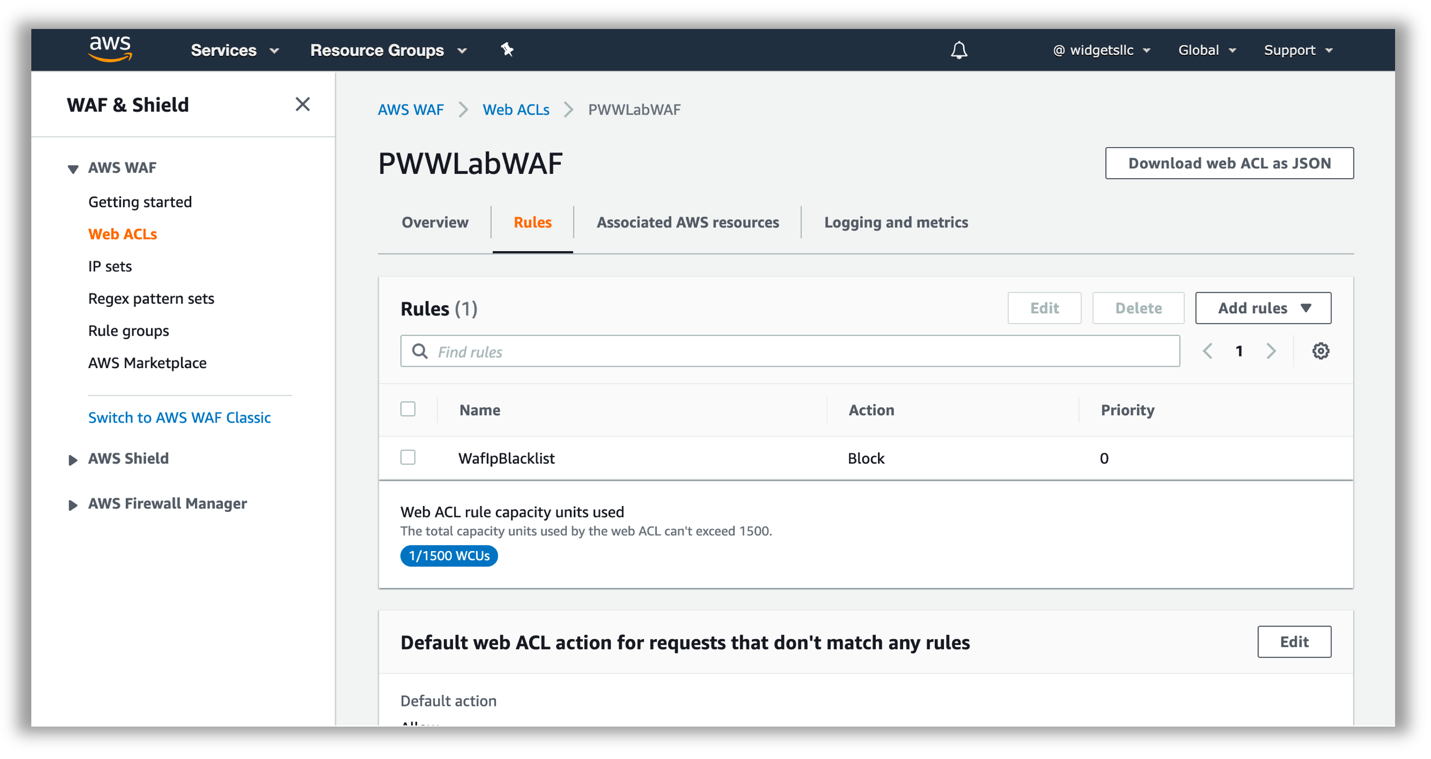
Please ensure you are **using the improved AWS WAF console and API experience** for this workshop.

## **Identify the WAF ACL for your Site**

1. If needed, go to <https://console.aws.amazon.com/console/home>. You will be redirected to the AWS Management Console dashboard on successful login:  Make sure you select the appropriate AWS Region when working in the AWS Management Console (top right corner, on the menu bar).
2. From the Management Console dashboard, navigate to the AWS WAF & Shield service console. You can do that several ways:
   * Type “waf” in the AWS services panel search box and select the resulting option
   * Expand the Services drop down menu (top left on the menu bar) and choose WAF & Shield
   * Expand the All services area of the AWS services panel and choose WAF & Shield Once selected, you will be redirected to the AWS WAF & AWS Shield service console. You may see an initial landing page at first. Choose Go to AWS WAF:

 3. In the side bar menu on the left, pick the Web ACLs option under the AWS WAF heading.

 4. Click on the WAF Web ACL Name to select the existing Web ACL. Once the detail pane is loaded on the left of your screen, you will see three tabs: Requests, Rules, and Logging. Toggle to Rules:

 Validate that you are able to see a pre-existing rule, configured to block requests, and that your Web ACL is associated with an Application load balancer resource. You can drill down further into the properties of the existing rule, by selecting the rule name and clicking **Edit**. This rule references IP sets for the loopback/localhost IP addresses (127.0.0.0/8, ::1/128).

**Viewing and Logging Requests**

In the ***Overview*** tab for your Web ACL, you can view a [*sample of the requests*](https://docs.aws.amazon.com/waf/latest/developerguide/web-acl-testing.html#web-acl-testing-view-sample) that have been inspected by the WAF. For each sampled request, you can view detailed data about the request, such as the originating IP address and the headers included in the request. You also can view which rule the request matched, and whether the rule is configured to allow or block requests. You can refer to the sampled requests throughout this exercise to monitor activity and look for suspicious activity. Although not used in this workshop, in the ***Logging and metrics*** tab, [*you can enable full logging*](https://docs.aws.amazon.com/waf/latest/developerguide/logging.html) to get detailed information about traffic that is analyzed by your web ACL.

## **AWS WAF Rule Design and Considerations**

### **Basics**

You use AWS WAF to control how an Amazon CloudFront distribution, an Amazon API Gateway API, or an Application Load Balancer responds to web requests.

**Web ACLs** – You use a web access control list (ACL) to protect a set of AWS resources. You create a web ACL and define its protection strategy by adding rules. Rules define criteria for inspecting web requests and specify how to handle requests that match the criteria. You set a default action for the web ACL that indicates whether to block or allow through those requests that pass the rules inspections.

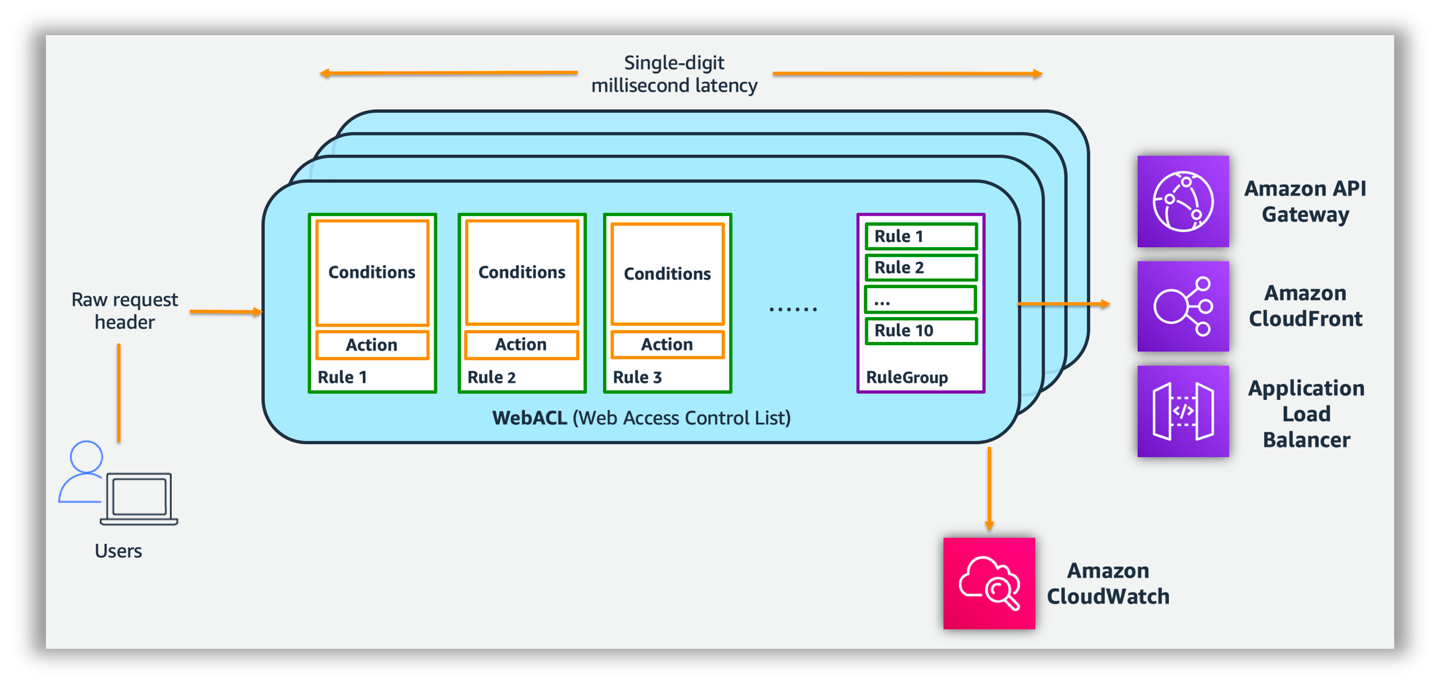
**Rules** – Each rule contains a statement that defines the inspection criteria, and an action to take if a web request meets the criteria. When a web request meets the criteria, that's a match. You can use rules to block matching requests or to allow matching requests through. You can also use rules just to count matching requests.

**Rules groups** – You can use rules individually or in reusable rule groups. AWS Managed Rules and AWS Marketplace sellers provide managed rule groups for your use. You can also define your own rule groups.

Rule statements are the part of a rule that tells AWS WAF how to inspect a web request. When AWS WAF finds the inspection criteria in a web request, we say that the web request matches the statement. Every rule statement specifies what to look for and how, according to the statement type.

Every rule in AWS WAF has a single top-level rule statement, which can contain other statements. Rule statements can be very simple. For example, you could have a statement that provides just a set of originating countries to check your web requests for. Rule statements can also be very complex. For example, you could have a statement that combines many other statements with logical AND, OR, and NOT statements.

#### How WAF Works



After you create your web ACL, you can associate it with one or more AWS resources. The resource types that you can protect using AWS WAF web ACLs are Amazon CloudFront distributions, Amazon API Gateway APIs, and Application Load Balancers.

**Note About This Section**

**In order to illustrate the process of creating WAF rules, we will walk through the creation of the first rule in your WAF ACL.** The complete list of threats and solutions is available in the WAF Rule Creation and Solutions section.

### **Rule Design Considerations:**

Use the following guiding questions when planning WAF rules:

1. What is the intended purpose of the rule?
2. What HTTP request components apply to the purpose of the rule?
3. Do you already have rules or rule groups targeting those request components that you can expand? Is that desirable?
4. How can you define the purpose of the rule in a Boolean logic expression?
5. Will the rule require nested statements under logical AND or OR rule statements?
6. Are any transformations relevant to my input content type?

#### Sample Rule purpose:

* **Detect SQL Injection in query string, use ‘block’ action in Web ACL**

#### HTTP request components:

* **Request Method** – form input typically gets submitted using a GET HTTP request method
* **Query String** – the SQL injection attempt is located in the query string

#### Define the purpose of the rule using Boolean logic:

* If **Query String contains suspected SQL Injection** then **block**

#### Sample Rule - Statement to implement:

* **Contains SQL injection attacks Match type** targeting the request **Query string**

#### Relevant transformations:

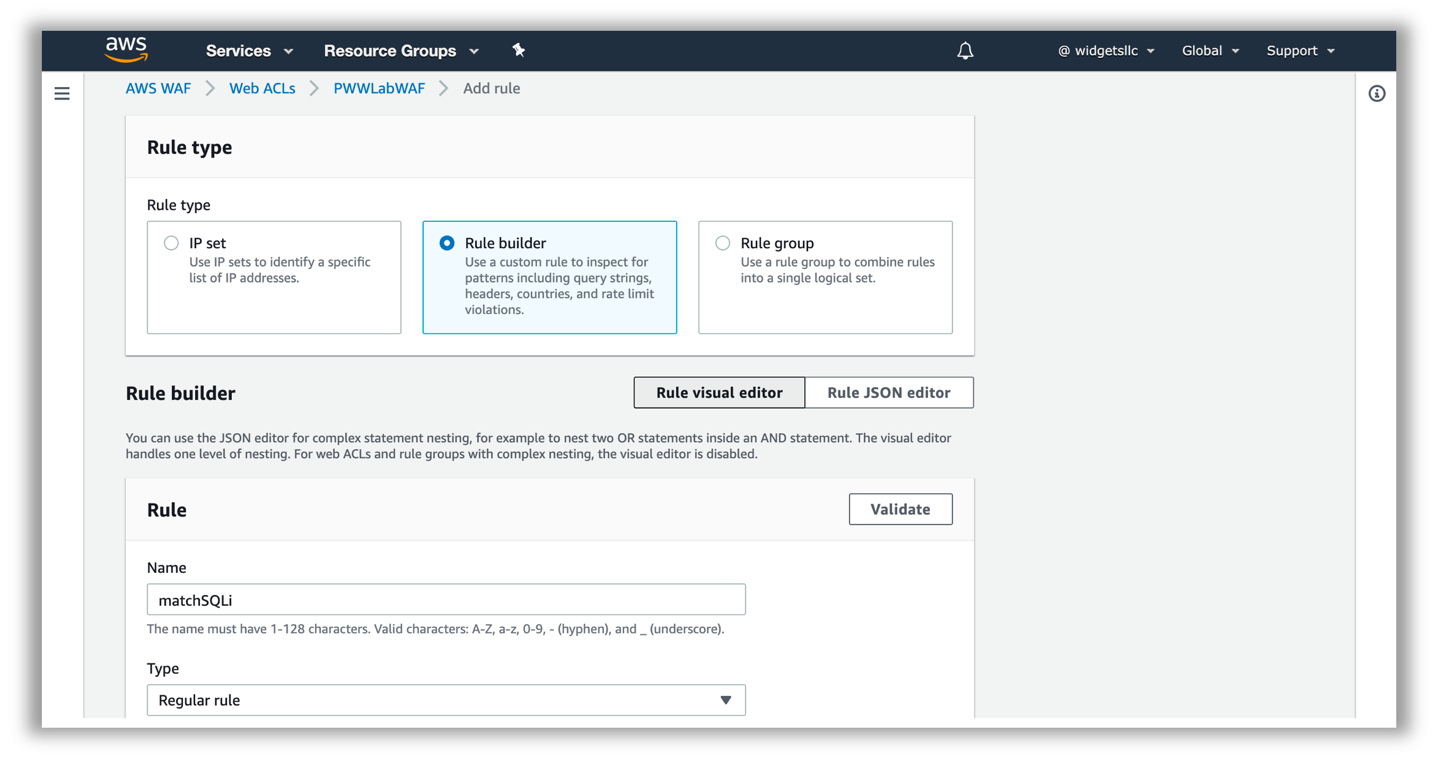
* **Contains SQL injection attacks Match type** query string is URL encoded, so we will apply the **URL\_DECODE** transformation.

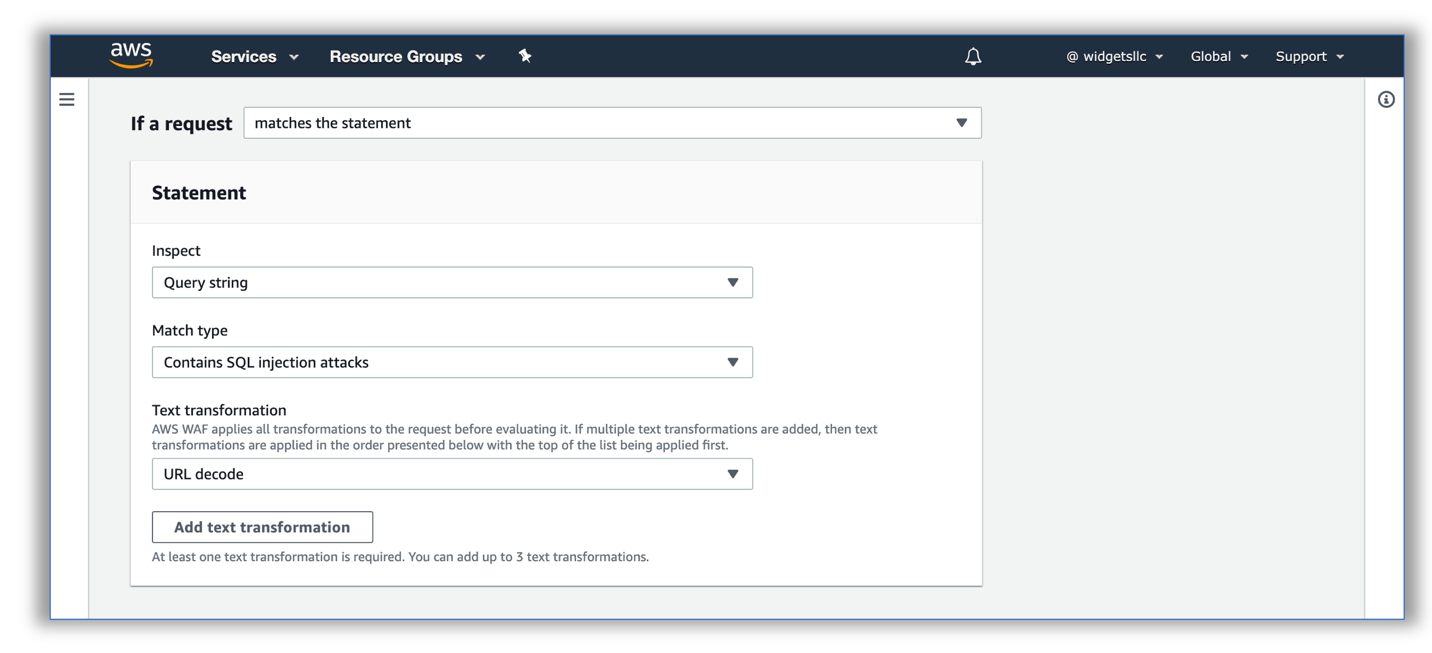
#### Rules to implement:

* Rule with 1 predicate Contains SQL injection attacks Match type

## **Console Walkthrough - Creating a Rule**

1. In the AWS WAF console, create a SQL injection rule by clicking the Web ACL, selecting the **Rules** tab,  **Add rules**, **Add my own rules and rule groups**
2. Click on **Rule builder**, provide **matchSQLi** for the **Name** and keep **Regular rule** for **Type**:

 3. For **If a request** select **matches the statement**. Under **Statement**, for **Inspect** select **Query string**, for **Match type** select **Contains SQL injection attacks**, for **Text transformation** select **URL decode** and for **Action** select **Block**.

 4. Click on **Add Rule** and then click **Save**

1. View the **matchSQLi** rule in the **Rule visual editor** to confirm it is correct.

**Additional Resources**

For a more comprehensive discussion of common vulnerabilities for web applications, as well as how to mitigate them using AWS WAF, and other AWS services, please refer to the [Use AWS WAF to Mitigate OWASP’s Top 10 Web Application Vulnerabilities whitepaper](https://d0.awsstatic.com/whitepapers/Security/aws-waf-owasp.pdf).

## **WAF Rule Creation and Solutions**

In this phase, we will have a set of 3 exercises walking you through the process of building a basic mitigation rule set for common vulnerabilities. We will build these rules from scratch, so you can gain familiarity with the AWS WAF programming model and you can then write rules specific to your applications.

**Note About Exercise Solutions**

For the exercises below, you will find the high level description and solution configuration for your web ACL. You can test your ACL ruleset at any time using the Red Team Host. For AWS sponsored event, you can also view test results on the WAF Lab Dashboard.

### **1. SQL Injection & Cross Site Scripting Mitigation**

Use the SQL injection, cross-site scripting, as well as string and regex matching to build rules that mitigate injection attacks and cross site scripting attacks.

Consider the following:

* How does your web application accept end-user input (whether directly or indirectly). Which HTTP request components does that input get inserted into?
* What kind of content encoding considerations do you need to factor in for the input format?
* What considerations do you need to account for in regards to false positives? For example, does your application legitimately need to accept SQL statements as input?

How do the requirements derived from the above questions affect your solution?

SQL Injection Solution

1. Update the **matchSQLi** rule with 2 additional statements
   1. Select the **matchSQLi** rule and click **Edit** (You should have created this rule in [*the walk through above*](https://protecting-workloads.awssecworkshops.com/workshop/perimeter-layer/remediate/#console-walkthrough-creating-a-rule))
   2. Change **If a request** to **matches at least one of the statements (OR)**
   3. Click **Add another statement**: body, contains sql injection attacks, html entity decode and URL decode
   4. Click **Add another statement**: header, cookie (type manually), contains sql injection attacks, url decode
2. View the existing matchSQLi rule to confirm additional conditions
3. Re-run the WAF test script (runscanner) from your red team host to confirm requests are blocked

Cross Site Scripting Solution

1. Create a new rule named **matchXSS** and for **If a request** choose **matches at least one of the statements (OR)**. Add statements:
   1. all query parameters, contains xss injection attacks, url decode
   2. body, contains xss injection attacks, html entity decode and url decode
   3. header, cookie (*type manually*), contains xss injection attacks, url decode
   4. Click on **Add Rule** and then click **Save**
2. **Edit** the rule, click the **Rule JSON editor** and note the structure and syntax of the rule logic.
3. Add an exception statement for the XSS rule to allow access to */reportBuilder/Editor.aspx*. *Note that we are using the JSON editor here due to the nested logic required for the exception.*
   1. After reviewing it, clear the existing editor content for the matchXSS rule and paste the following JSON

Nested Statement with XSS Exception Solution

{

"Name": "matchXSS",

"Priority": 2,

"Action": {

"Block": {}

},

"VisibilityConfig": {

"SampledRequestsEnabled": true,

"CloudWatchMetricsEnabled": true,

"MetricName": "matchXSS"

},

"Statement": {

"AndStatement": {

"Statements": [{

"NotStatement": {

"Statement": {

"ByteMatchStatement": {

"SearchString": "/reportBuilder/Editor.aspx",

"FieldToMatch": {

"UriPath": {}

},

"TextTransformations": [{

"Priority": 0,

"Type": "NONE"

}],

"PositionalConstraint": "STARTS\_WITH"

}

}

}

},

{

"OrStatement": {

"Statements": [{

"XssMatchStatement": {

"FieldToMatch": {

"QueryString": {}

},

"TextTransformations": [{

"Priority": 0,

"Type": "URL\_DECODE"

}]

}

},

{

"XssMatchStatement": {

"FieldToMatch": {

"Body": {}

},

"TextTransformations": [{

"Priority": 0,

"Type": "HTML\_ENTITY\_DECODE"

},

{

"Priority": 1,

"Type": "URL\_DECODE"

}

]

}

},

{

"XssMatchStatement": {

"FieldToMatch": {

"SingleHeader": {

"Name": "cookie"

}

},

"TextTransformations": [{

"Priority": 0,

"Type": "URL\_DECODE"

}]

}

}

]

}

}

]

}

}

}

1. Click **Save rule**
2. Re-run the WAF test script (runscanner) from your red team host to confirm requests are blocked

### **2. Mitigate File Inclusion & Path Traversal**

Use the string and regex matching to build rules that block specific patterns indicative of unwanted path traversal or file inclusion.

Consider the following:

* Can end users browse the directory structure of your web folders? Do you have directory indexes enabled?
* Is your application (or any dependency components) use input parameters in filesystem or remote URL references?
* Do you adequately lock down access so input paths cannot be manipulated?
* What considerations do you need to account for in regards to false positives (directory traversal signature patterns)?

Build rules that ensure the relevant HTTP request components used for input into paths do not contain known path traversal patterns.

Solution

1. Create a new rule named **matchTraversal** and for **If a request** choose **matches at least one of the statements (OR)**. Add statements:
   1. uri\_path, starts with string, /include, url\_decode
   2. all\_query\_parameters, contains string, ../, url\_decode
   3. Click on **Add Rule** and then click **Save**
2. Re-run the WAF test script (runscanner) from your red team host to confirm requests are blocked

### **3. Enforce Request Hygiene**

Use the string and regex matching, size constraints and IP address matching to build rules that block non-conforming or low value HTTP requests.

Consider the following:

• Are there limits to the size of the various HTTP request components relevant to your web application? For example, does your application ever use URIs that are longer than 100 characters in size?

• Are there specific HTTP request components without which your application cannot operate effectively (e.g. CSRF token header, authorization header, referrer header)?

Build rules that ensure the requests your application ends up processing are valid, conforming and valuable.

Solution

1. In the left pane, choose **Regex pattern sets**, **Create regex pattern set**
   1. **Regex pattern set name** *csrf*, **Regular expressions** *^[0-9a-f]{40}$*
      1. The Regex pattern above is a simple example that matches the string length (40) and characters (0-9 or a-f). Copy the Regex pattern set ID into a scratch file to refer to it later.
      2. Note your AWS account Id (*in CloudFormation Stack Outputs*) and region and add them to the scratch file.
2. Create a new rule and choose **Rule JSON editor**
   1. Delete any existing text and paste the following JSON below. Review the statements in the JSON
   2. **Update the region, AWS account Id and Regex pattern ID** with the one created in the previous stepd

{

"Name": "matchCSRF",

"Priority": 3,

"Action": {

"Block": {}

},

"VisibilityConfig": {

"SampledRequestsEnabled": true,

"CloudWatchMetricsEnabled": true,

"MetricName": "matchCSRF"

},

"Statement": {

"AndStatement": {

"Statements": [{

"NotStatement": {

"Statement": {

"RegexPatternSetReferenceStatement": {

"ARN": "arn:aws:wafv2:YOUR\_REGION:ACCOUNT\_ID:regional/regexpatternset/csrf/YOUR\_REGEX\_PATTERN\_ID",

"FieldToMatch": {

"SingleHeader": {

"Name": "x-csrf-token"

}

},

"TextTransformations": [{

"Priority": 0,

"Type": "URL\_DECODE"

}]

}

}

}

},

{

"OrStatement": {

"Statements": [{

"ByteMatchStatement": {

"SearchString": "/form.php",

"FieldToMatch": {

"UriPath": {}

},

"TextTransformations": [{

"Priority": 0,

"Type": "NONE"

}],

"PositionalConstraint": "STARTS\_WITH"

}

},

{

"ByteMatchStatement": {

"SearchString": "/form.php",

"FieldToMatch": {

"UriPath": {}

},

"TextTransformations": [{

"Priority": 0,

"Type": "NONE"

}],

"PositionalConstraint": "EXACTLY"

}

}

]

}

}

]

}

}

}

1. Click **Save rule**
2. Re-run the WAF test script (runscanner) from your red team host to confirm requests are blocked

Mitigating Common Web Application Attack Vectors Using AWS WAF - Verify Phase

In the previous remediation phase, you implemented an AWS WAF ruleset to protect your site from common attack vectors. You are now going to reassess the posture of the site to confirm the rules are performing as intended and blocking the simulated malicious requests.

1. Confirm malicious requests are blocked by WAF policy
2. Implement WAF monitoring dashboard using Amazon CloudWatch (Optional)

Confirm malicious requests are blocked by WAF policy

If needed, start a Session Manager connection to your Red Team Host, the scanner script can be invoked by typing the following command:

runscanner

Confirm that all of the tests in the script pass. If requests (other than canary) are not being blocked, go back to the remediate phase and confirm your conditions and rules are properly configured.