**Task 1. Create the VPC network**

* In Cloud Shell, enter the following command to create a VPC network:

gcloud compute networks create Network Name --subnet-mode custom

Copied!

content\_copy

Created

NAME SUBNET\_MODE BGP\_ROUTING\_MODE

Network Name CUSTOM REGIONAL

Create a subnet

* In Cloud Shell, enter the following command to create a subnet:

gcloud compute networks subnets create Subnet Name \

--network Network Name --range 10.0.0.0/24 --region Region

Copied!

content\_copy

Created

NAME REGION NETWORK RANGE

{{{my\_primary\_project.startup\_script.subnet\_name| Subnet Name}}} {{{my\_primary\_project.startup\_script.region\_1| Region}}} {{{my\_primary\_project.startup\_script.network\_name| Network Name}}} 10.0.0.0/24

Create VPC firewall rules

After creating the VPC and subnet, set up a few firewall rules.

* The first firewall rule named allow-js-site allows all IPs to access the external IP of the test application's website on port 3000.
* The second firewall rule named allow-health-check allows health-checks from source IP of the load balancers.

1. In Cloud Shell, enter the following command to create a firewall rule to allow all IPs to access the application:

gcloud compute firewall-rules create Firewall Name --allow tcp:3000 --network Network Name

Copied!

content\_copy

Output:

Creating firewall...done.

NAME NETWORK DIRECTION PRIORITY ALLOW DENY DISABLED

Firewall Name Network Name INGRESS 1000 tcp:3000 False

1. In Cloud Shell, enter the following command to create firewall rule to allow health-checks from the Google health-check ranges:

gcloud compute firewall-rules create Firewall Name1 \

--network=Network Name \

--action=allow \

--direction=ingress \

--source-ranges=130.211.0.0/22,35.191.0.0/16 \

--target-tags=allow-healthcheck \

--rules=tcp

Copied!

content\_copy

Output:

Creating firewall...done.

NAME NETWORK DIRECTION PRIORITY ALLOW DENY DISABLED

Firewall\_Name1 Network Name INGRESS 1000 tcp False

Click **Check my progress** to verify the objective.

Create the VPC network

Check my progress

**Task 2. Set up the test application**

Create the test application, in this case, the OWASP Juice Shop web server. When you create the compute instance, you use a container image to ensure the server has the appropriate services. You deploy this server in the Zone and has a network tag that allows health checks.

Create the OWASP Juice Shop application

* Use the open source well-known OWASP Juice Shop application to serve as the vulnerable application. You can also use this application to do OWASP security challenges through the [OWASP website](https://owasp.org/www-project-juice-shop/).

gcloud compute instances create-with-container vm\_instance --container-image bkimminich/juice-shop \

--network Network Name \

--subnet Subnet Name \

--private-network-ip=10.0.0.3 \

--machine-type n1-standard-2 \

--zone Zone \

--tags allow-healthcheck

Copied!

content\_copy

Output:

NAME ZONE MACHINE\_TYPE PREEMPTIBLE

{{{my\_primary\_project.startup\_script.vm\_instance| vm\_instance}}} {{{my\_primary\_project.startup\_script.zone| Zone}}} n1-standard-2

INTERNAL\_IP EXTERNAL\_IP STATUS

10.0.0.3 <public IP> RUNNING

Click **Check my progress** to verify the objective.

Set up the test application

Check my progress

Set up the Cloud load balancer component: instance group

1. In Cloud Shell, enter the following command to create the unmanaged instance group:

gcloud compute instance-groups unmanaged create Instance Group \

--zone=Zone

Copied!

content\_copy

Output:

NAME LOCATION SCOPE NETWORK MANAGED INSTANCES

Instance Group Zone zone 0

1. Add the Juice Shop Google Compute Engine (GCE) instance to the unmanaged instance group:

gcloud compute instance-groups unmanaged add-instances Instance Group \

--zone=Zone \

--instances=VM Instance

Copied!

content\_copy

Output:

Updated [https://www.googleapis.com/compute/v1/projects//zones/Zone/instanceGroups/Instance\_Group].

1. Set the named port to that of the Juice Shop application:

gcloud compute instance-groups unmanaged set-named-ports \

Instance Group \

--named-ports=http:3000 \

--zone=Zone

Copied!

content\_copy

Output:

Updated [https://www.googleapis.com/compute/v1/projects//zones/Zone/instanceGroups/Instance Group].

Click **Check my progress** to verify the objective.

Set up the Cloud load balancer component- instance group

Check my progress

Set up the Cloud load balancer component: health check

Now that you've created the unmanaged instance group, create a health check, backend service, URL map, target proxy, and forwarding rule.

* In Cloud Shell, enter the following command to create the health-check for the Juice Shop service port:

gcloud compute health-checks create tcp tcp-port-3000 \

--port 3000

Copied!

content\_copy

Output:

Created

NAME PROTOCOL

tcp-port-3000 TCP

Set up the Cloud load balancer component: backend service

1. In Cloud Shell, enter the following command to create the backend service parameters:

gcloud compute backend-services create juice-shop-backend \

--protocol HTTP \

--port-name http \

--health-checks tcp-port-3000 \

--enable-logging \

--global

Copied!

content\_copy

Output:

NAME BACKENDS PROTOCOL

juice-shop-backend HTTP

1. Add the Juice Shop instance group to the backend service:

gcloud compute backend-services add-backend juice-shop-backend \

--instance-group=Instance Group \

--instance-group-zone=Zone \

--global

Copied!

content\_copy

Output:

Updated [https://www.googleapis.com/compute/v1/projects/cythom-host1/global/backendServices/juice-shop-backend].

Set up the Cloud load balancer component: URL map

* In Cloud Shell, enter the following command to create the URL map to send incoming requests to the backend:

gcloud compute url-maps create juice-shop-loadbalancer \

--default-service juice-shop-backend

Copied!

content\_copy

Output:

NAME DEFAULT\_SERVICE

juice-shop-loadbalancer backendServices/juice-shop-backend

Set up the Cloud load balancer component: target proxy

* In Cloud Shell, enter the following command to create the Target Proxy to route incoming requests the URL map:

gcloud compute target-http-proxies create juice-shop-proxy \

--url-map juice-shop-loadbalancer

Copied!

content\_copy

Output:

NAME URL\_MAP

juice-shop-proxy juice-shop-loadbalancer

Set up the Cloud load balancer component: forwarding rule

* In Cloud Shell, enter the following command to create the forwarding rule for the Load Balancer:

gcloud compute forwarding-rules create juice-shop-rule \

--global \

--target-http-proxy=juice-shop-proxy \

--ports=80

Copied!

content\_copy

Output:

Created [https://www.googleapis.com/compute/v1/projects/cythom-host1/global/forwardingRules/juice-shop-rule].

Verify the Juice Shop service is online

1. From Cloud Shell:

PUBLIC\_SVC\_IP="$(gcloud compute forwarding-rules describe juice-shop-rule --global --format="value(IPAddress)")"

echo $PUBLIC\_SVC\_IP

Copied!

content\_copy

Output:

<public VIP of service>

Wait a few minutes before continuing on, else you may retrieve a HTTP/1.1 404 Not Found response.

1. From Cloud Shell:

curl -Ii http://$PUBLIC\_SVC\_IP

Copied!

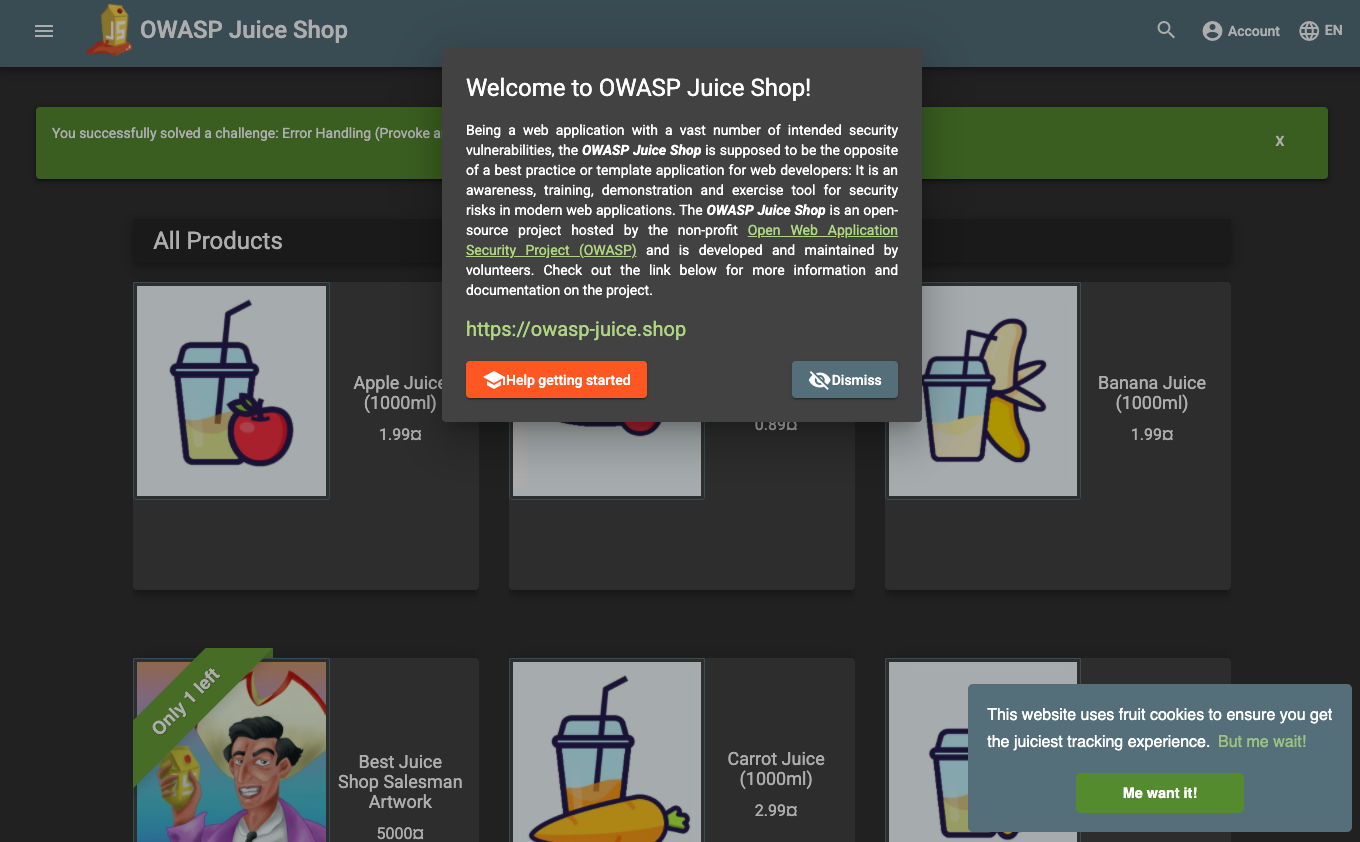
content\_copy

Output:

HTTP/1.1 200 OK

<...>

You can also go to the browser to view the Juice Shop!



You're now ready to explore the Juice Shop vulnerabilities and protect against them with Cloud Armor WAF rule sets.

Click **Check my progress** to verify the objective.

Set up the Cloud load balancer component- health check

Check my progress

**Task 3. Demonstrate known vulnerabilities**

In this lab, you demonstrate the states before and after Cloud Armor WAF rules are propagated in condensed steps.

Observe an LFI vulnerability: path traversal

Local File Inclusion is the process of observing files present on the server by exploiting lack of input validation in the request to potentially expose sensitive data. The following shows a path traversal is possible. In your browser or with curl, observe an existing path served by the application.

1. From Cloud Shell:

curl -Ii http://$PUBLIC\_SVC\_IP/ftp

Copied!

content\_copy

Output:

HTTP/1.1 200 OK

<...>

Observe that path traversal works too.

1. From Cloud Shell:

curl -Ii http://$PUBLIC\_SVC\_IP/ftp/../

Copied!

content\_copy

Output:

HTTP/1.1 200 OK

<...>

Observe an RCE vulnerability

Remote Code Execution includes various UNIX and Windows command injection scenarios allowing attackers to execute OS commands usually restricted to privileged users. The following shows a simple ls command execution passed in.

* From Cloud Shell:

curl -Ii http://$PUBLIC\_SVC\_IP/ftp?doc=/bin/ls

Copied!

content\_copy

Output:

HTTP/1.1 200 OK

<...>

Remove the curl flags to observe the full output.

Observe a well-known scanner's access

Both commercial and open source scan applications for various purposes, including to find vulnerabilities. These tools use well-known User-Agent and other Headers. Observe curl works with a well-known User-Agent Header.

* In Cloud Shell:

curl -Ii http://$PUBLIC\_SVC\_IP -H "User-Agent: blackwidow"

Copied!

content\_copy

Output:

HTTP/1.1 200 OK

<...>

Observe a protocol attack: HTTP splitting

Some web applications use input from the user to generate the headers in the responses. If the application doesn't properly filter the input, an attacker can potentially poison the input parameter with the sequence %0d%0a (the CRLF sequence that is used to separate different lines).

The response could then be interpreted as two responses by anything that happens to parse it, like an intermediary proxy server, potentially serving false content in subsequent requests. Insert the sequence %0d%0a into the input parameter, which can lead to serving a misleading page.

* From Cloud Shell:

curl -Ii "http://$PUBLIC\_SVC\_IP/index.html?foo=advanced%0d%0aContent-Length:%200%0d%0a%0d%0aHTTP/1.1%20200%20OK%0d%0aContent-Type:%20text/html%0d%0aContent-Length:%2035%0d%0a%0d%0a<html>Sorry,%20System%20Down</html>"

Copied!

content\_copy

Output:

HTTP/1.1 200 OK

<...>

Observe session fixation

* In Cloud Shell:

curl -Ii http://$PUBLIC\_SVC\_IP -H session\_id=X

Copied!

content\_copy

Output:

HTTP/1.1 200 OK

<...>

**Task 4. Define Cloud Armor WAF rules**

1. List the preconfigured WAF rules, using the following command in Cloud Shell:

gcloud compute security-policies list-preconfigured-expression-sets

Copied!

content\_copy

EXPRESSION\_SET

Sqli-canary

RULE\_ID

owasp-crs-v030001-id942110-sqli

owasp-crs-v030001-id942120-sqli

<...>

1. Create the Cloud Armor security policy using the following command in Cloud Shell:

gcloud compute security-policies create Policy Name \

--description "Block with OWASP ModSecurity CRS"

Copied!

content\_copy

1. In Cloud Shell, update the security policy default rule.

**Note:**The default rule priority has a numerical value of 2147483647.

gcloud compute security-policies rules update 2147483647 \

--security-policy Policy Name \

--action "deny-403"

Copied!

content\_copy

1. Since the default rule is configured with action deny, you must allow access from your IP. Please find your public IP (curl, ipmonkey, whatismyip, etc):

MY\_IP=$(curl ifconfig.me)

Copied!

content\_copy

1. Add the first rule to allow access from your IP (INSERT YOUR IP BELOW):

gcloud compute security-policies rules create 10000 \

--security-policy Policy Name \

--description "allow traffic from my IP" \

--src-ip-ranges "$MY\_IP/32" \

--action "allow"

Copied!

content\_copy

1. In Cloud Shell, update the security policy to block LFI attacks.

Apply the OWASP ModSecurity Core Rule Set that prevents path traversal for local file inclusions.

gcloud compute security-policies rules create 9000 \

--security-policy Policy Name \

--description "block local file inclusion" \

--expression "evaluatePreconfiguredExpr('lfi-stable')" \

--action deny-403

Copied!

content\_copy

1. In Cloud Shell, update the security policy to block Remote Code Execution (rce).

Per the OWASP ModSecurity Core Rule Set, apply rules that look for rce, including command injection. Typical OS commands are detected and blocked.

gcloud compute security-policies rules create 9001 \

--security-policy Policy Name \

--description "block rce attacks" \

--expression "evaluatePreconfiguredExpr('rce-stable')" \

--action deny-403

Copied!

content\_copy

1. Update the security policy to block security scanners.

Apply the OWASP ModSecurity Core Rule Set to block well-known security scanners, scripting HTTP clients, and web crawlers.

gcloud compute security-policies rules create 9002 \

--security-policy Policy Name \

--description "block scanners" \

--expression "evaluatePreconfiguredExpr('scannerdetection-stable')" \

--action deny-403

Copied!

content\_copy

1. In Cloud Shell, update the security policy to block protocol attacks.

Per the OWASP ModSecurity Core Rule Set, apply rules that look for Carriage Return (CR) %0d and Linefeed (LF)%0a characters and other types of protocol attacks like HTTP Request Smuggling.

gcloud compute security-policies rules create 9003 \

--security-policy Policy Name \

--description "block protocol attacks" \

--expression "evaluatePreconfiguredExpr('protocolattack-stable')" \

--action deny-403

Copied!

content\_copy

1. Update the security policy to block session fixation.

Per the OWASP ModSecurity Core Rule Set, apply the following rules using Cloud Shell:

gcloud compute security-policies rules create 9004 \

--security-policy Policy Name \

--description "block session fixation attacks" \

--expression "evaluatePreconfiguredExpr('sessionfixation-stable')" \

--action deny-403

Copied!

content\_copy

1. Attach the security policy to the backend service:

gcloud compute backend-services update juice-shop-backend \

--security-policy Policy Name \

--global

Copied!

content\_copy

Rules may take some time to propagate (but not more than 10 mins).

1. Once sufficient time has passed, test the vulnerabilities previously demonstrated to confirm Cloud Armor WAF rule enforcement in the next step.

Click **Check my progress** to verify the objective.

Create the Cloud Armor security policy

Check my progress

Observe Cloud Armor protection with OWASP ModSecurity Core Rule Set

1. In Cloud Shell, confirm the LFI vulnerability is mitigated:

curl -Ii http://$PUBLIC\_SVC\_IP/?a=../

Copied!

content\_copy

Output:

HTTP/1.1 403 Forbidden

<...>

1. In Cloud Shell, confirm the RCE attack is mitigated:

curl -Ii http://$PUBLIC\_SVC\_IP/ftp?doc=/bin/ls

Copied!

content\_copy

Output:

HTTP/1.1 403 Forbidden

<..>

1. In Cloud Shell, confirm well-known scanner detection.

curl -Ii http://$PUBLIC\_SVC\_IP -H "User-Agent: blackwidow"

Copied!

content\_copy

Output:

HTTP/1.1 403 Forbidden

<..>

1. In Cloud Shell, confirm a protocol attack is mitigated.

Per the OWASP ModSecurity Core Rule Set ver.3.0.2, the protocol attack is mitigated by:

curl -Ii "http://$PUBLIC\_SVC\_IP/index.html?foo=advanced%0d%0aContent-Length:%200%0d%0a%0d%0aHTTP/1.1%20200%20OK%0d%0aContent-Type:%20text/html%0d%0aContent-Length:%2035%0d%0a%0d%0a<html>Sorry,%20System%20Down</html>"

Copied!

content\_copy

Output:

HTTP/1.1 403 Forbidden

<..>

1. In Cloud Shell, confirm session fixation attempts are blocked:

curl -Ii http://$PUBLIC\_SVC\_IP/?session\_id=a

Copied!

content\_copy

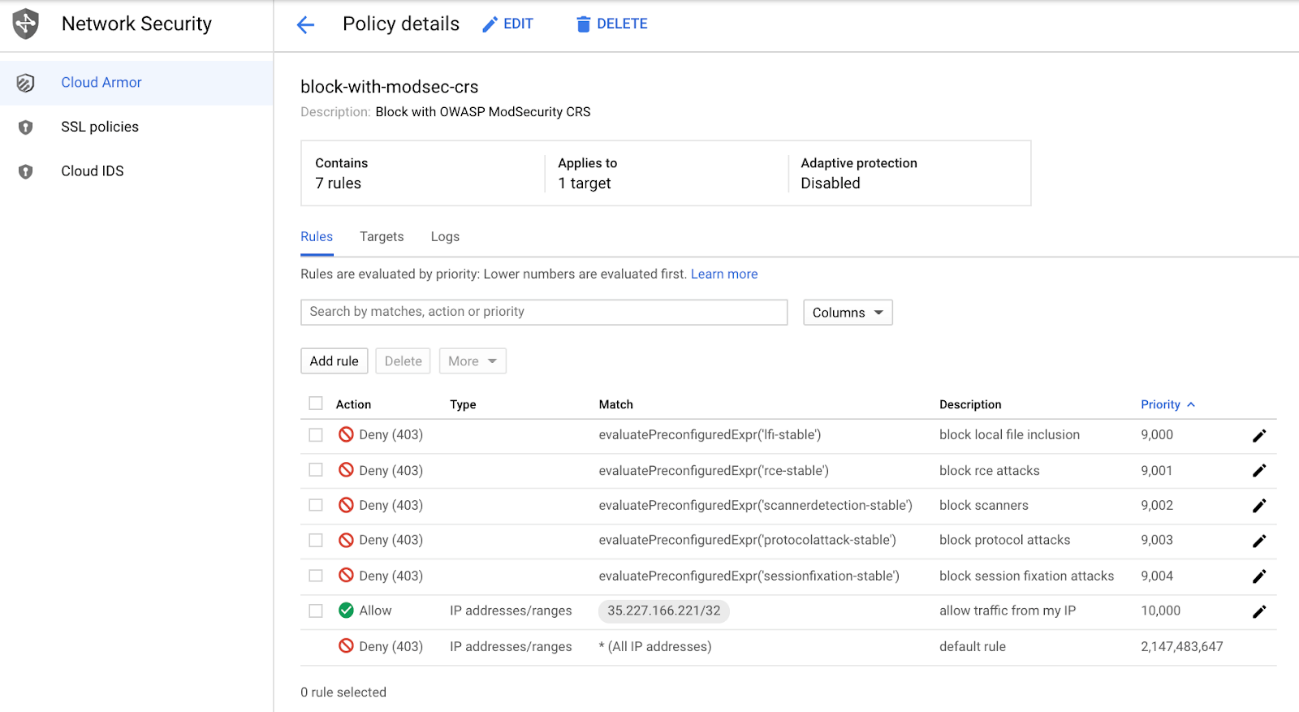
Output:

HTTP/1.1 403 Forbidden

<..>

**Task 5. Review Cloud Armor Security rules**

Now that you've created the security policy, look at what rules have been configured.



Rules are evaluated by priority: lower numbers are evaluated first and once triggered, processing does not continue for rules with higher priority values.

* Priority 9000 - Block LFI (local file inclusion)
* Priority 9001 - Block RCE (remote code execution/command injection)
* Priority 9002 - Block Scanners Detected
* Priority 9003 - Block Protocol Attacks like HTTP splitting and HTTP smuggling
* Priority 9004 - Block Session Fixation Attacks
* Priority 10000 - Allow your IP to access the Website
* Priority Default - Deny.

**Note:**Notice the "allow your IP" rule is configured with the highest priority number to allow access to the site, however blocks any attack.

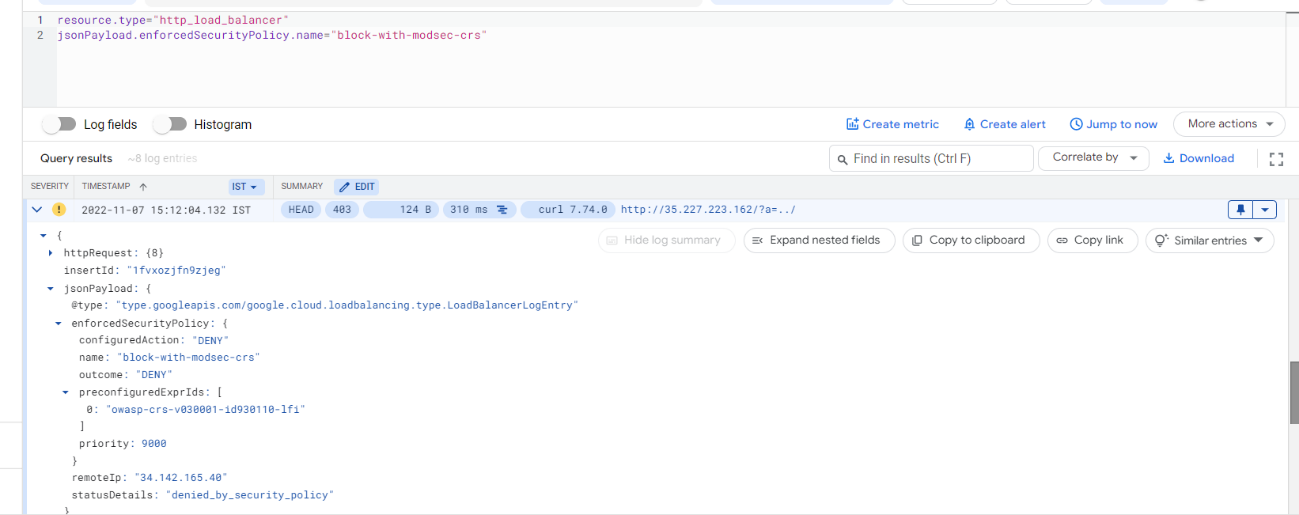
**Task 6. Observe Cloud Armor security policy logs**

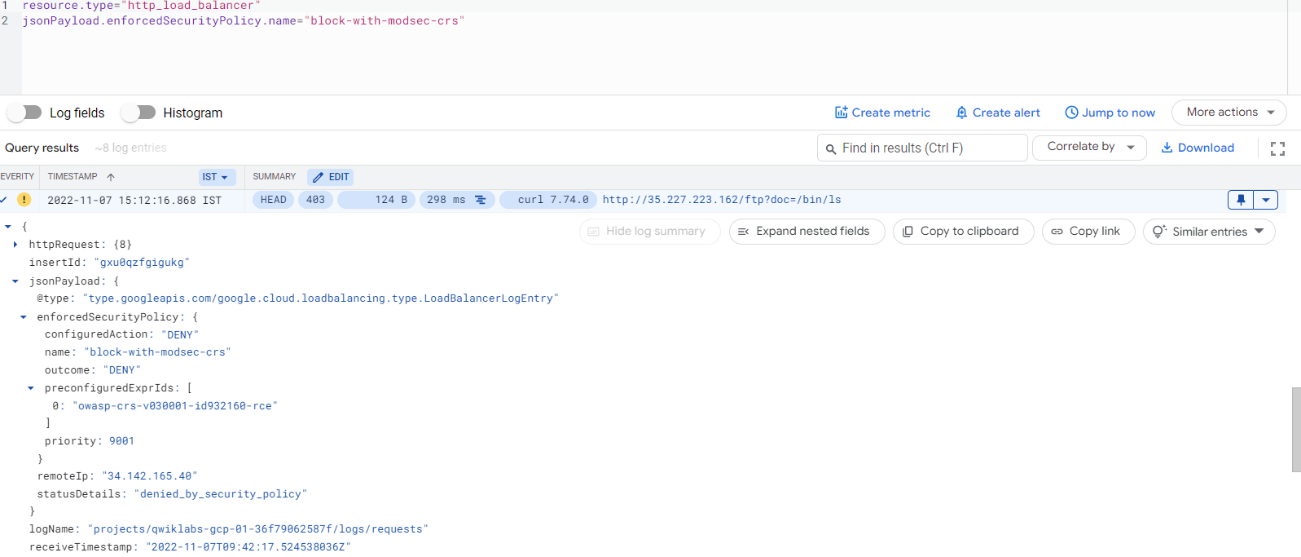
From the Cloud Armor console page, view details of the security policy and click the Logs tab followed by the View policy logs link to be directed to the Cloud Logging page. It automatically filters based on the security policy of interest, for example, resource.type:(http\_load\_balancer) AND jsonPayload.enforcedSecurityPolicy.name: Policy Name . Observe the 403 error response codes and expand the log details to observe the enforced security policy's name, matched field value, and further down the preconfigured expression IDs (or the signature id).

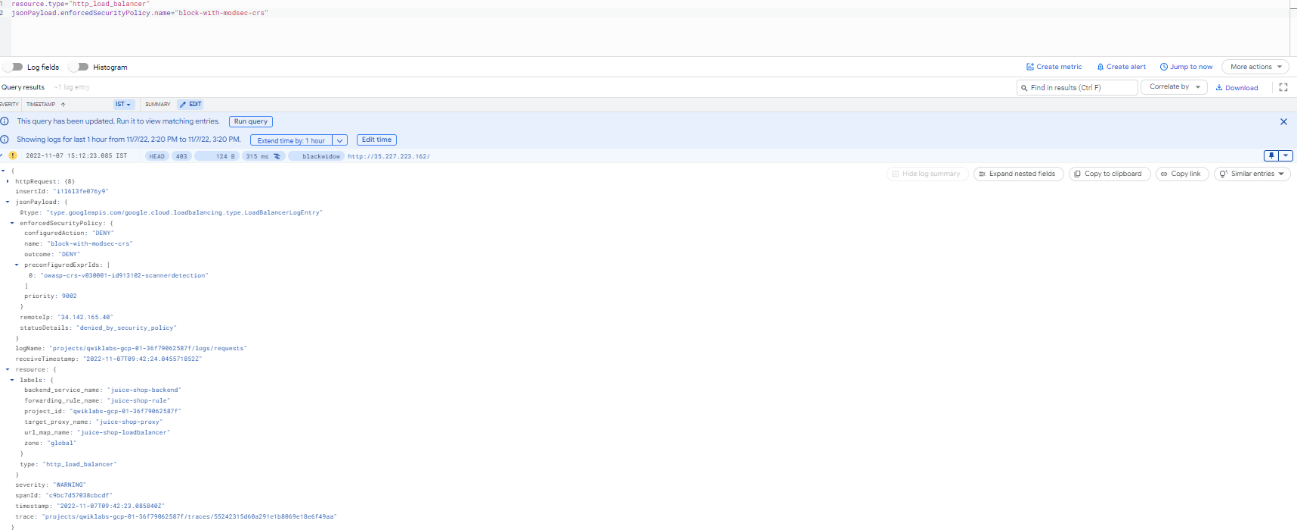
It automatically filters based on the security policy of interest, for example, resource.type:(http\_load\_balancer) AND jsonPayload.enforcedSecurityPolicy.name:( Policy Name ).

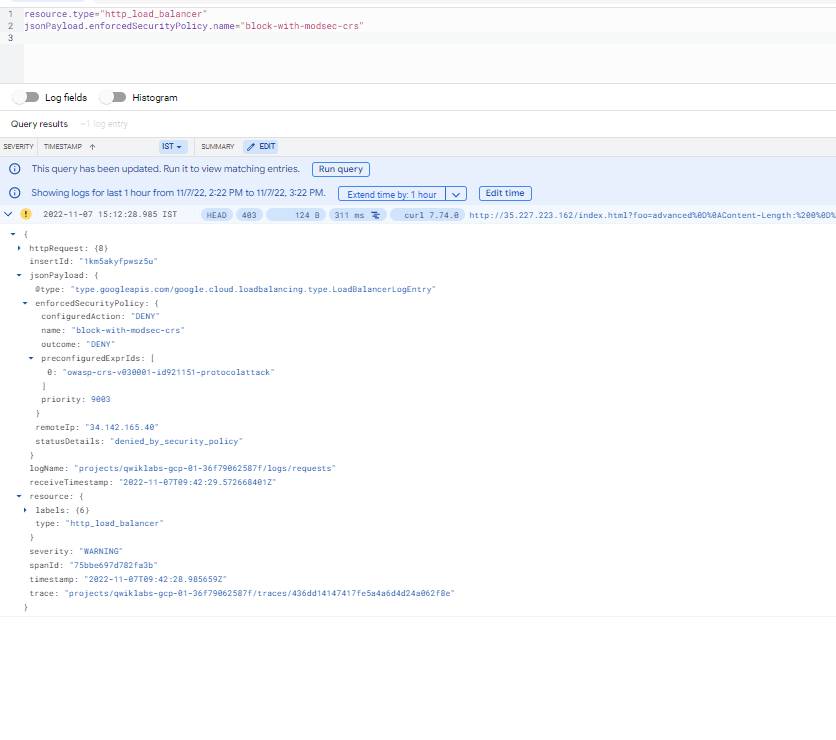
* Observe the 403 error response codes and expand the log details to observe the enforced security policy's name, matched field value, and further down the preconfigured expression IDs (or the signature id).

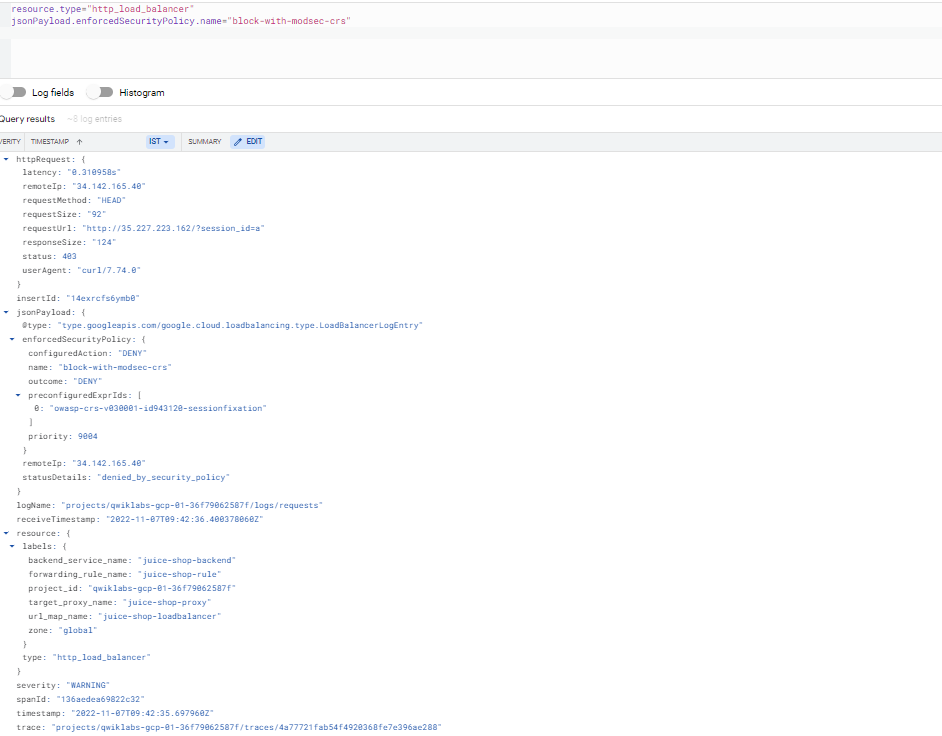
The following screenshots show examples of the logs for the enforced security policies configured in this lab.

**LFI log** 

**RCE log** 

**Scanner detection log** 

**Protocol attack log** 

**Session fixation log** 

**Congratulations!**

You've successfully mitigated some of the common vulnerabilities by using Google Cloud