**Lab 1 – Deploy Arcadia API Security-as-Code in Dev**

Scenario - A production instance of Arcadia Finance app is deployed with a WAF policy providing OWASP top10 protection.

Objective - The Arcadia Finance management team wants to migrate the Finance app to a microservices architecture. Your team has been tasked with developing an API Security strategy around F5’s Advanced WAF capabilities. The production app consists of three microservices for which your team has begun to develop an OpenAPI v3.0 spec file for the purpose of incorporating it into Arcadia’s security-first app deployment practice. This file is in your Ubuntu App server at /home/ubuntu/repo/arcadia/dev/arcadia-oas3-dev.json.  
  
Task - In this lab you will take on the role of a Security-focused DevOps engineer where you will leverage F5’s declarative App Services API (AS3) to deploy a test instance of Arcadia Finance while automatically attaching a consumable-as-code API security profile based on modern OpenAPI Spec practices and use OpenAPI standards to resolve any issues.

**Module1 – Review Arcadia Application**

**Task 1 – Review Web UI**

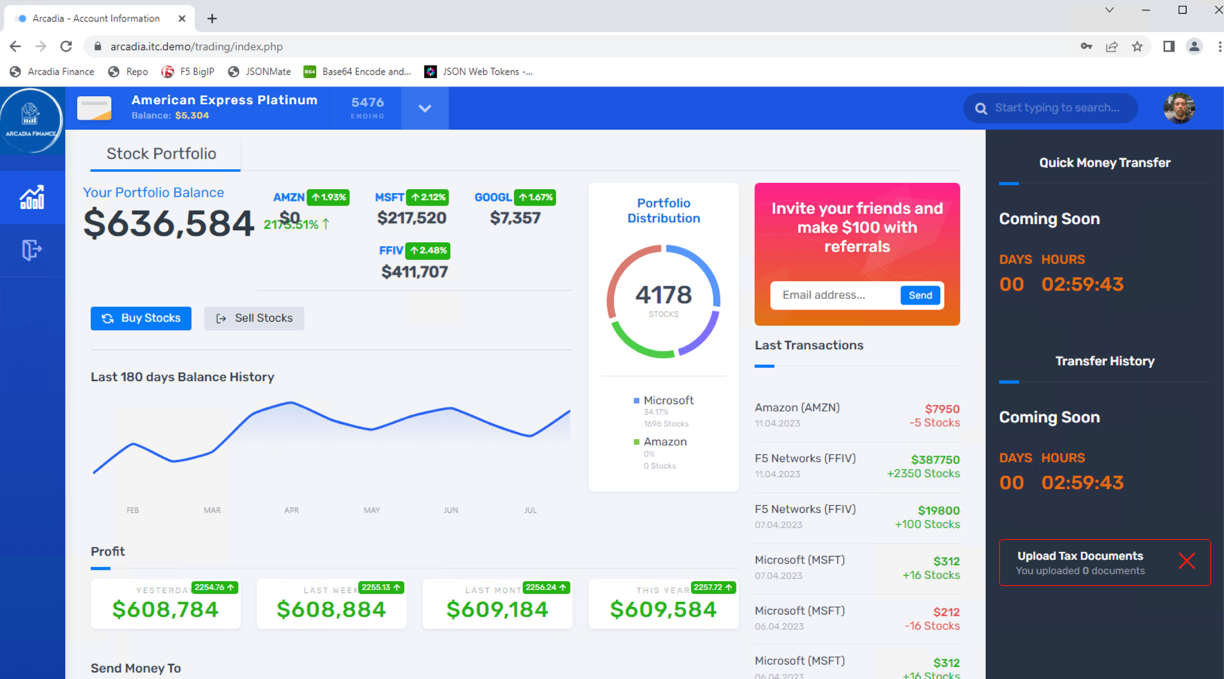
In the Lab Components deployment page -> Click the Access menu under the “Client Workstation” and select RDP

Open Chrome and Postman (will take several seconds)

In Chrome > click the Arcadia bookmark

Find the Login page and login using admin/iloveblue as user/password

Navigate the site and perform a couple of actions to familiarize yourself with the apps running. I.e. Buy stocks, sell stocks, view your last transactions



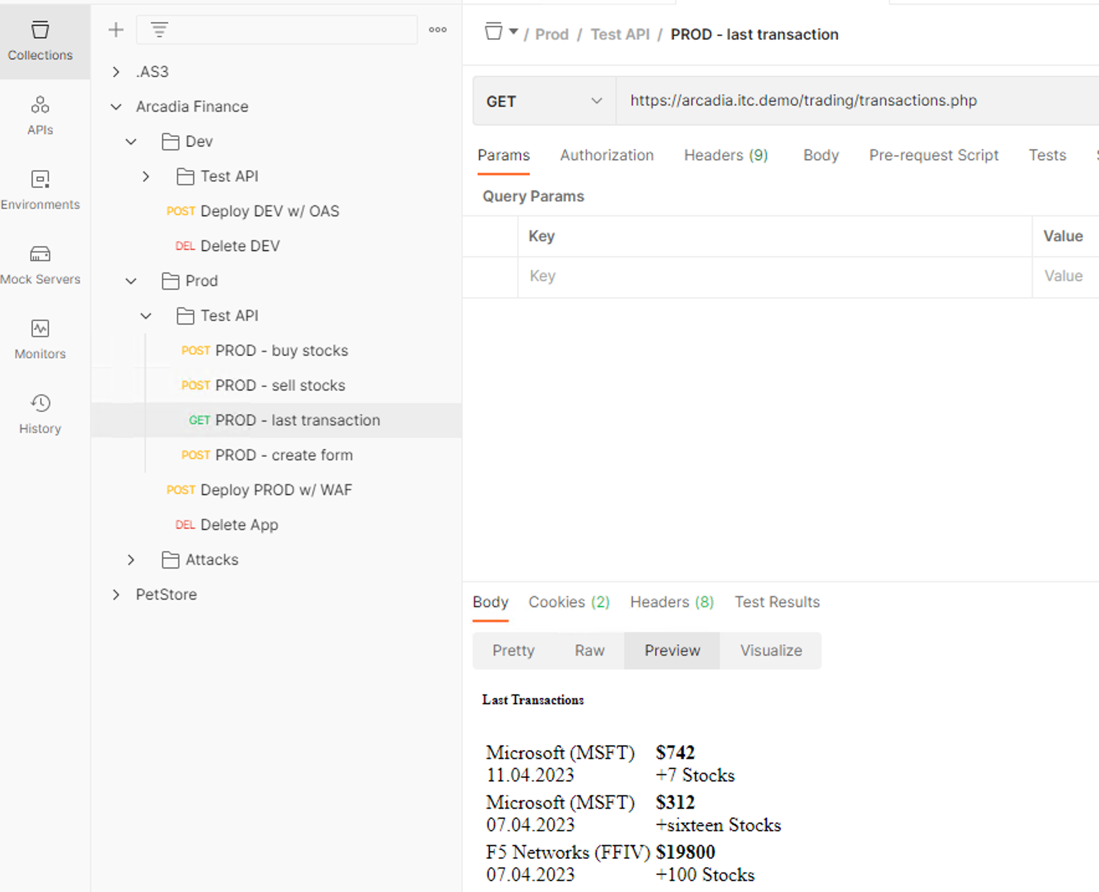
These three applications (buy, sell, last trans) are serviced by APIs populated by your selections on this frontend site.

Notice the “Coming Soon” countdown. We will be deploying that feature later in this lab.

**Task 2 – Review activity within Postman**

Click on Postman and expand Arcadia Finance collection.

Click Prod > Test API > PROD – last transaction and click Send  
Set the response body to “Preview” for a nicer presentation

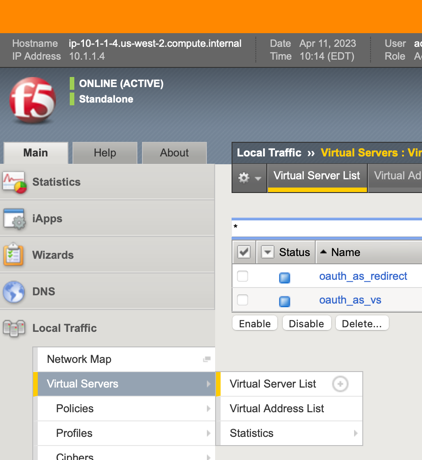


Note - Your transactions are stored in a NoSQL database (MongoDB)

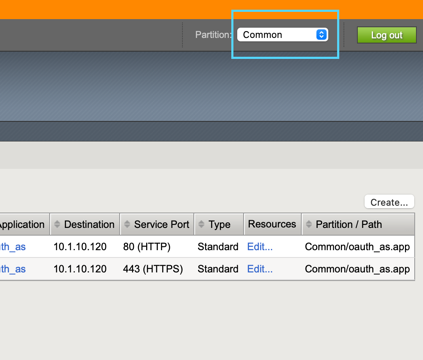
**Task 3 – Review F5 Configuration**

Back on the Lab Components deployment page, click Access under BigIP and select TMUI

Login with admin/admin for user/password and go to Local Traffic > Virtual Servers



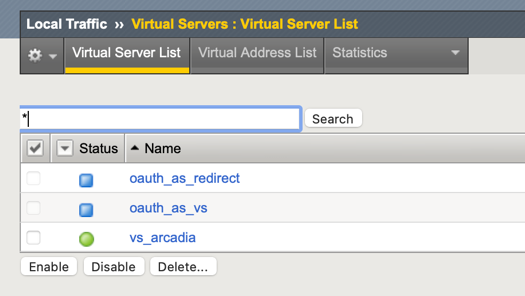
On the top-right, you should see the Partition drop-down.



Select the “Arcadia Prod” partition.

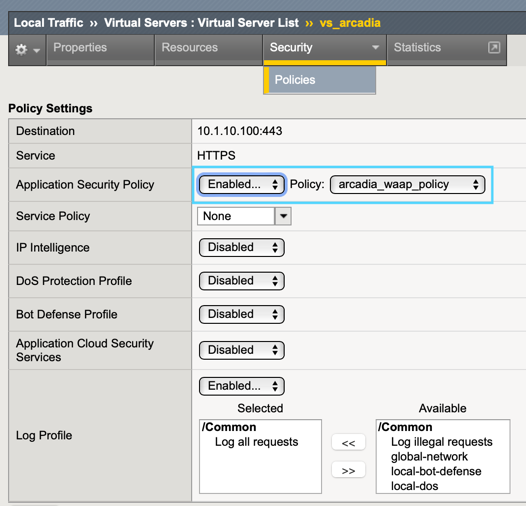
Note - If it’s grayed out, make sure you have clicked on Local Traffic > Virtual Servers

You should now see the Arcadia VIP



Click on the VIP and select the Security tab at the top.

Verify the security policy ‘arcadia\_waap\_policy’ is attached.



Let’s review the security policy.

Click on Security > Application Security > Security Policies

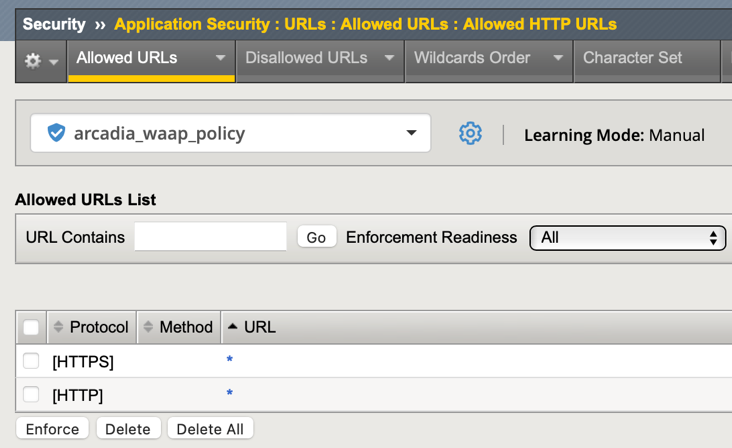
Click on arcadia\_waap\_policy

Review the settings, notice there is no Swagger file associated with this policy   
(Note: If there was a swagger file associated with the policy, you would see the name of the swagger file instead of the upload button)



Click on App Security > URLs > Allowed URLs

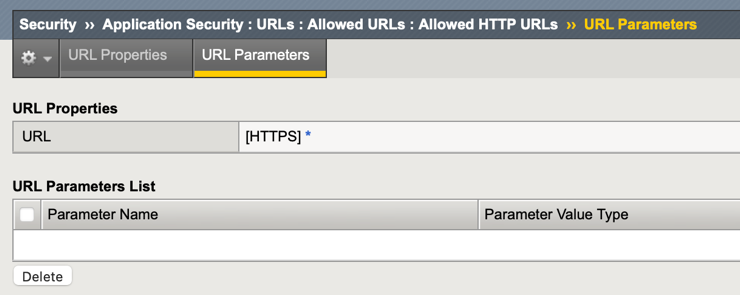
The policy is allowing all HTTP/S URLs via a wildcard.



Click on either of the asterisks.

Notice we are still enforcing attack signatures on the URLs.

Click on URL Parameters at the top and again we are allowing all via wildcard.



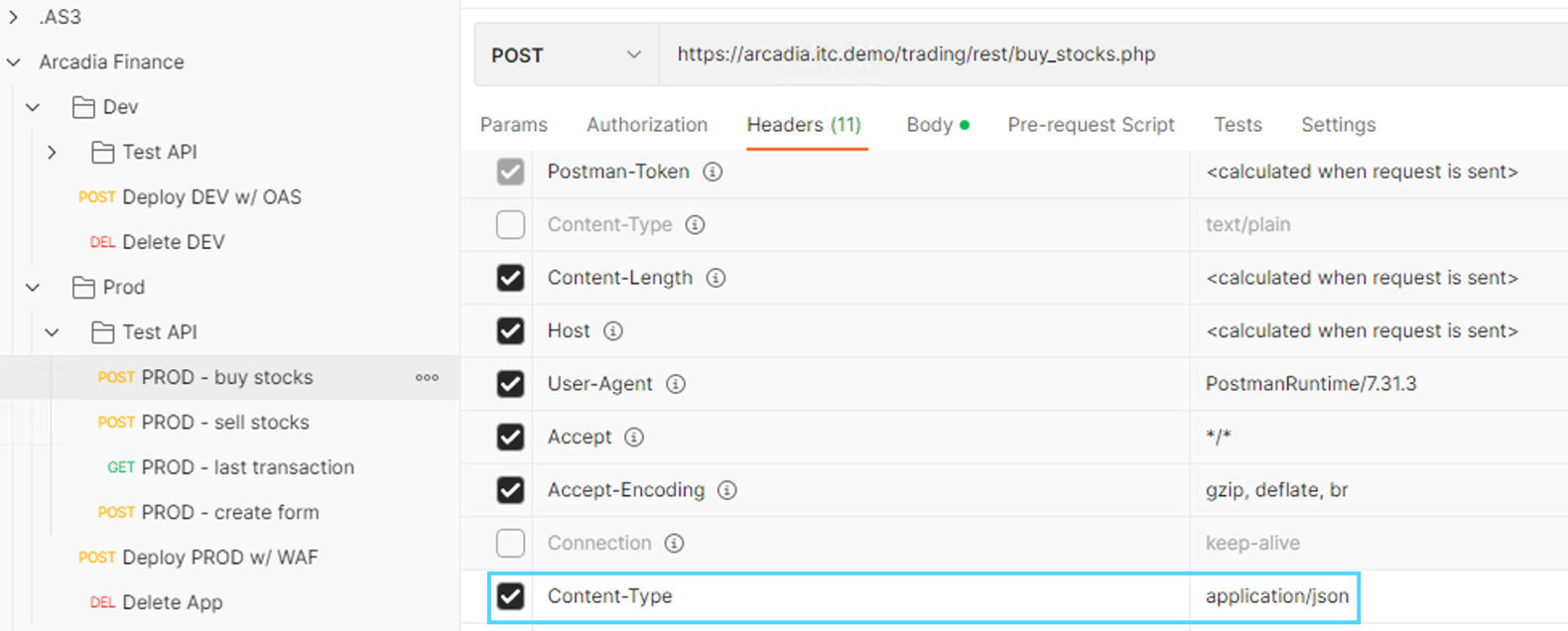
Click the asterisk parameter and again notice we are enforcing attack signatures.

Question?

What is this policy doing as far as the APIs are concerned? Nothing beyond OWASP Top10

Go back to your Windows RDP and got to Postman

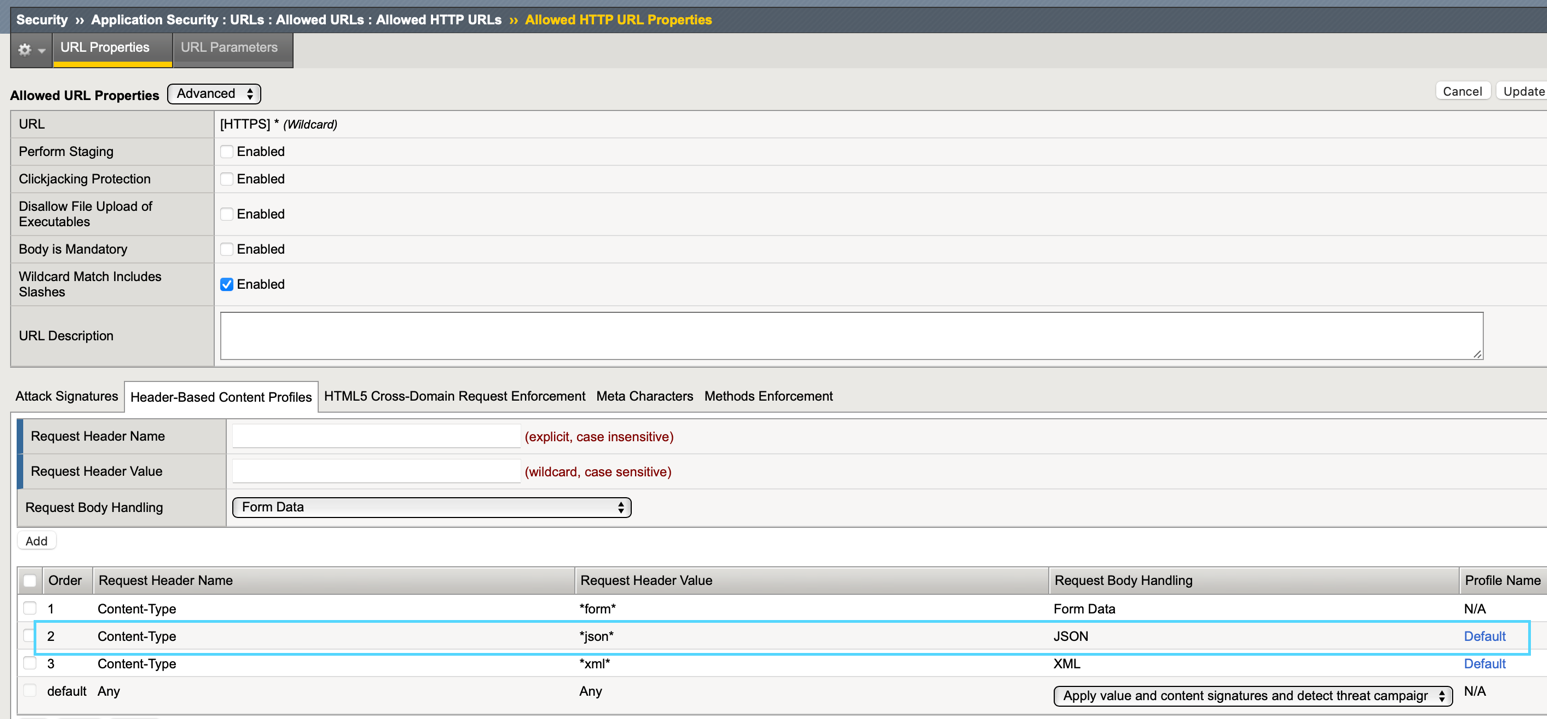
Select the Arcadia > Prod > Test API > buy stocks request and review the headers. Specifically, the Content-Type header. This value is important for enabling AWAF’s JSON parser.



Go back to the BIG-IP security policy, go to allowed URLs and Click one of the asterisks

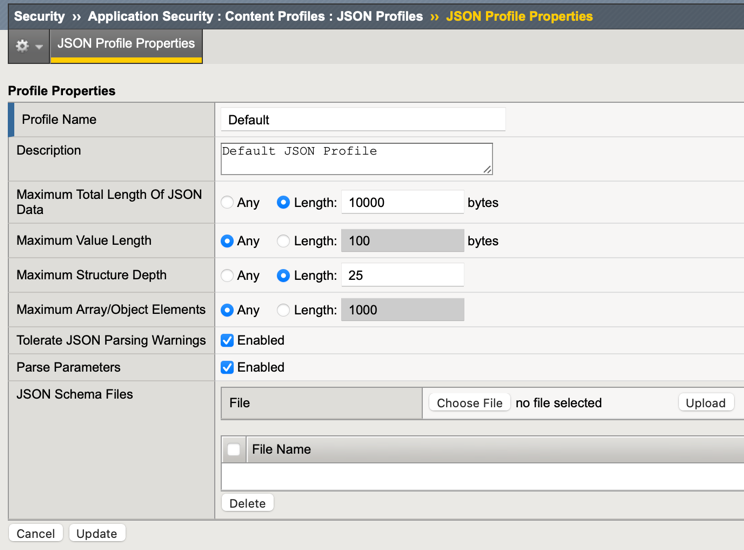
Make sure the URL Properties drop-down is set to Advanced (not Basic)

On the bottom of the page, select Header-based Content Profiles



This is saying if the policy sees application/json as the Content-Type header, it will invoke the “Default” json content profile

Click the Default json profile

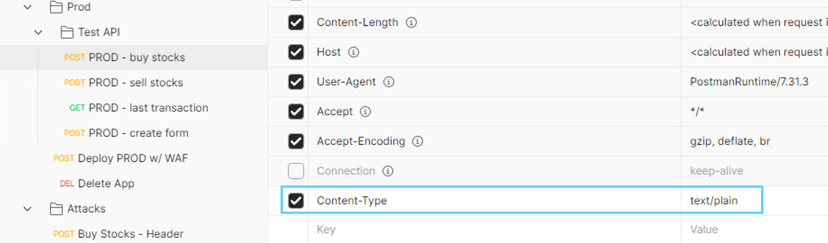


This component tells AWAF to inspect JSON content and parse parameter values for potential security violations. Notice we can also support custom JSON Schema files which can allow your App/Dev teams to get very granular with OpenAPI/Swagger request schemas. For this lab we will not be using this feature.

**Task 4 – Test Prod Security Policy**

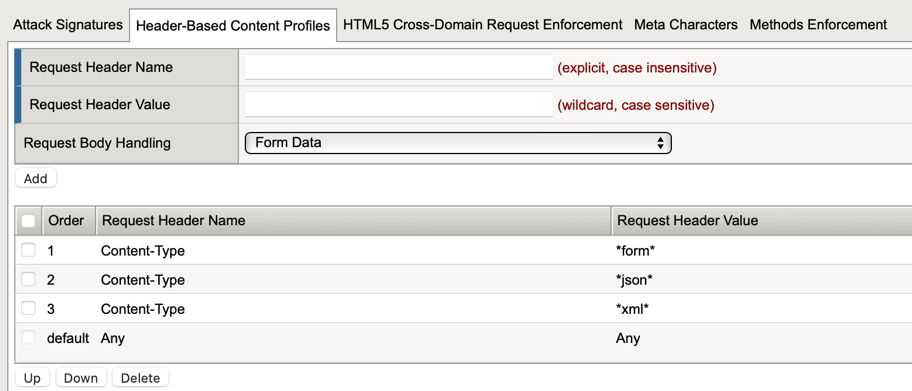
Now go back to Postman > Prod > buy\_stocks and click Send (You should get a 200 response)

Select Headers and change the content-type to text/plain instead of application/json and click Send again.



Given that our backend API should only accept JSON requests, is this acceptable?

Go back to the Arcadia Prod security policy and select allowed URLs > asterisk and review the Header-based Content profiles again. Notice there is a default rule that allows “Any” header value if it passes signature check validation. That is why your request was accepted.

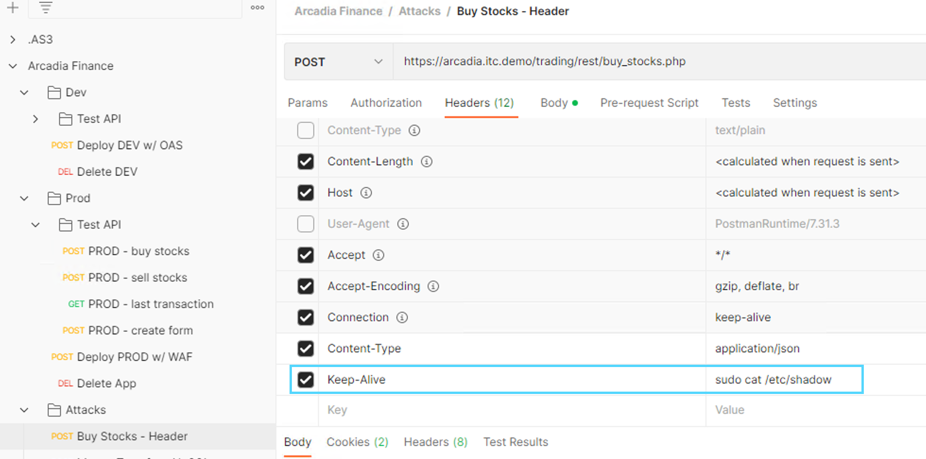


Let’s see if something more malicious will get through...

From the Windows RDP go to Postman and expand the Prod > Attacks folder

Select Buy Stocks – Header and then select Headers on the right pane

That Keep-Alive header looks suspicious!



Click Send on the request. Was it a success?

Review the ASM Security Event Logs under Security > Event Logs > Application > Requests

Go back to Postman > Test API folder and send the sell stocks and last transactions requests to verify they work

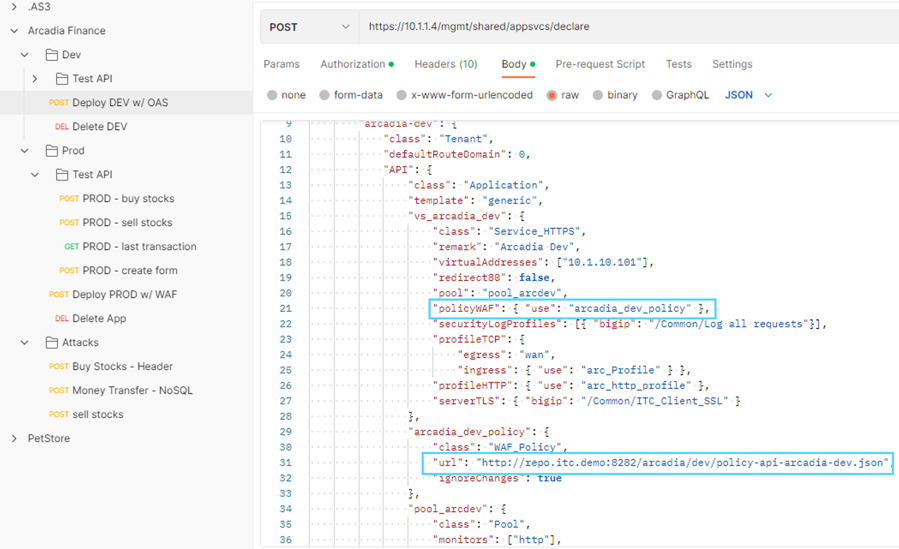
Now that we have reviewed our Production instance, let’s deploy our development version.

**Module 2 – Deploy Dev instance of Arcadia**

Objective - You will use F5’s declarative API, known as AS3, for deploying your new application instance. In your production security policy, recall that there is no Swagger/OAS file being used. While you could manually upload an OAS file to an existing policy, that is not the strategy Arcadia’s management wants to rely on going forward. For this lab, we will deploy the application along with the security policy which will be configured based on the OpenAPI spec file that is ready for testing.

**Task 1 – Review and Deploy AS3 Declaration**

In Postman, expand the Arcadia > Dev folder and click on Deploy DEV w/ OAS



In the declaration, we are deploying a new security policy called arcadia\_dev\_policy.

Near the bottom, you can see we are defining this policy by calling a json file.

Let’s take a look at this policy file

In the Components section click Access under the App Server (Ubuntu) and select Web Shell or SSH if you provided your own keypair.

In the CLI type or paste: cat /home/ubuntu/repo/arcadia/dev/policy-api-arcadia-dev.json



This is a basic Owasp Top10 policy based on our Production instance with one difference.

Notice at the bottom we are referencing an OpenAPI file. Let’s review this file.

From CLI type or paste: less /home/ubuntu/repo/arcadia/dev/arcadia-oas3-dev.json

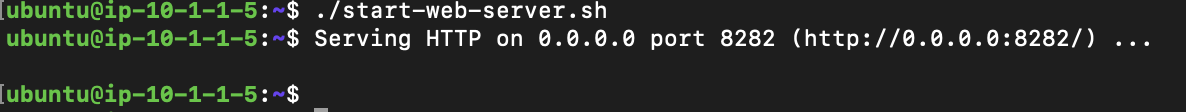
Hit SPACE or Enter to scroll through the file and review the settings.

The paths and parameters should look familiar as you’ve already used them in Postman.

Hit Q when you’re done.

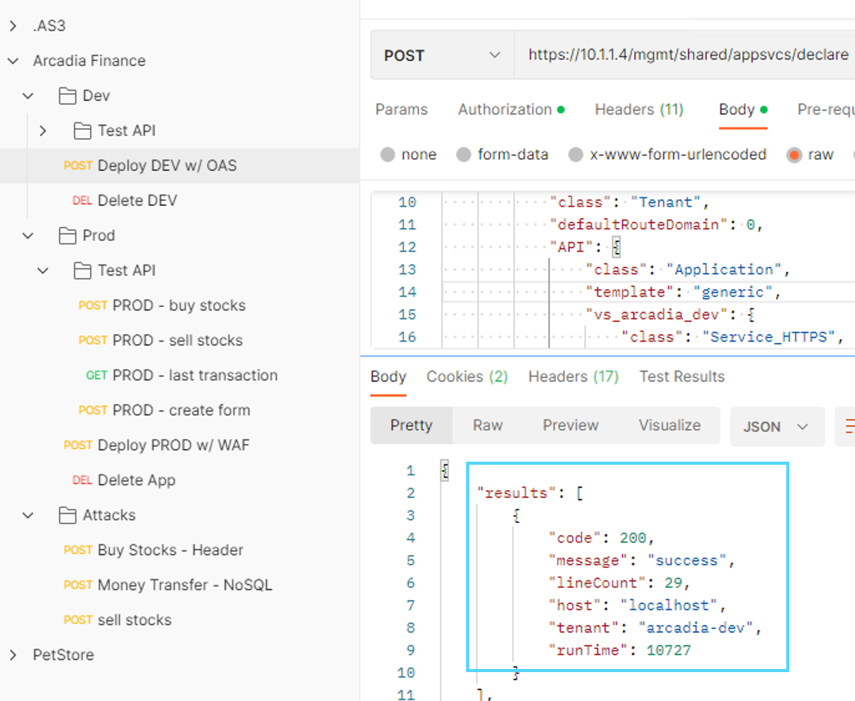
Now that you have examined how the declaration ties these files together, it’s time to start your repo service and deploy the Development VIP

In Ubuntu cd /home/ubuntu and type: ./start-web-server.sh and hit Enter a couple times



From Postman click Arcadia > Dev > Deploy Dev w/ OAS and hit Send (will take a few seconds)

You should get a 200 OK response with the details of your deployment.

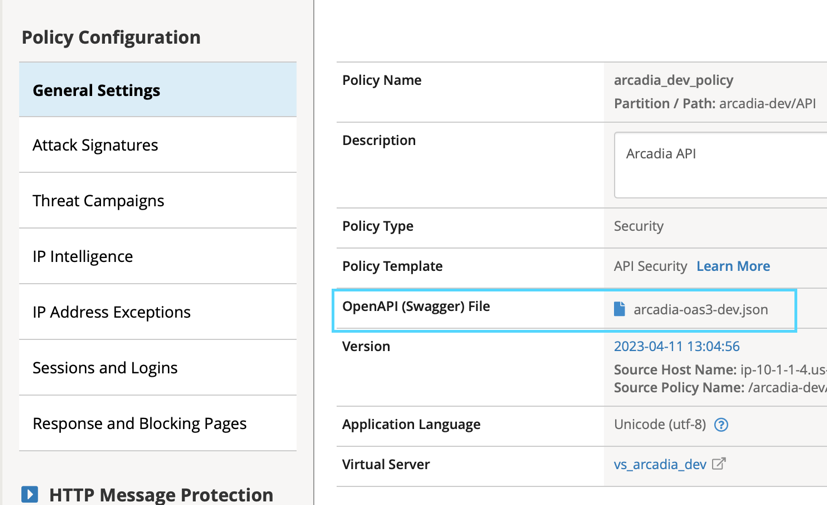
Verify the deployment was a success by clicking the Arcadia > Dev > Test API > Buy Stocks request and hit Send (JSON response body should indicate Success)  


In the BigIP select the new “arcadia-dev” partition at the top-right

Go to Security > Application Security > Security Policies

Click on the arcadia\_dev\_policy

Unlike our previously reviewed arcadia-prod policy, notice how now we are referencing an OpenAPI file in the security policy



Under Application Security go to URLs > Allowed URLs

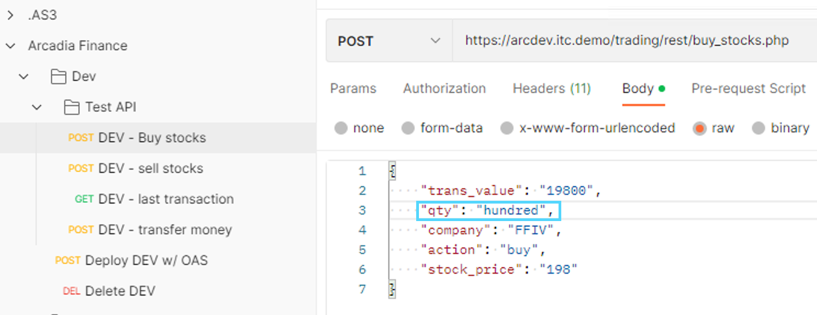
No more wildcard paths allowed since this is for API gateway enforcement not just basic web security.

Select buy stocks URL and click URL Parameters at the top

Notice how there are no wildcard parameters and parameters names are now defined.

**Task 2 – Test Security Policy**

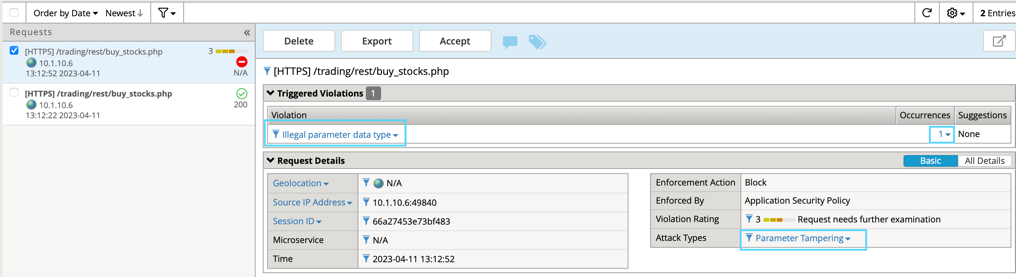
In the Postman Arcadia Finance > Dev > Test API, click on Buy Stocks and edit the “qty” parameter value to a word “hundred” instead of “100” and click Send



Review support ID by going to Application Security > Event Logs > Application

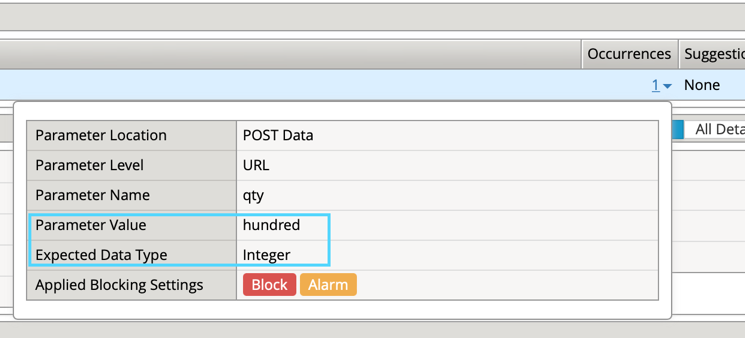
It should be the request at the top, if not click the filter icon and search for your Support ID

Select the request



The request is flagged for an illegal parameter type

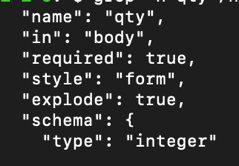
On the right-side, under Occurrences, click the number 1



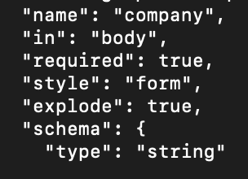
The policy expects an integer value, based on the OpenAPI file we imported

In Ubuntu CLI, type or paste:

*grep -n qty /home/ubuntu/repo/arcadia/dev/arcadia-oas3-dev.json -A 6 | head –8*

Notice the schema type for qty is “integer” – this sets our security policy configuration  
  
   
  
Grep for “company” instead of “qty” using the same command above:

*grep -n company /home/ubuntu/repo/arcadia/dev/arcadia-oas3-dev.json -A 6 | head -8*



How does the “string” value reflect in your security policy for this parameter?

In the BIG-IP, go to the Security > Application Security > URLs > Allowed URLs and select buy\_stocks URL and click URL parameters

Select the “company” parameter and notice it’s Data Type is set to Alpha-Numeric

Review how the OpenAPI parameter settings (from our last grep command) influence your WAAP policy

required: **true**/false  
 in: body/query

\*Note - Style: form is required in order to support { key: value } pairs in this format.

\*Note - Explode: true tells the parser to create separate objects for each parameter

In Postman click on Buy stocks again and set **qty** back to 100

Change **company** to “fubar” and hit Send

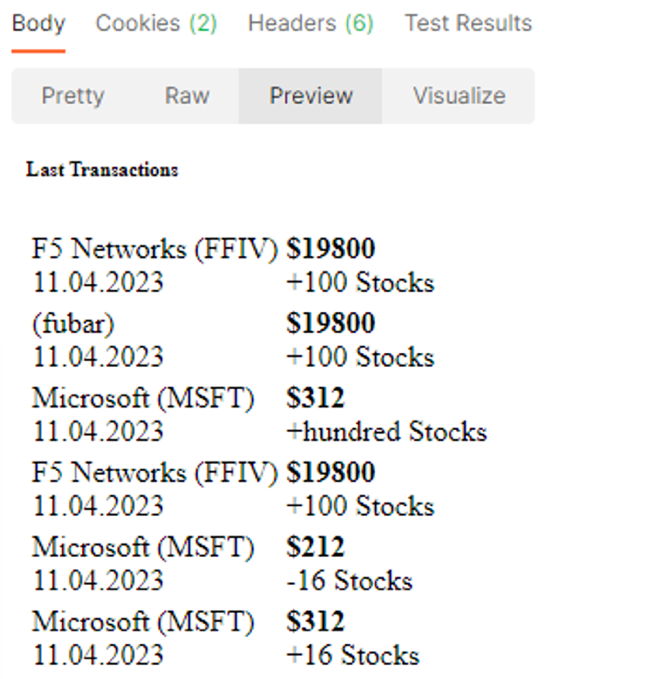
Did the request succeed? Should it have?

Change the company value to “cat /etc/ssl/certs/\*.\*” and click Send  
 Did the request succeed? Review the security event log

In Postman select the last transaction request and hit Send

Set the response body in Postman to “Preview”

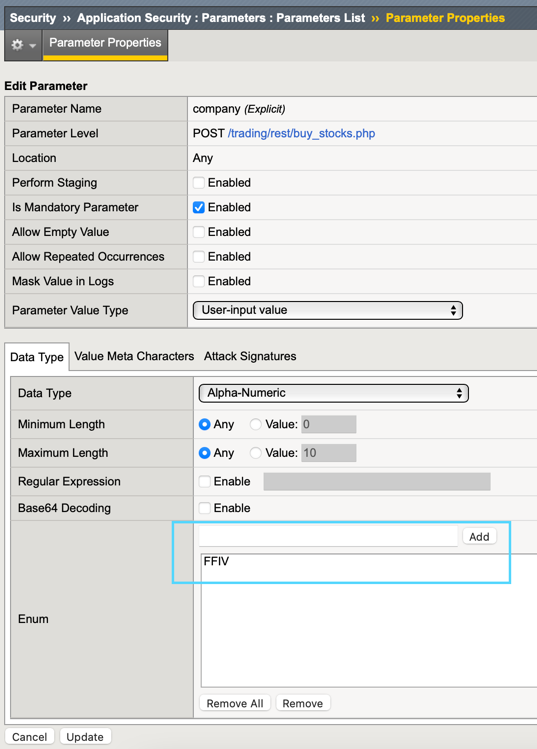
Notice the blocked request does not make it to the backend database



Since we don’t want erroneous values (like fubar) being accepted in our transaction DB, let’s see how we can lock this down.

On the BIG-IP, go back to the **company** parameter for the buy\_stocks URL (Security > Application Security > Parameters > Parameter List )

In the Enum field, add FFIV to the list



Click Update and then be sure to **Apply the policy**

In Postman click the Buy Stocks request and change the company name to FFIV and click send

Request should succeed

Change company value to lower-case ffiv or any word other than FFIV and click Send

Review the security logs

We have proven we can lock the parameter down to specific values but since we are deploying as code, any changes we make directly to the security policy in the BigIP config will be lost if the application is redeployed. The changes need to be made permanent by adding them to our OpenAPI Spec file. Fortunately, OpenAPI allows you to define parameter enumerations.

In your Ubuntu CLI, cd /home/ubuntu/repo/arcadia/dev

Use vi/nano to edit arcadia-oas3-dev.json

Under the buy\_stocks.php path, find the company parameter settings and edit the schema object to include enumerations as below.

\*Note - If using vi, to edit the json hit ‘i’ and the editor will enter the INSERT state. When done editing hit ESC then type :wq! And hit enter.

{

"name": "company",

"in": "body",

"required": true,

"style": "form",

"explode": true,

"schema": {

"type": "string",

"enum": [

"MSFT",

"AMZN",

"FFIV"

]

}

},



When we make changes to our OpenAPI file, AS3 needs to re-import the file for settings to take effect. In this case, we need to delete the current Dev VIP and re-deploy it.

Go back to Postman and select the Arcadia > Dev > Delete DEV request and hit Send

Once the Delete DEV request succeeds, Send the Deploy DEV w/OAS request to re-deploy and import the new parameter settings.

If the deployment fails, the confirm that you can load the swagger file from the webserver from the Windows RDP session to the following path: <http://repo.itc.demo:8282/arcadia/dev/arcadia-oas3-dev.json>

Making a keying error in the swagger edit can cause the python web server to fail. In the event your deployment fails, you can relaunch the webserver by using the following commands:

cd /home/ubuntu

pgrep python3 |xargs kill

./start-web-server.sh

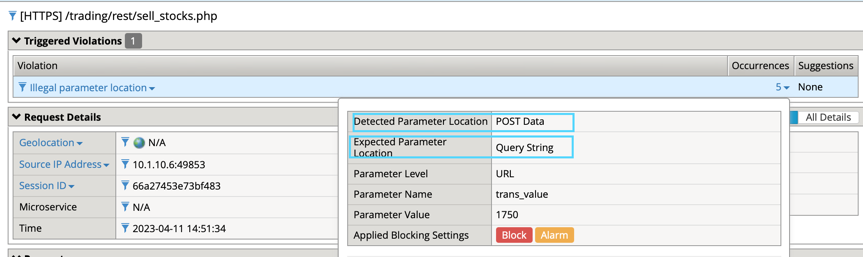
# hit enter a couple of times

In the BIG-IP, review the changes to the company parameter for buy stocks URL. In theory, we would apply enumerations to several parameters (the action parameter should only be buy or sell, as an example) but time constraints allow for only so much JSON editing...

Now let’s test the other API’s

In Postman, select the Arcadia Finance > Dev > Test API > DEV - sell stocks request and click Send

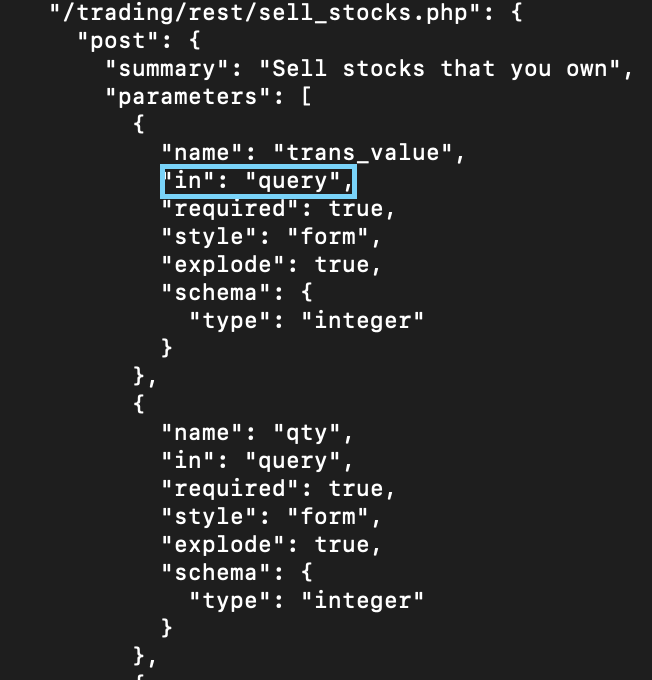
Why was the request blocked? Review the security event logs

On the right side, click the “5” under Occurrences   
  
 

The policy is expecting our parameters to be sent as part of the query string rather than in the content. This would mean passing parameters in the form of:   
  
*https://arcdev.itc.demo/trading/rest/sell\_stocks.php?trans\_value=1750&qty=100&....*   
  
Based on the production Arcadia deployment, we know this is incorrect and that our parameters are passed as json content. Let's look at the OAS file to find the problem.

In Ubuntu CLI, less /home/ubuntu/repo/arcadia/dev/arcadia-oas3-dev.json

Scroll down until you see the sell\_stocks path then review the parameters



Chances are whoever created this file did a copy/paste and forgot to edit this value...and then turned it into a lab task.

Compare the parameter settings in OAS file between buy\_stocks and sell\_stocks

Since we know the requests have the same format, edit the arcadia-oas3-dev.json file sell\_stocks parameters so they are located correctly in our security policy.

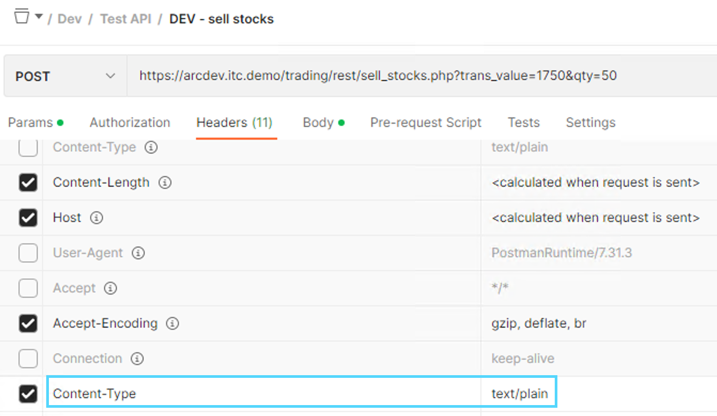
When you have finished editing the file, you will need to delete and re-deploy your VIP from Postman as you did earlier.

Arcadia > Dev > Delete Dev

Arcadia > Dev > Deploy Dev w/OAS - this will update our policy with new settings.

Ensure the security policy is accepting requests for DEV Buy, Sell and Last Transactions

Select either Buy or Sell Stocks and change the Content-Type header to text/plain as you did earlier in Prod, hit Send.



Why is the request blocked this time?  
  
In this scenario, our security policy is providing more value than just Owasp Top10, we are also enforcing the API’s rules as a gateway, which is a great way to reduce noise/load on your backend containers.

Now that you are an OpenAPI pro and a JSON editing whiz, it’s time to put your skills to the test by adding the new Money Transfer microservice to our API gateway.

**Arcadia Lab 2 – Add a New Microservice**

**Lab 2-Module 1 – Test Money Transfer app**

Scenario: One of the Arcadia App teams has released a new microservice that is ready for security testing. You have been tasked with incorporating the new Money Transfer microservice into our API gateway security policy.

In order to achieve this we will need to:

* Launch the container running your new micro-service
* Update your Nginx Container Ingress to include the service
* Functional test with the existing security policy
* Incorporate all changes into OpenAPI file and deploy as code

Task 1 – Deploy container running new microservice

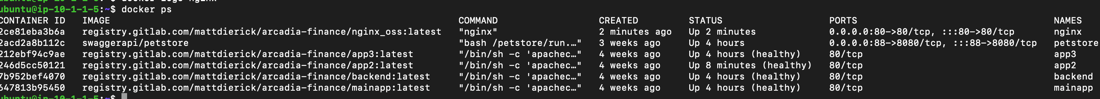
From Ubuntu CLI enter: docker start app2

Task 2 – Update Nginx Container Ingress  
 Since this is a lab, Prod and Dev are both using the same pool. Therefore, technically this step breaks Prod.

Diagram

Description automatically generated with medium confidence

* In the Ubuntu CLI, vi /home/ubuntu/nginx.conf and remove the # in front of upstream/api and in front of /api proxypass (lines 11-13 and 33-35). Do not remove the # from any other line. Save the file
* From the Ubuntu CLI enter the command: Docker stop nginx && docker rm nginx
* Now enter (copy/paste all as one line):   
  docker run -dit -h nginx --name=nginx --net=internal --restart=unless-stopped -p 80:80 -v /home/ubuntu/nginx.conf:/etc/nginx/conf.d/default.conf [registry.gitlab.com/mattdierick/arcadia-finance/nginx\_oss:latest](http://registry.gitlab.com/mattdierick/arcadia-finance/nginx_oss:latest)
* Enter “docker ps” and you should see six containers are running



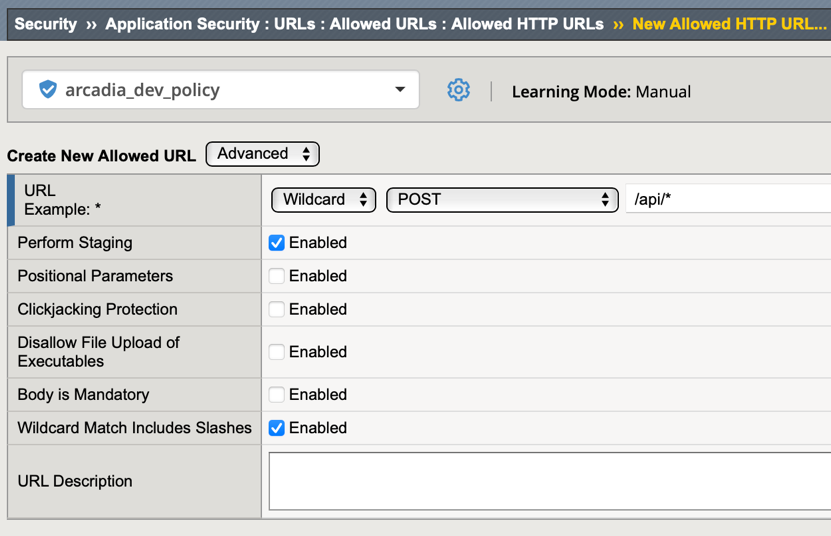
Task 3 – Test Microservice

In Postman, select Arcadia Finance > Dev > Test API > DEV transfer money and click Send

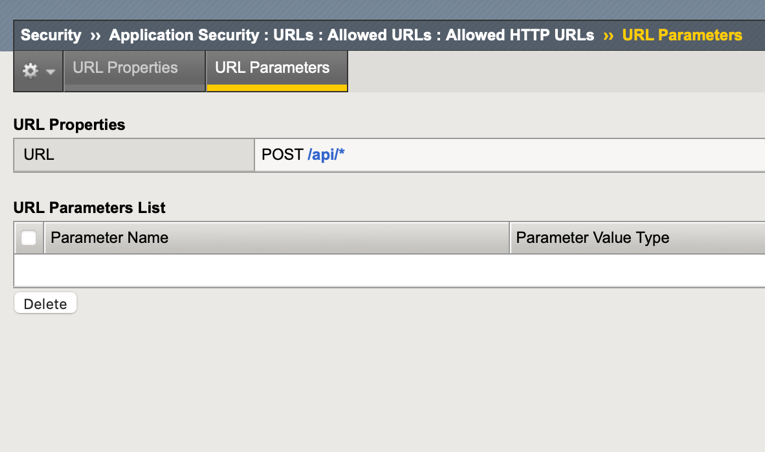
The request is blocked  
Review the security event log and correct the issue

For testing purposes, we will simulate the Prod policy and add wildcard URL and Parameter values to the existing policy.

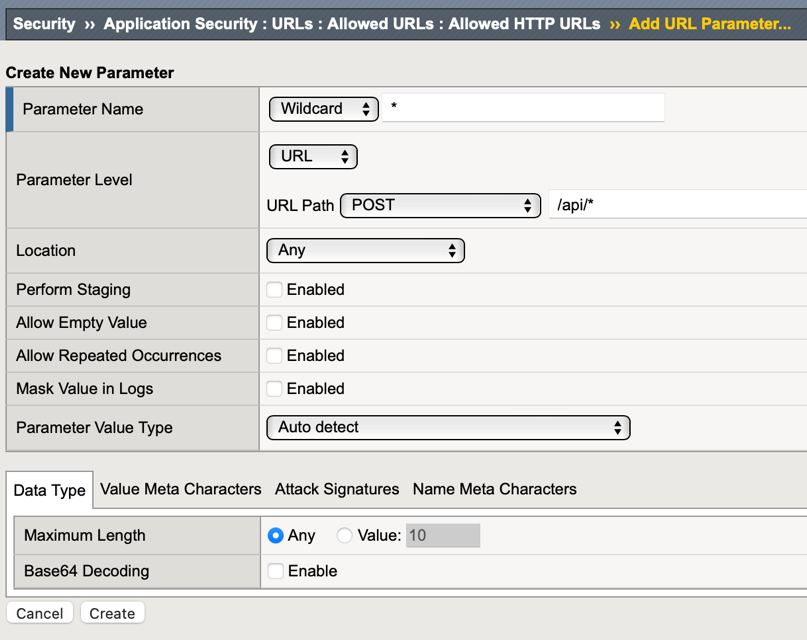
Go to Application Security > URLs > allowed URL’s > create new http URL:



Since we will be passing parameters in the request, we need to account for that in the policy. Go back to your list of allowed URL’s and click on /api/\* then select URL Parameters



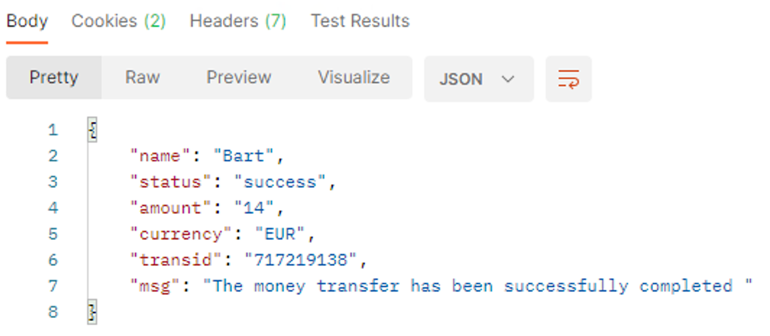
Click Create and configure as below:



Click Create

Go back to Security Policy and **Apply Policy**

In Postman retry the Money Transfer request and it should now succeed



\*Note\* If you get ASM Block with “Illegal HTTP Status Code” and a 404 server response then delete the app2 container and rebuild by the following CLI commands:

docker stop app2 mainapp nginx

docker rm app2

docker run -dit -h app2 --name=app2 --net=internal registry.gitlab.com/mattdierick/arcadia-finance/app2:latest

docker start mainapp nginx

If requests are successful, it is time to make these changes permanent in your OpenAPI file

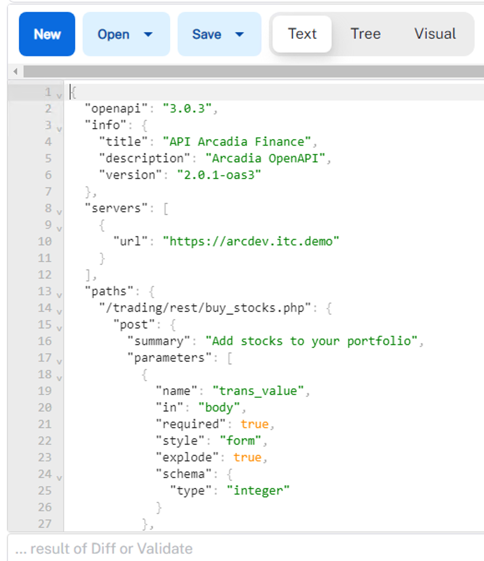
**Lab2:Module2 – Add Money Transfer Service to OpenAPI File**

Task 1 – Configure OpenAPI Spec Parameters for Money Transfer

Open a new tab in Chrome and click the JSONMate bookmark

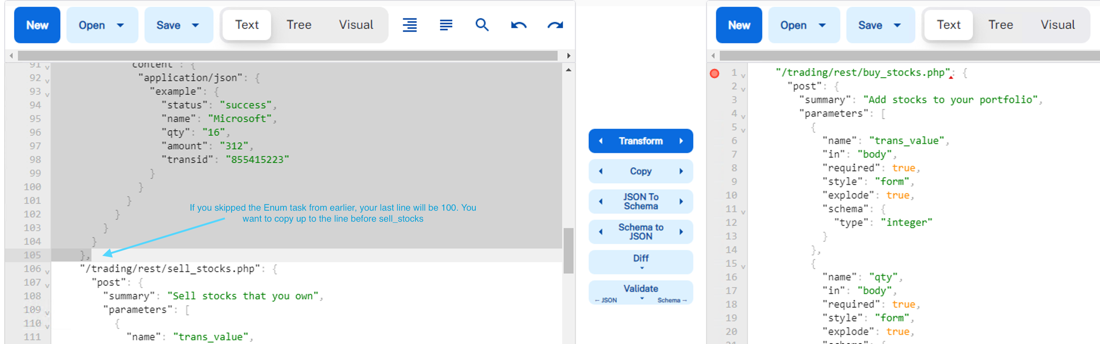
Click “New” on the left and right JSONMate windows panes

Copy /repo/arcadia/dev/arcadia-oas3-dev.json (also accessible from: <http://repo.itc.demo:8282/arcadia/dev/arcadia-oas3-dev.json>) to your clipboard and paste in the left-side JSONMate window



Highlight and copy lines 14 thru 105 ( "/trading/rest/buy\_stocks.php": { to }, )

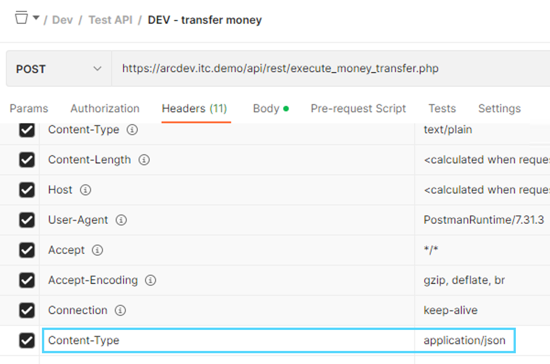
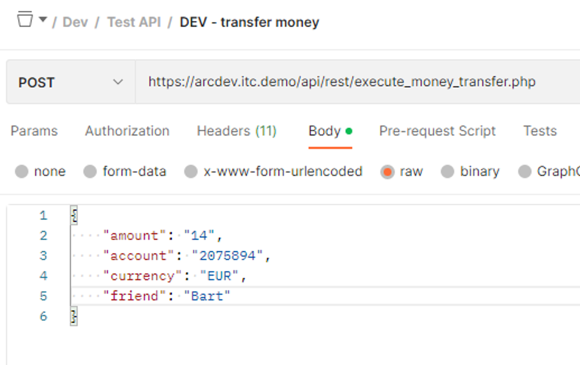
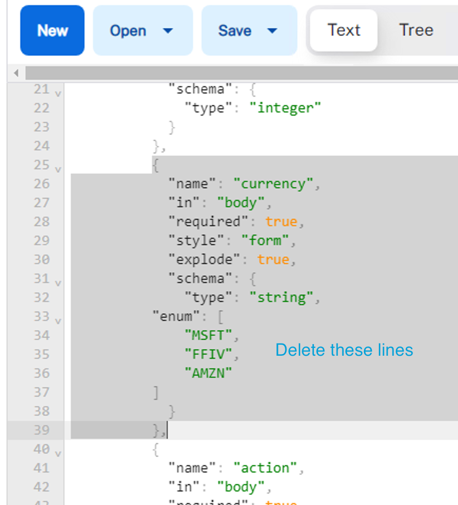
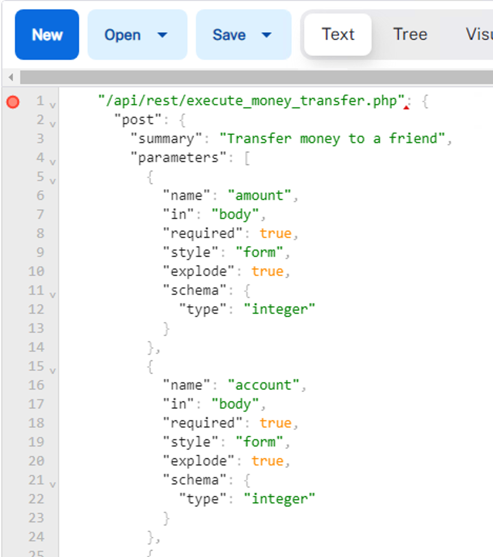
Paste the buy stocks path info on the right-side of JSONMate (ignore errors at bottom)



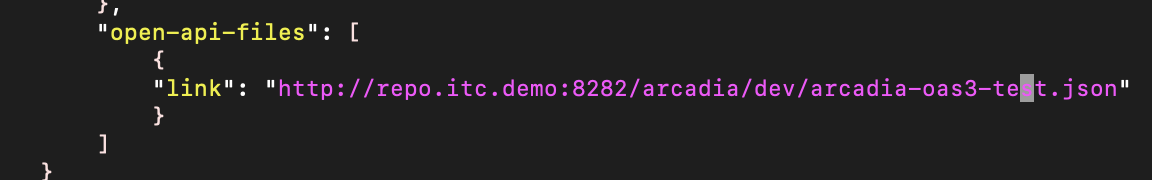
We are just using Buy Stocks on the right as a template for the new Money Transfer API

Edit the right-side so it reflects what we know about the new API

In Postman, review the transfer money request to help define the OpenAPI file configuration

* From the body and headers here we know the URI path, it contains 4 parameters, what schema type those parameters are (integer, string) and they are passed in JSON  
  
* Edit the path and parameter values on the right-side JSONMate pane with the new values. Since buy stocks has 5 parameters and we only need 4, delete the “company” parameter on the right-side (Line 25 – 39)   
  
* The right-side should resemble the image below:  
    
  make sure to edit the RequestBody and Response examples at bottom  
  You can review the Postman response body for how to configure your API spec response
* Once you have the new spec defined, paste it to the left-side JSONMate pane BEFORE the “/trading/rest/...” line. If you followed all labs, this should be line 106  
  
* Look for any formatting errors in red at bottom of left side. The last character in your output should be a comma ( , ) on line 181

**Task 2 – Create New OAS File for Testing**

* Copy the left-side JSONMate to clipboard and create a new OAS file in your repo  
  sudo vi /home/ubuntu/repo/arcadia/dev/arcadia-oas3-**test**.json > paste and save file
* Now you need to update the WAF policy file used in our AS3 declaration to use the newly created OAS file:  
  sudo vi /home/ubuntu/repo/arcadia/dev/policy-api-arcadia-dev.json  
  edit openapi spec file location to the test version we just created in the previous step  
  
* Redeploy the Dev VIP to update our security policy  
  In Postman, delete and then re-deploy the Dev VIP  
  As long as your JSON file is valid, you should receive a success indication
* In Postman send the DEV transfer money request. If you have any issues, resolve them in the OpenAPI file

Well done! You have successfully created an API gateway policy and added a new service!

“But what if I want even MORE security?” you ask...

Let’s look at an example of how Advanced WAF can enhance security

**Lab2:Module3 – Enhance Security with AWAF Features**

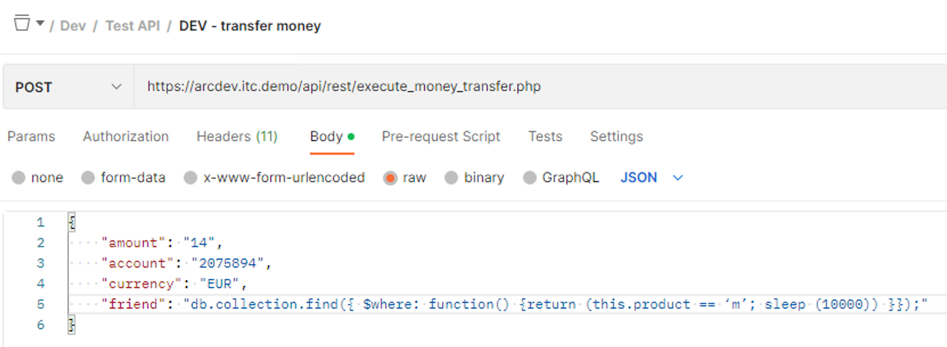
**Task 1 – Attack a Parameter**

In Postman, select the Dev > Test API > transfer money request again

Since we know the “friend” parameter is accepting alpha-numeric values, let’s see if we can sneak in a NoSQL Injection to our MongoDB

Change the friend value from Bart to the following and click Send

db.collection.find({ $where: function() {return (this.product == ‘m’; sleep (10000)) }});

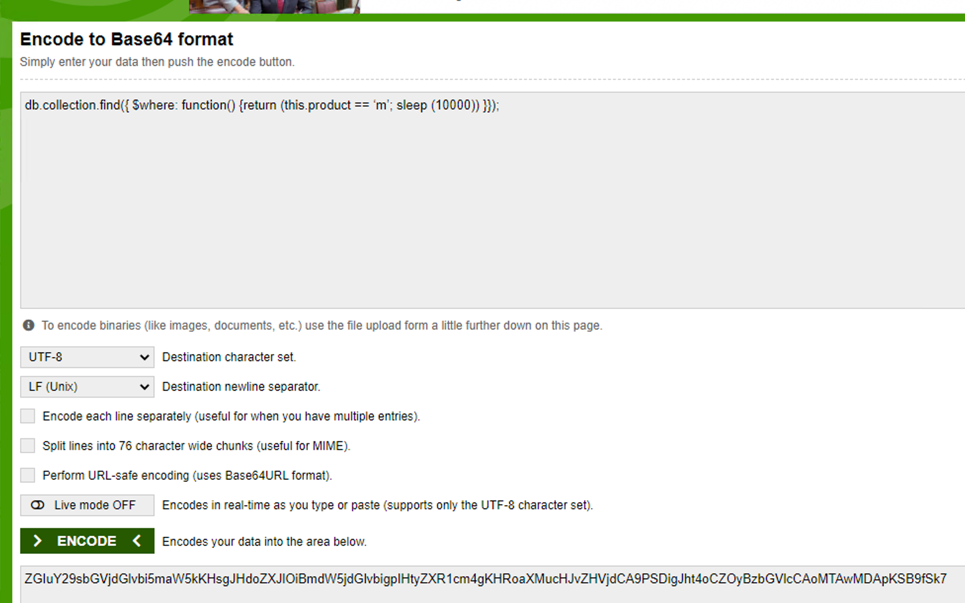


Predictably, we are blocked for a NoSQL Injection

**Task 2 – Encode the Attack and Try Again**

Open Chrome and go to the Base64 Encode bookmark

Encode your NoSQL Injection string and copy the results



Now in the Money Transfer request, change “friend” to the new encoded string and Send

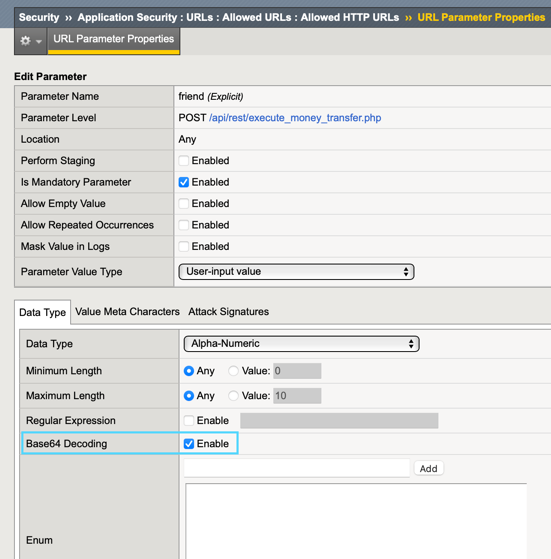




Unfortunately, this type of security is not something OpenAPI has a spec for. To protect your API against base64 encoded attacks, you would need to enable a WAF-specific feature on the parameter.

**Task 3 – Tune Security Policy**

Go to the Arcadia dev security policy > allowed URLs > money transfer > URL Parameters

Click the “friend” parameter and enable Base64 decoding  


Click Update and then Apply the Policy

Go back to Postman and re-send the same request with encoded “friend” parameter

Check the security logs - The parameter value is now decoded and flagged as a violation

Time for a well-deserved break! This concludes the Arcadia OpenAPI Security Lab.