

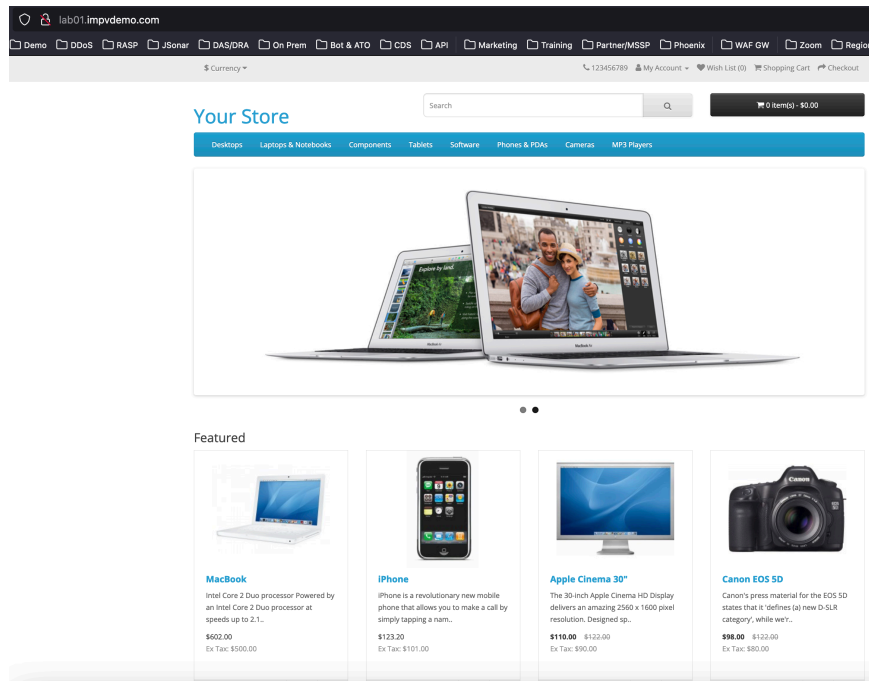
API and Advanced Bot LAB EXERCISES

Table of Contents

- API and Advanced Bot LAB EXERCISES1
- 1. Getting Started2
- 2. Advanced Bot Protection - Configuration4
 - 2.1 Advanced Bot Protection – Testing7
- 3. API Security – Configuration and Testing9
- 4. Log Review14

1. Getting Started

1. A website named **labxx.impvdemo.com** has already been onboarded to **my.imperva.com** where **xx** is your assigned user number. It has not yet been set up to prevent API attacks or attacks from Bot such as Account Take Over (ATO) or scraping attacks.
2. First, check that you are able to see your assigned website using a browser. You should see a display similar to lab01.impvdemo.com shown below, which is an example store website.



3. Logon to (my.imperva.com) MY using the following link, with the username and password you have been allocated. The first link will only move you to the correct place if you don't already have the same email address registered in MY. The second link will take you directly to the account for this exercise where you should replace the <account_id> field with the correct ID that you have been allocated.

<https://my.imperva.com/admin/login>

https://management.service.imperva.com/my/sites?caid=<account_id>

4. You should see one protected website similar to the example shown below. Choosing **Application** from the main menu will show you the onboarded website with menu options for API and Advanced Bot Protection.

imperva

CURRENT ACCOUNT
Lab1101

[Home](#) [Application](#) [Network](#) [Data](#)

Q Search

[? Help](#) [Account](#) [Admin](#) [Trial: 11 days left](#)

Application

Production

[Incapsula Sales](#) / [Imperva Enterprise Sales](#) / [Trials](#) / [NA](#) / [Demo AMS](#) / [Amplify AppSec 2022](#) / [Lab1101](#)

Websites

Viewing: Statistics (Last 7 days)

Filter by Keyword or ID

Site Name	Bandwidth (30 days)	Human Visits	Bot Visits	WAF Sessions	Creation Date	Status
lab01.impvdemo.com (46052984)	N/A	0	0	0	04 Mar 2022	✓

Showing 1 to 1 of 1 entries Show 25 rows

<< < 1 > >>

2. Advanced Bot Protection - Configuration

In this exercise, the configuration of Advanced Bot Protection (ABP) will be reviewed and configured to prevent a scraping attack that will capture the products and prices from the protected website using a script. The attack will be launched from a test website which allows for scraping and API attacks at the press of a button.

To begin with, the problem needs to be reviewed.

1. Open up the attack website, open the following URL:

`http://gentraffic.impvdemo.com/`

2. There is a button called **ABP Script** that will launch a Python script using Selenium and Chromium driver in the background to move to the website you define and scrape the products. Click the **ABP Script** button, which should move you to a new screen inviting you to fill out the website details. Enter the **target site** in the format shown below. The full URL must be specified as shown below.

`http://labxx.impvdemo.com/`

3. Once complete, press the **Attack** button. This should launch the attack script, which will launch a Chrome process in the background and start reviewing each of the pages on the website as a user might do. The output of the attack script should contain the prices for each website page scraped, and should be similar to the output shown below.

Successfully scraped the following: ['MacBook', '\$602.00', 'iPhone', '\$123.20', 'Apple Cinema 30"', '\$110.00']

4. Next, the Advanced Bot Protection needs to be configured to prevent the attack that is happening. To review Advanced Bot Protection configuration, move to **Advanced Bot Protection > Settings** on the left menu. Currently nothing is configured.
5. A Website Group that contains the settings should be first configured. To configure a Website Group, click the **+** button on the right-hand side of the screen.
6. Give the Website Group an appropriate name, an example for lab01 is shown below, but make sure your website is chosen from the Website field as shown, then click **Create**.

Create Website Group

Website Group Name

Select the website you want to protect. The required configurations are automatically generated for you.

Website

Cancel Create

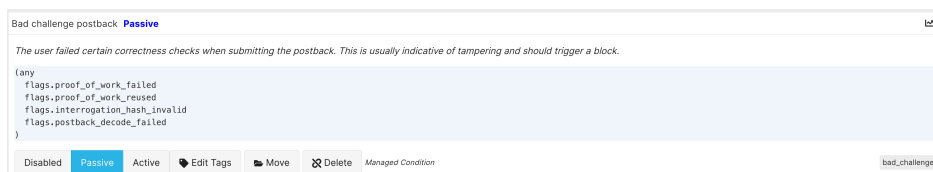
7. The Website Group should now be defined as shown below. Notice that there is an option to edit the Default Policy as well as assign different policies on a per-path basis. This means that the protection may be tailored depending on the use case of the path under the website.

For example, clicking the button for Desktops shows the path of <domain>/**desktops/mac** whereas the path is different for other products, such as Tablets, which is <domain>/**tablets**.

8. For a scraping use case that covers all products, it is easier to configure the **Default** policy which will apply to all paths for the websites. To do this, click the button to **Edit Default Policy** which moves to the **Policies** tab.
9. Notice that the policy is divided into several sections, listed below.
 - allow
 - block
 - captcha_cleared
 - captcha
 - identify
 - monitor
10. Usually, the sections **allow**, **block** and **captcha** are relevant to policy definition. Note that by default within a policy there is blue text to right of the **condition** which denotes the state of the condition. For example, examining the block section should show configuration similar to that below, where by default all the conditions are in a **Passive** state.



11. Each condition has a number of states, that may be examined or changed. To edit a state, click on the condition which opens up a sub-menu that describes the condition and allows the state of it to be changed between **Disabled**, **Passive** and **Active**, as per the example below. The condition may also be **Edited**, **Moved** to a different section and **Deleted**.

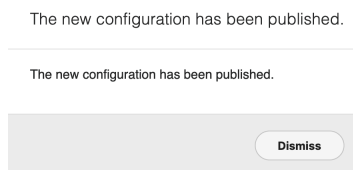


12. The condition that is going to be configured is one that detects automation used in browsers. The attack script is using Selenium and the Chromium driver to instantiate a new browser instance, automatically fill in some fields to move to the right page and capture the results. As such, this may be detected and blocked by Advanced Bot Protection.

Currently the condition to detect Automation is in a different section, known as a **Directive**.

Under which directive (section) is the condition that might stop Automation?	Captcha
If the Automation condition is set to active, what should happen when the script is run?	A Captcha should be sent to be filled out before the script can complete.

13. Change the condition for Automation to **Active**. This alone is not enough to update the configuration. The configuration change must also must be published to the cloud.
14. To publish the configuration, scroll to the top of the page and click the **Publish Configuration** button on the top right of the page. A message box should confirm the configuration was published, as below.



2.1 Advanced Bot Protection – Testing

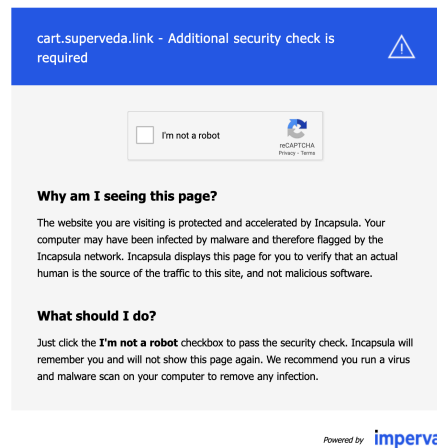
1. Re-run the scraping script to try it out.

What happens?	The script should break because it is trying to send Captcha in the background, which should be enough to break the script. Note that the first line of the script will always execute.
---------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

A screenshot of the error is below:

```
Message: no such element: Unable to locate element: {"method":"xpath","selector":"//*[@id="content"]/div[1]/div[2]/h1"}
(Session info: headless chrome=98.0.4758.102)
Stacktrace:
#0 0x55a0e6e5db33 <unknown>
#1 0x55a0e69266d8 <unknown>
#2 0x55a0e695c6f1 <unknown>
#3 0x55a0e695c8b1 <unknown>
#4 0x55a0e698f574 <unknown>
#5 0x55a0e697a08d <unknown>
#6 0x55a0e698d2fb <unknown>
#7 0x55a0e6979f53 <unknown>
#8 0x55a0e694fa0a <unknown>
#9 0x55a0e6950ad5 <unknown>
#10 0x55a0e6e8f2fd <unknown>
#11 0x55a0e6ea84bb <unknown>
#12 0x55a0e6e910d5 <unknown>
#13 0x55a0e6ea9145 <unknown>
#14 0x55a0e6e84aaf <unknown>
#15 0x55a0e6ec5ba8 <unknown>
#16 0x55a0e6ec5d28 <unknown>
#17 0x55a0e6ee048d <unknown>
#18 0x7f24799a144b <unknown>
```

Since the script is running on a server, you will not be able to see any Captcha. However, an example Captcha is shown below to show what would happen if using a real web browser with Chromium driver and Selenium.



- Captcha is typically good enough to break most automation scripts. In the case of this example, the script didn't wait for the Captcha to be filled out and broke. However, it would be simple to change the script so that it waits in the case of the website not returning results within a certain time period. It's also possible to hire people that would complete the Captchas on behalf of the script, so in this instance it might be better to Block than to Captcha.

How could the Automation condition be configured to block instead of Captcha?

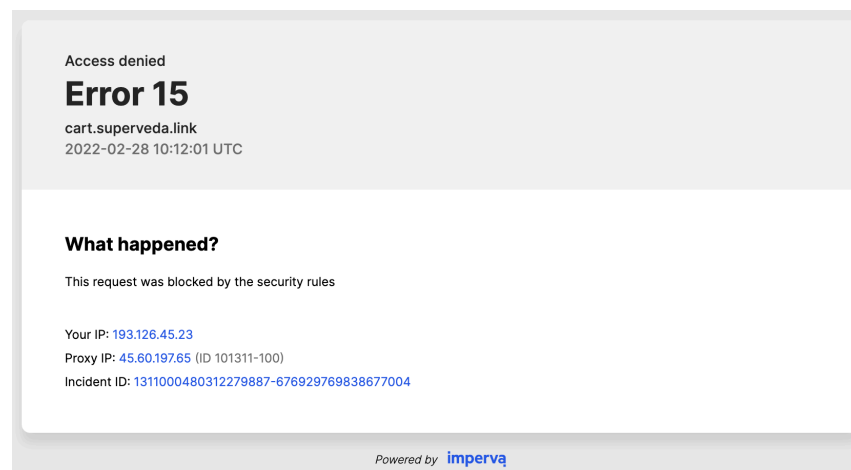
The condition needs to be moved to the block directive.

- Try out blocking by moving the **Automation** condition to the **block** directive using the **Move** button. Be sure to make sure the condition is **Active**. Finally, **Publish** the new configuration by using the button at the top right of the screen. Re-run the scraping script to see what happens.

What happens this time?

The script will break, but for different reasons. The backend website will display an Access denied page as shown below.

Since the script is running on a server, you will not be able to see the block page – just that the script breaks. However, an example Captcha is shown below to show what would happen if using a real web browser with Chromium driver and Selenium.



3. API Security – Configuration and Testing

In this exercise, API will be used instead of screen scraping to extract similar information to that which the Selenium scripts were able to extract, and which was prevented with Advanced Bot Protection configuration. It is an example showing that even the smallest flaw in a website design may be exploited to get results that may not have been considered when writing the application.

To begin with, the problem needs to be reviewed.

1. Open up the attack website, open the following URL:

`http://gentraffic.impvdemo.com/`

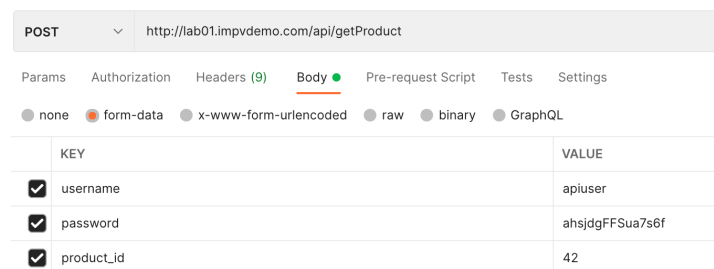
2. Click the button marked **API Security**. It will move to a new screen asking for details of the API call to be filled out. First type the **API URL** into the text box as shown below.

`http://labxx.impvdemo.com/api/getProduct`

3. We will be calling an API called **getProduct**, which requires three parameters – username, password and product_id. When the **Submit** button is pressed, parameters that were checked will be passed into the API call. The website is using the following cURL statement to make the call where the parameters are passed into the body of the call.

```
curl -d 'username=apiuser&password=ahsjdgFFSua7s6f&product_id=42'
http://labxx.impvdemo.com/api/getProduct
```

An example call using Postman is shown below, which may make things clearer. Note that, as with cURL, parameters are always passed in the Body of the API call.



The screenshot shows the Postman interface for a POST request. The URL is `http://lab01.impvdemo.com/api/getProduct`. The 'Body' tab is selected, and the 'form-data' radio button is chosen. A table lists the parameters to be sent in the request body.

KEY	VALUE
<input checked="" type="checkbox"/> username	apiuser
<input checked="" type="checkbox"/> password	ahsjdgFFSua7s6f
<input checked="" type="checkbox"/> product_id	42

4. The **username** and **password** authenticate the caller of the API, and typically would be public knowledge for an API such as this. In the attack website, ticking the box for **username** and **password** will send the appropriate credentials – no need to fill those out. The **product_id** is the database index of the product, and returns values related to it. If the box is checked on the attack website, it will send **product_id** 42. If unchecked it does not include the parameter at all. To begin with ensure all three boxes are checked and make sure the **API URL** is correct filled out. Press **Submit** to run the attack.

The API should be called and return the values of the product_id 42 from the website, as below. Scroll down the results to make sure only product_id 42 results are returned. This is the use of the API as intended.

```
{
  "success": 1,
  "data": [
    {
      "product_id": "42",
      "model": "Product 15",
      "sku": "",
      "upc": "",
      "ean": "",
      "jan": "",
      "isbn": "",
      "mpn": "",
      "location": "",
      "quantity": "990",
      "stock_status_id": "5",
      "image": "catalog/demo/apple_cinema_30.jpg",
      "manufacturer_id": "8",
      "shipping": "1",
      "price": "100.0000",
      "points": "400",
      "tax_class_id": "9",
      "date available": "2009-02-04".
    }
  ]
}
```

5. The call may also be run directly from a command line or website such as <https://reqbin.com/curl> using the cURL call as shown below. Optionally, try to make the API call directly using the cURL command below. The returned results should be the same as the website.

```
curl -d 'username=apiuser&password=ahsjdgFFSua7s6f&product_id=42'
http://labxx.impvdemo.com/api/getProduct
```

6. Send the call again, but this time remove the checkbox related to **product_id** as shown below or remove the &product_id=42 part, if using the cURL command. i.e.

```
curl -d 'username=apiuser&password=ahsjdgFFSua7s6f'  
http://labxx.impvdemo.com/api/getProduct
```

What happens this time?	The call still executes, but this time it returns ALL of the data contained in the database. This is similar to the results of a SQL Injection attack.
-------------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------


7. Since the API call with the missing parameter returned all of the data in the database, it is clear that this is another attack vector that needs to be investigated and fixed.

What other ways could this data be obtained theoretically? How could this be stopped?	SQL Injection would be the most obvious attack vector, which is covered by Cloud WAF, and Bot Scraping was covered in the previous exercise.
---------------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------

8. API protection uses an OpenAPI definition file known as a Swagger file as a baseline from which to protect each API call. Ideally, a Swagger file will exist for the application, but in many cases companies have public websites that have API sections, but do not have documentation or a Swagger file available. The API used in this example did not come with a Swagger file. In that case, there are a couple of options available to generate it as below.
- Allow Imperva API Protection to detect API calls passing through, and generate a Swagger file from that definition. Whilst the detection is instant, it can take a few days to capture all of the calls and create the Swagger file this way. However, this is definitely the simplest method.
 - Manually create the Swagger file. This will require either defining all API calls written in Postman as has been done in previous steps, creating documentation and converting the documentation to the correct format using a free or paid website.
9. The Swagger file for this website has not yet been uploaded to Imperva, so this is the next step. Instead of using a Swagger file, the API endpoints may be discovered and the security definition take from the discovered endpoints. In MY, the Imperva Cloud Security Console, choose the **Application** tab at the top, followed by **API Security > Settings**. Note that API Discovery has not been enabled for the website. Enable it if you would like to. The discovery works by reviewing the API calls made to the website, and may take a few hours to actually discover something. Reviewing this setting is considered “homework” therefore although the instructor should show the results of a typical discovery.
10. After enabling API Discovery, move to **API Security > Inventory** to review the discovered API calls. Since we are only ever calling one endpoint, API Discovery would only give one endpoint result although there might be GET and POST variations of that call.

11. We will upload a Swagger file, then configure the endpoints. To do that move to **API Security > Inventory** and choose the **My APIs** tab. There is a button to **Add OAS File**. Click the button and upload the file called **swagger.json** that has been supplied by the instructors. Optionally fill out a **Description**, but make sure to apply the file to your lab website (example for lab01.impvdemo.com shown). Check the Base Path comes up as **/api**. There is the chance to set **Violation Actions** to an action other than Alert (Default). For now, leave that part as the default setting and click the **Add File** button to add the Swagger file.

Swagger Details
Apply security configuration on your APIs by uploading your swagger file.


Upload swagger file
JSON, YAML file formats are supported.
[Upload file](#)

swagger.json x

Description (Optional)
IntroductionAllows use of OpenCart via API

Apply to
lab01.impvdemo.com

Base Path
/api

12. Check the Swagger file was added properly by clicking the **/API** endpoint shown on the **APIs Inventory** page. There should be two APIs defined as shown below. We will change policy on the **/getProduct** API.

APIs Inventory

Search .csv Download Swagger Specification Add OAS File

API/Endpoint	Host	Data Labels	Source	Policy Action
^ /api	lab01.impvdemo.com		OAS File	Default
POST /addEditProduct				Default
POST /getProduct				Default

1 - 1 of 1 Rows per page 10 >

13. To change the policy, choose **API Security > Policies** to review the policies settings. Notes that there is a **Site-level Policy** for the website itself, and below that an **API-level Policy** which allows policies to be set per API call. It is here that the changes will be made to fix the issue of missing parameters, although equally this could be done at the **Site-level** as well. Uncheck the check box that says **Show only modified APIs**, which will display the API that was imported.

- There are actions per parameter which may be **Alert (Default)**, **Ignore**, **Alert**, **Block IP**, **Block Request** and **Block User**.

To fix the issue of the website returning all of the data when there is a missing parameter in the call, what should be changed here? Note that when blocking, typically the request should be blocked. Blocking the user or IP will result in the website being unavailable to that user or IP.

Change the policy so that **Missing Required Parameters** will **Block Request**. This should just reset the request and return no data.

- The API call should break, with a display similar to the one shown below. The reason the error looks so nasty is simply because the script isn't handling the error nicely. Repeat the test with all the check boxes checked to make sure you only get data when you ask for Product_ID 42, but not when you miss the parameter completely. Optionally check using the cURL calls to see the actual data returned.

```
requests.exceptions.JSONDecodeError: [Errno Expecting value] html style="height:100%;>head>  
content=<"initial-scale=1.0"><meta http-equiv="X-UA-Compatible" content="IE=edge, chrome=1"></h<br/>nifm(1-3-1413727)>0.920000000000000073281646389554865100929%2B0x28000020-1x20-1x29x200x280020-1x  
nth=POST">frameborder=0 width="100%" height="100%" marginheight="0px" marginwidth="0px">Reque
```

Traceback (most recent call last)

```
File ~/home/ec2-user/.local/lib/python3.7/site-packages/requests/models.py, line 910, in json  
    return complexjson.loads(self.text, **kwargs)  
  
File ~/usr/lib64/python3.7/site-packages/simplejson/_init_.py, line 488, in loads  
    return _default_decoder.decode(s)  
  
File ~/usr/lib64/python3.7/site-packages/simplejson/decoder.py, line 374, in decode  
    obj, end = self._raw_decode(s)  
  
File ~/usr/lib64/python3.7/site-packages/simplejson/decoder.py, line 393, in _raw_decode  
    return self.scan_once(s, idx=_w(s, idx).end())  

```

During handling of the above exception, another exception occurred:

```
File ~/home/ec2-user/.local/lib/python3.7/site-packages/flask/app.py, line 2091, in __call__  
    return self.wsgi_app(environ, start_response)  
  
File ~/home/ec2-user/.local/lib/python3.7/site-packages/flask/app.py, line 2076, in wsgi_app  
    response = self.handle_exception(e)  
  
File ~/home/ec2-user/.local/lib/python3.7/site-packages/flask/app.py, line 2073, in wsgi_app  
    response = self.full_dispatch_request()  
  
File ~/home/ec2-user/.local/lib/python3.7/site-packages/flask/app.py, line 1518, in full_dispatch_request  
    rv = self.handle_user_exception(e)
```

4. Log Review

The final step is to review the event logs to understand the attacks that were tried and how Imperva reports on them.

1. To review the event logs, choose **Security Events** from the left-hand side of the menu. Events similar to those shown below should be reviewed. Click on the **More Details** button and the down arrow to the left of each alert to see the details.

Advanced Bot Protection - CAPTCHA (Fail)

Client App	Chrome	Time	28 Feb 2022, 09:51:35	Hits	83
Entry Page	cart.superveda.link/	Session ID	1311000480311717848	Page Views	5
Method	GET	Country	Portugal	HTTP	1.1
User Agent	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) ...	Source IP	193.126.45.23	Cookies	Supported

CAPTCHA (Fail) Advanced Bot Protection

More Details ^

Request ID	Type	URL	Method	Action
675940424122043404	Advanced Bot Protection CAPTCHA	http://cart.superveda.link/index.php?route=produ...	GET	Suspended

Advanced Bot Protection - CAPTCHA (Pass)

Client App	Chrome	Time	28 Feb 2022, 09:48:57	Hits	115
Entry Page	cart.superveda.link/abusion-borne-Mur-secrew...	Session ID	1311000480311645819	Page Views	7
Method	POST	Country	Portugal	HTTP	1.1
User Agent	Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:...	Source IP	193.126.45.23	Cookies	Supported

CAPTCHA (Pass) Advanced Bot Protection

More Details ^

Request ID	Type	URL	Method	Action
186803344858090502	Advanced Bot Protection CAPTCHA	http://cart.superveda.link/index.php?route=produ...	GET	Suspended

Advanced Bot Protection – Block Rule

Client App	Chrome	Time	28 Feb 2022, 10:11:54	Hits	74
Entry Page	cart.superveda.link/	Session ID	1311000480312279887	Page Views	5
Method	GET	Country	Portugal	HTTP	1.1
User Agent	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) ...	Source IP	193.126.45.23	Cookies	Supported

Advanced Bot Protection

More Details ^

Request ID	Type	URL	Method	Action
354744700799749129	Advanced Bot Protection	http://cart.superveda.link/index.php?route=produ...	GET	Blocked

API Specification Violation (Missing Parameter)

Client Type	Hacking Tool	Time	28 Feb 2022, 11:25:32	Hits	7	<div></div>
Client App	Postman	Session ID	1311000480314494715	HTTP	1.1	
Entry Page	cart.superveda.link/api/getProduct	Country	Portugal	Cookies	Supported	
Method	POST	Source IP	193.126.45.23			
User Agent	PostmanRuntime/7.28.4					
API Specification Violation (1)						More Details ^

	Request ID	Type	URL	Method	Action
^	571741631303060491	API Specification Violation	http://cart.superveda.link/api/getProduct	POST	Alerted

Request Details	
Incident ID	1311000480314494715-571741631303060491
Think Time	74
Response Time	144
HTTP Status Code	200
Endpoint ID	183823
Post Data	username=XXXXXXX&password=XXXXXXXXXXXXXXX
API Specification Violation(Alert raised)	
Violation Type	MISSING_PARAM
Parameter Name	product_id