**Mass-Assignment:**

**A Common Web Security Vulnerability**

Mass-assignment is a convenient programming technique that allows developers to assign multiple values or data fields to an object or database record in one go, rather than individually setting each field's value. This is often used when dealing with user input from forms or other sources.

For example, Django, a popular Python web framework, provides a similar mechanism for mass-assignment using its ModelForms. ModelForms are a subclass of forms that automatically generate a form field for each model field. They also provide methods to create or update an object based on the form data.

# Create a new object with data from a form

form = MyModelForm(request.POST)

if form.is\_valid():

instance = form.save()

# Update an existing object with form data

instance = MyModel.objects.get(id=my\_id)

form = MyModelForm(request.POST, instance=instance)

if form.is\_valid():

instance = form.save()

However, this functionality also comes with a security risk. If the developer does not specify a whitelist of fields that can be assigned by the user, an attacker may exploit this vulnerability to assign data to protected fields that are not intended to be exposed or modified by the user. This can lead to data leakage, data corruption, or unauthorized access.

For instance, suppose that the MyModel class has a field called is\_admin that determines whether the user has administrative privileges. If the developer does not exclude this field from the ModelForm, an attacker can submit a form with is\_admin=True and gain administrative access to the application.

To prevent this, Django provides a way to specify which fields are included or excluded from the ModelForm using the fields or exclude attributes. For example:

class MyModelForm(forms.ModelForm):

class Meta:

model = MyModel

exclude = ['is\_admin']

This way, only the fields that are not in the exclude list will be processed by the ModelForm.

Mass assignment is not a problem specific to Django or Python. It is a common web security vulnerability that affects many web frameworks and languages. Therefore, it is important for developers to be aware of this issue and take appropriate measures to protect their applications from mass assignment attacks.

**How to Detect and Exploit Mass Assignment Vulnerabilities**

One of the most common places to discover and exploit mass assignment vulnerabilities is in API requests that accept and process client input. Account registration, profile editing, user management, and client management are all common functions that allow clients to submit input using the API.

To detect mass assignment vulnerabilities, an attacker can try to submit additional parameters or fields that are not expected by the application logic, and observe how the application responds. For example, if the application has an API endpoint for creating a new user account, an attacker can try to submit a parameter called role with a value of admin and see if the application creates a new user with administrative privileges.

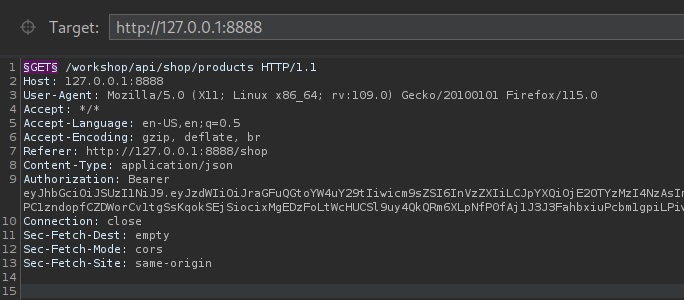
To exploit mass assignment vulnerabilities, an attacker can try to modify or inject data into sensitive fields that are not intended to be accessible or editable by the user. For example, if the application has an API endpoint for updating a user profile, an attacker can try to submit a parameter called password with a new value and see if the application changes the user's password without requiring the old password.

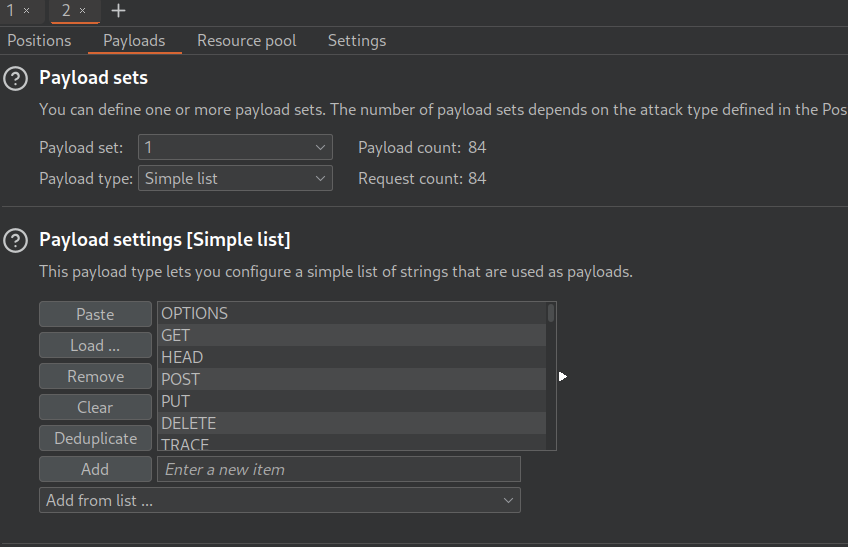
To prevent mass assignment vulnerabilities, developers should always use whitelists or blacklists to control which fields can be assigned by the user input. They should also validate and sanitize the user input before assigning it to the object or database record. Additionally, they should implement proper authentication and authorization mechanisms to restrict access to sensitive functions and data.

Armed with our cutting-edge mass assignment attack techniques, let's delve into testing crAPI. We'll begin by scrutinizing which requests accept input from clients and strategize how to exploit a potentially vulnerable variable to compromise the API.

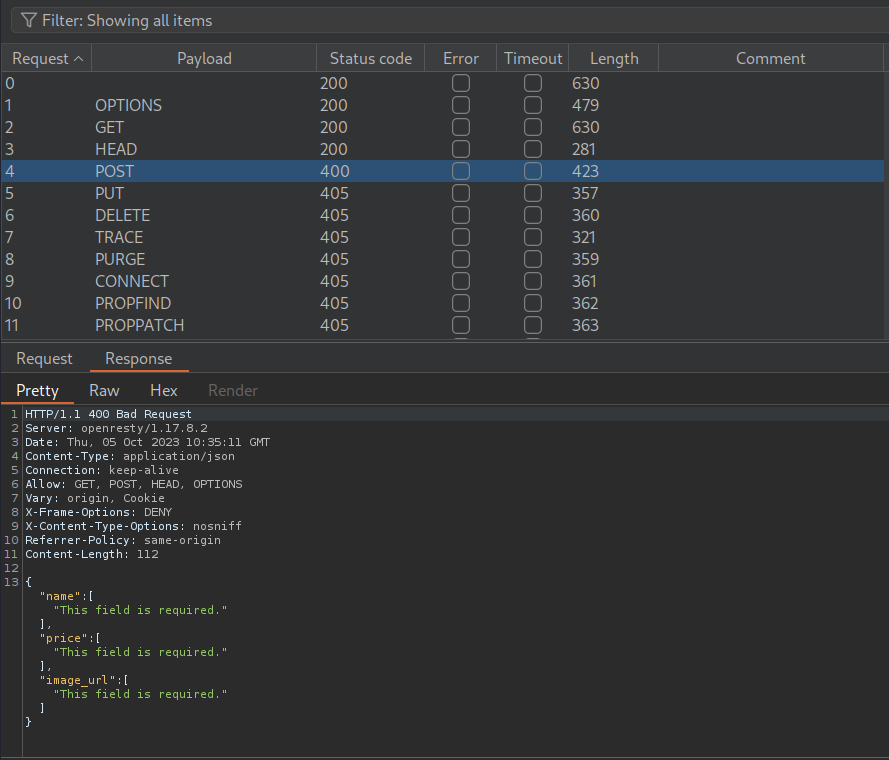
Our journey leads us to the GET request directed towards the /workshop/api/shop/products endpoint. This endpoint plays a crucial role in filling crAPI's storefront with various products. As we dissect the request using Repeater, we discover a JSON variable named "credit." This variable piques our interest as it could be a promising candidate for manipulation.

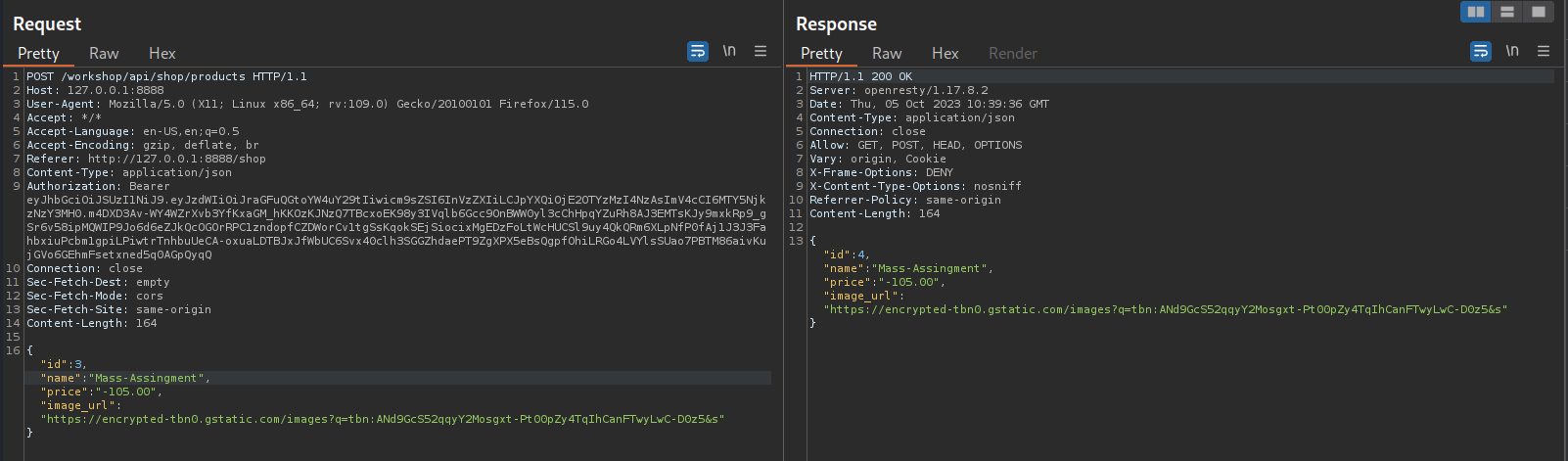
Although this GET request grants us access to the "credit" variable, it's important to note that we cannot directly modify its value through a GET request. To explore alternative avenues, we decide to initiate an Intruder scan. We conveniently send the request to Intruder by right-clicking it in Repeater. Within Intruder, we configure the attack position to target the request method itself





The subsequent response reveals that certain mandatory fields were omitted in the POST request. The intriguing part is that the API provides us with clear insights into the required parameters. Upon careful consideration, it becomes apparent that this request is intended for use by crAPI administrators to update the store. However, the absence of strict access restrictions means that we can exploit this opportunity to not only create a new item within the store but also simultaneously manipulate our credit to our advantage.

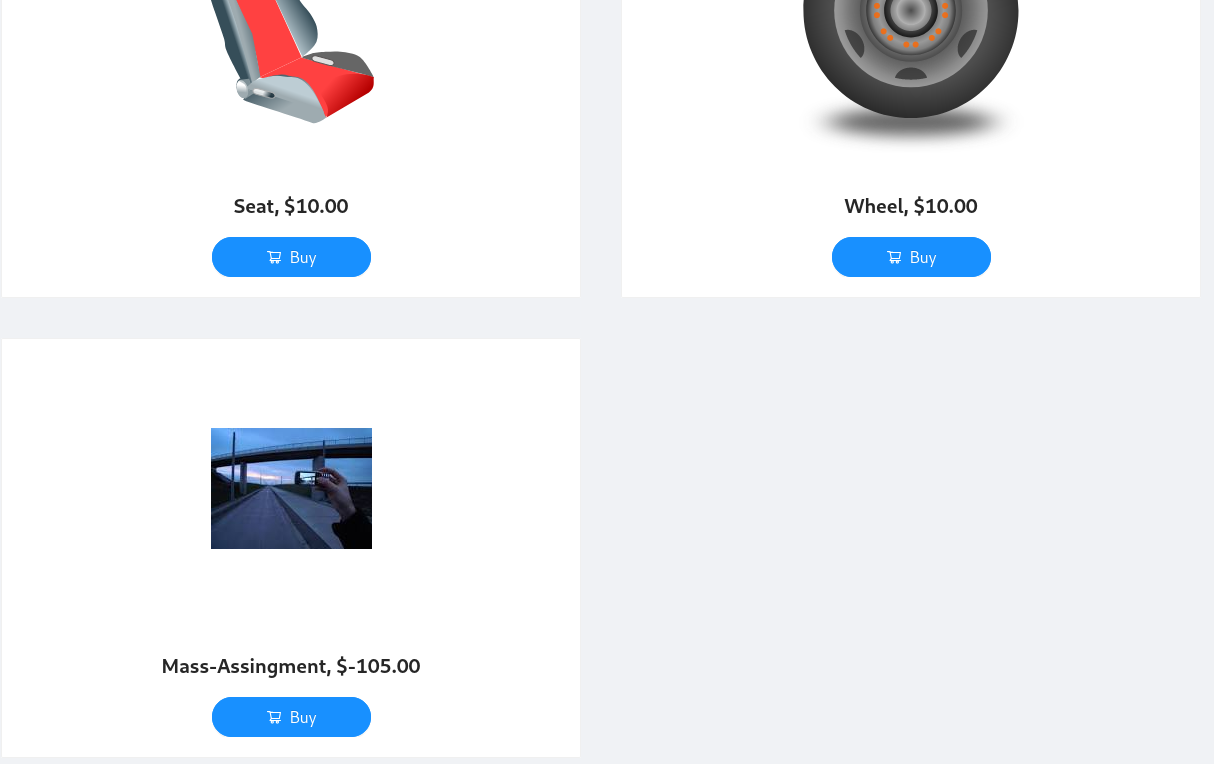




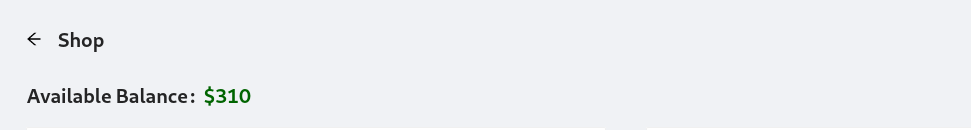
With this request, we achieve a favorable outcome marked by an HTTP 200 OK response! A quick visit to the crAPI store through a web browser confirms our success in creating a new item in the store, complete with a revised price of 105. However, there's a slight disappointment as our credit remains unaffected. Nevertheless, when we proceed to make a purchase of this newly added item, we observe that the system correctly deducts the corresponding amount from our credit, adhering to the standard store transaction process.

Now, let's don our adversarial mindset and delve into the intricacies of this business logic. In the role of crAPI consumers, we shouldn't have the privilege to add products to the store or alter their prices. But, intriguingly, we can. This raises questions about the assumptions made by the developers while designing the API. If they presumed that only trustworthy users would have access to add products to the crAPI store, it presents an opportunity for exploitation.

One intriguing possibility emerges – we could potentially grant ourselves an extraordinary discount on a product. Perhaps a deal so enticing that the price becomes a negative number, posing an unusual twist to the conventional pricing structure.



By purchasing (twice) this special deal, we add an extra $310 to our available balance.



As you can see, our mass assignment exploit would have severe consequences for any business with this vulnerability.

https://learn.snyk.io/lesson/mass-assignment/