CRUD API

We saw examples of a `City Search` web application that uses PHP parameters to search for a city name in the previous sections. This section will look at how such a web application may utilize APIs to perform the same thing, and we will directly interact with the API endpoint.

APIs

There are several types of APIs. Many APIs are used to interact with a database, such that we would be able to specify the requested table and the requested row within our API query, and then use an HTTP method to perform the operation needed. For example, for the `api.php` endpoint in our example, if we wanted to update the `city` table in the database, and the row we will be updating has a city name of `london`, then the URL would look something like this:

Code: bash

```
curl -X PUT http://<SERVER_IP>:<PORT>/api.php/city/london ...SNIP...
```

CRUD

As we can see, we can easily specify the table and the row we want to perform an operation on through such APIs. Then we may utilize different HTTP methods to perform different operations on that row. In general, APIs perform 4 main operations on the requested database entity:

Operation	HTTP Method	Description
`Create`	`POST`	Adds the specified data to the database table
`Read`	`GET`	Reads the specified entity from the database table
`Update`	`PUT`	Updates the data of the specified database table
`Delete`	`DELETE`	Removes the specified row from the database table

These four operations are mainly linked to the commonly known CRUD APIs, but the same principle is also used in REST APIs and several other types of APIs. Of course, not all APIs work in the same way, and the user access control will limit what actions we can perform and what results we can see. The Introduction to Web Applications module further explains these concepts, so you may refer to it for more details about APIs and their usage.

Read

The first thing we will do when interacting with an API is reading data. As mentioned earlier, we can simply specify the table name after the API (e.g. `/city`) and then specify our search term (e.g. `/london`), as follows:

CRUD API

```
MisaelMacias@htb[/htb]$ curl http://<SERVER_IP>:<PORT>/api.php/city/london

[{"city_name":"London","country_name":"(UK)"}]
```

We see that the result is sent as a JSON string. To have it properly formatted in JSON format, we can pipe the output to the `jq` utility, which will format it properly. We will also silent any unneeded cURL output with `-s`, as follows:

CRUD API

```
MisaelMacias@htb[/htb]$ curl -s http://<SERVER_IP>:<PORT>/api.php/city/london | jq

[
  {
    "city_name": "London",
    "country_name": "(UK)"
  }
]
```

As we can see, we got the output in a nicely formatted output. We can also provide a search term and get all matching results:

CRUD API

```
MisaelMacias@htb[/htb]$ curl -s http://<SERVER_IP>:<PORT>/api.php/city/le | jq

[
  {
    "city_name": "Leeds",
    "country_name": "(UK)"
  },
  {
    "city_name": "Dudley",
    "country_name": "(UK)"
  },
  {
    "city_name": "Leicester",
    "country_name": "(UK)"
  },
  ...SNIP...
]
```

Finally, we can pass an empty string to retrieve all entries in the table:

CRUD API

```
MisaelMacias@htb[/htb]$ curl -s http://<SERVER_IP>:<PORT>/api.php/city/ | jq
```

Cheat Sheet

Go to Questions

Table of Contents

HTTP Fundamentals

HyperText Transfer Protocol (HTTP)

Hypertext Transfer Protocol Secure (HTTPS)

HTTP Requests and Responses

HTTP Headers

HTTP Methods

HTTP Methods and Codes

GET

POST

CRUD API

```
[
  {
    "city_name": "London",
    "country_name": "(UK)"
  },
  {
    "city_name": "Birmingham",
    "country_name": "(UK)"
  },
  {
    "city_name": "Leeds",
    "country_name": "(UK)"
  },
  ...SNIP...
]
```

Try visiting any of the above links using your browser, to see how the result is rendered.

Create

To add a new entry, we can use an HTTP POST request, which is quite similar to what we have performed in the previous section. We can simply POST our JSON data, and it will be added to the table. As this API is using JSON data, we will also set the `Content-Type` header to JSON, as follows:

CRUD API

```
MisaelMacias@htb[/htb]$ curl -X POST http://<SERVER_IP>:<PORT>/api.php/city/ -d '{"city_name":"HTB_City", "country_nam
```

Now, we can read the content of the city we added (`HTB_City`), to see if it was successfully added:

CRUD API

```
MisaelMacias@htb[/htb]$ curl -s http://<SERVER_IP>:<PORT>/api.php/city/HTB_City | jq

[
  {
    "city_name": "HTB_City",
    "country_name": "HTB"
  }
]
```

As we can see, a new city was created, which did not exist before.

Exercise: Try adding a new city through the browser devtools, by using one of the Fetch POST requests you used in the previous section.

Update

Now that we know how to read and write entries through APIs, let's start discussing two other HTTP methods we have not used so far: `PUT` and `DELETE`. As mentioned at the beginning of the section, `PUT` is used to update API entries and modify their details, while `DELETE` is used to remove a specific entity.

Note: The HTTP `PATCH` method may also be used to update API entries instead of `PUT`. To be precise, `PATCH` is used to partially update an entry (only modify some of its data "e.g. only city_name"), while `PUT` is used to update the entire entry. We may also use the HTTP `OPTIONS` method to see which of the two is accepted by the server, and then use the appropriate method accordingly. In this section, we will be focusing on the `PUT` method, though their usage is quite similar.

Using `PUT` is quite similar to `POST` in this case, with the only difference being that we have to specify the name of the entity we want to edit in the URL, otherwise the API will not know which entity to edit. So, all we have to do is specify the `city` name in the URL, change the request method to `PUT`, and provide the JSON data like we did with POST, as follows:

CRUD API

```
MisaelMacias@htb[/htb]$ curl -X PUT http://<SERVER_IP>:<PORT>/api.php/city/london -d '{"city_name":"New_HTB_City", "co
```

We see in the example above that we first specified `/city/london` as our city, and passed a JSON string that contained `"city_name":"New_HTB_City"` in the request data. So, the london city should no longer exist, and a new city with the name `New_HTB_City` should exist. Let's try reading both to confirm:

CRUD API

```
MisaelMacias@htb[/htb]$ curl -s http://<SERVER_IP>:<PORT>/api.php/city/london | jq
```

CRUD API

```
MisaelMacias@htb[/htb]$ curl -s http://<SERVER_IP>:<PORT>/api.php/city/New_HTB_City | jq

[
  {
    "city_name": "New_HTB_City",
    "country_name": "HTB"
  }
]
```

Indeed, we successfully replaced the old city name with the new city.

Note: In some APIs, the `Update` operation may be used to create new entries as well. Basically, we would send our data, and if it does not exist, it would create it. For example, in the above example, even if an entry with a `london` city did not exist, it would create a new entry with the details we passed. In our example, however, this is not the case. Try to update a non-existing city and see what you would get.

DELETE

Finally, let's try to delete a city, which is as easy as reading a city. We simply specify the city name for the API and use the HTTP `DELETE` method, and it would delete the entry, as follows:

CRUD API

```
MisaelMacias@htb[/htb]$ curl -X DELETE http://<SERVER_IP>:<PORT>/api.php/city/New_HTB_City
```

CRUD API

```
MisaelMacias@htb[/htb]$ curl -s http://<SERVER_IP>:<PORT>/api.php/city/New_HTB_City | jq
[]
```

As we can see, after we deleted `New_HTB_City`, we get an empty array when we try reading it, meaning it no longer exists.

Exercise: Try to delete any of the cities you added earlier through POST requests, and then read all entries to confirm that they were successfully deleted.

With this, we are able to perform all 4 `CRUD` operations through cURL. In a real web application, such actions may not be allowed for all users, or it would be considered a vulnerability if anyone can modify or delete any entry. Each user would have certain privileges on what they can `read` or `write`, where `write` refers to adding, modifying, or deleting data. To authenticate our user to use the API, we would need to pass a cookie or an authorization header (e.g. JWT), as we did in an earlier section. Other than that, the operations are similar to what we practiced in this section.

Connect to Pwnbox
Your own web-based Parrot Linux instance to play our labs.

Pwnbox Location

UK 139ms

Terminate Pwnbox to switch location

Start Instance

∞ / 1 spawns left

Waiting to start...

Enable step-by-step solutions for all questions

Questions

Answer the question(s) below to complete this Section and earn cubes!

Cheat Sheet

Target(s): Click here to spawn the target system!

+2 First, try to update any city's name to be 'flag'. Then, delete any city. Once done, search for a city named 'flag' to get the flag.

HTB{crud_4p!_m4n!pul4t0r}

Submit Hint

Previous Finish