Securing Web Applications with FFP (FAUST, flagbot, polygl0ts)

Introduction to Python 2/2

Matteo Rizzo [3]

In collaboration with [1]FAUST (ctf@fsi.informatik.uni-erlangen.de), [2]flagbot (ctf@vis.ethz.ch) and [3]polygl0ts (ctf@epfl.ch)

October 9, 2021

Table of Contents

HTTP Basics

What is HTTP?

- HTTP is the protocol used to transmit data on the web.
- HTTP is how web browsers communicate with web servers.
- It's a fairly simple text-based protocol.
- A client makes a HTTP request to a server, and the server responds with some data.

Why you should know about HTTP

- This is a *web* workshop.
  - You will interact with *web* services a lot.
  - The only way to interact with these services is to use HTTP.

Why you should know about HTTP

- This is a *web* workshop.
    - You will interact with *web* services a lot.
    - The only way to interact with these services is to use HTTP.
- If you use a web browser, it will make HTTP request automatically for you.
- Sometimes it's useful to make these requests from Python instead.
- In order to do that, you should know at least the basics of how HTTP works.

Simple Example

```
curl -v http://ffp-ctf.0x4141.net/challenges
> GET /challenges HTTP/1.1
> Host: ffp-ctf.0x4141.net
> User-Agent: curl/7.64.1
> Accept: */*

< HTTP/1.1 200 OK
< Server: gunicorn/20.0.4
< Date: Sun, 03 Oct 2021 18:41:27 GMT
< Content-Type: text/html; charset=utf-8
< Content-Length: 4605
< Set-Cookie: session=aaa; HttpOnly; Path=/; SameSite=Lax

<!DOCTYPE html>
...
```

HTTP Method

```
GET /challenges HTTP/1.1
```

- The most important part of the request are the *verb* and the name of the resource that the client is requesting.
  - Here the verb is `GET` and the resource name is `/challenges`.
- The verb tells the server what the client wants. GET usually means that the client wants the server to send it some data (like a webpage).
- The name of the resource tells the server what data the client is asking for.

HTTP Headers

```
Host: ffp-ctf.0x4141.net
User-Agent: curl/7.64.1
Accept: */*
```

- A HTTP request can contain any number of *headers*.
- Headers contain short metadata about the request.
  - For example which browser the client is using.
- Each header has a name and a value. Both the name and the value can contain arbitrary text.

HTTP Response

```
HTTP/1.1 200 OK
Server: gunicorn/20.0.4
Date: Sun, 03 Oct 2021 18:41:27 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 4605
Set-Cookie: session=aaa; HttpOnly; Path=/; SameSite=Lax

<!DOCTYPE html>
...
```

- After the client has finished making a request, the server responds.
- The response contains a status code and the data that the client requested.
    - The status code tells the client if the request was successful.
- Responses can also have headers.

GET Request

```
GET /challenges HTTP/1.1
Host: ffp-ctf.0x4141.net
User-Agent: curl/7.64.1
Accept: */*
```

- GET is the most common type of request.
- Generally used to ask the server to send some data (e.g. a webpage).
- The request itself cannot have a body.
  - I.e., it cannot be used to upload data to a server.

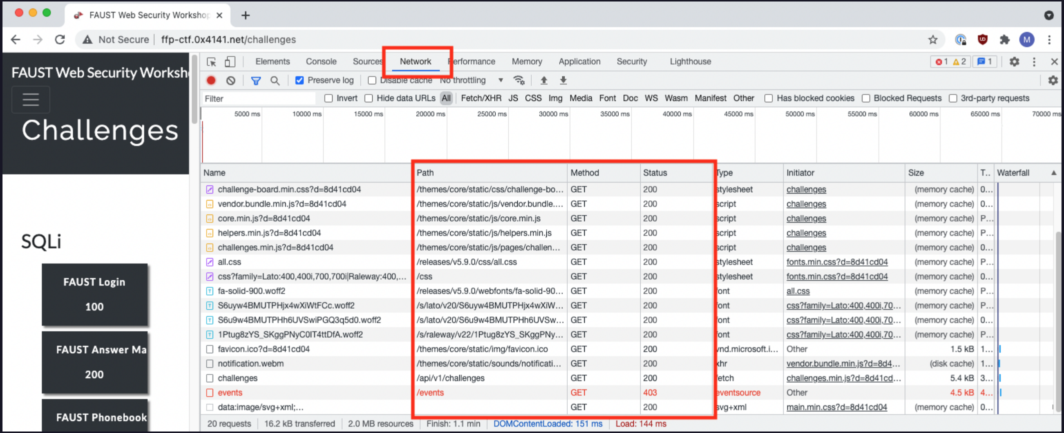
POST Request

```
POST /challenges HTTP/1.1
Host: ffp-ctf.0x4141.net
User-Agent: curl/7.64.1
Accept: */*
Content-Length: 16
Content-Type: application/x-www-form-urlencoded

foo=bar&baz=asdf
```

- POST is another common type of request.
- Generally used to instruct the server to do something or upload data.
  - For example, logging into a website.
- Unlike GET requests, POST requests can have a body containing arbitrary data.

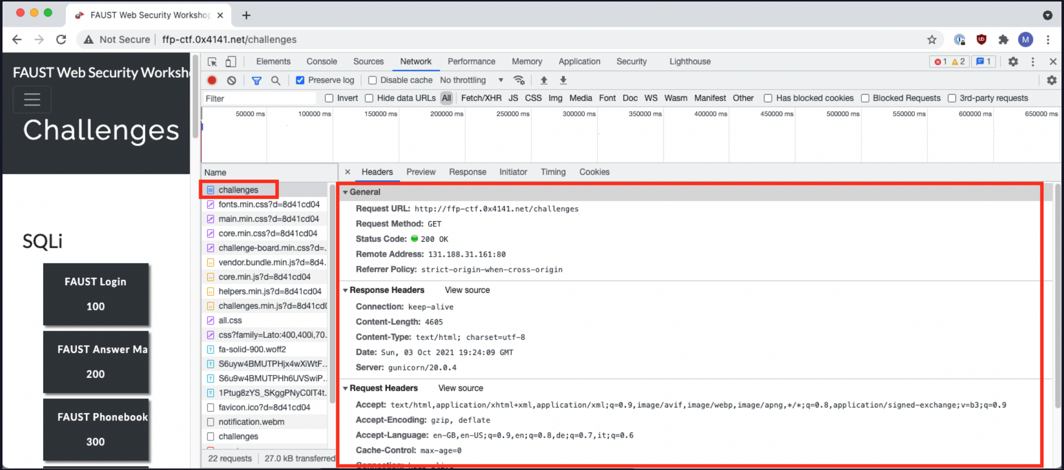
Viewing HTTP Requests in the Browser



Name	Path	Method	Status	Type	Initiator	Size	T..	Waterfall
challenge-board.min.css?d=8d41cd04	/themes/core/static/css/challenge-bo...	GET	200	stylesheet	challenges	(memory cache)	0...	
vendor.bundle.min.js?d=8d41cd04	/themes/core/static/js/vendor.bundle....	GET	200	script	challenges	(memory cache)	0...	
core.min.js?d=8d41cd04	/themes/core/static/js/core.min.js	GET	200	script	challenges	(memory cache)	P...	
helpers.min.js?d=8d41cd04	/themes/core/static/js/helpers.min.js	GET	200	script	challenges	(memory cache)	P...	
challenges.min.js?d=8d41cd04	/themes/core/static/js/pages/challen...	GET	200	script	challenges	(memory cache)	0...	
all.css	/releases/v5.9.0/css/all.css	GET	200	stylesheet	fonts.min.css?d=8d41cd04	(memory cache)	P...	
css?family=Lato:400,400i,700,700i|Raleway:400,...	/css	GET	200	stylesheet	fonts.min.css?d=8d41cd04	(memory cache)	0...	
fa-solid-900.woff2	/releases/v5.9.0/webfonts/fa-solid-90...	GET	200	font	all.css	(memory cache)	P...	
S6uyw4BMUTPHjx4wXiWtFCc.woff2	/s/lato/v20/S6uyw4BMUTPHjx4wXiW...	GET	200	font	css?family=Lato:400,400i,70...	(memory cache)	P...	
S6u9w4BMUTPHh6UVSwiPGQ3q5d0.woff2	/s/lato/v20/S6u9w4BMUTPHh6UVSw...	GET	200	font	css?family=Lato:400,400i,70...	(memory cache)	0...	
1Ptug8zYS_SKggPNyC0IT4ttDfA.woff2	/s/raleway/v22/1Ptug8zYS_SKggPNy...	GET	200	font	css?family=Lato:400,400i,70...	(memory cache)	P...	
favicon.ico?d=8d41cd04	/themes/core/static/img/favicon.ico	GET	200	vnd.microsoft.i...	Other	1.5 kB	1...	
notification.webm	/themes/core/static/sounds/notificati...	GET	200	xhr	vendor.bundle.min.js?d=8d4...	(disk cache)	1...	
challenges	/api/v1/challenges	GET	200	fetch	challenges.min.js?d=8d41cd...	5.4 kB	3...	
events	/events	GET	403	eventsource	Other	4.5 kB	4...	
data:image/svg+xml;...		GET	200	svg+xml	main.min.css?d=8d41cd04	(memory cache)	0...	

20 requests | 16.2 kB transferred | 2.0 MB resources | Finish: 1.1 min | DOMContentLoaded: 151 ms | Load: 144 ms

Viewing HTTP Requests in the Browser


FAUST Web Security Workshop
Not Secure | ffp-ctf.0x4141.net/challenges
Elements Console Sources Network Performance Memory Application Security Lighthouse
Preserve log
Disable cache
No throttling
Filter
Invert
Hide data URLs
All Fetch/XHR JS CSS Img Media Font Doc WS Wasm Manifest Other
Has blocked cookies
Blocked Requests
3rd-party requests
FAUST Web Security Worksh
Challenges
SQLi
FAUST Login
100
FAUST Answer Ma
200
FAUST Phonebook
300
Headers Preview Response Initiator Timing Cookies

Name
- challenges
- fonts.min.css?d=8d41cd04
- main.min.css?d=8d41cd04
- core.min.css?d=8d41cd04
- challenge-board.min.css?d=.
- vendor.bundle.min.js?d=8d4.
- core.min.js?d=8d41cd04
- helpers.min.js?d=8d41cd04
- challenges.min.js?d=8d41cd(
- all.css
- css?family=Lato:400,400i,70.
- fa-solid-900.woff2
- S6uyw4BMUTPHjx4wXiWtF..
- S6u9w4BMUTPHh6UVSwiP..
- 1Ptug8zYS_SKggPNyC0IT4t.
- favicon.ico?d=8d41cd04
- notification.webm
- challenges

22 requests | 27.0 kB transferred

General

- **Request URL:** `http://ffp-ctf.0x4141.net/challenges`
- **Request Method:** `GET`
- **Status Code:** ● `200 OK`
- **Remote Address:** `131.188.31.161:80`
- **Referrer Policy:** `strict-origin-when-cross-origin`

Response Headers View source

- **Connection:** `keep-alive`
- **Content-Length:** `4605`
- **Content-Type:** `text/html; charset=utf-8`
- **Date:** `Sun, 03 Oct 2021 19:24:09 GMT`
- **Server:** `gunicorn/20.0.4`

Request Headers View source

- **Accept:** `text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9`
- **Accept-Encoding:** `gzip, deflate`
- **Accept-Language:** `en-GB,en-US;q=0.9,en;q=0.8,de;q=0.7,it;q=0.6`
- **Cache-Control:** `max-age=0`

Basic HTTP Requests With Python

`urllib` vs. Requests

- Python's standard library includes a module (`urllib`) to make HTTP requests.
- Unfortunately the interface is a bit low level and not too easy to use...
- The Requests library provides a higher level and easier to use API.
- Install it with `$ python3 -m pip install requests`.

Making a Simple GET Request

```python
import requests

r = requests.get('https://httpbin.org/get/')
# r.text are the contents of the response as a string
print(r.text)
# {
#   "args": {},
#   "headers": {
#     "Accept": "*/*",
#     "Accept-Encoding": "gzip, deflate",
#     "Host": "httpbin.org",
#     "User-Agent": "python-requests/2.26.0",
#   },
#   "origin": "<your ip address>",
#   "url": "https://httpbin.org/get"
# }
```

Making Other Types of Requests

- All HTTP verbs are supported. However for this workshop you will only need GET and POST.

```
requests.post('http://ffp-ctf.0x4141.net/')
requests.head('http://ffp-ctf.0x4141.net/')
# ...
```

Passing Data to a Server

Passing Parameters in the Query String

- Clients can provide parameters to the web server in the URL. Parameters are key/value pairs that appear at the end of the requested URL.
- Parameters usually tell the web server to do something for us.
- Example: `https://en.wikipedia.org/w/index.php?search=CTF`
  - This searches for a page named "CTF" on English Wikipedia.
  - There is one parameter with key `search` and value `CTF`.
- Parameters are separated with the "&" symbol in the URL.

Passing Parameters in the Query String

- Clients can provide parameters to the web server in the URL. Parameters are key/value pairs that appear at the end of the requested URL.
- Parameters usually tell the web server to do something for us.
- Example: `https://en.wikipedia.org/w/index.php?search=CTF`
  - This searches for a page named "CTF" on English Wikipedia.
  - There is one parameter with key `search` and value `CTF`.
- Parameters are separated with the "&" symbol in the URL.
- In Python:

```python
r = requests.get('https://en.wikipedia.org/w/index.php',
    params={'search': 'CTF'})
print(r.url)
# https://en.wikipedia.org/w/index.php?search=CTF
```

Passing Parameters in the Query String

- "Why can't I simply use string formatting instead?"

```
requests.get(f'http://foo/bar?search={myquery}')
```

Passing Parameters in the Query String

- "Why can't I simply use string formatting instead?"

```
requests.get(f'http://foo/bar?search={myquery}')
```

- The answer is that some characters (e.g. space) are invalid in URLs and need to be encoded specially.
- Using `params={...}` takes care of the encoding for you.

Sending Data in POST Requests

- POST requests are the most common way to send data to a server.
  - Used to e.g., upload files, send username/password when logging in.
- The body of the request can technically contain any data. However most of the time the server expects key/value pairs like in the query string

Sending Data in POST Requests

- POST requests are the most common way to send data to a server.
    - Used to e.g., upload files, send username/password when logging in.
- The body of the request can technically contain any data. However most of the time the server expects key/value pairs like in the query string
- In Python:

```python
r = requests.post('https://httpbin.org/post',
    data={'key': 'value'})

print(r.text)
# {
#  "form": {
#    "key": "value"
#  },
#  ...
# }
```

Sending and Accessing Cookies

HTTP Cookies

- A *cookie* is a small piece of data sent by a HTTP server in a response.
- When a web browser receives a cookie from a server, it stores the cookie and sends it back to that server in every subsequent request.
- Often used for authentication or storing user preferences.

HTTP Cookies

- A *cookie* is a small piece of data sent by a HTTP server in a response.
- When a web browser receives a cookie from a server, it stores the cookie and sends it back to that server in every subsequent request.
- Often used for authentication or storing user preferences.

```
HTTP/1.1 200 OK
...
Set-Cookie: session=aaa; HttpOnly; Path=/; SameSite=Lax
```

Sending Cookies in a Request

- Use the `cookies` parameter. Similar to passing parameters in the query string or in a POST request.

```python
r = requests.get('https://httpbin.org/cookies',
    cookies={'cookie_name': 'cookie_value'})
print(r.text)
# {
#   "cookies": {
#     "cookie_name": "cookie_value"
#   }
# }
```

Reading Cookies from a Response

- The response object has a `.cookies` property (a dictionary) which containing the values of all cookies sent by the server.

```
r = requests.get('https://httpbin.org/cookies/set/mycookie/myvalue',
    allow_redirects=False)

print(r.cookies['mycookie'])
# myvalue
```

HTTP Basic Authentication

HTTP Basic Authentication

- HTTP has built-in support for authenticating users.
- The client sends its credentials (username/password) to the server in a special `Authorization` header.
- If the server doesn't accept the credentials, it replies with a special message (`401 Authorization Required`).

HTTP Basic Authentication

- HTTP has built-in support for authenticating users.
- The client sends its credentials (username/password) to the server in a special `Authorization` header.
- If the server doesn't accept the credentials, it replies with a special message (`401 Authorization Required`).
- **If you get "Authorization Required" responses from the challenge server then you are almost certainly not sending credentials (or sending the wrong credentials).**

HTTP Basic Authentication in Python

- The easiest way is to pass an `HTTPBasicAuth` object as the `auth` parameter when making a request.

```python
import requests
from requests.auth import HTTPBasicAuth

r = requests.get('https://httpbin.org/basic-auth/myuser/mypassword')
print(r.status_code)
# 401, authentication failed because we didn't provide credentials

r = requests.get('https://httpbin.org/basic-auth/myuser/mypassword',
    auth=HTTPBasicAuth('myuser', 'mypassword'))
print(r.status_code)
# 200, we provided the right credentials
```