System and Web Security

SS21
Homework 6

Team members:

Name	Matriculation number
Sharvari Waghmare	3514276
Akshay Patel	3514739

Problem 1: HTTP Headers (4 points)

Use your browser's developer tools (or similar tools) to record the retrieval of this homework sheet from ILIAS. You should only consider the request/response pair of the download itself, not any other steps that lead to the download link.

- Show the transmitted HTTP headers from the request and the response.
- Explain all headers that occur in these messages and briefly explain their contents.
- Which parts of these headers must never leak to an attacker? You must obfuscate such parts (and only such parts) in your submission by replacing them with `XXXXX`.

Request headers :

```
GET /goto_Uni_Stuttgart_file_2521333_download.html HTTP/1.1

Host: ilias3.uni-stuttgart.de
Connection: keep-alive
sec-ch-ua: " Not;A Brand";v="99", "Microsoft Edge";v="91", "Chromium";v="91"
sec-ch-ua-mobile: ?0
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/91.0.4472.101 Safari/537.36 Edg/91.0.864.48
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,
application/signed-exchange;v=b3;q=0.9
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9
```

```
Cookie: ilClientId=XXXXX; PHPSESSID=XXXXX
```

Response headers:

```
HTTP/1.1 200 OK
Server: nginx
Date: Sun, 20 Jun 2021 20:46:07 GMT
Content-Type: application/pdf
Content-Length: 409458
Connection: keep-alive
Set-Cookie: ilClientId=Uni_Stuttgart; path=/; HttpOnly
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
X-ILIAS-FileDelivery-Method: mod_xsendfile
Content-Disposition: inline; filename="sws-homework-06.pdf"
Content-Description: sws-homework-06.pdf
Accept-Ranges: bytes
Last-Modified: Tue, 15 Jun 2021 10:22:12 GMT
ETag: "63f72-5c4cb58e8c131"
Strict-Transport-Security: max-age=31536000
```

- **Connection**: Controls whether the network connection stays open after the current transaction finishes.
- **User-Agent**: Contains a characteristic string that allows the network protocol peers to identify the application type, operating system, software vendor or software version of the requesting software user agent.
- **Accept**: Informs the server about the types of data that can be sent back.
- **sec-ch-ua**: considered 'low-entropy hints', which will be sent by default
- **sec-ch-ua-mobile**: a boolean header, describing whether the client a mobile device, either ?1 (yes) or ?0 (no)
- Upgrade-Insecure-Requests: The HTTP Upgrade-Insecure-Requests request header sends a signal to the server expressing the client’s preference for an encrypted and authenticated response, and that it can successfully handle the upgrade-insecure-requests CSP directive.
- Sec-Fetch-Site: It is a request header that indicates the relationship between a request initiator's origin and its target's origin.
- Sec-Fetch-Mode: It is a request header that indicates the request's mode to a server. It is a Structured Header whose value is a token with possible values cors, navigate, no-cors, same-origin, and websocket.
- Sec-Fetch-User: It is a request header that indicates whether or not a navigation request was triggered by user activation.It is a Structured Header whose value is a boolean so possible values are ?0 for false and ?1 for true.
- Sec-Fetch-Dest: It is a request header that indicates the request's destination to a server. It is a Structured Header whose value is a token

- Accept-Encoding: The encoding algorithm, usually a compression algorithm, that can be used on the resource sent back
- Accept-Language: Informs the server about the human language the server is expected to send back.
- Cookie: Contains stored HTTP cookies previously sent by the server with the Set-Cookie header.
- Server: Contains information about the software used by the origin server to handle the request.
- Content-Type: Indicates the media type of the resource.
- Content-Length: The size of the resource, in decimal number of bytes.
- Set-Cookie: Send cookies from the server to the user-agent.
- Expires: The date/time after which the response is considered stale.
- Cache-Control: Directives for caching mechanisms in both requests and responses.
- Pragma: Implementation-specific header that may have various effects anywhere along the request-response chain. Used for backwards compatibility with HTTP/1.0 caches where the Cache-Control header is not yet present.
- X-ILIAS-FileDelivery-Method: file delivery method from ILIAS
- Content-Disposition: Indicates if the resource transmitted should be displayed inline (default behavior without the header), or if it should be handled like a download and the browser should present a “Save As” dialog.
- Content-Description: description of the content
- Accept-Ranges: Indicates if the server supports range requests, and if so in which unit the range can be expressed.
- Last-Modified: The last modification date of the resource, used to compare several versions of the same resource.
- ETag: A unique string identifying the version of the resource
- Strict-Transport-Security: Force communication using HTTPS instead of HTTP.

Problem 2: Session Management (4 points)

Extend the Python script `session-example.py` from the lecture (see Slide Set 7) as follows:

- Currently, a user who is logged in cannot log out in a convenient way. Add a way for the user to log out such that the user just has to click on a button on the web page.

[1] SHOULD, MUST, and MUST NOT are used as defined in RFC2119.

- For session management, it is a best practice to change the session id whenever a privilege level change of the user's session is performed (for example, when the user logs in or out). Implement this best practice.

Solution:

The script for the server is attached separately.

Problem 3: HTTP Transmission and XSS (4 points)

In the ZIP file of this homework sheet, you find the files `xss-search-1a.py` and `xss-search-1b.py`. Both files contain a web application that allows you to look up user names. As usual, these web applications can be run with Python 3 and then start a web server that is reachable at `http://localhost:8081`.

1. Have a look at the location bar of your browser when you enter the following strings in one of both pages:

   `Aß Bü C+ D& E% F.`

   Which encoding is used and what is the purpose of the encoding?
2. Construct an input for the search field such that both pages produce a different output.
3. Show the relevant part of the HTML source code of both pages after the input has been processed. Explain why there is a difference.
4. Why does one page pose a security risk and the other page does not?

- Which encoding is used and what is the purpose of the encoding? **Utf-8**

  UTF-8 is a compromise character encoding that can be as compact as ASCII (if the file is just plain English text) but can also contain any unicode characters (with some increase in file size).

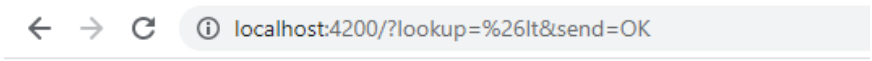
  UTF stands for Unicode Transformation Format. The '8' means it uses 8-bit blocks to represent a character. The number of blocks needed to represent a character varies from 1 to 4.

  One of the really nice features of UTF-8 is that it is compatible with nul-terminated strings. No character will have a nul (0) byte when encoded.

- Construct an input for the search field such that both pages produce a different output

Input string = &lt

File name => **xss-search-1a.py**



SEC Community

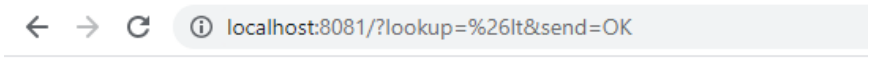
Exiting security vulnerabilities and more!

User Lookup

Please enter a query string:
OK

Search Results for <

File name => **xss-search-1b.py**



SEC Community

Exiting security vulnerabilities and more!

User Lookup

Please enter a query string:
OK

Search Results for &lt

- Show the relevant part of the HTML source code of both pages after the input has been processed. Explain why there is a difference.

xss-search-1b - Notepad

File Edit Format View Help

```
        self.send_header('Content-Type', 'text/html;charset=utf-8')
        self.send_header('X-XSS-Protection', '0') # disables XSS protection in the browser

        self.end_headers()

        result = ''
        if 'lookup' in get_dict:
            lookup = get_dict['lookup'][0]
            result = f'<h3>Search Results for {escape(lookup)}</h3>'
            for user in users:
                if user.lower().find(lookup.lower()) >= 0:
                    result += f'{user}<br>'

        output = main_doc.format(result=result)
        self.wfile.write(bytes(output, 'UTF-8'))

if __name__ == '__main__':
    server = HTTPServer(('', 8081), MyHandler)
    print ("Starting web server on http://localhost:8081/")
    server.serve_forever()
```

xss-search-1a - Notepad

File Edit Format View Help

```
        self.send_header('Content-Type', 'text/html;charset=utf-8')
        self.send_header('X-XSS-Protection', '0') # disables XSS protection in the browser

        self.end_headers()

        result = ''
        if 'lookup' in get_dict:
            lookup = get_dict['lookup'][0]
            result = f'<h3>Search Results for {lookup}</h3>'
            for user in users:
                if user.lower().find(lookup.lower()) >= 0:
                    result += f'{user}<br>'
```

- Both the files have a difference in their html for using Escape() function.
- Escaping in HTML means that you are replacing some special characters with others. In HTML it means usually, you replace e. e.g < or > or " or &. These characters have special meanings in HTML.
- Xss-search-1a file does not use Escape() function whereas Xss-search-1b file does use Escape() function.

- Why does one page pose a security risk and the other page does not?
- Inside a JavaScript context, data is kept safe by JavaScript-escaping.

- An attacker gives your web application JavaScript tags on input. When this input is returned to the user unsanitized, the user's browser will execute it. It can be as simple as crafting a link and persuading a user to click it, or it can be something much more sinister. On page load the script runs and, for example, can be used to post your cookies to the attacker.
- Usually, the workaround is simply converting all html entities , so that `<script>` is returned as `&lt;script&gt;`. The other often employed method of sanitization is using regular expressions to strip away HTML tags using regular expressions on `<` and `>`, but this is dangerous as a lot of browsers will interpret severely broken HTML just fine. Better to convert all characters to their escaped counterparts.
- Converting all characters to their escaped counterparts can be a simple web security solution here.
- Hence, Xss-search-1a file can be proved insecure as it does not use Escape() function whereas Xss-search-1b file does use Escape() function.