HTTP Security Headers in Web APIs

Seminar Project

17 December 2019

Patrick Frischknecht
HTTP security headers

“HTTP response headers that your application can use to increase the security of your application. Once set, these HTTP response headers can restrict modern browsers from running into easily preventable vulnerabilities.”

OWASP Secure Headers Project
Support in browsers

<table>
<thead>
<tr>
<th></th>
<th>IE</th>
<th>Chrome</th>
<th>Safari</th>
<th>Firefox</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strict-Transport-Security</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Content-Security-Policy</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>X-Frame-Options</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>
## Support in HTTP client libraries

<table>
<thead>
<tr>
<th></th>
<th>Glide</th>
<th>OkHttp</th>
<th>Volley</th>
<th>ION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strict-Transport-Security</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>Content-Security-Policy</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>X-Frame-Options</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
</tbody>
</table>
Reasons for lacking support

Browsers

- must be able to access any site
- must provide HTTP (backwards compatibility)
- rich set of potentially harmful features (e.g., JavaScript execution)

API client libraries

- can restrict access to a set of whitelist pages
- can completely prevent HTTP connections
- offer limited features
- are not responsible for most potentially harmful features
Is there more we can do?
What do we want to mitigate?

The most common threats:

- sensitive information leaks
- code injection attacks
Example 1: sensitive information leak

POST /login

{ "username": "user", "password": "pass" }

10:32; Request /login; payload: { "username": "user", "password": "pass" }
Persistence-Allowed header field

Syntax

Persistence-Allowed: any | only-hashed | none

Semantics

any         allows to store content everywhere
only-hashed allows to store content only after hashing
none        does not allow to store content
           (content must reside in memory)
Example 1: sensitive information leak

Using our header

POST /login
Persistence-Allowed: none

{"username": "user",
 "password": "pass"}
Prototype demo
Prototype flow diagram

Request

Persistence-Allowed

None

any

taint

tainted?

yes

no

refuse to log

log

Persistenced-Allowed middleware

logging middleware
Interpretation-Allowed header field

Syntax

Interpretation-Allowed: any | language1,language2 | none

Semantics

any content can be interpreted/executed in any language
language1,... content can be interpreted in the provided languages
none content must not be interpreted
Example 2: Code injection

Request
apple

Server-side code
db.query("SELECT * FROM articles WHERE name='" + response.body + '"")

Result
An error is raised, since there is a potential SQL injection vulnerability

Example attack value: 10'; DROP TABLE 'articles
Future work

- Implementation of framework or language runtime that supports both headers
- Evaluation on open-source and commercial projects
Conclusion

1. HTTP security headers are poorly supported by web API clients

2. Code injection attacks and data leaks are major threats

3. Proposed HTTP headers
   a. Persistence-Allowed mitigates data leaks
   b. Interpretation-Allowed mitigates code injection attacks