Web Security Audit Report

List of issues discovered:

- missing http headers
- source map not disabled
- potentially unsafe auth session
- iframe cross site scripting
- insecure direct object reference

Here is a list of endpoints tested.

Missing HTTP Headers

Reference: **OWASP A05**

Severity: **low**

The following critical security headers are missing from the site:

strict-transport-security	It strengthens your implementation of TLS by getting the User Agent to enforce the use of HTTPS. Recommended value: **max-age=31536000**
x-frame-options	It tells the browser whether you want to allow your site to be iframed, which can help prevent clickjacking. Recommended value: **sameorigin**

x-content-type-options	Setting it to **nosniff** stops a browser from trying to mime-sniff the content type and forces it to stick with the declared content-type. Recommended value: **nosniff**
referer-policy	It allows a site to control how much information the browser includes with navigations away from a document and should be set by all sites. Recommended value: **strict-origin-when-cross-origin** Recommended value: **sameorigin**
content-security-policy	It allows you to whitelist trusted script sources to prevent the browser from loading malicious assets and executing injected inline scripts, which can fully prevent xss. Recommended value: **script-src 'self' apis.google.com www.googletagmanager.com www.acsbapp.com www.gstatic.com widget.helpcrunch.com js.stripe.com www.redditstatic.com connect.facebook.net www.google-analytics.com m.stripe.network;**

The site reveals the server information via **server** header. This makes it easy for malicious actors to identify security misconfigurations in the server and exploit resulting vulnerabilities.

For instance, index.html and static resources respond with **server** header equal to **AmazonS3**. This lets anyone know that the site is statically hosted on aws s3.

Similarly, ***https://dig.shovelapp.io/backend/**** responds with the **server** header set to **nginx/1.20.0**. This again reveals that the API server is behind nginx reverse proxy.

It is worthwhile to configure these servers so that they no longer broadcast **server** header.

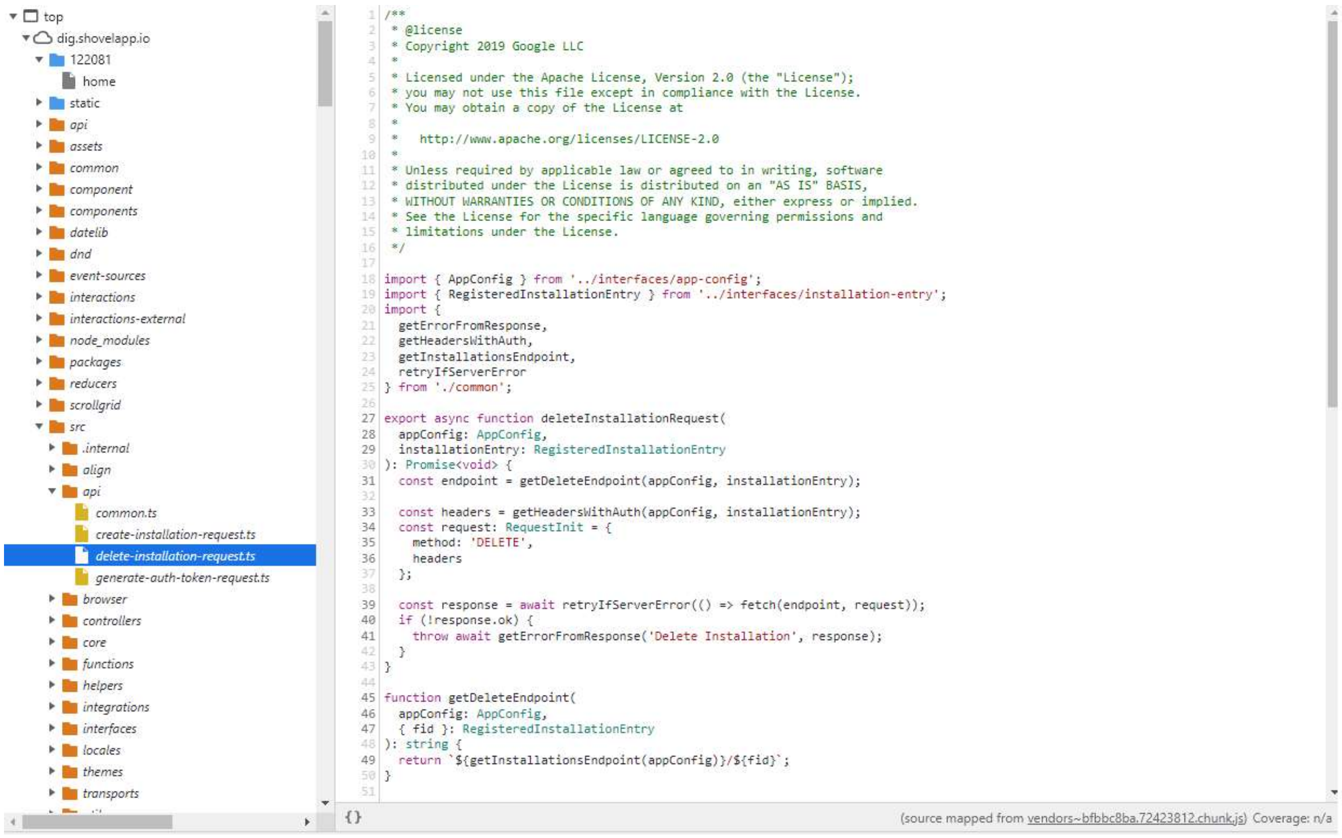
Sourcemap not disabled

Reference: **OWASP A05**

Severity: **low**

The entirety of the client side application source code is plainly visible in the browser developer tool thanks to sourcemap. By analyzing the source code, attackers can easily decipher the entire flow of the application and find security flaws to exploit. That wouldn't have been so easy, had the sourcemap been turned off.

Therefore, it is extremely important that the sourcemap be turned off during production build.



```
/**
 * @license
 * Copyright 2019 Google LLC
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *   http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

import { AppConfig } from '../interfaces/app-config';
import { RegisteredInstallationEntry } from '../interfaces/installation-entry';
import {
  getErrorFromResponse,
  getHeadersWithAuth,
  getInstallationsEndpoint,
  retryIfServerError
} from './common';

export async function deleteInstallationRequest(
  appConfig: AppConfig,
  installationEntry: RegisteredInstallationEntry
): Promise<void> {
  const endpoint = getDeleteEndpoint(appConfig, installationEntry);

  const headers = getHeadersWithAuth(appConfig, installationEntry);
  const request: RequestInit = {
    method: 'DELETE',
    headers
  };

  const response = await retryIfServerError(() => fetch(endpoint, request));
  if (!response.ok) {
    throw await getErrorFromResponse('Delete Installation', response);
  }
}

function getDeleteEndpoint(
  appConfig: AppConfig,
  { fid }: RegisteredInstallationEntry
): string {
  return `${getInstallationsEndpoint(appConfig)}/${fid}`;
}
```

(source mapped from vendors~bfbbc8ba.72423812.chunk.js) Coverage: n/a

Potentially unsafe authentication state

Reference: **OWASP A07**

Severity: **low**

The application authentication state is stored in localstorage. The app uses **refreshToken** *value* from localstorage to retrieve auth token from the backend.

This works great, but still isn't completely secure. If someone finds xss vulnerability in the future, they can exploit it to fully take over the victim's account. That's because javascript injected to the app can retrieve the **refreshToken** from localStorage deliver it to an attacker. This refresh token is all that is needed to compromise the account and perform dangerous actions like account deletion.

Cookies is a safer alternative. When properly designed, auth token stored in a cookie will not be vulnerable to the aforementioned scenario. You could, for example, have the backend send an auth_token cookie, with **httpOnly**, **secure** and **sameSite** flags, during a successful login response.

However, that may not be feasible to implement sbecause that would require a massive overhaul in the applicaton architecture. Nonetheless, it is definitely something to keep an eye on.

Cross Site Scripting on iframe

References: **OWASP A03** and **OWASP A06**

Severity: **low** to **medium**

note: *this is a vulnerability in an external library (helpcrunch sdk)*

When you click on the chat icon at the bottom right corner, helpcrunch chat modal pops up. The contents of this modal is an iframe generated by helpcrunch sdk. The sdk doesn't properly filter the message content for xss payloads.

If you send the following text through the message box, the popup disappears.

```
<img

    onError = "javascript:document['body']['innerHTML']=''"
    src =  "-"

>
```

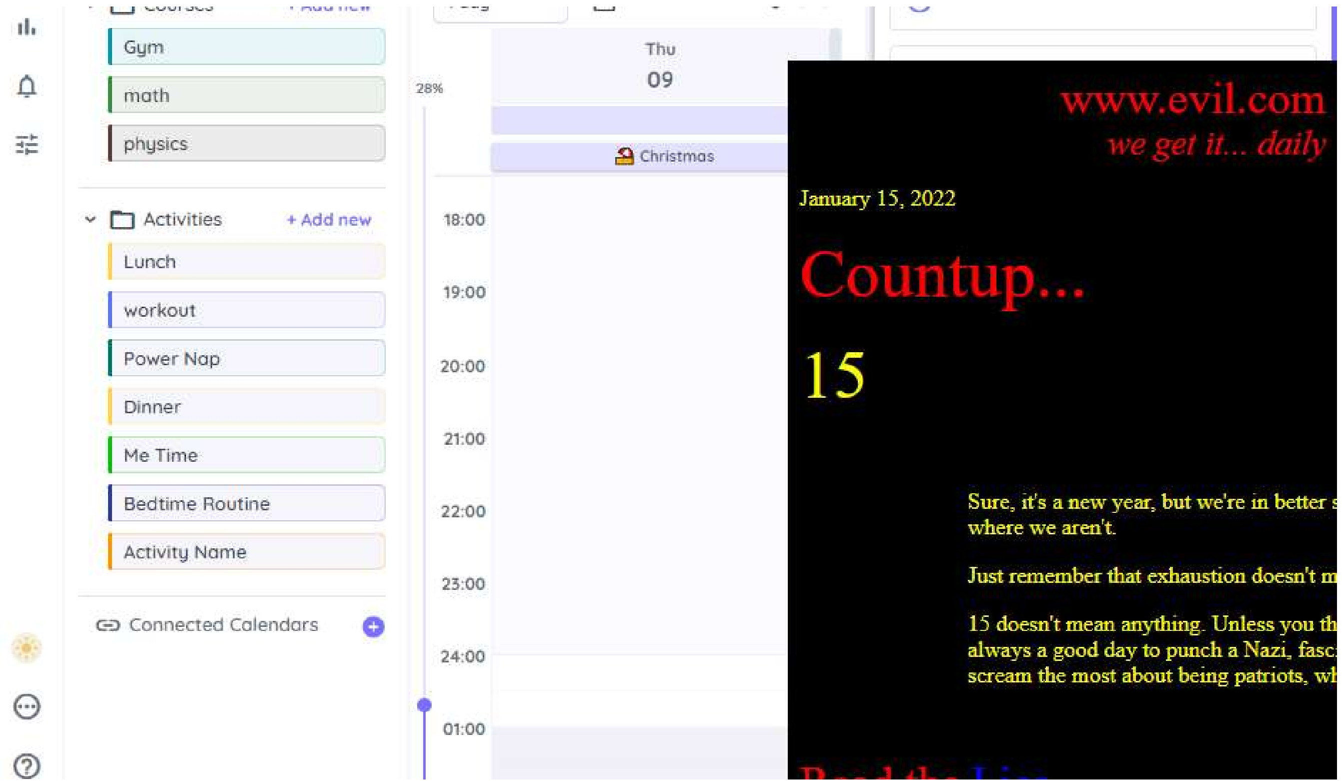
In case the admin is using the exact same interface to reply, they are going to be effected similarly as well. In fact, they can no longer reply, because the chat disappears as soon as they open it.

Here is another payload which redirects anyone who opens the popup, including the user sending this payload, to an external site. And the user gets stuck there until another page refresh.

```
<img

    onError = "javascript:location['href']=['https://evil', 'com']['join']('.')
    src =  "-"

>
```



Sure, it's a new year, but we're in better s
where we aren't.

Just remember that exhaustion doesn't m

15 doesn't mean anything. Unless you th
always a good day to punch a Nazi, fasc
scream the most about being patriots, wh

There is not much you can do about the external sdk. But because the vulnerable iframe cannot access the parent frame's context, it only has a minimal impact.

Insecure Direct Object Reference (IDOR)

Reference: **OWASP A04**

Severity: **low** to **medium**

It can occur when a backend web application uses an ***identifier*** for direct access to an object in an internal database but does not check for access control or authentication.

In our case the highlighted parameter in the following api url is the ***identifier*** and it identifies a user of the application.

```
https://dig.shovelapp.io/backend/api/semesters/122081/activities/1091213
```

So when we have an IDOR vulnerability, any logged in user can make an API request to the above url by replacing his ***identifier*** with that of another user, and perform actions on their behalf like deleting their activity or reading their timeline.

Having tested the site for this vulnerability, I can confirm that the site does enough server-side checks to protect itself from update IDOR actions, meaning that a malicious user cannot perform **post**, **update**, and **delete** actions on behalf of another user.

However, a couple of endpoints are vulnerable to **get** IDOR actions, meaning that a malicious user can indeed read information about another user, which they are probably not supposed to.

Anyone can read anyone else's analytics, simply by tweaking the highlighted ***identifier*** parameter in the following request.

```
GET https://dig.shovelapp.io/backend/api/semesters/122180/analytics HTTP/2
Host: dig.shovelapp.io
Accept: application/json
Authorization: Bearer <auth_token>
```

Here is another endpoint that is similarly vulnerable.

```
GET /backend/api/semesters/122180/ics/check-updates?defaultDaysAhead=3 HTTP/2
Host: dig.shovelapp.io
Accept: application/json
```

```
    Authorization: Bearer <auth_token>
```

This can be mitigated by enforcing that the ***identifier*** url parameter matches the ***identifier*** represented by <auth_token>.

List of endpoints tested

Client side routes tested:

- https://dig.shovelapp.io/:id/home
- https://dig.shovelapp.io/:id/charts/analytics
- https://dig.shovelapp.io/:id/charts/timeline
- https://dig.shovelapp.io/:id/charts/cushion
- https://dig.shovelapp.io/settings/terms
- https://dig.shovelapp.io/gpa-calculator
- https://dig.shovelapp.io/notifications
- https://dig.shovelapp.io/:id/awake-time
- https://dig.shovelapp.io/:id/study-time
- https://help.shovelapp.io/