Part 5

In this assignment, we will extend our ecommerce web application by creating public and private profile web pages for users. We want to give the registered users the possibility to sell products in our shop. Thus, the products are sold either from other users, from *thin air LTD* themselves, or from a partner business. The goal of this assignment is to understand how cross-site information leakage attacks (XS-Leaks) work, and how your implementation should be robust against those attacks.

Introduction

The e-commerce website for *thin air LTD* seems to be finished, and you are waiting for the final response from your boss, who should come any minute now from the meeting with the company representative.

You meet with her and get to know that the client was not completely satisfied, due to a feature missing in the prototype: there was no page where the client could see a user's personal details, or order history for example. You try to reassure your boss by telling him that you were already working on this feature, but have not presented it yet, due to a specific concern. In fact, you were recently at a security conference, where you heard of how users of certain websites can be de-anonymized, or have personal information leaked.

As your boss is not entirely convinced (and now your colleagues know of this as well), you ought to implement this properly. The client requested a user profile page containing the user's private information. Additionally, the client came up with a new requirement, i.e., a public "store front" for every user who also sells products. Thus, users of the ecommerce web application can now sell products in their publicly available "store front". To estimate how often these pages are viewed, the client has asked you to use the Google Analytics tag (i.e., the Google's `analytics.js` library). The client owns another ecommerce web applications too, known as `high-air` (i.e., `https://example.high-air.com`), and they want us to identify the returning customers between the two applications leveraging the Google Analytics tag. Thus, you are asked to enable cross-domain tracking of users (i.e., between the domain of your application, `thin-air`, and the domain of the client's second ecommerce web application, `high-air`).

As these are both functionalities intended for the end user, you not only have to implement the routines providing the service, but also the corresponding HTML pages. After thinking it through, you realize that, in fact, these use-cases are subject to potential information leakage (as other users could, purposefully or accidentally, exfiltrate information from there), so you probably have to read that research work again and find out how to avoid that happening.

Task 5.1 Private Profile Web Page

Each user of the site must have a *private* profile page that contains any confidential information about the user. The page should be available at:

- `/profile/<email>/private◀`

It should be accessible only to the owner of the profile (whose email match the email specified in the URL), and other users should be banned from accessing the same content.

Note. You are free to show any confidential information in this web page (e.g., user's past orders, etc).

Task 5.2 Public Profile Web Page

Each user of the site must have a *public* profile page that acts as a shopping store of that user, where he/she can sell his/her products. The page should be available at:

- `/profile/<email>/public-store◀`

Page Views. In this task, we will add the Google Analytics library to the public store front web page (see `https://developers.google.com/analytics/devguides/collection/analyticsjs`). Google Analytics library anonymously identifies each browser interface with a unique client id, and by default stores this id in a cookie bound to the domain of the web site, which means it can only be accessed by pages on the same domain. To track the same client id for a given user across different domains, we use cross-domain tracking. In this assignment, we want to share the client id from the domain of the `thin-air` to the domain of the `high-air` web application (and set it in a cookie via the analytics library), so that the `high-air` web application can identify returning customers of `thin-air`.

In order to implement the analytics functionality, we are provided with the following piece of code. To enable cross-domain user tracking, we embed the iframe of a target page belonging to `high-air`, and pass the client id via the postMessage API from within the web page of `thin-air` to the iframe of `high-air` (see `https://developer.mozilla.org/en-US/docs/Web/API/Window/postMessage`).

```
<script type="text/javascript">
// library setup
window.ga=window.ga||function(){(ga.q=ga.q||[]).push(arguments)};ga.l=+new Date;
// create a tracker
ga('create', 'UA-XXXXX-Y', 'auto');
// send a page view commmand to the command queue.
ga('send', 'pageview');
</script>
<script async src='https://www.google-analytics.com/analytics.js'></script>

<iframe id="destination-frame" src="http://127.0.0.1:2000/high-air/"></iframe>
<script>
ga(function(tracker) {
  // Gets the client ID of the default tracker.
  var clientId = tracker.get('clientId');
  // Gets a reference to the window object of the destionation iframe.
  var frameWindow = document.getElementById('destination-frame').contentWindow;
  // Sends the client ID to the window inside the destination frame.
  frameWindow.postMessage(clientId, 'http://127.0.0.1:2000');
});
</script>
```

Example JavaScript code for the web page of the `high-air` is shown below. Once the target page receives the client id via the postMessage API event handler, it creates a tracker object with the client id and store it in a cookie within the domain of `high-air`. Web pages of `high-air` can thus check for the client id cookie and send a page view command providing the stored client id in the cookie to designate a returning customer.

```
window.addEventListener('message', function(event) {
  // Ignores messages from untrusted sources, only allow thin-air's origin
  if (event.origin != 'http://127.0.0.1:3000') return;

  // Create a tracker and store the client id
  ga('create', 'UA-XXXXX-Y', 'auto', {
    clientId: event.data
```

```
  });
});
```

Access Control. The public store front web page should be accessible to every logged user (not accessible to users not logged in). If the visitor is logged and is the owner of the public store front, you must not include the analytics library to send any page views, as a user should not be able to rank up the visits of his/her profile, nor should you allow/include cross-domain user tracking. On the other hand, if the visitor is logged, and is not the owner of a public store-front, you must include the analytics library, send a page view command, and enable cross-domain user tracking.

Second Ecommerce Application. For this assignment, we need a second ecommerce web application hosted on a different domain that acts as the `high-air` web site. However, since we don't actually have that available, we can run a second instance of our application on port 2000 ◄, and then create a simple web page for the second web application available at `/high-air/` ◄ that contains the analytics JavaScript code (postMessage event handler). Thus, the iframe source in your application should be set to `http://127.0.0.1:2000/high-air/` ◄.

Note. For simplicity and for the purpose of this assignment, the content of all `pages` can be mocked, e.g., placed *statically* in the HTML templates, or come from the backend server without interaction with the actual database. The public store page of each user at the moment is a showcase. However, a fundamental step is how you make use of the analytics library. You do not need to implement the actual purchase functionality for this assignment. If a user has no products to sell, you can consider showing a corresponding message in the page (i.e., "This user has currently no items for selling").