Secure Programming Excersises Lesson 3

B.L.Schopman

30 November 2019

Contents

1 Introduction

Dit zijn de oefeningen voor de *derde* week van **Secure Programming**

2 Online banking

2.1 Probleem

Een van de problemen zijn dat de wachtwoorden van oma en henk niet sterk zijn. De wachtwoorden zijn **correcthorsebatterystaple** en **omaisthebest**. Deze wachtwoorden zijn makkelijk te kraken. Het tweede probleem is dat het mogelijk is om CSRF te gebruiken.

2.2 Bewijzen

De bewijzen daarvoor zijn hun wachtwoorden en er word ook geen 2FA gevraagd aan oma en Henk. Het bewijs voor het tweede probleem is dat er met behulp van Postman de gegevens kunnen worden aangepast. Daardoor kan een hacker zoals MR. Robot geld naar zich zelf overmaken.Dat is dus CRSF

2.3 Oplossing

Oma en Henk moeten een wachtwoord-kluis gebruiken om sterke wachtwoorden daar in op te kunnen slaan. Een oplossing is bijvoorbeeld **LastPass**. En daarnaast zijn dit de volgende do's voor wachtwoorden van Pieter en Wouter [1].

- Store passwords and credentials in hashed form.
- Do not share passwords between accounts
- Create awareness among users AND ADMINISTRATORS!
- Apply a password policy that encourages the use of strong passwords, enforce this policy among all types of accounts equally, e.g. do not allow administrators to bypass this policy.
- Use two factor authentication

De volgende dingen kan je volgens Pieter en Wouter doen voor CSRF[1];

- Add and verify a unique secret token to all sensitive requests.
- When changing state (POST)
- `https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html` En de volgende dingen kan je doen volgens OWASP: [2]
  - You Need a Security Encoding Library
  - **RULE #0**
    Never Insert Untrusted Data Except in Allowed Locations
  - **RULE #1**
    HTML Escape Before Inserting Untrusted Data into HTML Element Content
  - **RULE #2**
    Attribute Escape Before Inserting Untrusted Data into HTML Common Attributes
  - **RULE #3**
    JavaScript Escape Before Inserting Untrusted Data into JavaScript Data Values

- **RULE #3.1**
  HTML escape JSON values in an HTML context and read the data with JSON.parse
- **RULE #4**
  CSS Escape And Strictly Validate Before Inserting Untrusted Data into HTML Style Property Values
- **RULE #5**
  URL Escape Before Inserting Untrusted Data into HTML URL Parameter Values
- **RULE #6**
  Sanitize HTML Markup with a Library Designed for the Job
- **RULE #7**
  Avoid JavaScript URL's
- **RULE #8**
  Prevent DOM-based XSS
- **Bonus Rule #1**
  Use HTTPOnly cookie flag
- **Bonus Rule #2**
  Implement Content Security Policy
- **Bonus Rule #3**
  Use an Auto-Escaping Template System
- **Bonus Rule #4**
  Use the X-XSS-Protection Response Header
- **Bonus Rule #5**
  Properly use modern JS frameworks

3 Online shop

3.1 Probleem

Het probleem is dat het mogelijk is om meer dan 10 **items** toe te voegen aan de winkelmand,

3.2 Bewijzen

```
if request.args.get("action") == "checkout":
          baskettotal = 0
          order = {}
          for item in basket:
              order[item] = { 'description':basket[item]['description'], 'quantity':bas
              baskettotal = baskettotal + basket[item]['quantity']
              basket[item]['quantity'] = 0
```

```
            return render_template('checkout.html', basket = order)

    return render_template('shop.html', basket = basket, credits=10, baskettotal=0)
```

Het probleem is dat er niet wordt gekeken naar de aantal **items** in de winkelmand. Dat kan je zien op de volgende regels:

```
        for item in basket:
                order[item] = { 'description':basket[item]['description'], 'quantity':bas
                baskettotal = baskettotal + basket[item]['quantity']
```

Je kan dus gewoon zoveel items toevoegen als je wilt.

3.3 Redemption

Je kan het volgende doen om te voorkomen dat je meer dan 10 **items** kan toevoegen

```
       for item in basket:
                order[item] = { 'description':basket[item]['description'], 'quantity':bas
                baskettotal = baskettotal + basket[item]['quantity']
                if{baskettotal > 10}{
                return render_template('checkout.html', basket = toomuchitems)
                }
```

4 File download

Er is **broken access control** bij de **file download**. Dit is omdat het mogelijk is om naar bestanden te gaan waar je helemaal niet heen mag als gebruiker.

4.1 Bewijzen

```
    @app.route("/get-document/")
def get_document(documentid):
    try:
        fileid = base64.b64decode(documentid).decode()
        return send_from_directory("./documents", filename=fileid, as_attachment=True, a
    except FileNotFoundError:
        abort(404)

@app.route('/document', methods=['GET'])
def document():
    return render_template('document.html'
```

Het is te zien dat het document word gedecode door de code en daarna wordt het uit de map documents gevraagd. Daardoor kan je dus een andere document in een andere map invoeren en wordt die gebruikt en dan kan je die dowbloaden.

4.2 Oplossing

De OWASP heeft hier het volgende over gezegd [0]:

The most important step is to think through an application's access control requirements and capture it in a web application security policy. We strongly recommend the use of an access control matrix to define the access control rules. Without documenting the security policy, there is no definition of what it means to be secure for that site. The policy should document what types of users can access the system, and what functions and content each of these types of users should be allowed to access. The access control mechanism should be extensively tested to be sure that there is no way to bypass it. This testing requires a variety of accounts and extensive attempts to access unauthorized content or functions.

Some specific access control issues include:

- **Insecure Id's**
  Most web sites use some form of id, key, or index as a way to reference users, roles, content, objects, or functions. If an attacker can guess these id's, and the supplied values are not validated to ensure the are authorized for the current user, the attacker can exercise the access control scheme freely to see what they can access. Web applications should not rely on the secrecy of any id's for protection.

- **Forced Browsing Past Access Control Checks**
  many sites require users to pass certain checks before being granted access to certain URLs that are typically 'deeper' down in the site. These checks must not be bypassable by a user that simply skips over the page with the security check. Path Traversal – This attack involves providing relative path information

  ```
  e.g., \../../target_dir/target_file"
  ```

  as part of a request for information. Such attacks try to access files that are normally not directly accessible by anyone, or would otherwise be denied if requested directly. Such attacks can be submitted in URLs as well as any other input that ultimately accesses a file (i.e., system calls and shell commands).

- **File Permissions**
  Many web and application servers rely on access control lists provided by the file system of the underlying platform. Even if almost all data is stored on backend servers, there are always files stored locally on the web and application server that should not be publicly accessible, particularly

configuration files, default files, and scripts that are installed on most web and application servers. Only files that are specifically intended to be presented to web users should be marked as readable using the OS's permissions mechanism, most directories should not be readable, and very few files, if any, should be marked executable.

- **Client Side Caching**
  Many users access web applications from shared computers located in libraries, schools, airports, and other public access points. Browsers frequently cache web pages that can be accessed by attackers to gain access to otherwise inaccessible parts of sites. Developers should use multiple mechanisms, including HTTP headers and meta tags, to be sure that pages containing sensitive information are not cached by user's browsers.

  There are some application layer security components that can assist in the proper enforcement of some aspects of your access control scheme. Again, as for parameter validation, to be effective, the component must be configured with a strict definition of what access requests are valid for your site. When using such a component, you must be careful to understand exactly what access control assistance the component can provide for you given your site's security policy, and what part of your access control policy that the component cannot deal with, and therefore must be properly dealt with in your own custom code.

  For administrative functions, the primary recommendation is to never allow administrator access through the front door of your site if at all possible. Given the power of these interfaces, most organizations should not accept the risk of making these interfaces available to outside attack. If remote administrator access is absolutely required, this can be accomplished without opening the front door of the site. The use of VPN technology could be used to provide an outside administrator access to the internal company (or site) network from which an administrator can then access the site through a protected backend connection.

References

[1] Wouter Wessels Pieter Meulenhof. Secure Programming Lesson 3. `https://moodle.inholland.nl/course/view.php?id=6952#section-15`, 2019. [Online; accessed 30-November-2019].

[2] OWASP. $Cross_Site_Scripting_Prevention_Cheat_Sheet.$, n.d. [Online; accessed 30-November-2019].

OWASP. Broken Access Control: How to protect yourself. `https://www.owasp.org/index.php/Broken_Access_Control#How_to_Protect_Yourself`, n.d. [Online; accessed 30-November-2019].