Tutorial 04
- Web security -

1 Question 1

1. In class we discussed Cross Site Request Forgery (CSRF) attacks. Explain what a CSRF attack is.

2. Can HTTPS prevent CSRF attacks? Explain your answer.

3. Can session cookies solely prevent CSRF attacks? Explain your answer.

4. The `Referer` HTTP header contains the URL of the previous page visited. If you click on a link on a page, a GET/POST request is issued with the URL of this page as the value for the referer header. Explain how the `Referer` HTTP header can be used to prevent CSRF.

5. A common defense against CSRF attacks is the introduction of anti-CSRF tokens in the DOM of every page (often as a hidden form elements) in addition to the cookies. An HTTP request is accepted by the server only if it contains both a valid cookie and a valid anti-CSRF token in the POST parameters. Why and when does this prevent CSRF attacks? Explain your answer.

6. One way of choosing the anti-CSRF token is to choose it as a fixed random string. The same random string is used as the anti-CSRF token in all HTTP responses from the server. Does this prevent CSRF attacks? If your answer is yes, explain why. Otherwise describe an attack.

7. Suppose `bank.com` has a 'Transfer money' link which links to the following page:

```
<p> Transfer details:
<form action="/transfer" method="post">
   <input type="hidden" name="user" value="{{username}}">
   <input type="hidden" name="CSRFToken" value="OWY4NmQwODE4ODRjN2Q2NTlhMmZlYWEw">
   Recipient's account: <input type="input" name="account"/><br>
   Amount to transfer: <input type="input" name="amount"/><br>
   <input type="submit" value="Transfer now"/>
</form>
```

   The web server replaces {{username}} with the username of the logged-in user who wants to do the transfer. The implementation of /transfer is given by the following pseudocode:

```
if validate_login_cookie(request.parameters['user'], request.cookies['login_cookie'])
then transfer_money(request.parameters['user'],
                    request.parameter['account'],
```

```
                    request.parameters['amount']);
    return '<p>Money successfully transfered</p>'
else return '<p>Sorry, ' + request.parameters['user'] + ', an error occurred.</p>'
```

where `validate_login_cookie()` checks that the cookie sent by the browser is authentic and was issued to the specified username. Assume that `login_cookie` is tied to the user's account and difficult to guess. Is `bank.com` vulnerable to a CSRF attacks?

2 Question 2

TheBookShop allows clients to order books online visiting a URL of the form
`https://www.thebookshop.com/order?title=OliverTwist`

```
1. def order_handler(cookie, param):
2.     print "Content-type: text/html\r\n\r\n",
3.
4.     user = check_cookie(cookie)
5.     if user is None:
6.         print "You first need to log in"
7.         return
8.
9.     book = param['title']
10.    if in_stock(title):
11.        ship_book(title, user)
12.        print "Order succeeded"
13.    else:
14.        print "Book", title, "is currently not in stock"
```

The `param` argument contains the query parameters in the HTTP request (*i.e.*, the part of the URL after the question mark). The function `check_cookie` checks the cookie and returns the username of the authenticated user.

1. Briefly define cross-site scripting (XSS) attacks. Explain the difference between reflected and stored XSS attacks.
2. Is this website vulnerable to an XSS attack? If your answer is yes, explain how an attacker could exploit it, and explain how TheBookShop could fix it.
3. Briefly define cross-site request forgery (CSRF) attacks.
4. Is this website vulnerable to a CSRF attack? If your answer is yes, explain how an attacker could exploit it, and explain how TheBookShop could fix it.
5. Suppose a user uses two browsers: one for surfing the visiting low-security websites, and a different one for visiting "sensitive". For example, the user could use Chrome to read blogs and Firefox for banking. You will assume that each browser store temporary files and cookies in a different directory.
   (a) Would using two browsers in such a way prevent reflected XSS attacks?
   (b) Would using two browsers in such a way prevent stored XSS attacks?
   (c) Would using two browsers in such a way prevent CSRF attacks?

-CSRF

1. CSRF is a type of malicious exploit of a website where unauthorised commands are transmitted from a user that the web application trusts.

TARGET: User who has an account on vulnerable server

MAIN STEPS:
1) Build an exploit URL
2) Trick victim into making a request to the vulnerable server as if intentional

ATTACKER TOOLS:
a) Ability to get the user to click exploit link
b) Ability to have victim visit attackers server while logged-in to vulnerable server

KEY INGREDIENT: Requests to vulnerable server have predictable structure

2. NO! The malicious code can issue HTTP request over TLS (i.e. HTTPS) The victim browser will establish a TLS connection w/ the vulnerable server and the malicious HTTP request will then be sent by the victim browser causing the unwanted state change.

3. NO! If the attacker tricks his victim to issue an HTTP request to the vulnerable server, the victim's browser will include all cookies for the vulnerable server in any HTTP request.

So if the vulnerable website relies on session cookies, and such a cookie has been set when the victim authenticated themselves, it will be sent along the HTTP request.

4. • The remote server verifies that the hostname in the referer header matches the target origin.

   • This will prevent CSRF attacks as the attacker cannot manipulate the header, thus malicious requests will not have the target origin as hostname in the header.

   • This method does not require any per-user state

   (+) ↳ Useful when memory is scarce

   (-) ↳ Raise privacy concerns

5. Any state changing operation requires a secure random token (e.g. CSRF token) to prevent CSRF attacks.

   → The token is added as a hidden field for forms or within the URL if the operation occurs via a GET

   → The server rejects the requested action if the CSRF token fails validation

   - Unique per user session
   - Large random value
   - Generated by a cryptographically secure random no. generator

6. NO! The attacker can just run an honest session w/ the remote website to learn the value of the anti-CSRF token, and incl. it in the malicious URL

7. YES! The server only checks the validity of the cookie and not the anti-CSRF token value sent w/ the HTTP request.

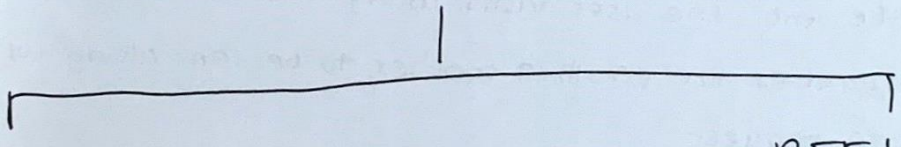
   The attacker can thus choose any value for the token in his malicious code

- XSS

XSS attacks are a type of injection, in which malicious scripts are injected into otherwise benign and trusted websites.

The goal of an attacker is to slip code into the browser under the guise of conforming to the same-origin policy:

1) Site evil.com provides a malicious script
2) Attacker tricks vulnerable server (bank.com) to send attacker's script to the user's browser
3) Victim's browser believes tht. the script's origin is bank.com (bc. it really does)
4) Malicious script runs w/ bank.com's access privileges



STORED

- injected script is permanently stored on the target servers (i.e. database)
- Victim retrieves the malicious script frm the server when it requests the stored info.

REFLECTED

- injected script is reflected off the web server, such as in an error msg, search result or any response tht. incl. input sent to server as part of the request
- Attacks are delivered to victims via another route (i.e. email/other site)

2. YES! Line 14 is vulnerable to XSS.

An attacker can supply a value of book title like

```
<script> alert(document.cookie)</script>
```

and assuming the in_stock fn returned false for that book ID, the web server would print that script tag to the browser and the browser will run the code frm. the URL.

To prevent: cgi.escape(title)

3. —

4. YES! An adversary can set up a form that submits a request to order a book to http://www.thebookshop.com/order?title=anytitle and this request will be honoured by the server

↳ To solve, incl. a random token w/ every legitimate request and check that cookie['csrftoken'] == param['csrftoken']

5. (a) YES if the malicious request comes from an untrusted origin (i.e. a website tht. the user visits using the low-security browser) This attack requires the sensitive cookies to be sent along w/ the malicious HTTP request

BUT the malicious HTTP request will be issued by the low-security browser which doesn't have access to the sensitive cookie.

(b). NO! The attacker can still store the malicious script on the honest dbase and have the honest website serve it to the victim's browser for sensitive sites

(c). YES! If the malicious request comes from an untrusted origin. This attack requires sensitive data such as cookies to be stored in the browser issuing the request.

BUT the low-security browser doesn't have access to sensitive data such as cookies.