Tutorial 04 solutions
- Web security -

1 Question 1

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

   Solution

   This 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 of attack:

   (a) build an exploit URL

   (b) trick the 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 the victim visit attackers server while logged-in to vulnerable server

   Key ingredient: requests to vulnerable server have predictable structure

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

   Solution

   No. The malicious code can issue HTTP request over TLS (*i.e.* HTTPS). The victim browser will establish a TLS connection with the vulnerable server, and the malicious HTTP request will then be sent by the victim browser causing the unwanted state change.

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

Solution

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 `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.

Solution

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 referer header, and thus malicious requests will not have the target origin as hostname in the referer header. This method does not require any per-user state. This makes Referer a useful method of CSRF prevention when memory is scarce. It might however raise privacy concerns more generally.

5. A common defence 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.

Solution

- Any state changing operation requires a secure random token (e.g., CSRF token) to prevent CSRF attacks
- Characteristics of a CSRF Token
  - Unique per user session
  - Large random value
  - Generated by a cryptographically secure random number generator
- The CSRF token is added as a hidden field for forms or within the URL if the state changing operation occurs via a GET
- The server rejects the requested action if the CSRF token fails validation

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.

Solution

No. The attacker can just run an honest session with the remote website to learn the value of the anti-CSRF token, and include it in the malicious URL.

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?

Solution

Yes. The server only checks the validity of the cookie and not the anti-CSRF token value sent with the HTTP request. The attacker can thus choose any value for the anti-CSRF token in his malicious code as it will not get checked by the server. This is exactly the twitter 2013 attack saw in class.

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.

Solution

Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into otherwise benign and trusted web sites The goal of an attacker is to slip code into the browser under the guise of conforming to the same-origin policy:

(a) site `evil.com` provides a malicious script

(b) attacker tricks the vulnerable server (`bank.com`) to send attacker's script to the user's browser!

(c) victim's browser believes that the script's origin is bank.com... because it does!

(d) malicious script runs with bank.com's access privileges

XSS attacks can generally be categorised into two categories: stored and reflected
Stored attacks are those where the injected script is permanently stored on the target servers, such as in a database, in a message forum, visitor log, comment field, etc. The victim then retrieves the malicious script from the server when it requests the stored information.
Reflected attacks are those where the injected script is reflected off the web server, such as in an error message, search result, or any other response that includes some or all of the input sent to the server as part of the request. Reflected attacks are delivered to victims via another route, such as in an e-mail message, or on some other web site.

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.

Solution

Yes, line 14 is vulnerable to cross-site scripting. An attacker can supply a value of book title like `<script>alert(document.cookie)</script>`, and assuming the in_stock function returned false for that book ID, the web server would print that script tag to the browser, and the browser will run the code from the URL. To prevent this vulnerability, wrap book in that line in `cgi.escape(title)`.

3. Briefly define cross-site request forgery (CSRF) attacks.

Solution

CSRF forces a user to execute unwanted actions on a web application in which they're currently authenticated. CSRF attacks target state-changing requests, not theft of data, since the attacker has no way to see the response to the forged request. Target: user who has an account on vulnerable server Main steps of attack:

(a) build an exploit URL

(b) trick the 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 the victim visit attacker's server while logged-in to vulnerable server

Example: Alice wishes to transfer $100 to Bob using the bank.com web application. This money transfer operation reduces to a request like:
`GET http://bank.com/transfer.do?acct=BOB&amount=100 HTTP/1.1`
The bank.com server is vulnerable to CSRF: the attacker can generate a valid malicious request for Alice to execute!!
The attack comprises the following steps:

(a) Eve crafts the following URL `http://bank.com/transfer.do?acct=Eve&amount=100000`

(b) When Alice visits Eve's website she tricks Alice's browser into accessing this URL

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.

Solution

Yes, an adversary can set up a form that submits a request to buy a book to `http:// benbooks.mit.edu/buy?book=anyid`, and this request will be honoured by the server. To solve this problem, include a random token with every legitimate request, and check that `cookie['csrftoken']==param['csrftoken']`.

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?

Solution

Yes, if the malicious request comes from an untrusted origin (*i.e.* a website that the user visits using the low-security browser). This attack requires the sensitive cookies to be sent along with the malicious HTTP request. However the malicious HTTP request will be issued by the low-security browser which doesn't have access to the sensitive cookies.

(b) Would using two browsers in such a way prevent stored XSS attacks?

Solution

No. The attacker can still store the malicious script on the honest DB and have the honest website serve it to the victim's browser for sensitive sites

(c) Would using two browsers in such a way prevent CSRF attacks?

Solution

Yes, if the malicious request comes from an untrusted origin (*i.e.* a website that the user visits using the low-security browser). This attack requires sensitive data such as cookies to be stored in the browser issuing the request. However the the low-security browser doesn't have access to sensitive data such as cookies.