Handout – Cross Site Request Forgery(XSRF)

Tagtäglich surfen sehr viele Menschen. Da sich viele auf bekannten und sicheren Seiten befinden kennen die meisten den drittgrössten Angriff: CSRF gar nicht. Es könnte sein, dass sogar kleine Seitenbetreiber selber nichts davon wissen und somit ihre Seiten für Nutzer gefährlich sein könnten. Wie es möglich ist, sich in Accounts von anderen Nutzern anzumelden wird unten genauer erklärt.

Was ist Cross Site Request Forgery?

Mit Hilfe von Cross Site Request Forgery, ist es möglich für einen Hacker Besitz von einem Account jemanden andere zu erlangen oder zum Beispiel Aktionen auf verschiedene Seiten auszuführen. Aber was genau muss der Hacker machen um sein Ziel zu erreichen?

Als erstes muss die Person, welche er angreifen möchte auf die Seite, welche der Hacker interessiert ist angemeldet sein. Ist der Benutzer ja auf dieser Seite angemeldet wird eine Session gestartet und es befindet sich ein Cookie auf dem Rechner des Benutzers, damit die Seite erkennt, dass der schon eingeloggt ist. Der Benutzer klickt dann zum Beispiel auf verschiede Knöpfe oder füllt Formulare aus. Egal was er macht, bei jedem Befehl sendet der Browser eine Anfrage an dem Server und der Server führet diese aus und die Seite wird wieder auf dem Browser angezeigt.

Hier kommt dann der Angreifer ins Spiel. Er schickt dem Benutzer einen Link, welcher auf eine Webseite, welche der Angreifer erstellt hat weiterleitet. Zum Beispiel eine Blogseite wo der Benutzer ein Formular ausfüllt und auf einen Knopf drücken muss, oder einer Seite, bei welcher ein Formular schon beim Aufruf der Seite gesendet wird. Wird der Formular gesendet, wird der Benutzer auf die Webseite weitergeleitet, auf welcher er sich eingeloggt hat und dort führet dann der Server, den gewünschten Befehl des Hackers, wie zum Beispiel, das Ändern des Passworts vom Benutzer oder einen anderen gefährlichen Befehl.

Wie funktioniert Cross Site Request Forgery?

Wie oben erwähnt, erstellt der Angreifer zum Beispiel eine Seite mit einem Formular. Diese sendet er dann dem Benutzer. In diesem Formular gibt er dann als «action» den Link auf dem der Benutzer weitergeleitet werden soll. Somit existieren für dieses Beispiel zwei Webseiten. Einmal die Seite auf der der Benutzer angemeldet ist: https://bank.ch/ und dann gibt es die Seite, welche der Angreifer geschickt hat: https://Blog.ch/. Heisst der Link um das Passwort auf der Bank Seite zu ändern: https://bank.ch/changePassword/ wird der Angreifer zum Beispiel diesen Link als «action» auswählen. Der Angreifer schaut dann den Namen der Felder des Formulars der Bank um das Passwort zu ändern an. Diese heissen zum Beispiel «password» und «password_confirm». Der Angreifer platziert dann in seinem eigenen Formular zwei versteckte Felder mit denselben Namen wie auf der Banks Seite. Dort gibt er dann sein gewünschtes Passwort als Value ein. Geht der Benutzer auf der Seite des Angreifers wird dieser die Felder nicht sehen können. Führet er dann das Formular des Angreifers, wird automatisch auch das Formular der Bank mit den Werten, welche der Angreifer angegeben hat ausgeführt und somit wird das Passwort des Benutzers verändert, da er noch eingeloggt ist und der Bsrowser natürlich nicht unterscheiden kann von welcher Seite aus die Request kam und somit das Passwort des Angreifers als Werte an dem Server schickt. Somit wird das Passwort des Benutzers geändert ohne dass dieser es mitkriegt. Der Angreifer erfährt damit zwar nicht die Login Daten des Benutzers, aber er hat trotzdem den Zugriff des Accounts übernommen.

Massnahmen gegen den Angriff?

- Es gibt mehrere Massnahmen, welche Webseitenbetreiber benutzen können. Als erstes wäre das Verlangen des Passworts beim Ausführen von wichtigen Befehlen. Da der Angreifer das Passwort nicht kennt, kann dieser auch das Passwort nicht in seinem Formular verstecken.
- Der Webseitenbetreiber kann auch ein Captcha in seinem Formular platzieren. Somit wird der Benutzer darauf aufmerksam und das Formular wird nicht ausgeführt.
- Dasselbe gilt auch mit einem Pop-Up, in welchen der Benutzer bestätigen soll, dass er die Aktion Ausführen will.
- Der Betreiber könnte ebenfalls eine E-Mail-Bestätigung verlangen, welcher der Benutzer bestätigen muss.
- Als letztes kann auch ein Token beim Ausführen des Formulars generiert werden und dieser Token wird dann, wenn der PHP Code ausgeführt wird überprüft. Da der Angreifer den Token nicht kennt, kann der auch den Token nicht mitgeben.

Codebeispiele (Angriff)

Als erstes sieht man ein Passwortveränderungs-Formular auf der Seite, welche der Benutzer eingeloggt ist und als zweites sieht man das Formular auf die Seite des Angreifers.

https://bank.ch/changePassword/:

```
<form action="https://bank.ch/changePassword/" method="post">
<br>Password:<br>
<input type="password" size="40" maxlength="250" name="password"><br><br>

Confirm Password:<br>
<input type="password" size="40"  maxlength="250" name="password_confirm"><

<input type="submit" value="Abschicken" name="send">
</form>
```

https://Blog.ch/:

```
<form action="https://bank.ch/changePassword/" method="post">
<textarea rows="4" cols="50" placeholder="Kommentar schreiben..."
<input type="hidden" name="password" value="123456"/>
<input type="hidden" name="password_confirm" value="123456"/>

<input type="submit" value="Comment">
</form>
```

Auf den Bildern wird erkennbar, dass der Angreifer dieselbe «action» angibt (blau markiert) Er benutzt ein Kommentar Feld um den Benutzer dazu zu bringen etwas reinzuschreiben aber das Textfeld bewirkt nichts. Dann versteckt er zwei Input-Felder mit dem Typ «hidden» (orange markiert). Er benutzt dann für die Namen der Felder dieselben Namen wie auch auf der Seite (rot markiert). Zum Schluss schreibt er als Wert, das Passwort, welches er haben möchte (grün markiert). Drückt der Benutzer auf «Comment» wird sein Passwort bei der Bank sofort geändert.

Codebeispiele (Massnahme)

Als Beispiel für die Schutzmassnahmen, wird hier die Schutzmassnahme mit dem Token gezeigt.
Als erstes wird eine Klasse erstellt, welche das Token generiert und überprüft.

```php
class Token {
  public static function generate() {
    return $_SESSION['token'] = md5(uniqid());
  }

  public static function check($token) {
    if (isset($_SESSION['token']) && $token === $_SESSION['token']){
      unset($_SESSION['token']);
      return true;

    }
    return false;
  }
}
```

Mit Hilfe von «md5(uniqid());» wird ein zufälliger String erzeugt. Beim «check()» wird überprüft ob der angegeben Token gleich ist, wie der generierte Token, welcher in der Session gespeichert wurde. Versucht also der Angreifer, den Benutzer auszutricksen, wird die Variable «$token» leer sein, da er die ja nicht kennt.

Das generieren des Tokens befindet sich dann im Formular für das Wechseln des Passworts.

https://bank.ch/changePassword/:

```php
<form action="https://bank.ch/changePassword/" method="post">
<br>Password:<br>
<input type="password" size="40" maxlength="250" name="password"><br><br>

Confirm Password:<br>
<input type="password" size="40"  maxlength="250" name="password_confirm"><br><br>
<input type="hidden" name="token" value="<?php echo Token::generate();?>"/>

<input type="submit" value="Abschicken" name="send">
</form>
```

Im PHP-Code der Seite, wird dann die Klasse importiert und zusammen mit dem Überprüfen der Variablen, wird auch der Token geprüft und der Code wird nur fortgesetzt, falls der Token übereinstimmt.

```php
<?php
include ('connect.php');
session_start();
require_once 'Token.php';
```

```php
if (isset($_POST['username']) && isset($_POST['betrag']) && isset($_POST['token'])){
  $send_username = $_POST['username'];
  $send_betrag = $_POST['betrag'];
  if (Token::check($_POST['token'])){
```