Techniques de test

Compte rendu TP4

WebGoat - Failles de sécurité Application web

Ricardo Rodríguez

Techniques de test
M1 ILSEN
Université d'Avignon et des Pays de Vaucluse

Introduction

Le TP 4 présente une application web avec des défauts qui permet aux étudiants d'apprendre sur les failles de sécurité des applications web, les raisons et réfléchir sur le solutions possibles.

Les 9 leçons WebGoat à traiter :

1. Access Control Flaws > Using an Access Control Matrix
2. Code Quality >Discover Clues in the HTML
3. Injection Flaws > String SQL Injection
4. Injection Flaws > Database Backdoors > Stage 1
5. Improper Error Handling > Fail Open Authentication Scheme
6. Parameter Tampering > Bypass HTML Field Restrictions
7. Cross-Site Scripting (XSS) > Cross Site Request Forgery (CSRF)
8. Session Management Flaws > Spoof an Authentication Cookie
9. Malicious Execution > Malicious File Execution

Rendu pour chaque leçon

- Indiquez les éléments utilisés pour atteindre le but
    - Uniquement l'énoncé (Lesson Plan)
    - Le ou les indices (Hints)
    - Cookies, Paramètres ou Code source Java
    - La solution de la leçon
- Qualifiez les différents tests en lien avec le cours
    - Objectif(s) de sécurité et menace(s) liées à l'exploitation de la vulnérabilité telle que proposée dans la leçon
    - Top 10 des Risques de Sécurité des Applications OWSAP
    - Indiquer le ou lignes de code java à l'origine de la vulnérabilité. (Si le code est disponible)

Lessons

1) Access Control Flaws > Using an Access Control Matrix

Éléments utilisés pour atteindre le but :
- Lesson Plan
    Pour atteindre le but il fallait tester l'accès à "Account Manager" avec les idfférents utilisateurs.

Solution :
L'utilisateur Larry a réussi à accéder à cette ressource.

Objectif de sécurité menacé :
Autorisation, un utilisateur ne devrait pas accéder à une ressource qui lui est interdite.
Dans la leçon, il suffit de changer le nom de l'utilisateur pour accéder à une ressource interdite.

Top 10 OWSAP : A7

Code Java :
Contrôle de rôles

```
265     private boolean isAllowed(String user, String resource)
266     {
267     List roles = getRoles(user);
268     List resources = getResources(roles);
269     return (resources.contains(resource));
270     }
```

2) Code Quality >Discover Clues in the HTML

Éléments utilisés pour atteindre le but :
- Lesson Plan
    Pour atteindre le but il fallait chercher dans le code du formulaire des indices qui nous permettent de nous connecter.
- Code source HTML

Solution :
Dans le code HTML il y a ce commentaire :
    <!-- FIXME admin:adminpw --><!-- Use Admin to regenerate database -->
En utilisant User = admin et Password = adminpw j'ai réussi à me connecter.

Objectif de sécurité menacé :
Authentification, un utilisateur non enregistré ne devrait pas accéder à l’application.
Dans la leçon, les identifiants étaient visible au public.

Top 10 OWSAP : A2

3) Injection Flaws > String SQL Injection

Éléments utilisés pour atteindre le but :
- Lesson Plan
    Pour atteindre le but il fallait trouver le bon nom d'utilisateur qui nous permette de lister les numéros de carte.
- Code SQL

Solution :
Utiliser le nom d'utilisateur : Smith' OR '1'='1
En faisant ainsi, on modifie la SQL qui sera exécutée et on accède à tous les numéros de carte.

Objectif de sécurité menacé :
Confidentialité, les données sont privées, aucun utilisateur non authorisé devrait pouvoir accéder aux données des autres.
Dans la leçon, l'injection SQL nous permet de visualiser toutes les cartes de crédit.

Top 10 OWSAP : A1, A6

Code Java :
```
94      String query = "SELECT * FROM user_data WHERE last_name = '" + accountName + "'";
...
101  ResultSet results = statement.executeQuery(query);
```

4) Injection Flaws > Database Backdoors > Stage 1

Éléments utilisés pour atteindre le but :
- Lesson Plan
    Il faut trouver un moyen d'augmenter le salaire de l'employé en exécutant une requête supplémentaire.
- Code SQL

Solution :
Dans l'id du compte, utiliser : 101; UPDATE employee SET salary = 80000
En faisant ainsi, la BDD exécute une deuxième requête qui augmente le salaire.

Objectif de sécurité menacé :
Intégrité, les données du système ne peuvent être modifiées que par les personnes habilitées.
Dans la leçon, l'injection SQL nous permet de modifier facilement les données en BDD, il suffit de connaître ou deviner le schéma.

Top 10 OWSAP : A1

Code Java :
La requête est créée et exécutée sans contrôler les données reçues.
```
131     String userInput = s.getParser().getRawParameter(USERNAME, "");
132     if (!userInput.equals(""))
133     {
```

```
134     userInput = SELECT_ST + userInput;
```

5) Improper Error Handling > Fail Open Authentication Scheme

Éléments utilisés pour atteindre le but :
- Lesson Plan
  Il faut trouver un moyen de se connecter avec l'utilisateur "webgoat" sans connaître son mot de passe.
- Hints :
  You can force errors during the authentication process.
  You can change length, existance, or values of authentication parameters.

Solution :
J'ai modifié le code HTML du formulaire et j'ai changé le nom du champ "Password" pour un autre. Le système n'a pas trouvé le paramètre Password dans la requête et il m'a permis donc d'accéder en tant que "webgoat" sans donner de mot de passe.

Objectif de sécurité menacé :
Authentification, autorisation, Confidentialité.
Un mauvais contrôle dans l'accès au système peut permettre aux gens de se connecter sans avoir le droit, en plus d'avoir accès aux données privées de l'utilisateur concerné.

Top 10 OWSAP : A2

Code Java :
Si le paramètre Password n'est pas trouvé, cela lance une exception qui est mal gérée et on peut se connecter sans mot de passe.

```
73  try
74  {
75      username = s.getParser().getRawParameter(USERNAME);
76      password = s.getParser().getRawParameter(PASSWORD);
77
78      // if credentials are bad, send the login page
79      if (!"webgoat".equals(username) || !password.equals("webgoat"))
80      {
81      s.setMessage("Invalid username and password entered.");
...
85      } catch (Exception e)
86      {
...
96      }
```

6) Parameter Tampering > Bypass HTML Field Restrictions

Éléments utilisés pour atteindre le but :
- Lesson Plan
  Il faut trouver un moyen d'envoyer une valeur incorrecte pour chaque élément du formulaire.

Solution :
Pour résoudre l'exercice j'ai modifié le code HTML du formulaire avec les options du WebDeveloper de Firefox.
Cela m'a permis d'ajouter des options au select, d'habiliter le dernier champ de texte, etc.

Objectif de sécurité menacé :
Intégrité.
Un mauvais contrôle dans la validation des formulaires peut permettre que des données invalides soient envoyées au système.

Top 10 OWSAP : A5

7) Cross-Site Scripting (XSS) > Cross Site Request Forgery (CSRF)

Éléments utilisés pour atteindre le but :
- Lesson Plan
    Écrire un email avec une image HTML. L'image doit contenir un lien vers une URL mal intentionnée.

Solution :
Pour résoudre l'exercice il faut écrire dans le texte du mail l'image suivante :

```
<image
src="http://localhost:8080/WebGoat-5.4/attack?Screen=109&menu=900&transferFunds=4000"
height="1px" width="1px" />
```

Cela fera que le navigateur, lors de la charge du mail, enverra une requête vers l'URL et celle-ci fera le transfert d'argent depuis le compte de l'utilisateur connecté vers celui de l'attaquant.

Objectif de sécurité menacé :
Autorisation.
Le formulaire d'envoi de mails devrait contrôler les messages écrits, afin d'éviter qu'un attaquant puisse effectuer des action non autorisées sur les comptes d'autres utilisateurs.

Top 10 OWSAP : A3

Code Java :
Le message est stocké sans contrôler son contenu.

```
91      String title = HtmlEncoder.encode(s.getParser().getRawParameter(TITLE, ""));
92      String message = s.getParser().getRawParameter(MESSAGE, "");
...
96      String query = "INSERT INTO messages VALUES (?, ?, ?, ?, ? )";
97
98      PreparedStatement statement = connection.prepareStatement(query,
                ResultSet.TYPE_SCROLL_INSENSITIVE,
99              ResultSet.CONCUR_READ_ONLY);
100     statement.setInt(1, count++);
101     statement.setString(2, title);
102     statement.setString(3, message);
103     statement.setString(4, s.getUserName());
104     statement.setString(5, this.getClass().getName());
105     statement.execute();
```

8) Session Management Flaws > Spoof an Authentication Cookie

Éléments utilisés pour atteindre le but :
- Lesson Plan
    Il faut trouver un moyen de se connecter en tant qu'alice. Pour cela, on peut partir de deux utilisateurs dont on connaît le nom et mot de passe afin d'identifier la méthode utilisée par le système.
- Cookies

Solution :
En me connectant avec les identifiants webgoat et aspect, et en regardant les cookies, j'ai découvert que le système utilise un cookie AuthCookie pour identifier l'utilisateur connecté. Après d'avoir envoyés les identifiants à travers le formulaire, le système stocke cette cookie pour pouvoir revenir à la page tout en restant connecté.

Il fallait donc trouver un moyen de trouver le cookie qui correspondrait à alice et l'envoyer au serveur. J'ai commencé par chercher entre les outils DevTools de Firefox mais je n'ai pas trouvé un moyen pour modifier les cookies envoyés. J'ai donc téléchargé l'application recommandé par WebGoat, WebScarab. Avec celle-ci j'ai réussi à envoyer des requêtes au serveur.

L'algorithme qui crée le cookie pour les utilisateurs prend le nom, inverse les lettres et puis il remplace chaque lettre pour le suivant. Ainsi, le cookie pour alice est :
    65432fdjmb

J'ai envoyé les cookies suivants au serveur pour résoudre le problème :
    JSESSIONID=5DEFF52A7C0F21189247DEBE4AFCBEA2;AuthCookie=65432fdjmb

Objectif de sécurité menacé :
Authentification, Autorisation.
Un mauvais contrôle dans le contrôle du login peut permettre à des utilisateurs non autorisés à se connecter. En plus, avec un minimum de connaissances sur le système (par exemple en ayant les identifiants de deux personnes) on peut déduire la méthode pour se connecter avec un autre compte.

Top 10 OWSAP : A2

9) Malicious Execution > Malicious File Execution

Éléments utilisés pour atteindre le but :
- Lesson Plan
    Utiliser le formulaire d'upload d'images pour télécharger un fichier mal intentionné qui va créer un autre fichier sur le serveur.
- Hints :
    Where are uploaded images stored? Can you browse to them directly?
    What type of file can you upload to a J2EE server that will be executed when you browse to it?

Solution :
En lisant le deuxième hint je me suis rendu compte qu'effectivement on n'a pas besoin de grande chose pour envoyer un fichier exécutable à un serveur J2EE. J'ai donc écrit le fichier JSP suivant :

```
<%@ page contentType="text/html; charset=ISO-8859-1" language="java"
   import="java.io.*"
%>
<%
   File fo = new File("G:\\Faculté\\2016-2017 M1 S2\\Tests\\M
Capon\\TP4\\apache-tomcat-7.0.59\\webapps\\WebGoat-5.4\\mfe_target\\basic.txt");
   fo.close();
%>
```

Quand le navigateur veut récupérer l'image, il va envoyer une requête au fichier JSp qui va ensuite créer le fichier.

Objectif de sécurité menacé :
Intégrité.
Les fichiers de l'application ne devraient pas être modifiés par des tiers sans autorisation. Dans ce cas, il y a un grave problème dans le contrôle des fichiers téléchargés. Il suffirait au minimum de restreindre le type de fichiers qui peuvent être envoyés.

Top 10 OWSAP : A5, A9