

RAPPORT TP

SECURITE APPLICATIVE

Réalisé par :

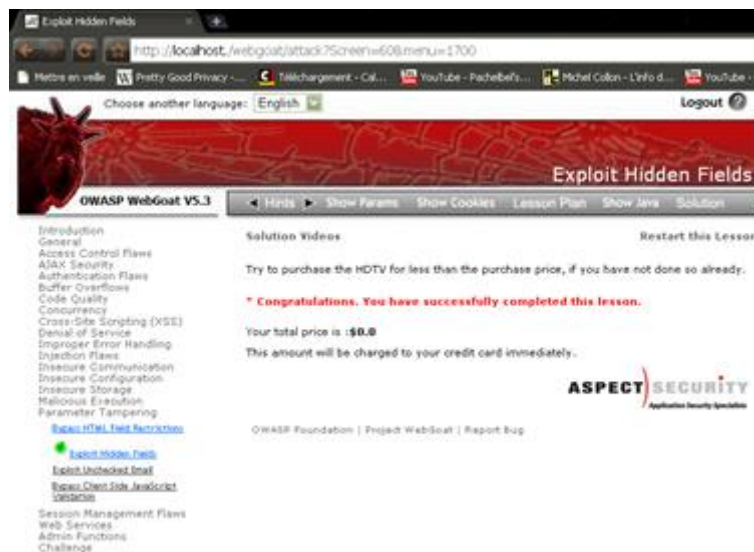
ELFARISSI Abdessamad

KHALIL El mahdi

Encadré par :

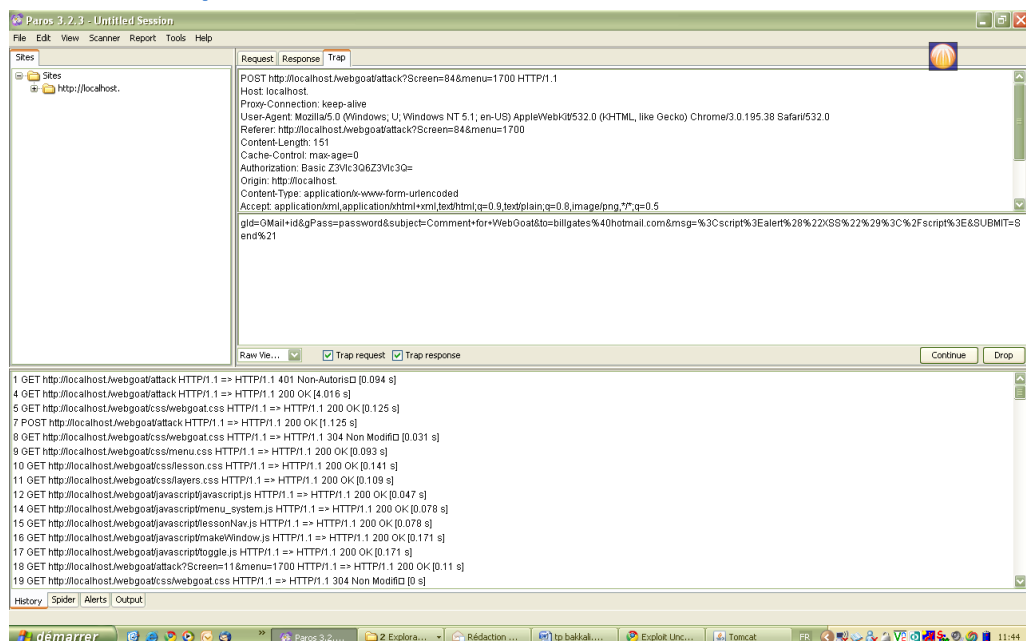
ELBAKKALI Hanane

I) Première attaque : Changement de prix d'une commande



Le problème c'est que la vérification se fait cotée client. Donc un client mal intentionné peut faire ce qu'il veut du côté de son client. Par exemple, il peut utiliser un outil comme Paros (webScramble) qui lui permet de récupérer les requêtes HTTP avant qu'elles soient envoyées à destination du serveur. Par l'exemple ici, il passe une commande avec un prix falsifié.

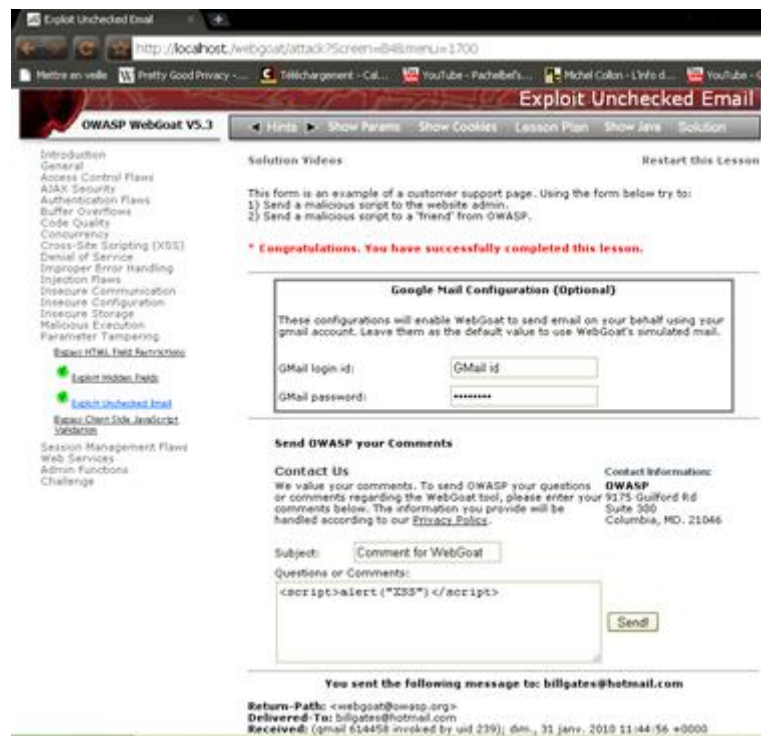
II) Deuxième attaque : Envoi de code malicieux via un mail.



Le problème ici c'est que y'a aucun contrôle sur le contenu du mail qui peut contenir des scripts (JavaScript, flash, VB Script,...) malicieux qui peuvent comporter des bouts de code dangereux pour contrôler à distance le serveur à l'insu de son utilisateur (webmaster).

On a envoyé une alerte par JavaScript qui est la suivante dans le champ mail : `<script>alert(\"c'est piraté\")</script>`

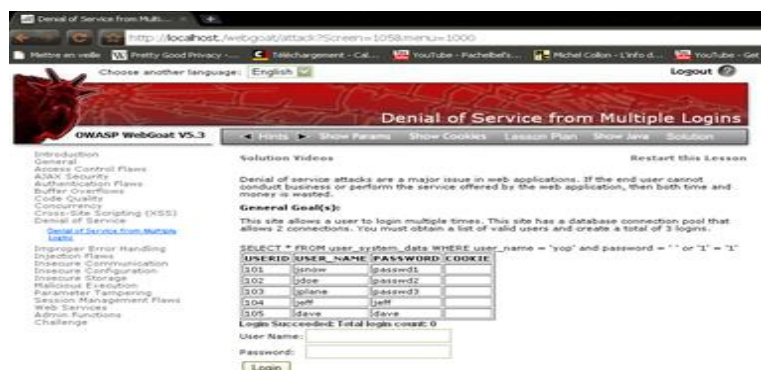
La seconde partie à été de modifier le destinataire, en interceptant le message via Paros et en changeant le paramètre 'to' de webgoatadmin@webgot.org vers billgates@hotmail.com (exploiter la confiance de client vers l'administrateur du site pour faire passer des code malicieux dans le mail).



III) Troisième attaque : injection SQL (causé un déni de service par connecté multiples comptes à la fois).

`SELECT * FROM user_system_data WHERE user_name = 'yop' and password = '' or '1' = '1'`

Ici '1' = '1' est toujours une condition vrai, ce qui donne que sans mot de passe on peut accéder au contenu de la table user_system_data pour l'utilisateur qui nous intéresse. Cette vulnérabilité est due au fait qu'il n'y a aucun contrôle sur les requêtes passées (Si elle contient des caractères spéciaux, ou respecte un format bien donné par exemple).



VI) 4ème injection Injection flaws : Injection SQL numérique

Accéder à des informations, préalablement non accéssibles (toujours problème de non contrôle des requetes passé par l'utilisateur).

Numeric SQL Injection

OWASP WebGoat V5.3

Introduction
General
Access Control Flaws
AJAX Security
Authentication Flaws
Buffer Overflows
Code Quality
Concurrency
Cross-Site Scripting (XSS)
Denial of Service
Improper Error Handling
Injection Flaws
Command Injection
Numeric SQL Injection
Log Spoofing
XPath Injection
LDAP SQL Injection
Stage 1: String SQL Injection
Stage 2: Parameterized Query
Stage 3: Numeric SQL Injection
Stage 4: Parameterized Query
String SQL Injection
Modify Data with SQL Injection
Add Data with SQL Injection
Database Backdoors
Blind Numeric SQL Injection
Blind String SQL Injection
Insecure Communication
Insecure Configuration
Insecure Storage
Malicious Execution
Parameter Tampering
Session Management Flaws
Web Services
Admin Functions
Challenge

Solution Videos

Restart this Lesson

SQL injection attacks represent a serious threat to any database-driven site. The methods behind an attack are easy to learn and the damage caused can range from considerable to complete system compromise. Despite these risks, an incredible number of systems on the internet are susceptible to this form of attack.

Not only is it a threat easily instigated, it is also a threat that, with a little common-sense and forethought, can easily be prevented.

It is always good practice to sanitize all input data, especially data that will be used in OS command, scripts, and database queries, even if the threat of SQL injection has been prevented in some other manner.

General Goal(s):

The form below allows a user to view weather data. Try to inject an SQL string that results in all the weather data being displayed.

* Congratulations. You have successfully completed this lesson.
* But you can't do it again! This lesson has detected your successful attack and has now switched to a defensive mode. Try again to attack a parameterized query.

Select your local weather station:

Go

SELECT * FROM weather_data WHERE station = 101 or 1=1

STATION	NAME	STATE	MIN_TEMP	MAX_TEMP
101	Columbia	MD	-10	102
102	Seattle	WA	-15	90
103	New York	NY	-10	110
104	Houston	TX	20	120
10001	Camp David	MD	-10	100
11001	Ice Station Zebra	NA	-60	30

OWASP Foundation | Project WebGoat | Report Bug

V) 5ème attaque: Configuration non sécurisée (FORCED BROWSING):

Le but de cette attaque est de découvrir des url réservés à des personnes privilégiées.

La vulnérabilité qui sera exploitée, c'est qu'un développeur d'une application web, utilise des URL connus et communs pour données des accès privilégiés.

Ici nous avons essayé de deviner l'url du webmaster : après avoir remplacé le chemin de l'URL /attack par /configuration, on a essayé /admin, et ça a donné lieu à la page suivante.

Forced Browsing

OWASP WebGoat V5.3

Introduction
General
Access Control Flaws
AJAX Security
Authentication Flaws
Buffer Overflows
Code Quality
Concurrency
Cross-Site Scripting (XSS)
Denial of Service
Improper Error Handling
Injection Flaws
Insecure Communication
Insecure Configuration
Forced Browsing
Insecure Storage
Malicious Execution
Parameter Tampering
Session Management Flaws
Web Services
Admin Functions
Challenge

Solution Videos

Restart this Lesson

* Your goal should be to try to guess the URL for the "config" interface.
* The "config" URL is only available to the maintenance personnel.
* The application doesn't check for horizontal privileges.

* Congratulations. You have successfully completed this lesson.

Welcome to WebGoat Configuration Page

Set Admin Privileges for:

Set Admin Password:

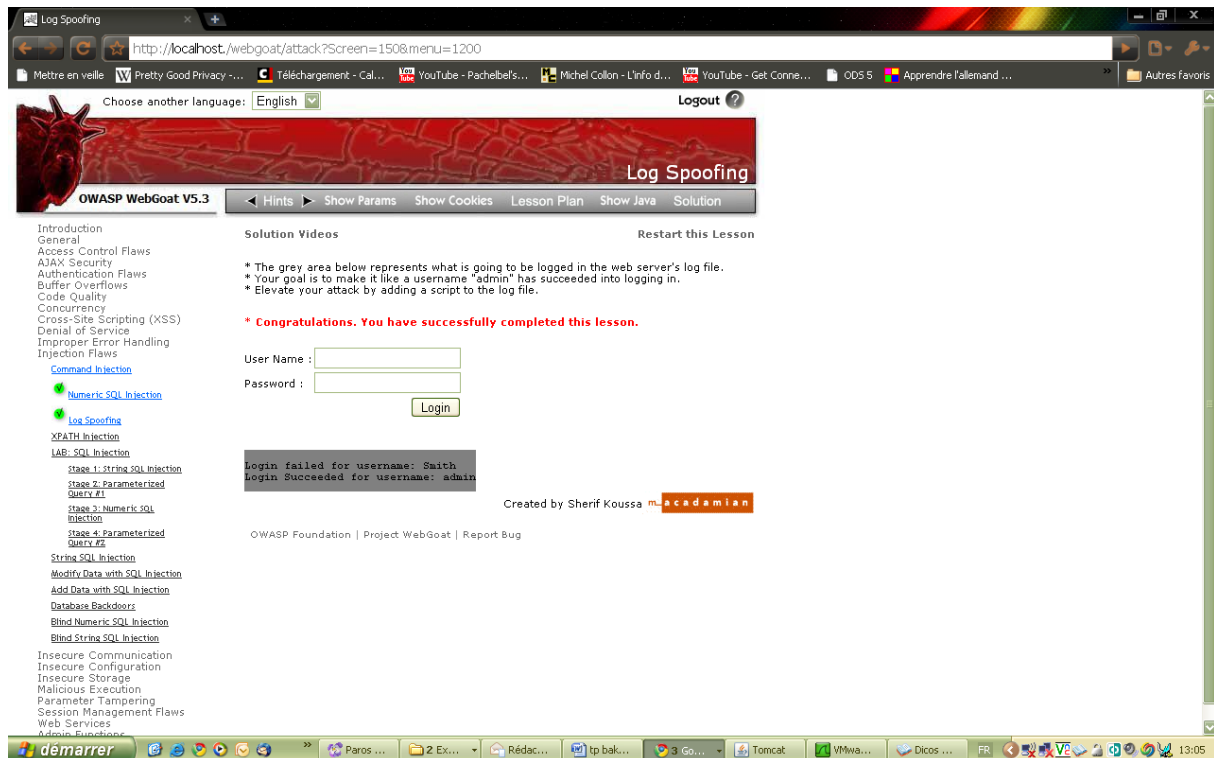
Submit

Created by Sherif Koussa

OWASP Foundation | Project WebGoat | Report Bug

VI) 6ème attaque : Log spoofing :

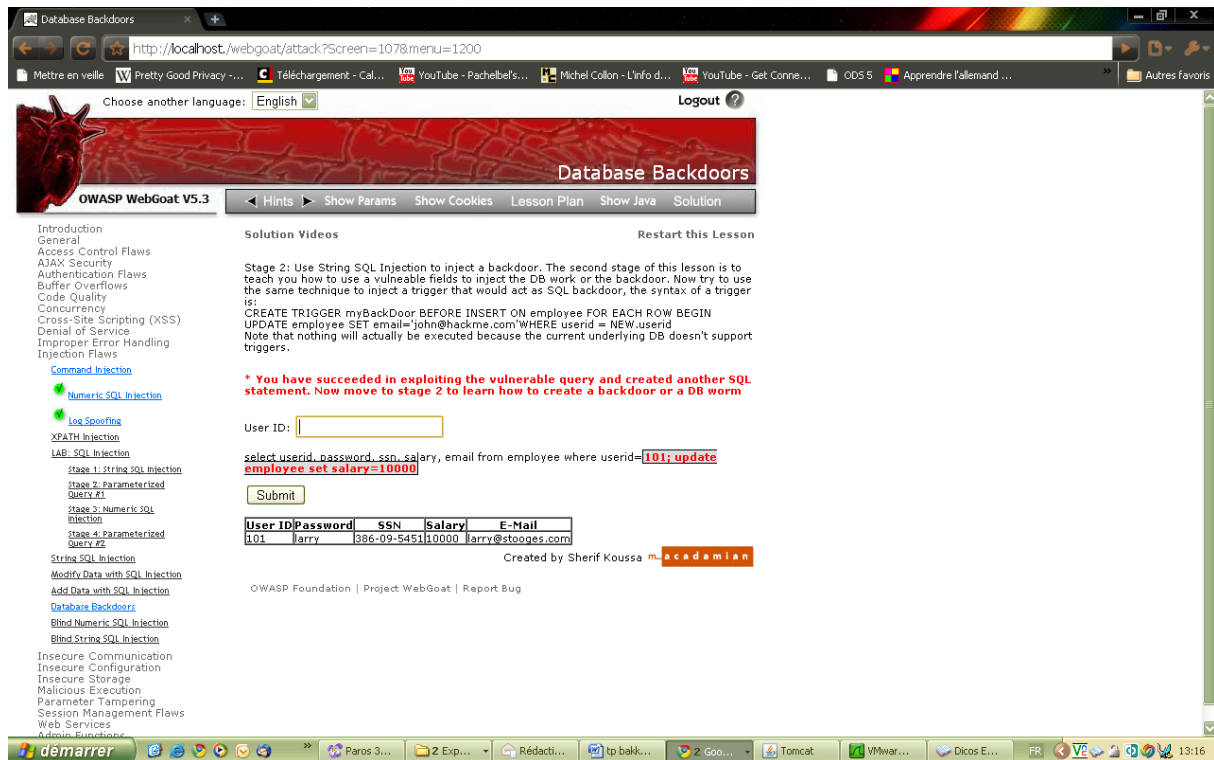
Cette attaque a pour but de toucher au fichier log qui regroupe les différentes connexions qui ont eu lieu. Cette attaque peut être profitable à un hacker qui a envie d'effacer ses traces.



Le champ de remplissage du logger n'admet pas de restrictions, on y injecte alors le texte suivant : 'Smith%0d%0aLogin Succeeded for username: admin'

Ce qui va simuler le fait que Smith n'a pas été connecté alors que l'utilisateur admin a été connecté. Un bout de code malicieux en JavaScript peut être envoyé au fichier log du site web pour une telle attaque.

VII) 7ème DATABASE BACKDOORS



1ere étape: on a rempli le champ id par : 101 ;update employee set salary = 100000 ;

On ajoute cette requête SQL : CREATE TRIGGER myBackDoor BEFORE INSERT ON employee FOR EACH ROW BEGIN UPDATE employee SET email='john@hackme.com'WHERE userid = NEW.userid

Afin de rediriger tous les mails sortant de l'admin vers celui de l'attaquant qui pourra tout espionner .

VIII) SESSION MANAGEMENT FLAWS : SESSION FIXATION :

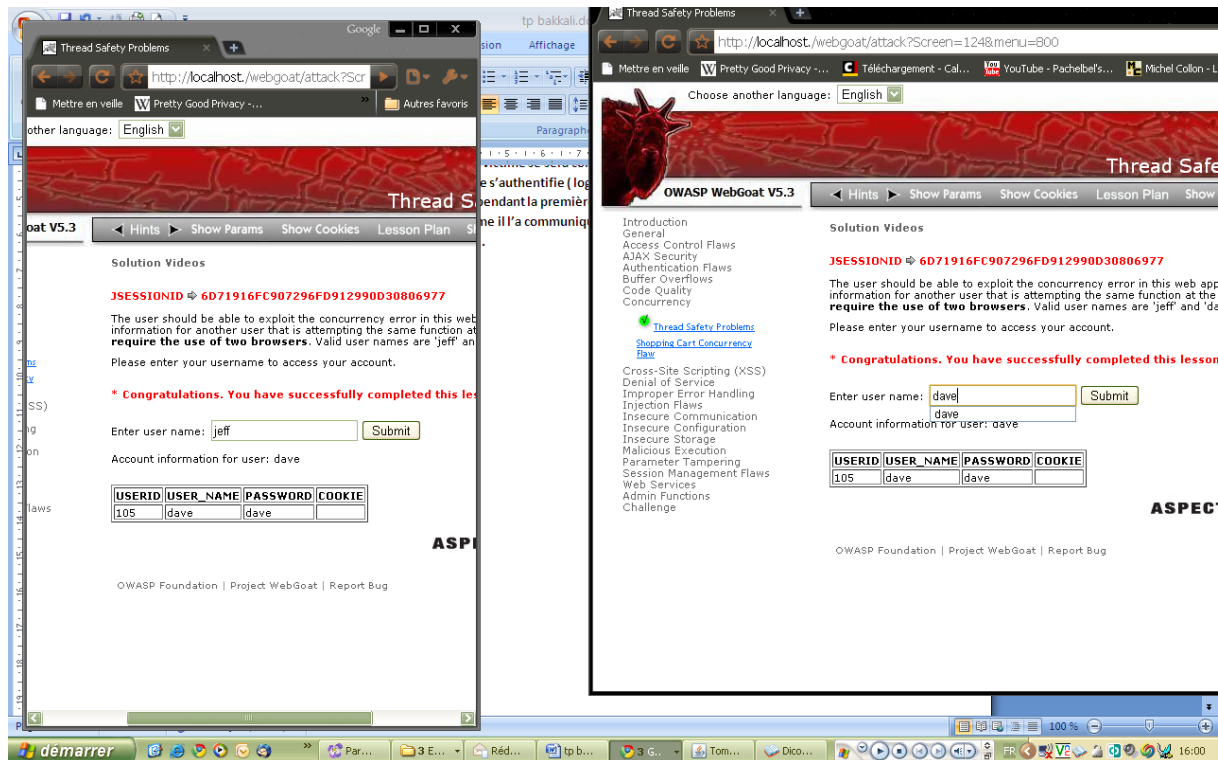
Dans les étapes 1 et 4 je suis le pirate (hacker) , dans les étapes 2 et 3 je suis la victime

la procédure est la suivante :

Au début, le pirate rédige un message en bonne et due forme , respectant les normes d'écriture et exempt de fautes d'orthographe , afin de se passer pour l'administrateur de l'entreprise .

Il y inclut également une url contenant un SID (id de session , que l'on va fixer à la valeur 100 par exemple) , une fois que la victime se sera connectée elle aura comme SID celui fixé par l'attaquant , à la 3eme étape la victime s'authentifie (login et mot de passe) , 4ème étape , l'attaquant connaît déjà l'url qu'il va poser cependant la première SID est invalide , et il la remplace alors par la suivante : SID = 100 (comme il l'a communiquée à la victime) , l'attaquant peut donc se connecter au compte de la victime !! .

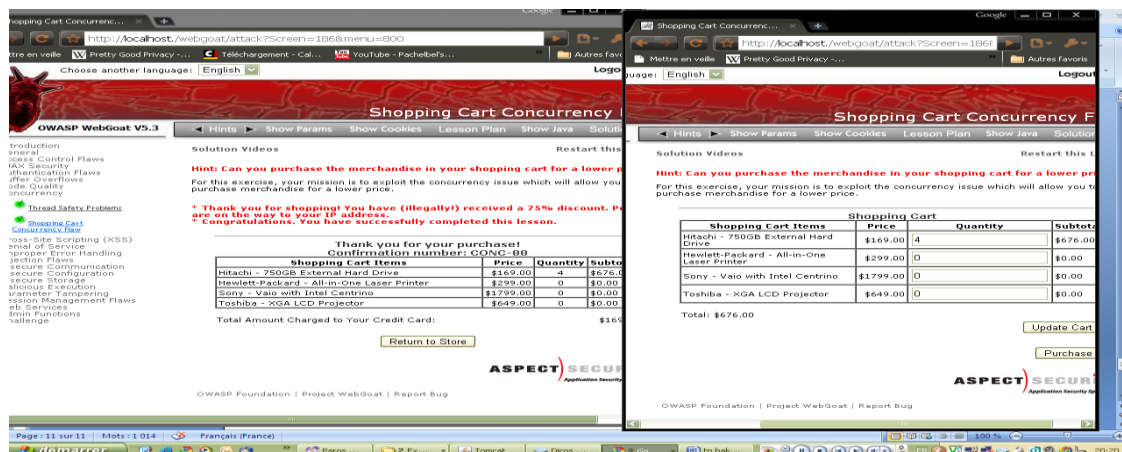
IX) PROBLEME DE CONCURRENCE DES REQUETES (THREAD SAFETY PROBLEMS)



Le principe étant d'ouvrir deux navigateurs en même temps et essayer de se loguer avec deux utilisateurs différents, en général l'appel fait travailler les mêmes fonctionnalités systèmes et méthodes de programmation, parfois il y a des bugs dans les appels à ces méthodes qui utilisent les mêmes variables et procédures de programmation, donc les mêmes ressources.


Le résultat est le suivant : si on se logue presque simultanément dans les deux fenêtres, on accède à la requête la plus récente (enfin l'une des deux), et un utilisateur se trouve connecté dans le compte d'un autre . !!

X) SESSION MANAGEMENT FLAWS : Paiement par carte de crédit :



On profite de la même faille, c'est à dire la grande défaillance sécuritaire des variables face aux threads multiples. on parvient à réduire de façon considérable le prix d'achat d'un produit.

XI) Qualité du code source :

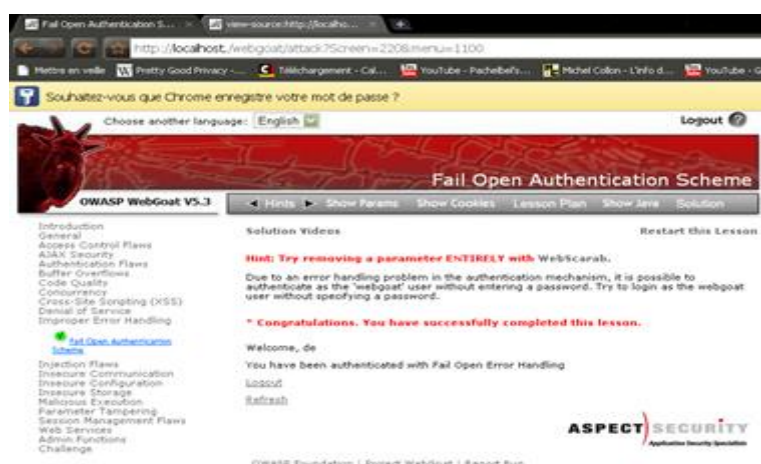


```
559 Below is an example of a forms based authentication form. Look for clues to help you log in.
560 <!-- Stop Instructions -->
561 <br>
562 <p><b>General Goal(s):</b></p>
563 The user should be able to bypass the authentication check.
564
565 <br>
566 <br>
567 <a href="javascript:toggle('lessonPlans')" target="_top" onclick="MM_nbGroup('down','group1','plans','',1)">Close this
568 Window</a>
569 </div>
570 <div id="lessonContent">
571     Developers are notorious for leaving statements like FIXME's, TODO's, Code Broken, Hack, etc... inside the source code.
572     <br>Review the source code for any comments denoting<br> passwords, backdoors, or something doesn't work right.<br>
573     Below is an example of a forms based authentication form. Look for clues to help you log in.
574 </div>
575 <div id="message" class="info"></div>
576
577 <div id="lessonContent"><form accept-charset='UNKNOWN' method='POST' name='form' action='attack?Screen=221&menu=700'
578 enctype='><!-- FIXME admin:adminpw --><!-- Use admin to regenerate database --><h1>Sign In</h1><table align='center' cellpadding='0' width='90%'
579 border='0' cellspacing='2'><tr><th colspan='2' align='left'>Please sign in to your account. See the OWASP admin if you do not have an
580 account.</th></tr><tr><td width='30%'>*Required Fields</td></tr><tr><td colspan='2'><br></td></tr><tr><td colspan='2'><b>User Name : </b></td><td><input
581 name='Username' type='TEXT' value=''></td></tr><tr><td colspan='2'><b>Password : </b></td><td><input name='Password' type='PASSWORD'
582 value=''></td></tr><tr><td colspan='2'><input name='SUBMIT' type='SUBMIT' value='Login'></td></tr></table></form></div>
583
584 <div id="credits">
585 <table align="RIGHT" cellpadding="0" width="90%" border="0" cellspacing="0"><tr><td valign="MIDDLE" width="100%"
586 align="RIGHT"></td><td valign="MIDDLE" align="RIGHT"><a href="http://www.aspectsecurity.com"></a></td></tr></table>
588 </div>
589
590 <div id="bottom">
591 <div align="center"><a href="http://www.owasp.org">OWASP Foundation</a> |
592 <a href="http://www.owasp.org/index.php/OWASP_WebGoat_Project">Project WebGoat</a> |
593 <a href="mailto:WebGoat@owasp.org?subject=WebGoat_Bug_Report_-_Lesson:">Report Bug</a>
594 </div>
595 </div>
```

Parfois on tombe sur des informations très précieuses juste en décortiquant le code source de la page web , commentaires , champs 'hidden' ou autres entités html constituent des goulots susceptibles de casser la sécurité de toute une application ,gare à ces erreurs , considérées de débutant ☺.

L'exemple ci-dessus, on a trouvé le login de l'admin ainsi que son mot de passe (admin :adminpw)

XII) Improper error handling (fail open authentication scheme) :



Le but de cette attaque est de profiter d'une défaillance de code, en particulier lors du traitement de l'exception JAVA (POINTEUR NULL).

Lorsque le champ html password est laissé vide, la variable Java correspondant à la chaîne de caractère relative est du type NULL, et donc le bloc { Catch throw correspondant à cette exception est appelé } . Or ce bloc d'erreur nous redirige vers une page où l'on considère authentifiés, Cette exception aurait dû être traitée autrement !!! d'où le nom IMPROPER ERROR HANDLING .