M1 - Outils de l'Internet (Année 2010/2011)

Miller's Crossing

Exercice 1. *Cross-Site Scripting*

Dans cet exercice nous allons nous intéresser à un type de vulnérabilités de sites webs générant du contenu dynamiquement : le *cross-site scripting* (abrégé XSS pour éviter les confusions avec le langage CSS).

1. Expliquez comment vous réaliseriez une page web qui demande à un utilisateur d'entrer son nom puis renvoie une page personalisée avec le nom entré en utilisant un formulaire et un script PHP.

On suppose que la page renvoyée est :

```
<html>
  <head><title>Bonjour</title></head>
  <body><p>Bonjour [nom]</p></body>
</html>
```

2. Expliquez ce qui se passe dans l'exemple précédent si l'on entre dans le formulaire le nom `Berenice</p><script language="Javascript">alert("Haha")</script>`

Toute page qui réutilise directement l'entrée de l'utilisateur dans son code est sujette à une telle attaque.
L'attaque peut sembler inutile puisque l'utilisateur qui remplit le formulaire est le seul à voir la page de résultat (donc le seul à exécuter le script)...

3. Rappelez comment sont transmis les paramètres d'un formulaire utilisant la méthode `GET`.

4. Décrivez ce qui se passe lorsque Bérénice, qui ne se doute de rien, clique sur un lien envoyé par Abélard pointant vers l'url [1] :
`http://sitevulnerable.fr/index.html?nom=Berenice%3C/p%3D%3Cscript%20language%3D%22Javascript%22%3Dalert(%22Haha%22)%3C/script%3D`

5. Expliquez comment Abélard peut utiliser cette téchnique pour obtenir les cookies de Bérénice correspondant à un site vulnérable aux attaques XSS.
Indication : En DOM, il existe une variable appelée `document.cookie`...

Remarque : Il n'est pas possible de transmettre un URL de plus de 256 caractères ce qui limite la taille du script inséré. Pour contourner cette restriction, on peut alors faire appel à un script se trouvant sur un autre site (d'où le terme « *cross-site* ») :
`<script>document.location='http://monsite/script.cgi'</script>`

6. Expliquez comment effectuer une attaque XSS sur un forum de discussion permettant d'obtenir les cookies de tous les utilisateurs qui tentent de lire un message envoyé par l'attaquant.

1. Rappel : le caractère spécial `%` permet d'encoder des caractères dans des URL. `%20` est un espace, `%22` est un guillemet, `%3C` et `%3D` sont les symboles inférieur et supérieur.

7. À votre avis, comment peut-on éviter ce genre d'attaques ? (il existe plusieurs solutions, à la fois côté serveur et côté client)

Une variante de l'attaque précédente appelée *cross-site request forgery* consiste à faire effectuer une requête à un utilisateur authentifié sur un site à l'aide d'un cookie sans qu'il ait conscience de le faire.

8. En supposant qu'un virement bancaire sur le site de la banque de Bérénice puisse être fait par le simple envoi d'une requête GET de la part d'un utilisateur authentifié (session par cookie), expliquez comment Abélard peut détourner de l'argent à Bérénice en l'invitant à aller voir son site.

9. Comment éviter ce genre d'attaques ?

Afin de parer aux attaques XSS, Microsoft a introduit en 2003 la notion de cookies `HTTPOnly` qui ne peuvent pas être lus à l'aide de l'instruction `document.cookie`. Ces cookies ne peuvent donc pas être volés directement par un script, et ne sont transmis que lors d'une requête HTTP. L'idée du XST est donc de provoquer une requête HTTP qui permette de renvoyer le cookie. On utilise pour cela la méthode TRACE.
Par exemple, sur la requête

```
TRACE / HTTP/1.1
Host: www.bases-hacking.org
User-Agent: blogodo
Nimportequoi: nimportequoi
Cookie: secret
```

on recevrait la réponse

```
HTTP/1.1 200 OK
Date: Fri, 19 Feb 2010 06:53:37 GMT
Server: Apache
Transfer-Encoding: chunked
Content-Type: message/http

TRACE / HTTP/1.1
Host: www.bases-hacking.org
User-Agent: blogodo
Nimportequoi: nimportequoi
Cookie: secret
```

10. Expliquez comment utiliser la méthode TRACE pour voler les cookies `HTTPOnly` d'un utilisateur.

Exercice 2. *HTTP Response Splitting*

Nous allons maintenant nous intéresser à une autre vulnérabilité de certains sites webs appelée *response splitting*. Le but de cette attaque est d'envoyer à un serveur HTTP une requête qui produira plus d'une réponse. Pour cela on exploite une fonctionnalité du HTTP 1.1 qui permet d'envoyer à la suite plusieurs requêtes HTTP sur une même connexion TCP/IP (on ne ferme pas la connexion après chaque requête, pour gagner du temps en évitant d'établir des dizaines de connexions).
Considérons un script PHP recevant un argument. À la requête :

```
GET http://site.com/index.php?page=accueil HTTP/1.1
Host: http://site.com
```

correspond la réponse :

```
HTTP/1.1 200 OK
Location: www.site.com/index.php?page=accueil
Content-Type: text/html
Content-Length: 72

<html><head><title>Titre</title></head>
<body>Bienvenue !< body></html>
```

1. Donnez la réponse renvoyée par le serveur web à la requête précédente dans laquelle on remplace le paramètre `accueil` par [2]
`accueil%0d%0aContent-Lenght:%200%0d%0a%0d%0aHTTP/1.1%20200%20OK%0d%0a`
`Content-Type:text/html%0d%0aContent-Lengh:%2045%0d%0a%0d%0a`
`<html><head></head><body>Haha</body></html>%0d%0a`
(les caractères encodés sont décodés lorsqu'ils sont lus par le serveur et apparaissent donc sous leur forme décodée dans la réponse)

2. Comment le navigateur interprète-t-il la réponse obtenue ? Que va-t-il afficher à l'écran ?

3. Quelle est la signification du champ `Content-Length` dans l'en-tête de la réponse HTTP. Expliquez comment cette signification a été exploitée dans l'exemple donné.

4. Expliquez comment Abélard peut utiliser cette attaque pour obtenir les cookies de Bérénice sur un site vulnérable (indication : c'est pareil que dans l'exercice précédent...).

5. Comment peut-on éviter ce genre d'attaques ?

Il est également possible d'utiliser cette technique dans d'autres buts...

Empoisonnement de cache

Considérons un gros fournisseur d'accès à Internet (FAI). Toutes les requêtes provenant de ses clients passent par ses machines, qui contactent alors les sites demandés, et renvoient les réponses aux clients. Une telle machine intermédiaire est appelée *proxy* (pluriel *proxies*).
Lorsqu'une page est fréquemment demandée à un proxy, celui-ci l'enregistre dans sa mémoire locale (appelée *cache*) et c'est cette version en mémoire qu'il transmettra aux autres clients qui demandent la page (pendant un certain temps, puis il redemandera la page au site d'origine pour voir si la page a été mise à jour).

6. Expliquez l'intérêt du cache des proxies pour le FAI et pour les clients.

7. Décrivez comment la technique de *response splitting* vue précédemment peut être utilisée pour *empoisonner* un cache, c'est-à-dire que l'on fait croire à un proxy que la page `www.site.com/page.html` contient des informations qui ne sont pas les vraies si une page du site `www.site.com` est vulnérable (par exemple la page `index.php` considérée précédemment).
Indication : on fait une requête sur `index.php` qui produit deux réponses puis une requête sur `page.html`...

8. Que se passe-t-il alors quand le cache a été empoisonné ? (la situation est assez proche de celle de la question 6 de l'exercice précédent).

2. L'encodage `%0d%0a` correspond à un retour à la ligne

Vol de réponses

Lorsque plusieurs clients d'un même FAI émettent des requêtes vers un même serveur simultanément (ou dans un intervalle de temps petit), le FAI « groupe » les requêtes et les émet sur la même connexion TCP/IP (il ouvre une connexion, envoie la première requête, puis la seconde et transmet la première réponse au premier client, la seconde au second...).

9. Quel est l'intérêt de grouper les requêtes (pour le FAI, les clients et le serveur distant) ?

10. Expliquez comment Abélard peut utiliser le *response splitting* pour recevoir la réponse d'une requête de Bérénice à un site web, tout en envoyant à Bérénice une fausse réponse qu'il contrôle entièrement.
Indication : Abélard envoie deux requêtes, une juste avant celle de Bérénice et l'autre juste après.
Remarque : Cette technique est assez difficile à mettre en pratique parce qu'il faut qu'Abélard réussisse à émettre ses requêtes quasiment en même temps que Bérénice... En général il n'est pas possible de cibler la victime de l'attaque, mais en envoyant beaucoup de requêtes sur un site très fréquenté on arrive à voler des réponses.

Requêtes invalides

Les attaques précédentes reposent sur le fait qu'un serveur web effectue des requêtes contenant des chaînes de caractères envoyées par un utilisateur sans les sécuriser. Il existe également un moyen d'empoisonner le cache d'un proxy en envoyant une requête invalide et en jouant sur les différentes interprétations possibles.
Considérons par exemple la requête suivante :

```
POST /nimporte_quelle_page.html HTTP/1.1
Host: poc.bases-hacking.org
Connection: Keep-Alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 0
Content-Length: 73

GET /fausse_page.html HTTP/1.1
Host: poc.bases-hacking.org
User-Agent: GET /page_empoisonnee.html HTTP/1.1
Host: poc.bases-hacking.org
Connection: Keep-Alive
```

L'attaque repose sur le fait que la norme HTTP ne précise pas ce qu'il faut faire d'une requête contenant plusieurs champs `Content-Length` dans son en-tête. Certains serveurs considèreront que le premier est le vrai, d'autres que c'est le dernier et enfin certains refuseront la requête.

11. En supposant que la requête précédente passe par un proxy qui considère le premier champ `Content-Length` comme vrai puis arrive sur un serveur qui tiendra compte du second champ, que va-t-il se passer ?