Bypasser une authentification

Shiney

June 2, 2011

Préface

Cet article ne prône absolument pas le piratage mais la connaissance. Par conséquent, vous devez savoir qu'en cas de piratage, vous êtes seul et unique responsable des conséquences de vos actes. Je vous déconseil fortement de mettre en pratique la démonstration qui va suivre sur un site qui ne vous appartient pas.
Ce document s'adresse à des professionnels et à des passionnés de sécurité informatique[1]. Des connaissances du protocole HTTP, en SQL, en PHP et en sécurité des applications web sont nécessaires à sa bonne compréhension.

[1]Un lamer n'est pas un passionné de sécurité informatique.

Contents

Introduction

Aujourd'hui, nous allons étudier une méthode pour bypasser une authentification faillible à une injection SQL. Pour cela, je prendrai comme exemple un site académique de la NASA. Notre étude sera en grande partie basée sur des injections SQL et nous utiliserons l'extension hackbar[2] avec Firefox.

[2]Live Header ou Tamper Data peuvent être une alternative.

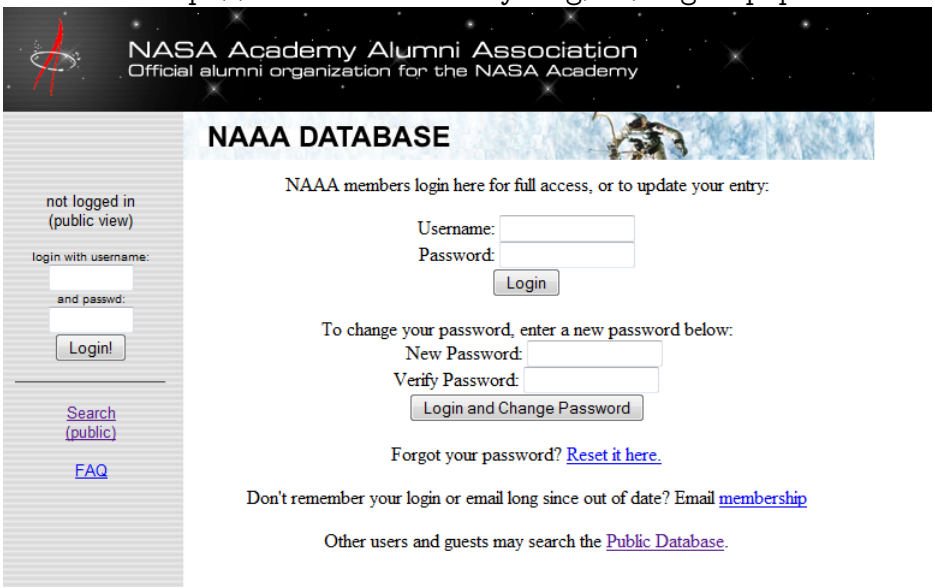
1 Analyse

Dans cette première partie, nous allons chercher une faille sur un script d'authentification. Pour se faire, une bonne analyse des informations envoyées vers ce dernier est primordiale. Plusieurs méthodes permettent de connaitre ces informations. Dans notre cas, nous nous contenterons d'analyser le code source du formulaire.

1.1 Page d'authentification

Voici notre page d'authentification classique, constitué d'un nom d'utilisateur et d'un mot de passe:

`http://www.nasa-academy.org/db/login.php`



1.2 Formulaire

Pour commencer, analysons le code source du formulaire qui se situe à gauche:

```
1.  <FORM METHOD=POST ACTION="login.php">
2.      <p align="center">
3.          <font size="2" face="Arial, Helvetica, sans-serif">
4.              not logged in<br>(public view)
5.          </font>
6.      </p>
7.      <p align="center">
8.          <font face="Arial, Helvetica, sans-serif" size="1">login with username:</font>
9.          <font face="Arial, Helvetica, sans-serif" size="1">
10.             <input name="username" size="7" type="text">
11.         </font><br>
12.         <font face="Arial, Helvetica, sans-serif" size="1">and passwd:</font><br>
13.         <font face="Arial, Helvetica, sans-serif" size="1">
14.             <input name="password" size="7" type="password">
15.         </font><br>
16.         <input name="pagefrom" value="alumni.php" type="hidden">
17.         <input value="Login!" type="submit">
18.     </p><hr>
19.     <p align="center">
20.         <font size="2" face="Arial, Helvetica, sans-serif">
21.             <a href="alumni.php">Search<br>(public)</a>
22.         </font> 23.      </p>
24.     <p align="center">
25.         <font size="2" face="Arial, Helvetica, sans-serif">
26.             <a href="http://www.nasa-academy.org/members/DatabaseFAQ.htm">FAQ</a>
27.         </font>
28.     </p>
29. </FORM>
```

On constate que 3 paramètres sont envoyés au script 'login.php' via une methode `POST`:

username : le nom du compte

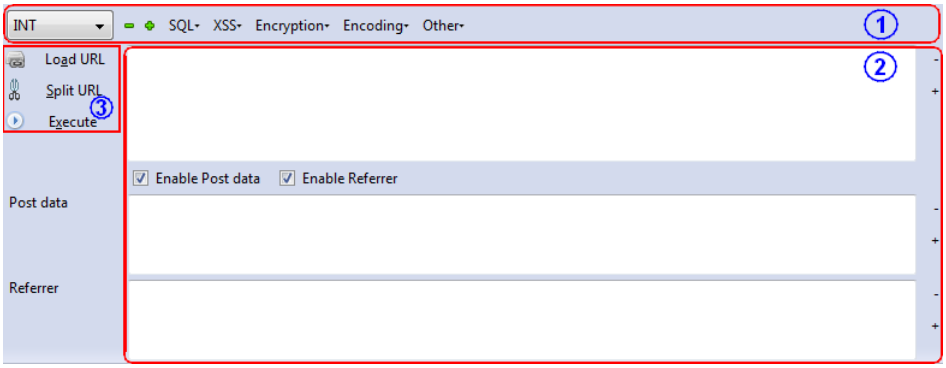
password : le mot de passe du compte

pagefrom : un paramètre caché initialisé à 'alumni.php'

1.3 Hackbar

1.3.1 Rapide présentation

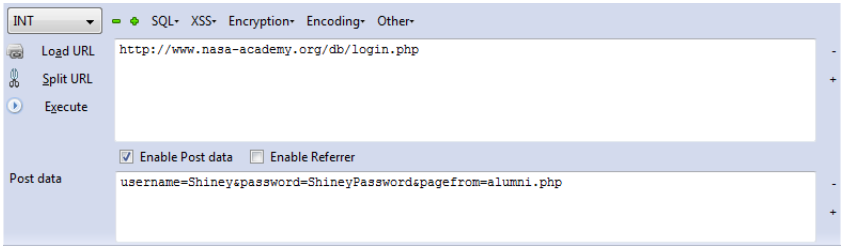
Pour éditer les données que nous enverrons, nous allons utiliser hackbar. Cette extension firefox se divise en 3 zones:



1. un menu avec des fonctions très utiles sur des chaines de caractères
2. un espace permettant d'éditer le champ url, d'éditer les données d'une requête POST et d'éditer l'entête `referer`
3. charger les informations de la dernière requête dans la zone 2, mettre en forme la zone de texte courante et exécuter la requête

1.3.2 Utilisation

Une authentification se fait de la manière suivante:



1.4 Recherche d'une faille

La plupart du temps les mots de passes sont encryptés en MD5, par conséquent nous n'attaquerons pas le paramètre `password`. Nous allons nous focaliser sur le paramètre `username` et tenter une fausse authentification dite témoin et deux injections:

Témoin. username = Shiney & password=Shiney

1. username = "' : pour provoquer une erreur
2. username = Shiney' OR '1'='1 : pour essayer de bypasser une authentification de manière classique

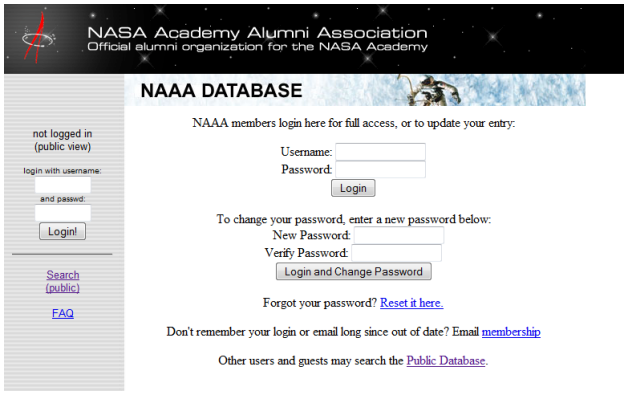


Figure 1: Authentification témoin

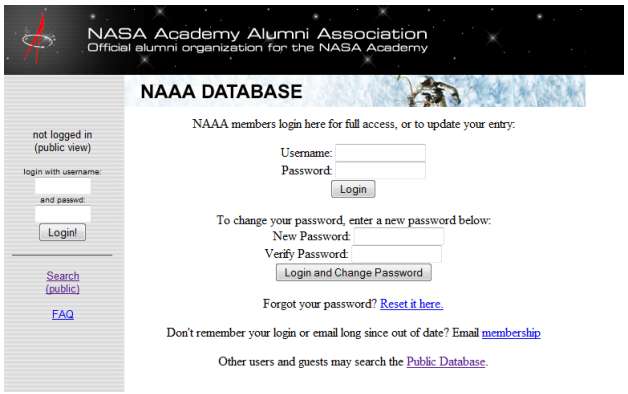


Figure 2: Injection 1

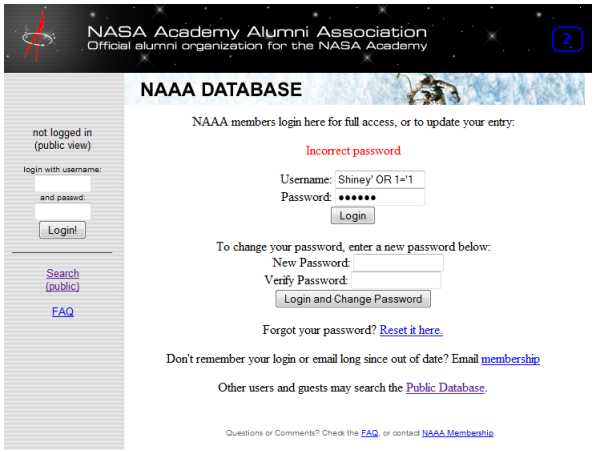


Figure 3: Injection 2

Notre authentification témoin nous montre que le script nous renvoie exactement la même page lorsque l'identifiant n'est pas bon.

On remarque que l'injection 1 n'a pas été concluante, aucun message d'erreur ou de réponse notable de la part du script.

En revanche, l'injection numéro 2 est beaucoup plus interessante. En effet, la page nous indique que le `password` est incorrect ce qui montre que pour le script, notre `username` est valide. Notre injection a permis de retourner n'importe quel `username` de la base de donnée, c'est donc pour cela que le script à accepté notre `username` mais pas notre `password`.

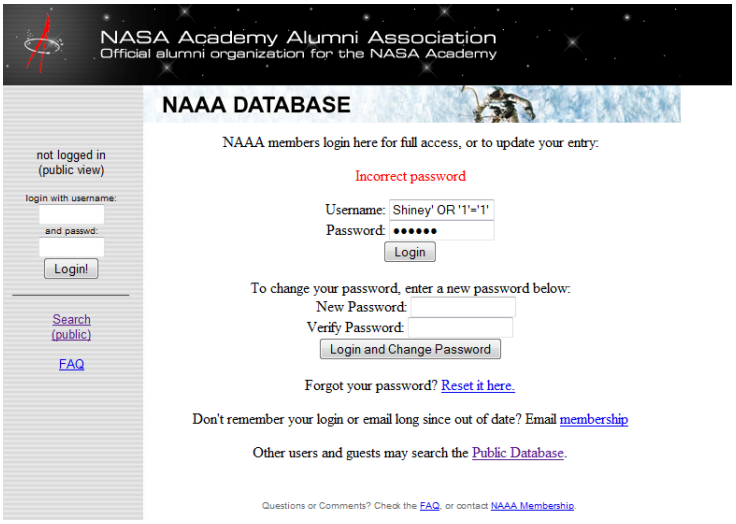
1.5 Prise de contrôle de la requête SQL

L'objectif après avoir localisé une faille SQL, c'est de comprendre cette requête puis d'en prendre le contrôle. Grâce à notre injection 2, nous avons pu en cerner une partie:

SQL : [...] username = 'Shiney' OR '1'='1' [...]

Cependant, nous n'avons aucune idée de ce qui se situe avant ou après (signalé par `[...]` dans l'encadré ci-dessus). Nous allons donc dire au script que la suite de notre injection représente du commentaire à l'aide du caractère dièse :

username = 'Shiney' OR '1'='1' #



La page nous indique encore que notre `password` est incorrecte, notre injection a donc de nouveau fonctionnée. Nous avons donc pris le contrôle de la fin de la requête SQL.

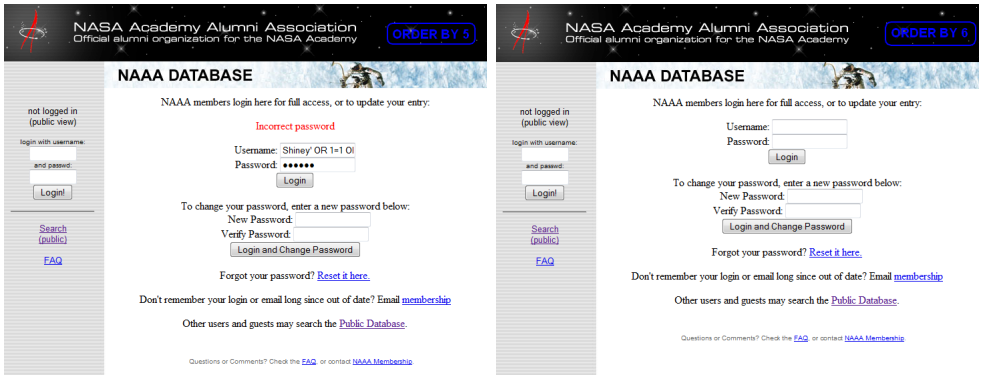
2 Exploitation

Nous arrivons à expliquer au script que notre `username` est valide mais pas notre `password`. Nous allons donc faire une `UNION` afin de donner au script les informations nécessaires pour qu'il accepte notre authentification.

2.1 Recherche du nombre de colonne

Il est nécessaire de connaitre le nombre de colonne de la première requête afin de pouvoir faire une `UNION`. Pour cela, nous allons utiliser l'expression `ORDER BY n` où `n` correspond au numéro d'une des colonnes de la requête. Cette expression permet de trier les resultats en fonction de la `n`-ème colonne. Par conséquent, si on augmente petit à petit la valeur de `n` jusqu'à obtenir une erreur [3] , alors la valeur précédente de `n`, nous donne le nombre de colonne:

username = Shiney' OR 1=1 ORDER BY 5 #	username = Shiney' OR 1=1 ORDER BY 6 #



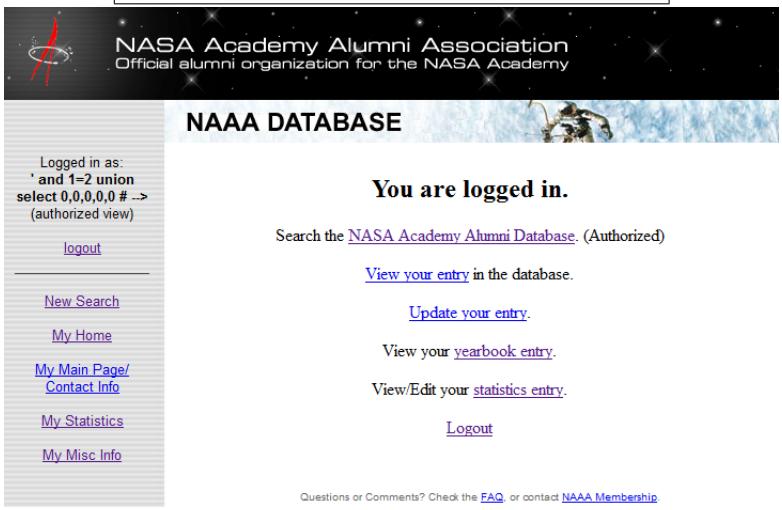
2.2 Syntaxe de l'UNION

Nous savons que la requête nécessite 5 colonnes, mais nous ne savons laquelle correspond à l'`username` ou au `password`. Nous allons donc tester une `UNION` afin d'observer la réaction du script. Les colonnes seront initalisées à 0 et nous remplacerons ' OR 1=1 par ' and 1=2 afin que le début de la requête SQL ne perturbe pas la suite de notre `UNION`.

[3] Dans notre cas, nous considérerons qu'une erreur correspond à l'absence du message 'Incorrect password', étant donné que rien ne peut être trié si il n'arrive pas à retourner un `username` .

Notre injection aura donc une syntaxe du type:

username = ' and 1=2 UNION SELECT 0,0,0,0,0 #

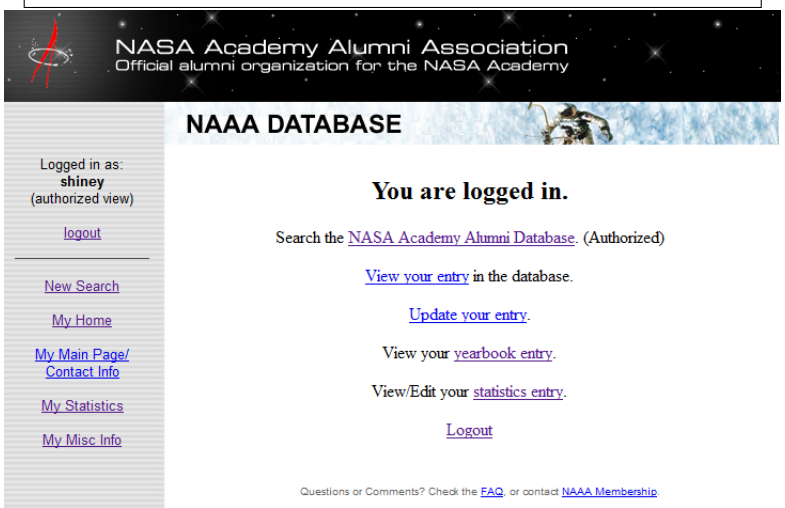


Nous nous sommes connectés en tant que : ' and 1=2 UNION SELECT 0,0,0,0,0 #. Il semblerait que le script accepte notre authentification, si la requête SQL renvoie un `password` qui a pour valeur 0.

2.3 Final

Pour le plaisir des yeux, nous allons mettre en commentaire HTML, la partie 'moche' de notre pseudo:

username = Shiney <!- - ' AND 1=2 UNION SELECT 0,0,0,0,0 # - ->



Conclusion

Cette démonstration exploite très clairement une injection SQL, nous aurions pu récupérer les identifiants d'un des comptes directement depuis la base de donnée. Cela aurait pris beaucoup plus de temps et nous nous serions exposés à de nombreuses contraintes notamment avec les `password` conservés en MD5. J'ai volontairement épargné certaines explications tel que la signification de l'injection 1 ou de l'injection 2 afin d'inviter le lecteur à poursuivre ses recherches dans les injections SQL classique et à l'aveugle.