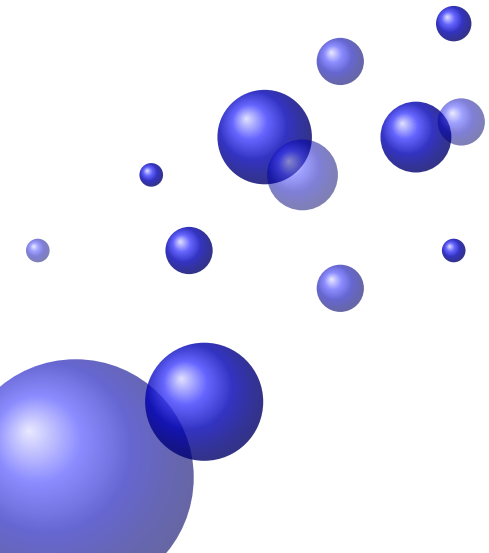
Distribution trouée Damn Vulnerable Web Application

Florian Barbarin - Abdelkader Beldjilali - Alexis Letombe



Le 24 avril 2017

Table des matières

Introduction

Comme TV5 monde en 2015, les pertes des entreprises victimes de cyberattaques se comptent souvent en dizaines de millions d'euros. L'Open Web Application Security Project (OWASP : https://www.owasp.org), publie régulièrement la liste des 10 menaces les plus critiques qui concernent les applications web. Une manière de s'en prémunir consiste à pratiquer le hacking web éthique. C'est là qu'entre en scène l'outil DVWA. Damn Vulnerable Web App (http://www.dvwa.co.uk), est un environnement PHP qui permet de se former à la sécurité des sites web. Le hacker en herbe peut ainsi se former et tester légalement ses compétences sur une application hébergée en local.

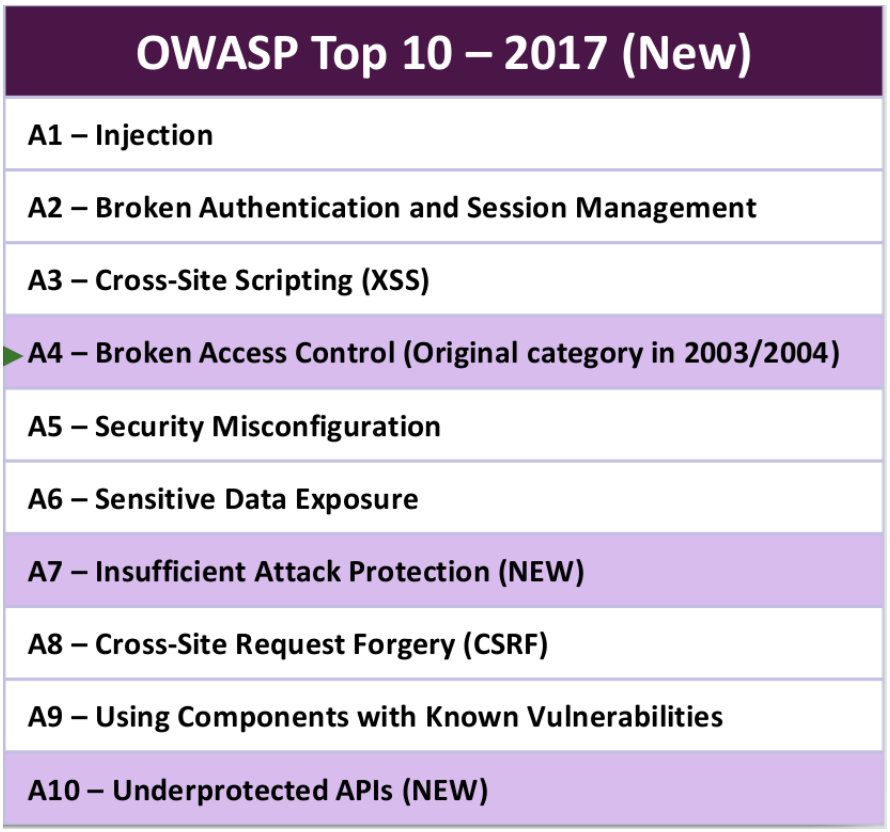


FIGURE 1 – Le top 10 des menaces web 2017 publiées par l'OWASP

1 Installation de la plateforme de test

Une solution possible consiste à installer la distribution kali et le site web DVWA sur une machine virtuelle virtualBox. Il suffit ensuite de modifier la configuration réseau NAT de la machine virtuelle en accès par pont. La commande hostname -I sur la machine virtuelle fournit son adresse IP, par exemple 192.168.1.8. Il ne reste plus qu'à se connecter via firefox au site dvwa via http ://192.168.1.8/dvwa/. Le login par défaut est admin/password.

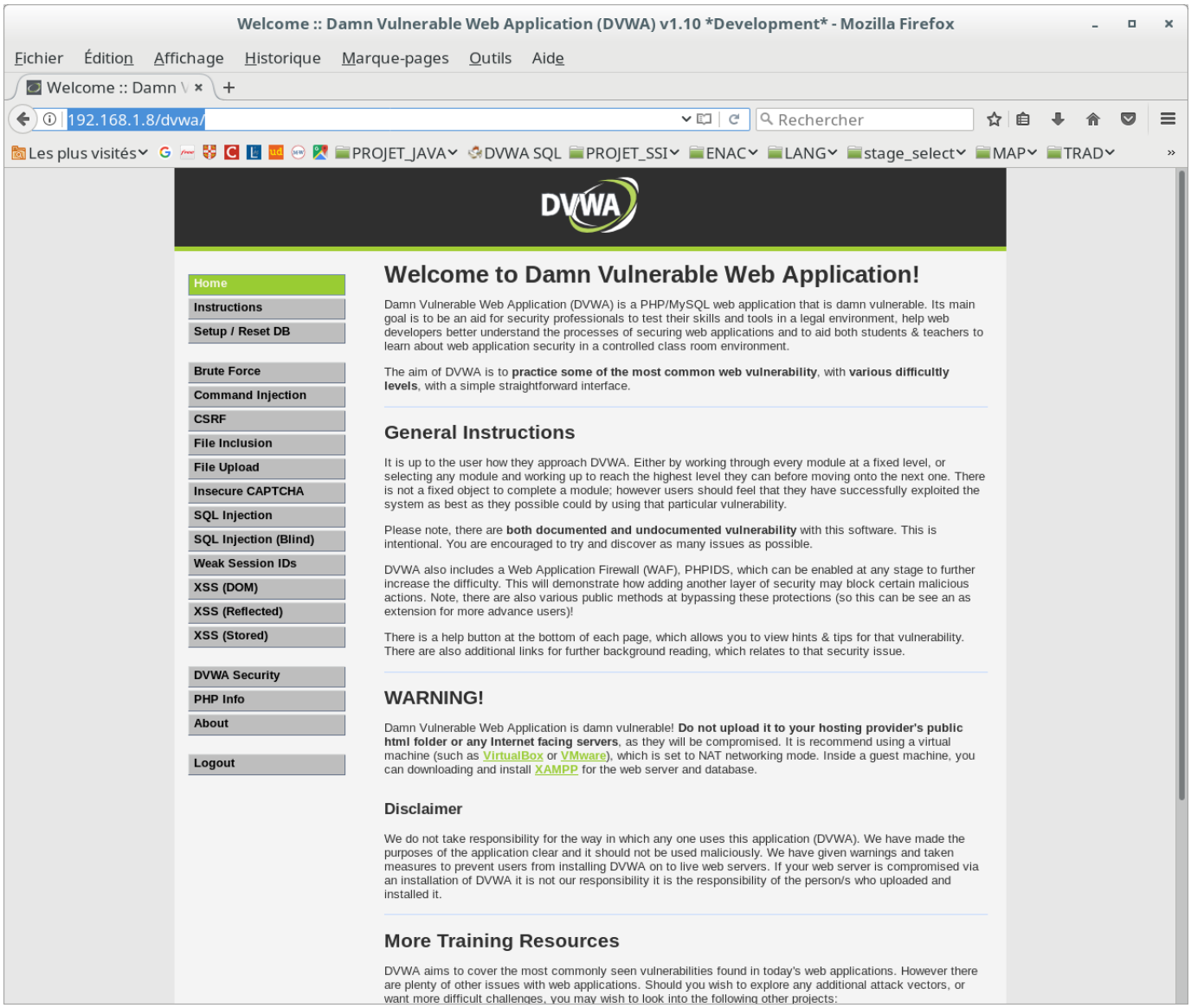


FIGURE 2 – Accès à DVWA à partir du navigateur de la machine hôte. Le site web DVWA est installé sur une machine virtuelle kali linux.

2 Titre partie 1

Comme nous avons pu le voir en introduction de cette étude, le point de départ d'un système cryptographique est de trouver un problème dans la classe **NP** voire **NPC** avant de l'adapter aux exigences de chiffrement et déchiffrement des messages. Une fois ces étapes explicitées, nous verrons comment mettre en œuvre le chiffrement de Merkle-Hellman.

2.1 Choix d'un problème difficile

2.2 Adaptation du problème

2.2.1 Fonction à sens unique

2.2.2 Introduction d'une trappe

2.2.3 Perturbation

2.2.4 Application au chiffrement à clé publique

Dans le cadre d'un système de chiffrement à clé publique, si Alice souhaite envoyer un message à Bob, cette première chiffre son message avec la clé publique de Bob qui le déchiffrera alors avec sa clé privée. Il est donc

nécessaire de définir les éléments constitutifs de la clé publique et de la clé privée de Bob.

Clé privée de Bob Comme nous l'avons esquissé à la section 2.2.3, Bob va tout d'abord générer une suite d'entiers super-croissante

— dans le « pire cas » et par le caractère super-croissant de la clé privée, le dernier élément s_n vaudra $D2^{n-1}$;

— on aura alors p de l'ordre de grandeur de $D2^n$ à $FD2^n$;

— les éléments de la clé publique T sont majorés par p par le modulo et un bloc chiffré y sera donc borné par $n \times FD2^n$.

Ce dernier bloc est chiffré comme les précédents. La figure 3 illustre le chiffrement d'un flux d'entrée avec une clé de taille 11. Cette méthode de bourrage est simple et ne crée pas d'ambiguïté puisque l'on sait dans les deux cas de figure que l'information utile se termine exactement au bit qui précède le premier bit non nul, en partant de la fin. Néanmoins, certaines méthodes de *padding* introduisent des vulnérabilités dans le code que la cryptanalyse pourra exploiter pour obtenir de l'information sur le message en clair. Il existe ainsi des méthodes standardisées de *padding* aléatoire que nous n'aborderons pas dans notre étude [1].

FIGURE 3 – Chiffrement d'un flux avec une clé de taille $n = 11$

1. Utiliser du *padding* aléatoire peut avoir l'avantage, y compris lorsque le message a déjà une taille multiple de celle d'un bloc, de produire des messages chiffrés différents pour un même message clair.

3 File Inclusion

De nombreux langages de programmation permettent d'inclure des portions de code contenues dans d'autres fichiers que celui en cours d'exécution. Le mécanisme mis à disposition permet de recopier dans le script principal le code contenu dans un autre fichier. Cette procédure est transparente à l'œil de l'utilisateur et peut être très avantageuse pour le développeur d'un site internet.

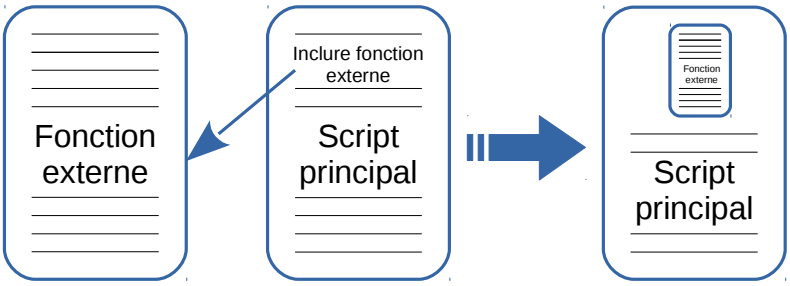


FIGURE 4 – Mécanisme d'inclusion d'un fichier

En effet, inclure du code contenu dans un autre fichier permet, entre autre, les deux utilisations suivantes :

— inclure des portions de code différentes en fonction de choix de l'utilisateur ou de l'environnement de ce dernier ;

— inclure des portions de code utilisées dans plusieurs scripts (par exemple une fonction de connexion à une base de données) afin de ne pas avoir à recopier les mêmes lignes à différents endroits et de ne modifier qu'un seul fichier en cas de modification de la fonction.

Nous voyons donc que le premier point ci-dessus permet d'obtenir une réelle adaptabilité du code alors que le second point donne la possibilité au développeur d'écrire du code concis et factorisé. Nous allons cependant voir que ce mécanisme n'est pas dépourvu de vulnérabilités.

3.1 Description de la vulnérabilité

La principale vulnérabilité connue dans le mécanisme que nous venons d'expliciter intervient lorsque l'inclusion d'un script est gérée par une variable pouvant être contrôlée par un attaquant. On se retrouve alors plutôt dans le premier cas d'utilisation indiqué, c'est à dire inclure des portions de code différentes en fonction de choix de l'utilisateur ou de l'environnement de ce dernier. En effet, dans le second cas d'utilisation, l'inclusion du fichier est généralement écrite "en dur" dans le script principal et ne peut donc pas être facilement modifiée par un attaquant.

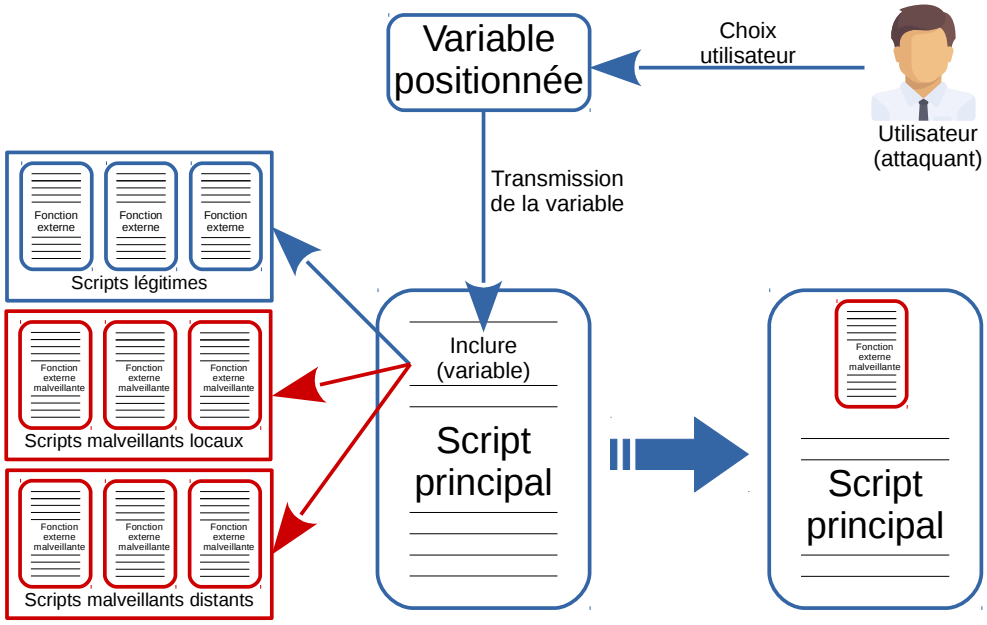


FIGURE 5 – Vulnérabilité d'inclusion d'un fichier

On remarque dans le schéma 3.1 que dans le cas où un attaquant peut avoir accès à la variable permettant de sélectionner le script légitime, celui-ci peut en modifier le contenu de deux façons. On parle alors de Local File Inclusion (LFI) et de Remote File Inclusion (RFI).

3.1.1 Local File Inclusion

Une fois que l'attaquant est en capacité de modifier le contenu de la variable indiquant le nom du script à inclure, celui-ci peut y indiquer un chemin local (i.e. directement sur le serveur) vers un script contenant du code malveillant. Il peut s'agir d'un script que l'attaquant a au préalable placé sur le serveur ou d'un script déjà présent qui effectue des opérations pouvant porter atteinte à la disponibilité de la machine voire à l'intégrité ou la confidentialité des données.

3.1.2 Remote File Inclusion

L'attaquant peut également indiquer dans la variable un chemin distant (i.e. vers un autre serveur) pointant vers un script contenant du code malveillant. Cette technique a pour avantage de faciliter la gestion du contenu du script malveillant par l'attaquant qui peut y inclure toutes les fonctionnalités qu'il souhaite voire le faire évoluer en fonction de la réponse de la machine attaquée.

Dans les deux cas, les scripts malveillants sont recopiés au sein du code du script principal qui sera au final exécuté par le serveur. Cette vulnérabilité offre donc de vastes possibilités à un attaquant qui peut alors faire exécuter par un serveur n'importe quelles fonctionnalités qu'il souhaite.

3.2 Exploitation de la vulnérabilité

3.3 Contre-mesure

4 File Upload

De très nombreuses applications web laissent la possibilité à l'utilisateur de charger des fichiers qu'il détient en local sur son propre ordinateur. On peut bien entendu citer à ce titre les serveurs de messagerie électronique ainsi que les clients web permettant la gestion d'un *cloud*. En effet, sans cette fonctionnalité de chargement, il serait impossible de joindre un fichier à mail ou même de sauvegarder ses fichiers dans le *cloud*.

Nous allons voir que cette fonctionnalité doit être bien encadrée et supervisée afin de ne pas créer de très importantes vulnérabilités.

4.1 Description de la vulnérabilité

4.1.1 Généralités

Le chargement d'un fichier local vers un serveur web est une fonctionnalité nécessaire voire inhérente à certaines applications web. Sans cette possibilité, certaines applications n'auraient même pas de raison d'être. Ainsi, *Gmail*, *Google Drive* ou *DropBox* reposent sur cette possibilité. Il est donc nécessaire de trouver le moyen de faire face aux différentes vulnérabilités de cette fonction.

Un grand nombre d'attaques sur les applications web se déroulent en deux phases :

1. injection d'un code malicieux sur le serveur web ;
2. exécution du code malicieux précédemment injecté.

On peut donc clairement voir que le chargement d'un fichier local vers un serveur web offre une possibilité claire à un attaquant de charger sur le serveur web un fichier contenant du code malicieux. Il ne lui restera alors plus qu'à trouver un moyen d'exécuter ce code.

De plus, il est à noter que n'importe quel type de script peut, grâce à cette fonction, être importé sur le serveur web. Cela laisse donc la porte ouverte à un grand nombre d'attaques différentes : déni de service, défacement, prise en main du système, etc...

Afin de retirer ces possibilités d'action à un attaquant éventuel, il sera alors nécessaire de restreindre le chargement à certains types de fichiers ou d'effectuer certains contrôles.

4.1.2 Différentes vulnérabilités

La plateforme OWASP classe les vulnérabilités liées au chargement de fichiers en deux catégories : les vulnérabilités liées aux méta-données et les vulnérabilités liées à la taille ou au type des fichier.

Les vulnérabilités liées aux méta-données : Ces vulnérabilités sont dues aux différents champs échangés dans les requêtes HTTP, principalement le champ indiquant le chemin de destination du fichier ainsi que le champ indiquant le nom du fichier.

En effet, en modifiant les champs des requêtes HTTP multi-part (utilisées dans le chargement et le téléchargement de fichiers sur un serveur web), il est possible d'écraser des fichiers déjà existants ou même d'atteindre des répertoires normalement non autorisés à l'écriture par un utilisateur.

Cette classe de vulnérabilités permet donc par exemple d'effectuer un défacement, d'insérer du code malicieux dans des dossiers stratégiques ou encore de modifier le comportement normal d'un script.

Les vulnérabilités liées à la taille ou au type des fichiers : Ces vulnérabilités sont principalement dues à l'absence de certaines vérifications avant le début du chargement du fichier.

Ainsi, la non vérification de la taille d'un fichier laisse la possibilité aux utilisateurs de charger des fichiers aussi volumineux qu'ils le souhaitent. Il est alors possible sur certains serveurs dont l'espace de stockage est limité d'atteindre la capacité maximale de cet espace et, par là, de rendre la fonctionnalité de chargement voire le serveur lui-même indisponible. On peut donc aboutir très rapidement à une attaque du type déni de service.

De même, la non-vérification du type de fichier peut permettre à un utilisateur de charger sur le serveur un script exécutable alors que le fichier attendu était par exemple une photographie. Un tel script pourra alors être *a posteriori* exécuté par l'attaquant lui donnant ainsi de nombreuses possibilités.

On voit donc que le chargement de fichier sur un serveur web est loin d'être anodin et doit être encadré afin d'éviter qu'un utilisateur ait à sa disposition un large éventail d'attaques possibles.

4.2 Exploitation de la vulnérabilité

4.3 Contre-mesure

5 Insecure CAPTCHA

De nombreux robots (ou *bots*) agissent de façon automatique sur internet afin, principalement, d'envoyer des messages, qu'ils soient publicitaires ou malveillants. La plupart des spams échangés sur internet sont d'ailleurs le fait de tels robots. Il est donc nécessaire, dans certaines applications web permettant notamment l'envoi de messages, de vérifier s'il s'agit d'un humain ou d'une machine qui souhaite effectuer cette opération.

C'est dans ce but que les contrôles CAPTCHA ont été créés. En effet, CAPTCHA est l'acronyme de *Completely Automated Public Turing test to tell Computers and Human Apart*. Comme son nom l'indique, ce test est censé permettre de confirmer ou non la présence d'un humain derrière la machine souhaitant réaliser une certaine opération. Ce test peut prendre plusieurs formes : reconnaissance de caractères déformés ou encore reconnaissance d'éléments dans un puzzle.

Pour que ce test soit efficace, il est nécessaire qu'il soit implémenté de façon correcte et, ainsi, ne pas laisser la possibilité à un attaquant d'exploiter une faille qu'il pourra par la suite réutiliser dans un script automatique.

5.1 Description de la vulnérabilité

Différentes catégories de vulnérabilités peuvent être mises en évidence concernant les contrôles CAPTCHA.

Transmission de la solution : La première et la plus simple à exploiter est lorsque la solution se trouve être transmise en texte clair au navigateur client. Il ne suffit que de repérer l'endroit où se trouve la solution pour réaliser un script qui résoudra à coup sûr chaque contrôle CAPTCHA. En effet, les solutions peuvent être passées en argument de l'URL, dans le nom de l'image, dans un champ caché d'un formulaire HTML ou encore en commentaire du fichier HTML chargé.

Mauvaise vérification de la réussite au test : La deuxième catégorie de vulnérabilités concerne le fait que le contrôle CAPTCHA peut être outrepassé si la vérification de la réussite au test n'est pas suffisamment sécurisée. En effet, la ou les variables indiquant une réussite ou non au test doivent être protégées et ne pas être facilement modifiées sans avoir à réaliser le test au préalable. Dans le cas contraire, il est encore possible de trouver une parade à la réalisation du test par un humain et de l'implémenter dans un script automatique.

Résolution automatique : Enfin, la troisième catégorie de vulnérabilités concerne la possibilité de résoudre de façon automatique les contrôles CAPTCHA. Dans cette catégorie, beaucoup de problèmes sont à prendre en compte : la difficulté du problème, le temps de résolution du problème ou encore l'étendue de la base de données CAPTCHA. En effet, ce dernier point est important car si la base de données d'images par exemple n'est pas assez grande, une entité

pourrait facilement trouver un avantage financier à payer des personnes pour résoudre tous les problèmes CAPTCHA afin de pouvoir les exploiter automatiquement par la suite.

5.2 Exploitation de la vulnérabilité

5.3 Contre-mesure

6 Injection SQL

6.1 Description

L'injection SQL, en anglais SQL injection ou SQLi en abrégé, est une des attaques les plus dangereuses. Comme pour le Cross Site Scripting présenté dans la suite de ce document, il s'agit ici de tirer parti de l'absence de filtrage des entrées utilisateurs. Cette absence de contrôles permet à un hacker d'insérer du code qui sera interprété par l'analyseur cible, par exemple SQL.

Dans la cas particulier de l'injection SQL et du site DVWA, les requêtes SQL, imbriquées dans des scripts PHP qui récupèrent les saisies des utilisateurs, peuvent être détournées sur la base de la syntaxe du langage.

6.2 Exploitation

La base de données contient 5 utilisateurs identifiés par les entiers de 1 à 5. La mission proposée par le DVWA est de voler leurs mots de passe par injection SQL.

6.2.1 DVWA - level "low"

On règle la "DVWA security" sur low de manière à avoir un site web "damn vulnerable". On saisit dans le champ User Id, une simple apostrophe i.e. '. Le site retourne le message "*You have an error in your SQL syntax ; check the manual that corresponds to your MariaDB server version for the right syntax to use near ''''' at line 1*"

Cette simple apostrophe démontre que le site est vulnérable pour deux raisons : d'abord on sait que nos saisies sont interprétées directement par l'analyseur SQL ; elle ne sont pas filtrées. Ensuite parce que le site est "bavard".

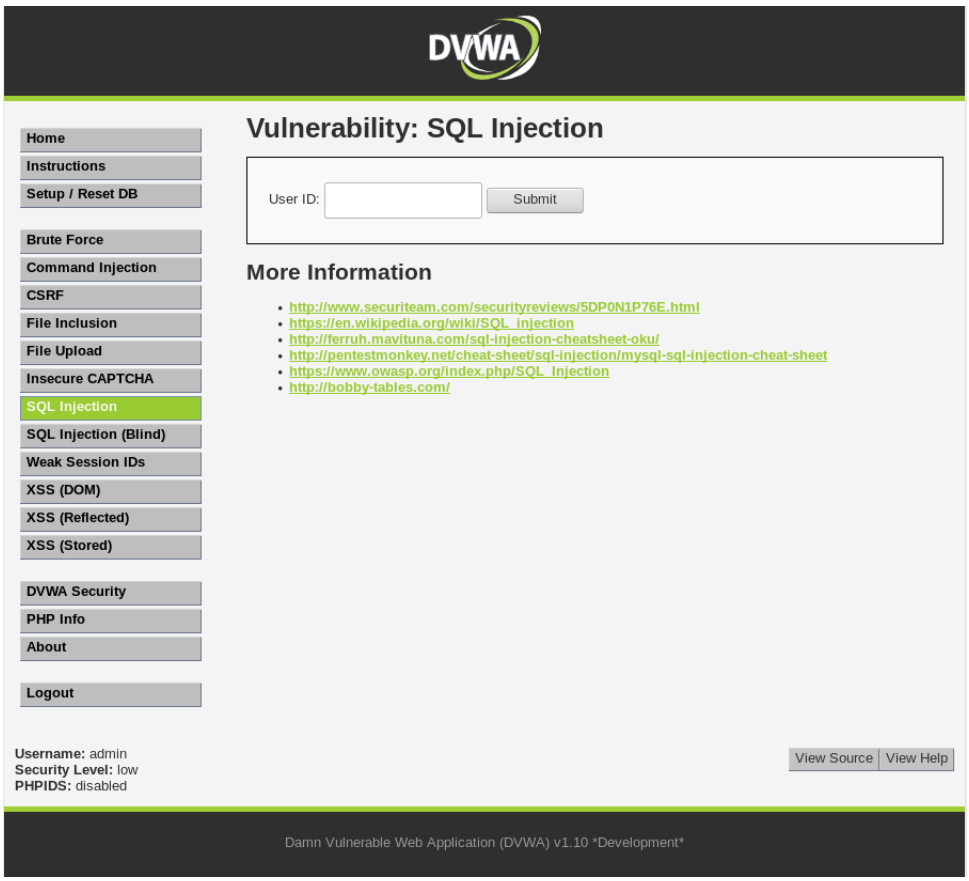


FIGURE 6 – Les saisies incorrectes donnent lieu à un message d'erreur SQL qui informe le hacker potentiel de l'absence de protection contre les SQLi. D'autres informations importantes sont dévoilées comme le type de base de donnée, ici MariaDB, version libre de MySql rachetée par Oracle.

Un clic sur le bouton "View Source" affiche le code PHP de la page. On constate, en effet, qu'on peut saisir n'importe quoi dans le champ User Id, il sera transmis sans modification à la requête $query via $id.

```
// Check database
$query  = "SELECT first_name, last_name FROM users WHERE user_id = '$id';";
```

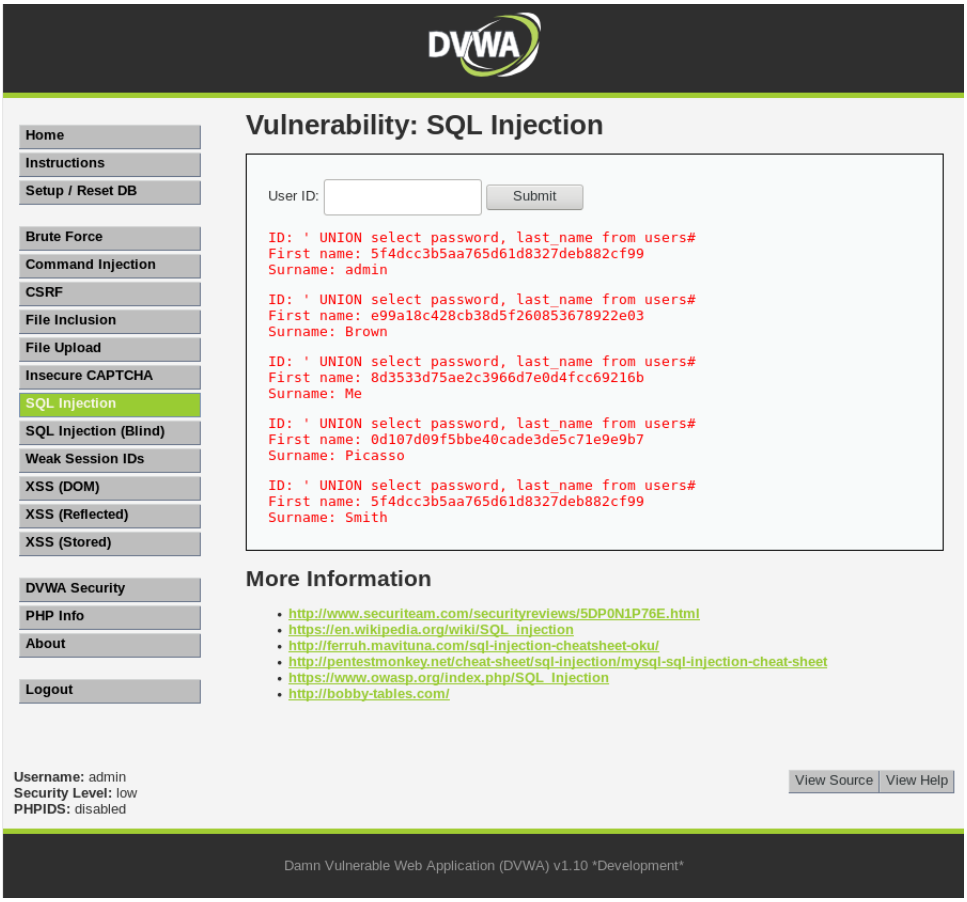


FIGURE 7 – Vol des mots de passe par l'injection SQLi sur site non protégé.

L'injection SQL suivante :

```
' UNION select password, last_name from users#
```

donne la requête suivante en remplaçant $id dans le script PHP :

```
$query  = "SELECT first_name, last_name FROM users WHERE user_id = '' UNION
          SELECT password, last_name from users#   ';";
```

Elle indique donc qu'on effectue l'union au sens mathématique des éléments recueillis par les deux requêtes. Le premier SELECT donne l'ensemble vide, le second donne tous les password et noms de la table users. On obtient donc les mots de passe faussement associés au champ "First name". Ces mots de passe sont cryptés. On pourra utiliser des techniques de révélation par ingénieurie sociales, recherche internet, force brute, dictionnaires ou rainbow tables. On note que Smith est certainement aussi admin car ces deux noms d'utilisateurs ont le même hash donc le même mot de passe. Le hacker peut être confiant quant à la suite des opération car Smith ne semble pas être un adepte de la SSI. Le caractère # en fin d'injection évite que PHP n'interprête la suite du code en particulier les caractères apostrophes et guillements qui donneraient une erreur SQL.

NB : C'est une technique répandue que de forcer l'analyseur SQL à ignorer le reste de la requête, en utilisant le symbole commentaire SQL double tiret - - ou les symboles de commentaires PHP dièse #, / */, // pour assurer que ce qui suit l'injection ne sera pas interprété.*

6.2.2 DVWA - level “medium”

6.2.3 DVWA - level “high”

6.3 Contre-mesure

7 Injection SQL aveugle

Blind SQL injection ou BSQLi

7.1 Description

7.2 Exploitation

7.3 Contre-mesure

8 Attaques XSS

Reflected XSS

8.1 Description

désignées au choix par les acronymes CSS ou XSS et qui seront Un site web qui fournit d'une part un service de recueille d'informations via des formulaires et d'autre part un service de publication sur le site de ces mêmes informations

8.2 Exploitation

8.3 Contre-mesure

9 Attaques XSS enregistrées

9.1 Description

9.2 Exploitation

9.3 Contre-mesure

10 Conclusion et perspectives

Bien que le problème de somme de sous-ensembles (SSP) soit NP-complet, des algorithmes peuvent être mis en place pour tenter de casser le cryptosystème de Merkle-Hellman dans sa version basique. Si les algorithmes de force brute montrent très vite leurs limites, l'apport de théories plus ou moins récentes telles que la géométrie des nombres et la notion de réseau permet d'envisager des algorithmes nouveaux et élégants, bien que leur succès soit loin d'être garanti. Enfin, tous les sacs à dos ne se valent pas. Comme nous avons eu l'occasion de le voir, certaines méthodes comme la programmation dynamique ont horreur des sacs à dos dilatés alors que la densité est l'ennemie de l'algorithme LLL. La diversité des méthodes de cryptanalyse et la difficulté de se protéger contre l'une sans s'exposer à une autre justifie pleinement l'abandon de ce cryptosystème au profit de systèmes réputés plus sûrs, du moins à l'heure actuelle, comme RSA.

Il n'en reste pas moins que le SSP reste un problème intéressant à étudier sur le plan théorique et pratique. L'application de méthode hybrides associant les techniques de programmation dynamique et arborescentes avec heuristiques ont fait l'objet de nombreuses publications. Enfin, appliquer des algorithmes endémiques au domaine de l'intelligence et de l'apprentissage artificielle, tels les « colonies de fourmis », les algorithmes génétiques ou les réseaux de neurones pourraient fournir des pistes plus innovantes.

A Annexe

— *Pour écrire simplement les clés sur le disque, nous utilisons Marshal, qui garantit la compatibilité entre toutes les plateformes pour une même version de OCaml.*

— *Bien que BatIO propose une API pour manipuler les canaux au niveau du bit, nous avons préféré rester au niveau de l'octet car il s'agit d'une solution plus évolutive – rares sont les bibliothèques proposant ce genre de fonctions. D'ailleurs, son fonctionnement est identique à ce que nous implémentons, reposant sur une lecture octet par octet.*

— *Dans la version actuelle du code, les canaux d'entrées-sorties ne sont pas toujours fermés proprement lorsqu'une exception « fatale » est rencontrée. . .*

— *Les blocs chiffrés sont écrits sur le canal de sortie sous forme de chaîne de caractères. Ainsi, le message chiffré constitué de deux blocs « 1234 5678 » est écrit, sous forme hexadécimale, « 31 32 33 34 00 35 36 37 38 00 ». Pour déchiffrer, on lit donc le canal d'entrée « chaîne par chaîne ». Cela a l'avantage de produire une sortie lisible mais présente l'inconvénient de consommer bien plus d'espace qu'une représentation binaire qui serait spécialement conçue pour le problème.*

B Annexe

— *Pour écrire simplement les clés sur le disque, nous utilisons Marshal, qui garantit la compatibilité entre toutes les plateformes pour une même version de OCaml.*

— *Bien que BatIO propose une API pour manipuler les canaux au niveau du bit, nous avons préféré rester au niveau de l'octet car il s'agit d'une solution plus évolutive – rares sont les bibliothèques proposant ce genre de fonctions. D'ailleurs, son fonctionnement est identique à ce que nous implémentons, reposant sur une lecture octet par octet.*

— *Dans la version actuelle du code, les canaux d'entrées-sorties ne sont pas toujours fermés proprement lorsqu'une exception « fatale » est rencontrée. . .*

— *Les blocs chiffrés sont écrits sur le canal de sortie sous forme de chaîne de caractères. Ainsi, le message chiffré constitué de deux blocs « 1234 5678 » est écrit, sous forme hexadécimale, « 31 32 33 34 00 35 36 37 38 00 ». Pour déchiffrer, on lit donc le canal d'entrée « chaîne par chaîne ». Cela a l'avantage de produire une sortie lisible mais présente l'inconvénient de consommer bien plus d'espace qu'une représentation binaire qui serait spécialement conçue pour le problème.*

Damn Vulnerable Web App (DVWA) is a PHP/MySQL web application that is damn vulnerable. Its main goals are to be an aid for security professionals to test their skills and tools in a legal environment, help web developers better understand the processes of securing web applications and aid teachers/students to teach/learn web application security in a class room environment.