R5.B.09
« Cybersécurité »
TP 2 : Attaques Actives

L'objectif de ce TP est de connaître les dangers des attaques actives, plus précisément au niveau de l'intégrité et sécurité des sites webs et bases de données. Vous allez être confrontés à un site web simple pouvant insérer des éléments dans une base de données, et en afficher le contenu. Vous allez dans un premier temps vous adresser a la sécurité de la base de données, avant de tourner votre attention à page web elle même. À la fin du TP, vous devez rendre un rapport écrit (PDF) contentant vos réponses aux questions posés, les traces de votre analyse et les preuves de vos commandes pour appuyer vos propos.

Bon courage et sécurisez ce site web !

Il est évident que les attaques vues pendant ce TP ne sont pas a utiliser à l'extérieur ! Ces attaques peuvent avoir de lourdes conséquences à la fois sur l'entité ciblé, mais également sur vous [1] [2] !

La préparation ...

Lors de ce TP, vous allez étudier, exploiter et sécuriser une application web reposant sur Python avec une base de données SQLite. Dans un premier temps, nous allons configurer l'espace Python ainsi qu'étudier la composition de la base de données associé au site web.

Un monde virtuel Python

Pour vous aider dans votre travail, nous allons utiliser les environnements virtuels de Python3. Ces environnements sont une sorte d'installation « indépendante » et « auto-suffisante » de Python, dans lequel vous pouvez installer les modules dont vous avez besoin, sans impacter votre système. De plus, le tout se situe dans un répertoire désigné, facilitant son nettoyage à la fin du TP.

Il y a plusieurs façons de mettre en place un environnement virtuel, tel que conda et anaconda. Pour ce TP, nous allons présenter l'utilisation de l'environnement virtuel de Python elle même `python3-venv`. Bien sur, vous pouvez utiliser l'environnement que vous voulez.

Dans un premier temps, installez le package afin d'avoir accès à l'environnement virtuel :

```
sudo apt install python3-venv
```

Une fois installé, créez votre environnement virtuel a l'emplacement souhaité avec la commande suivante :

```
python3 -m venv <folder>
```

en remplaçant `<folder>` par le nom du répertoire dans lequel vous voulez mettre votre environnement.

Par exemple : `python3 -m venv TP_attaques`.

Le réveil sonne

Avant de pouvoir travailler dans votre nouvelle espace virtuel, il faut l'activer. Mettez la commande suivante :

```
source <folder>/bin/activate
```

en replaçant `<folder>` par le chemin vers votre espace crée précédemment.

Par exemple : `source TP_attaques/bin/activate`.

1. https://www.vice.com/en/article/teen-security-researcher-bill-demirkapi-suspended-for-exposing-vulnerabilities/
2. https://bustbyte.no/blog/how-we-hacked-blackboard-and-changed-our-grades

A partir de maintenant, et jusqu'à ce que vous fermez votre terminal, tout commande python lancé sera faite a l'intérieur de votre environnement virtuel. Vous pouvez le confirmer avec l'apparition du nom de votre environnement au début de votre ligne de commande.

Pour la suite du TP, tout commande python devrait se faire dans votre environnement. Si vous avez utilisé un autre outil de gestion d'environnement virtuel, vous devez être à l'intérieur de votre environnement à partir de ce point.

Installation des modules Python

Dans ce TP nous allons utiliser deux modules Python pour faire fonctionner notre site web :
— **cherrypy** - un framework d'applications web en python, utilisé pour héberger et rendre disponible le site web utilisé.
— **sqlite3** - une interface entre l'application python et une base de donnée SQLite.
Installez ces modules avec la commande `pip3`

```
pip3 install cherrypy
pip3 install sqlite3
```

La table SQLite

Pour ce TP, nous utilisons une base de données SQLite, un moteur petit, rapide et très complet, reposant sur l'utilisation de fichiers de base de données. Ainsi, nous n'avons pas besoin d'utiliser des serveurs SQL, facilitant la mise ne place et la gestion des données.

Ici, le fichier de base de données vous est fourni sur moodle `application.db` contenant une table `vulnerable_tab` dont la structure est la suivante.

```
CREATE TABLE vulnerable_tab (
    id integer primary key autoincrement,
    txt varchar(255) not null,
    who varchar(255) not null
);
```

Vous pouvez installer le package `sqlite3` sur Linux pour interagir directement avec la base de données via le terminal. Cela vous permettrai de visualiser, modifier, ainsi que supprimer des lignes de la table pendant le TP.

1 Lancement et analyse de l'application

Téléchargez le fichier `application.py` sur moodle avec sa base de données `application.db`. Lancez l'application et observez la sortie dans le terminal. Si vous avez des erreurs, vérifiez que vous êtes bien dans votre environnement virtuel, que vous avez installé les deux modules Python, et que le port 8080 de votre machine n'est pas occupé. Si tout fonctionne, vous devez voir une sortie sur le terminal comme ceci :

```
[30/Sep/2024:23:22:25] ENGINE Serving on http://127.0.0.1:8080
```

Vous devez garder ce terminal ouvert pendant toute l'utilisation de l'application. Pour l'arrêter, faites un `CTRL+C` dans votre terminal, et l'application s'arrêtera.

Maintenant ouvrez votre navigateur et naviguez à l'adresse indiqué sur votre terminal. Dés à présent vous devez voir la page s'afficher. Testez son bon fonctionnement en ajoutant quelque chose à la base de données. Vérifiez qu'elle apparaît bien sur la page par la suite. Prenez un moment avant de continuer pour regarder le code HTML de la page web, ainsi que le code source de l'application.

L'objectif de cette application est simple : ajoutez une chaîne de caractères dans la table de la base de données (dans la colonne txt), en ajoutant également l'adresse IP de la machine source (dans la colonne who).
Vous pouvez regarder la documentation de CherryPy [3] ou encore SQLite3 [4] si vous le souhaitez.

3. https://cherrypy.org/
4. https://docs.python.org/3/library/sqlite3.html

2 Première vulnérabilité - L'injection SQL

INFO *- Une attaque par injection SQL survient lorsqu'une requête SQL n'est pas construite de manière sécurisé. Par exemple, via la concaténation de chaînes de caractères, à partir d'éléments extérieurs fourni par un utilisateur potentiellement malveillant.*

2.1 Qu'est ce que l'injection SQL ?

Pour comprendre le principe, prenons un exemple. Supposons qu'on possede une table SQL nommé « `table` » avec un champ nommé « `field` ». Ainsi, une requette SQL pourrait ressembler à ceci :

```
value = "text"
requette = "SELECT * FROM table WHERE field='" + value + "';"

-- Requette execute
SELECT * FROM table WHERE field='text';
```

Maintenant supposons que la valeur de `value` est contrôlée par l'utilisateur. Il est donc possible d'ajouter autre chose qu'une simple valeur afin d'altérer la requête d'origine. Un exemple de ceci est l'utilisation de la commande `' OR 1=1; -`, qui retournera tout le contenu de la table. En ajoutant également le symbole `-` à la fin de la commande, tout ce qui est après est considéré comme un commentaire (comme // en C, # en Python, etc...), et est donc ignoré. Il ne faut pas oublier la présence du guillemet simple `'` afin de fermer les guillemets qui sont autour du variable `value`. Ainsi la commande SQL ressemblera ceci :

```
value = "' OR 1=1; --"
requette = "SELECT * FROM table WHERE field='" + value + "';"

-- Requette execute
SELECT * FROM table WHERE field='' OR 1=1; --';
```

Il est également possible de pousser l'attaque plus loin, en ajoutant directement une autre commande SQL, ar exemple `';DROP TABLE table; -`. La commande finale serait donc ceci :

```
value = "';DROP TABLE table; --"
requette = "SELECT * FROM table WHERE field='" + value + "';"

-- Requettes executes
SELECT * FROM table WHERE field='';
DROP TABLE table; --';
```

Avant de démarrer cette exercice, regarder le code source de l'application `application.py` et identifiez la vulnérabilité d'injection SQL. N'hésitez pas a ajouter des traces dans le code Python pour vous aider (ex : `print(request)`).

→ *Notez le et expliquer pourquoi.*

2.2 Méthodes de protection préliminaires

Examinez à présent le code HTML dans votre navigateur. Vous remarquez que du code JavaScript a été ajouté afin de sécuriser l'exploitation de cette vulnérabilité.

→ *Présentez ce mécanisme et son fonctionnement.*

→ *D'après vous, est-ce-que ce mécanisme est efficace ? Expliquez votre réponse.*

Utilisez l'outil `curl` et formulez la commande qui permettrai d'envoyer les données du formulaire au serveur, sans passer par cette validation. Utilisez la documentation de la commande `curl` [5] pour définir la structure approprié.

Essayez d'insérer des chaînes de caractères dans la base de données qui ne sont pas autorisés par la validation précédent. Vérifiez sur le site web que l'insertion a bien eu lieu. Comparez en passant la même chaîne via le formulaire en ligne.

→ *Notez et expliquer votre commande* ***`curl`*** *et vos observations.*

5. https://ec.haxx.se/http/index.html

2.3 Exploitation de la vulnérabilité

Reprenez votre commande `curl` et proposez une injection SQL permettant de replacer le contenu du champ `who` par une valeur de votre choix. Vérifiez que votre commande fonctionne. Vous devez voir une différence avec les données précédents.

→ *Présentez et expliquer votre commande* `curl` *et votre raisonnement derrière votre injection SQL.*

Votre base de données `application.db` ne contient pas seulement une table, mais une autre table nommé `hidden_tab`. Proposer une injection SQL permettant d'afficher le contenu de cette table. Vous remarquez une ligne en particulier vous demande de faire quelque chose. Proposez une autre injection SQL afin de l'effectuer.

→ *Présentez et expliquer vos deux commandes* `curl` *et leurs injections SQL. Donnez le résultat des deux commandes pour illustrer votre raisonnement.*

2.3.1 Pour aller plus loin ...

INFO *- Cette section comprend des exercices complémentaires pour ceux qui avancent rapidement. Il est recommandé de finir l'ensemble du TP avant de revenir ici.*

Utilisez vos connaissances en SQL pour évaluer la base de données `application.db` et récupérer le maximum d'informations possible. Pour chaque attaque, présentez votre commande `curl` ainsi que l'injection SQL associé. Donnez le résultat et expliquer ce que cela vous donne comme informations sur la BDD. Tout est permis sur cette base de données. **Attention** cependant à ne pas supprimer la table `vulnerable_tab` qui et nécessaire au bon fonctionnement de l'application. N'hésitez pas a faire un backup si vous voulez tester des injections.

Évaluez bien la BDD, il se peut que des points bonus soient cachés quelque part ...
Rendez votre BDD avec votre rapport si vous avez réussi.

2.4 Sécurisation de l'application

Maintenant que vous avec une bonne compréhension de la vulnérabilité, proposez une solution pour sécuriser l'application nommé `application_secure.py`. Essayez d'exploiter la vulnérabilité de nouveau avec vos commandes `curl`.

→ *Expliquez votre approche et comment votre sécurisation fonctionne. Donnez les preuves du fonctionnement de vos commandes* `curl` *sur la BDD.*

3 Deuxième vulnérabilité - XSS

INFO *- Une attaque par XSS, ou « Cross Site Scripting » survient lorsque des balises HTML sont injectés dans une page web. Ces balises sont interprétés par le navigateur, ayant des conséquences inattendus et potentiellement malveillant sur l'utilisateur.*

3.1 Qu'est ce que l'attaque XSS ?

Pour comprendre le principe, prenons de nouveau un exemple. Comme avec notre attaque SQL, imaginons un serveur web qui génère un affichage a partir d'une variable. Le serveur pourrait donc ressembler à ceci :

```
1    name = "Edward"
2    web_page = "<p>Bonjour, " + name + "</p>"
3
4    <!-- HTML -->
5    <p>
6        Bonjour, Edward
7    </p>
```

De nouveau, nous supposons que la valeur de `name` est contrôlé par l'utilisateur. Il est donc possible d'ajouter autre chose que du simple texte, par exemple des balises HTML. Un exemple est l'ajout de bases d'affichage, par exemple `<h1>Edward</h1>`, qui changera la taille de la police du

texte rentré. Contrairement au SQL, nous n'avons pas de commentaires ou de guillemets à gérer. Ainsi, notre HTML ressemblera ceci :

```
name = "<h1>Edward</h1>"
web_page = "<p>Bonjour, " + name + "</p>"

<!-- HTML -->
<p>
    Bonjour,
    <h1>
        Edward
    </h1>
</p>
```

Bien que changer des éléments d'affichage est un bon jeu, on peut également aller beaucoup plus loin. Par exemple, on peut injecter directement du code JavaScript dans la page comme `<script>alert("You have been hacked!!")</script>`, afin d'afficher une boite d'alerte dans le navigateur. Bien sur, il ne faut pas oublier de spécifier que ce que l'on ajoute est bien du JavaScript avec les balises `<script>...</script>`. La structure finale serait donc ceci :

```
name = "<script>alert("You have been hacked!!")</script>"
web_page = "<p>Bonjour, " + name + "</p>"

<!-- HTML -->
<p>
    Bonjour,
    <script>
        alert("You have been hacked!!")
    </script>
</p>
```

Bien sur, une personne malveillante ne s'intéressera pas à changer l'affichage ou afficher des alertes, mais poussera la choses beaucoup plus loin, par exemple la redirection vers une autre page web avec potentiellement le vol des cookies.

3.2 Exploitation de la vulnérabilité

NOTE *- N'hésitez pas à utiliser un outil SQLite afin de supprimer des lignes de la table* ***`vulnerable_tab`*** *si vous avez besoin. Vous pouvez utiliser la commande SQLite3.*

Pour commencer, reprenez vos commandes `curl` et modifiez les afin d'ajouter une boite d'alerte sur votre site web.

→ *Présentez et expliquez votre commande* ***`curl`***.

Une vois vérifié que votre attaque a fonctionné, préparez une autre attaque, cette fois ci qui effectue une redirection vers un autre site web. Pour cette attaque, nous allons utiliser la commande `Netcat - nc`, qui est d'après lui le « couteau suisse de TCP/IP ». Il permet, entre autres, d'écrire et de lire des données à travers des connexions réseau, que ce soit via TCP ou UDP. Ici, il nous servira de site web « malicieux » qui réceptionnera la requête HTTP. Vous pouvez lancer ce serveur via la commande suivante

```
nc -l -p <numero de port>
```

avec `<numéro de port` un numéro de port disponible sur votre machine (i.e. 8081). Effectuez votre attaque et observé la sortie au niveau de `nc`.

→ *Présentez et expliquer votre commande* ***`curl`***. *Appuyez vous sur la sortie* ***`nc`*** *pour argumenter l'efficacité de l'attaque.*

→ *Que voyez vous au niveau du serveur de l'attaquant ? Pensez vous que cette attaque a été un succès ?*

3.3 Patch de sécurité de l'application

Étudiez quels méthodes existent pour « sécuriser » des entrées HTML sur un site web. Proposez un patch de sécurité à l'application `application_secure.py` que vous nommerez `application_patch.py`

→ *Expliquez votre méthode de sécurisation et pourquoi c'est important.*

→ *D'après vous, où vaut-il mieux réaliser cette sécurisation ? Au moment de l'insertion, de l'affichage, ou les deux ? Argumentez votre choix.*

Finaliser ce TP en essayant une dernière fois d'exploiter la vulnérabilité, cette fois sur votre nouveau patch. Vérifiez que votre attaque ne fonctionne plus.

4 Rendu

Pour ce TP vous devez rendre quatre éléments :
— **Rapport** - le rapport écrit en PDF avec vos réponses et votre raisonnement.
— **Application Sécurisé** - votre sécurisation de l'application pour l'injection SQL nommé `application_secure.py`.
— **Application Patché** - votre sécurisation de l'application pour l'attaque XSS nommé `application_patch.py`.
— **BDD** - votre base de données après les opérations demandés sur `hidden_tab`. Si vous avez fait l'exercice supplémentaire (section 2.3.1), vous pouvez le rendre ici également.
N'oubliez pas d'ajouter votre explication et votre démarche utilisé dans votre **rapport**.

Bravo, vous êtes à présent des cyber-attaquants !