

NET 5535

Projet de Fin d'Etudes

Outil de détection de vulnérabilité CSRF
(Cross Site Request Forgery)

Sommaire

| | |
|--|----|
| Introduction | 3 |
| I) La vulnérabilité CSRF | 4 |
| 1) Présentation | 4 |
| 2) Exemple classique | 4 |
| 3) Les méthodes rendant cette attaque « invisible » | 5 |
| A) L'utilisation du Cross Site Scripting XSS | 5 |
| B) L'utilisation de la balise html « image » | 6 |
| 4) Se protéger contre l'attaque CSRF | 6 |
| A) Réduire le risque d'exécution de la requête | 6 |
| B) Rendre la requête imprédictible pour l'attaquant : les jetons | 7 |
| C) Ajouter une autre étape d'authentification | 7 |
| D) Confirmer la demande d'exécution de l'opération | 8 |
| II) Présentation du projet de fin d'études | 8 |
| 1) Présentation du projet | 8 |
| 2) Gestion du projet | 8 |
| A) Rencontre avec la société VulnIT | 8 |
| B) Les spécifications de notre outil | 9 |
| C) Les principales étapes du projet | 9 |
| III) Le livrable | 10 |
| 1) Etat de l'art des logiciels existants | 10 |
| A) Pinata | 10 |
| B) Tamper-data | 11 |
| C) Web Scarab | 11 |
| D) CSRFTester | 12 |
| E) MonkeyFist | 12 |
| F) W3af | 13 |
| G) Arachni | 14 |
| 2) Présentation de l'algorithme | 14 |
| 3) Développement en Python et implémentation | 17 |
| A) Le langage Python | 17 |
| B) L'implémentation de l'algorithme | 19 |
| 4) Tests et résultats du logiciel | 26 |
| A) Google Gruyère | 27 |
| B) RecordPress | 30 |
| 5) Améliorations à apporter | 34 |
| IV) Conclusion | 35 |
| V) Annexes | 36 |
| 1) Module HTTP_request.py : | 36 |
| 2) Module handle_html.py : | 37 |
| 3) Module comparison_method.py : | 40 |

Introduction

Ce document a pour objectif de présenter le projet de fin d'études réalisé par les élèves de Lachèze-Murel Thibault et Rovelli Jérémie de Telecom Sudparis, portant sur la détection de la vulnérabilité « Cross Site Request Forgery » (CSRF).

Ce projet vise à développer un outil de détection de vulnérabilité CSRF, outil destiné à être intégré à VulnIT (Vulnerability Identification Tool) un outil de détection de vulnérabilités. Les encadrants pour ce projet sont Monsieur Debar Hervé, responsable de la spécialité SSR (Sécurité des Systèmes et Réseaux) à Telecom SudParis, et Monsieur Maury Vincent de la société VulnIT.

Ce document décrit premièrement ce qu'est une vulnérabilité CSRF ainsi que les méthodes permettant de s'en protéger. Une deuxième partie présente ensuite ce qui est attendu de l'outil de détection, et le plan adopté pour réaliser son développement. La suite du document décrit l'algorithme utilisé par l'outil, puis son implémentation en langage python. Enfin, ce document présente les différents tests réalisés ainsi que les améliorations apportées suite à ces tests.

I) La vulnérabilité CSRF

1) Présentation

Une attaque de type Cross Site Request Forgery (CSRF), a pour but d'utiliser le contexte authentifié de la victime afin de lui faire réaliser des opérations sensibles. La victime devient alors complice de l'attaque sans en avoir conscience. Pour ce faire, l'attaquant profite du système de sessions d'authentification utilisé par la victime. Cette attaque permet donc de contourner les systèmes d'authentification.

2) Exemple classique

Supposons que la victime vienne de consulter son compte bancaire sur le site web mabanque.com. Lors de sa connexion au site, la victime s'authentifie grâce à son identifiant et son mot de passe. Son navigateur enregistre donc un cookie de session, valable pendant un certain temps, afin de ne pas avoir à s'authentifier à chaque requête http sur le site web.

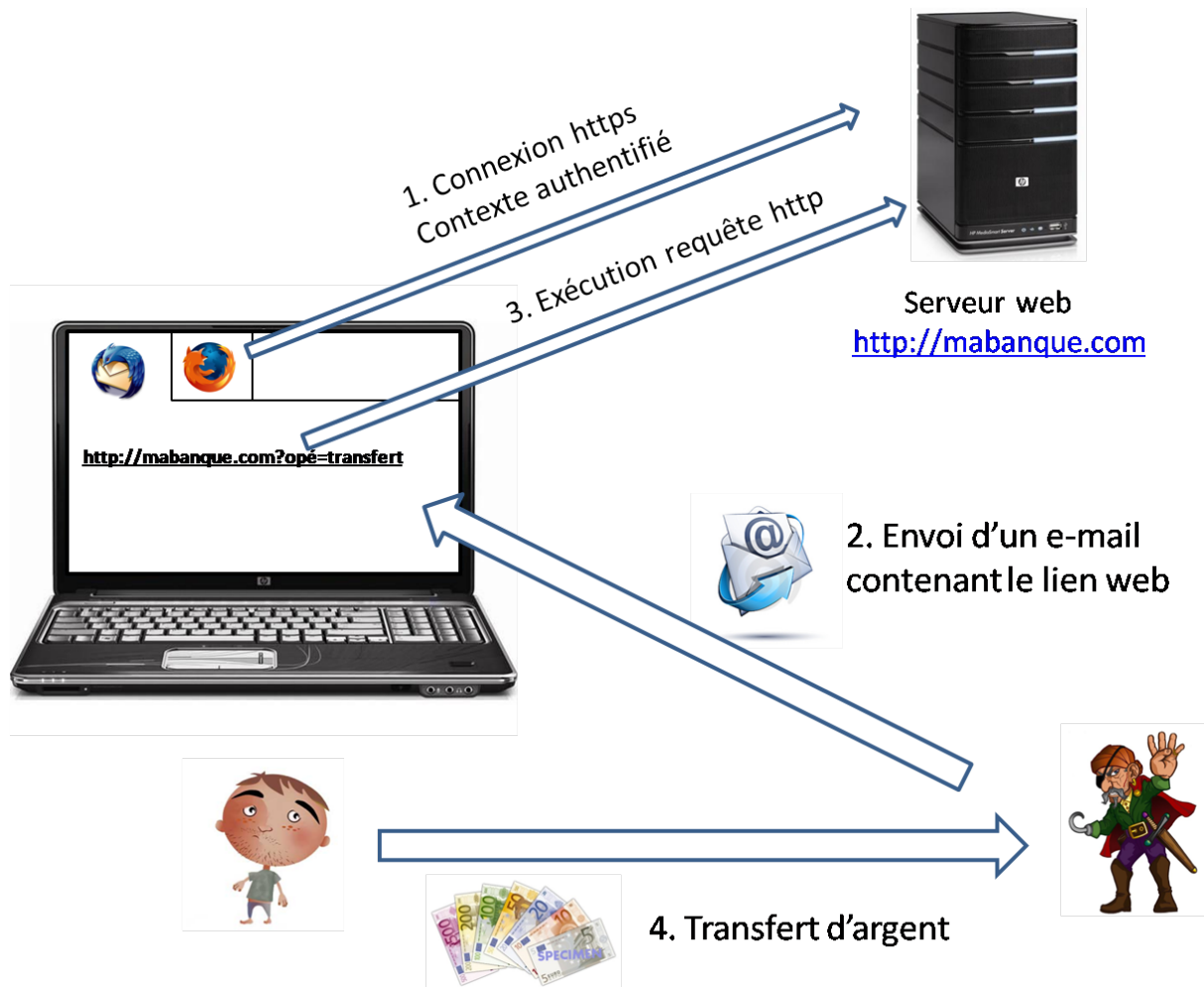
Admettons que la victime consulte ses e-mails, et ouvre un e-mail attirant son attention, celui de l'attaquant. Ce dernier a en effet envoyé un e-mail à sa victime, contenant un lien web, élaboré par ses soins :

<http://mabanque.com?opération=transfert&emetteur=victime&récepteur=attaquant&somme=1000>

Les paramètres de cette requête spécifient l'opération suivante : un transfert d'argent du compte de la victime vers le compte de l'attaquant. En cliquant sur ce lien, la victime participe à l'attaque sans en avoir conscience. Nous verrons plus tard que l'attaquant dispose de nombreux moyens afin de faire exécuter cette requête http au navigateur de la victime.

Evidemment, ce genre d'opération sensible ne peut être réalisé que dans un contexte authentifié, car le principal intéressé doit donner son autorisation. Mais puisque la victime est dans un contexte authentifié lorsqu'elle consulte cet e-mail, cette autorisation est en quelque sorte donnée à son insu.

Voici un schéma décrivant cet exemple :



3) Les méthodes rendant cette attaque « invisible »

A) L'utilisation du Cross Site Scripting XSS

Dans l'exemple précédent le lien web est en clair, et il est très probable qu'un utilisateur, même peu averti, se méfie d'un tel lien et ne l'exécute pas.

Pour rendre cette attaque invisible, l'attaquant dispose d'autres moyens pour faire exécuter la requête http à sa victime. En particulier, il peut insérer la requête http dans un script, qui sera lui-même destiné à être exécuté par la victime. Cette méthode est très intéressante pour l'attaquant car il peut ainsi utiliser une attaque de type Cross Site Scripting (XSS).

Ce type d'attaque consiste à écrire sur une page web, de telle sorte que le navigateur consultant cette page web interprète ce qui est écrit comme étant un script, et l'exécute.

Typiquement, cela consiste à insérer les balises html correspondant à l'écriture d'un script :

```
<script type="text/javascript"></script>
```

Il suffit alors d'insérer dans le script la requête http, qui sera exécutée par le navigateur de la victime.

Ces méthodes sont intéressantes pour l'attaquant, car il n'a pas forcément besoin d'être l'administrateur du site web pour insérer son script. En effet, il lui suffit par exemple d'insérer un « post » sur un forum.

B) L'utilisation de la balise html « image »

L'attaquant peut tout aussi bien inscrire sur un « post » un lien web supposé pointer vers une image, mais en réalité pointant vers un script malicieux contenant l'exécution de la requête http.

Pour cela, il utilise la balise html « image ». Cette balise offre une importante faille de sécurité dans la mesure où le navigateur de la victime exécute systématiquement l'url passé en paramètre.

```
<img src= « lien vers script malicieux » />
```

Grâce à ces méthodes, l'attaque CSRF est rendue très dangereuse et très difficile à détecter, car il suffit alors de visiter une page web sur un site tout à fait légitime pour en être victime, par exemple un réseau social.

Heureusement il existe des méthodes pour se protéger de cette attaque.

4) Se protéger contre l'attaque CSRF

A) Réduire le risque d'exécution de la requête

En tant que développeur du site web à protéger, il peut être intéressant de se protéger contre les attaques de XSS, afin de réduire le risque d'attaque CSRF. Pour cela, il est possible d'interdire l'écriture sur un post de certains caractères, tel que « > » ou « < », afin de se prémunir contre l'écriture de scripts à l'intérieur de « post ».

En tant que potentiel victime, il faut être prudent quant aux liens sur lesquels on nous propose de cliquer, car le lien présenté comme étant banal peut en fait conduire à l'exécution d'un script malicieux. Il convient donc de n'accorder qu'une confiance limitée aux liens proposés par notre environnement.

Les méthodes précédentes ne sont pas des barrières infaillibles, et l'attaquant trouvera toujours un moyen de faire exécuter le lien web à sa victime. Il convient donc de trouver un moyen de protéger le site web afin de ne pas permettre l'exécution d'actions de manière implicite. Pour cela, différentes méthodes existent.

B) Rendre la requête imprédictible pour l'attaquant : les jetons

En effet, il est assez simple de se protéger à l'aide de jetons. Pour cela, à chaque nouvelle visite sur un site web, une chaîne aléatoire est générée puis stockée dans une variable de session. Cette chaîne est alors transmise à chaque requête en tant que paramètre. Cette méthode permet de s'assurer que la requête vient d'être construite, et qu'elle n'a pas été préparée préalablement par un attaquant.

Lors du développement du site en php, ces quelques lignes de code suffisent :

Génération et stockage du jeton :

```
<?php
    $jeton = md5(time()*rand(1,10)); // génération du jeton
    $_SESSION[jeton] = $jeton; // stockage du jeton dans la session
```

Récupération du jeton et passage en paramètre de la requête :

```
<?php
    $jeton = $_SESSION[jeton]; // récupération du jeton
    echo '<a href="index.php?action=parametre1&jeton=$jeton">';
?>
```

Vérification du token:

```
<?php
    if(isset($_GET[jeton]) && $_GET[jeton] == $_SESSION[jeton]){
    }
    else{
        // erreur => jeton invalide ou inexistant ;
    }
?>
```

C) Ajouter une autre étape d'authentification

En effet, comme cela devient de plus en plus courant, on peut demander à l'utilisateur de fournir ses identifiants à chaque fois qu'il désire réaliser une opération sensible.

Par exemple, sur les sites web bancaires, il est nécessaire de fournir son mot de passe à l'aide d'un clavier virtuel s'affichant à l'écran, avant de réaliser tout transfert d'argent ou autre opération sensible et ce même si l'on est déjà authentifié sur le site web de la banque.

D) Confirmer la demande d'exécution de l'opération

Un moyen simple de se protéger contre une attaque CSRF peut aussi être de demander confirmation à l'utilisateur avant d'exécuter une opération sensible. Par exemple, une fenêtre demandant confirmation à l'utilisateur peut s'afficher, présentant le choix « oui » ou « non ». Cela peut sembler trivial, mais empêche néanmoins la victime d'être un complice de manière implicite.

II) Présentation du projet de fin d'études

1) Présentation du projet

Ce projet a pour but développer un outil de détection automatique de vulnérabilité CSRF, pour le compte de la société **VulnIT (Vulnerability Identification Tool)**, proposant un outil de détection de vulnérabilités.

Notre outil est destiné à être intégré à **VulnIT**, afin de proposer à ses utilisateurs de détecter si un site web est vulnérable aux attaques de type CSRF.

Il s'agit de déterminer quelles sont les fonctionnalités précises que doit fournir notre outil, quel doit-être l'algorithme utilisé, et en quel langage cet outil doit-être implémenté.

2) Gestion du projet

A) Rencontre avec la société VulnIT

Afin de mieux saisir ce qui est attendu de notre part par la société VulnIT, nous convenons d'un rendez-vous par vidéoconférence avec Monsieur Vincent Maury, notre contact chez la société VulnIT.

Lors de ce rendez-vous, il apparaît clairement qu'une première difficulté est de préciser quelles sont les limites de l'outil. En effet, il n'est pas évident de déterminer ce qui constitue une faille CSRF, car de nombreux éléments entrent en ligne de compte, comme cela a été vu dans la partie I.

En effet, de nombreuses questions se posent : est-on dans un contexte authentifié ? Le site web propose-t-il une protection contre le CSRF à l'aide d'une étape de confirmation ? A l'aide de jetons ? Quels sont les éléments dont nous disposons en entrée de notre outil ? Quels sont les éléments que doit renvoyer notre outil ?

Il convient alors de déterminer précisément ce que fera ou ne fera pas notre outil.

B) Les spécifications de notre outil

Après de nombreuses discussions, et le passage en revue des différents outils déjà existant concernant la faille CSRF, nous décidons que notre outil doit pouvoir passer en revue toutes les pages d'un site web, afin de déterminer quelles pages correspondent à un contexte authentifié. Il doit alors renvoyer tous les formulaires éventuellement contenus dans ces pages.

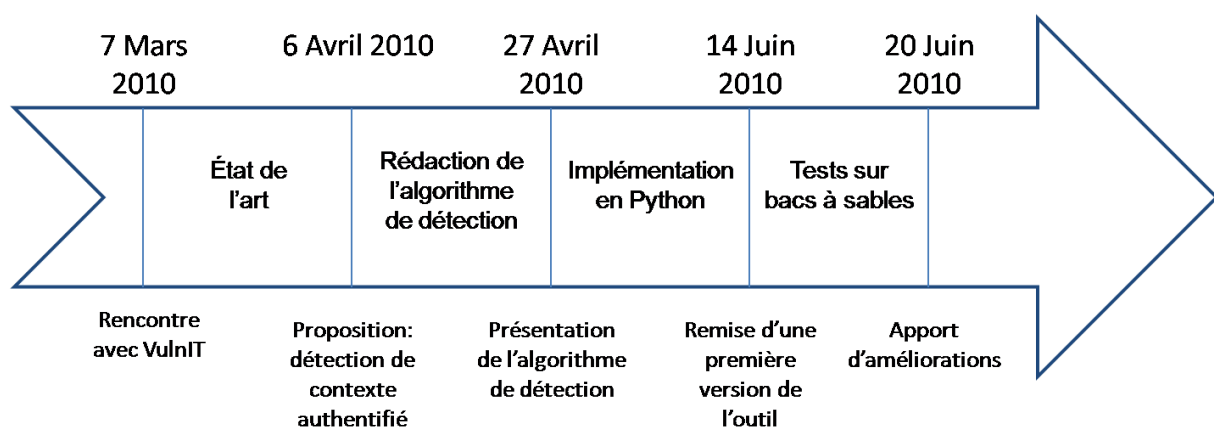
Nous présentons alors cette proposition à Monsieur Maury lors d'un entretien par vidéoconférence, afin d'obtenir son opinion, et ainsi mettre en place un planning contenant les différentes étapes du projet.

C) Les principales étapes du projet

A l'issue de cet entretien avec Monsieur Maury, notre proposition a été validée et nous avons convenu des étapes suivantes :

- Détermination de l'algorithme utilisé par notre outil (pour fin avril)
- Implémentation de l'outil (pour mi-Juin)
- Réalisation de tests sur bacs à sable (pour mi-Juin)

Pour permettre une meilleure représentation de notre projet, voici un schéma représentant les différentes étapes suivies et les dates effectives associées à chacune d'entre elles :



III) Le livrable

Les deux premières parties de ce document nous ont permis de mieux comprendre les attaques du type CSRF et de présenter la problématique de notre sujet. Nous allons maintenant voir en quoi consiste notre livrable en expliquant l'algorithme d'une manière formelle puis en regardant de plus près son implémentation dans le langage Python. Nous finirons cette partie par une présentation des tests et des résultats du livrable, ainsi que des améliorations possibles à apporter au logiciel. Cependant, avant de parler du programme développé, nous nous proposons de faire un état de l'art des différents logiciels qui existent déjà pour détecter les failles CSRF.

1) Etat de l'art des logiciels existants

Différents logiciels existent sur internet pour détecter des failles CSRF. Nous présenterons ici principalement des logiciels gratuits et open-source. En effet, il s'agit des logiciels les plus célèbres et autour desquels la communication est la plus importante. De plus il nous a été plus facile de tester des logiciels gratuits que des sharewares même si certains offrent la possibilité d'être utilisés gratuitement pendant une période donnée.

A) Pinata

Pinata est un logiciel gratuit et open-source. Il est développé en Python par Ahsan Mir l'auteur du blog : <http://secmir.blogspot.com/>. Trois releases ont été effectuées pour ce programme :

- Pinata-V0.91 en mai 2010
- Pinata-V0.92 en juillet 2010
- Pinata-V0.93 en novembre 2010

Le projet (ainsi que le blog) semble être arrêté depuis novembre 2010.

La page principale du logiciel se trouve à l'adresse suivante :

<http://code.google.com/p/pinata-csrf-tool/>.

Pinata ne permet pas de détecter directement les failles CSRF. Il s'agit d'un outil qui permet de produire des « proof of concept » à partir d'une url sensible au CSRF. En effet l'utilisateur du logiciel doit repérer un formulaire sensible aux attaques CSRF et copier l'ensemble de la requête (GET ou POST) dans un fichier texte se trouvant dans le même répertoire que Pinata. Ce dernier va se contenter de produire un document HTML qui encapsule l'url sensible dans une image. A chaque fois que l'utilisateur accède au fichier HTML, l'action décrite par l'url sensible sera effectuée. L'utilisation de ce logiciel pour la détection automatique de failles CSRF serait donc assez compliquée. En effet, il faudrait savoir quels paramètres le formulaire à tester est sensé modifier (le solde sur un compte bancaire par exemple), lancer Pinata sur le formulaire et regarder si les paramètres ont été modifiés (le solde a augmenté ou diminué). Cependant, une fois un formulaire sensible aux CSRF détecté, il peut-être

intéressant de l'utiliser pour démontrer la faille.

PaulDotCom a réalisé une vidéo très complète expliquant l'utilisation de Pinata.

Celle-ci se trouve à l'adresse suivante :

<http://www.youtube.com/watch?v=EmmNn1FRYm4>

B) Tamper-data

Tamper-data est un logiciel gratuit et open-source qui se présente sous la forme d'un Addon pour le navigateur firefox. Adam Judson est l'auteur de ce programme qui a eu de nombreuses releases depuis sa création en mars 2007. La dernière en date est la version 10.1.1. Le projet semble toujours actif et dynamique.

La page principale de Tamper-data se trouve à l'adresse :

<http://tamperdata.mozdev.org/index.html>, cependant il est possible d'obtenir l'addon directement depuis le site de Mozilla Firefox : <https://addons.mozilla.org/en-us/firefox/addon/tamper-data/>.

Ce logiciel se comporte comme un proxy qui intercepte toutes les requêtes du client vers le serveur et qui offre à l'utilisateur la possibilité de modifier à la volée l'ensemble des champs des formulaires de ces requêtes. Une des choses très intéressantes avec ce programme est qu'il permet de visualiser aussi bien les requêtes HTTP que HTTPS. Ainsi, cela permet à l'utilisateur de voir l'ensemble des champs qui sont envoyés au serveur web et de détecter de cette manière la présence ou l'absence de jetons aléatoires envoyés avec le formulaire. Cependant, il n'est pas possible d'automatiser l'utilisation de ce logiciel et ceci pour deux raisons. Premièrement, Tamper-data est un addon de firefox, il n'offre donc pas la latitude nécessaire à un lancement automatisé. Deuxièmement alors qu'un être humain est capable de détecter un jeton aléatoire dans les champs d'un formulaire, il est beaucoup plus difficile de faire réaliser cette détection à un ordinateur.

C) Web Scarab

WebScarab possède un fonctionnement assez similaire à Tamper-data. Il s'agit d'un logiciel développé par l'OWASP (Open Web Application Security Project), une organisation ouverte et gratuite travaillant sur la sécurité informatique. La page principale du projet se trouve à l'adresse :

https://www.owasp.org/index.php/Category:OWASP_WebScarab_Project.

La dernière release de ce projet date de mai 2007, cependant il est à noter qu'un projet portant le nom de WebScarab-ng a démarré depuis. Il s'agit d'un nouveau départ pour WebScarab, cette version nouvelle génération ayant pour but d'intégrer une interface graphique plus simple à utiliser. La page de ce nouveau projet se trouve à l'adresse :

https://www.owasp.org/index.php/OWASP_WebScarab_NG_Project.

La dernière release de WebScarab-ng date de janvier 2011, il s'agit de la version

0.2.1.

Ce logiciel se comporte comme Tamper-data dans le sens où il s'agit aussi d'un proxy interceptant les requêtes du client vers le serveur. Cependant, il est aussi capable de montrer à l'utilisateur les requêtes du serveur vers le client. Bien que développé en Java et donc plus facile à automatiser, WebScarab se retrouve toujours confronté au problème de la détection automatique de jetons au sein d'un formulaire. Cependant il s'est avéré très utile lors de notre étude pour bien comprendre les différentes spécificités de la vulnérabilité CSRF. Il nous a aussi permis de bien visualiser les moyens existants pour s'en protéger.

D) CSRFTester

CSRFTester est un autre logiciel open-source développé par l'OWASP. Sa dernière release (1.2) date de 2007 et sa page principale se trouve à l'adresse :

https://www.owasp.org/index.php/Category:OWASP_CSRFTester_Project.

Il s'agit d'un logiciel qui intègre à la fois les fonctions de WebScarab et de Pinata. En effet, tout comme WebScarab, CSRFTester est un proxy qui intercepte les requêtes HTTP et offre à l'utilisateur la possibilité de les modifier. S'ajoute à cette notion de proxy le fait de pouvoir créer des « proof of concept » sous la forme de fichier HTML. L'utilisation normale de CSRFTester pour tester des formulaires HTML suit les étapes suivantes :

- Lancer le logiciel en tant que proxy
- Exécuter la requête à tester sur son navigateur
- Dans CSRFTester changer les champs du formulaire qui sont intéressants (le montant d'un transfert d'argent par exemple)
- Produire un fichier HTML à partir de la requête interceptée par le logiciel
- Recharger plusieurs fois le fichier HTML produit et regarder si des données ont changé sur le site web (solde du compte en banque)
- Si les champs ont changé en adéquation avec le nombre de fois où le fichier HTML a été rechargé, alors la requête est sensible aux attaques CSRF

Une des options intéressantes de CSRFTester est la possibilité de générer plusieurs « proof of concept » différents. Alors que Pinata ne propose d'encapsuler la requête que dans une image HTML, avec CSRFTester, il est possible de l'encapsuler dans une iframe, une image, un objet XMLHttpRequest ou un simple lien internet (nécessitant donc l'intervention de la victime). Il est même possible de générer un formulaire qui sera automatiquement envoyé grâce à un script Javascript.

Tout comme les logiciels présentés précédemment, l'utilisation de ce logiciel part du postulat que l'utilisateur sait quelle donnée est modifiée par quel formulaire. Cela implique une connaissance profonde de l'architecture du site web, architecture qu'il est très difficile de faire apprendre à un ordinateur d'une manière automatique.

E) MonkeyFist

MonkeyFist est un outil gratuit et open-source développé en python par Nathan

Hamiel et Shawn Moyer. Il a été présenté à la conférence Defcon 17 ainsi qu'à la BlackHat 2009. La page principale du programme se trouve à l'adresse : <http://hexsec.com/misc/monkeyfist/>.

La dernière release est la 1.0 et date de Janvier 2010.

Il s'agit d'un logiciel un peu différent de tous ceux présentés jusqu'à maintenant. MonkeyFist agit lui aussi comme un proxy. Cependant il est nécessaire de renseigner l'url sensible aux attaques CSRF dans un fichier de configuration avant de lancer le programme. Lorsque l'utilisateur se trouve sur le site dont l'url sensible est issue, MonkeyFist va envoyer au client un message du type 302 Redirect avec, dans le champ location, l'url déclenchant l'attaque CSRF. Un message 302 Redirect, prévient le client d'une redirection. Suite à cette réponse du serveur, le navigateur du client est configuré pour envoyer une nouvelle requête GET vers l'url se trouvant dans le champ location du header HTML qui est, dans notre cas, l'url sensible.

MonkeyFist déclenche la redirection seulement lorsque l'utilisateur se trouve sur le site vulnérable parce qu'il s'agit du moment où celui-ci a le plus de chance d'y être authentifié. Cependant, il est aussi possible de déclencher la redirection à n'importe quel moment. Ceci est utile pour des sites comme Facebook où la plupart des utilisateurs sont constamment authentifiés.

On voit donc que MonkeyFist est un programme actif pour attaquer directement un utilisateur. Il ne permet pas, encore une fois, de détecter d'une manière automatique une vulnérabilité CSRF.

F) W3af

W3af est un framework d'audit et de test de vulnérabilités web. Il s'agit d'un logiciel gratuit et open-source développé par Andrés Riancho. La page principale du projet se trouve à l'adresse : <http://w3af.sourceforge.net/>.

Le 18 janvier dernier est sorti la version 1.0. Il s'agit d'un projet très actif qui est mondialement reconnu par la communauté informatique pour son efficacité.

W3af permet de détecter automatiquement de nombreuses failles web. C'est un des logiciels qui se rapproche le plus de notre projet. En effet, contrairement aux autres programmes, W3af ne requiert pas de connaître la requête sensible aux attaques CSRF et ne se contente pas de générer un simple « proof of concept ».

Il tente de détecter d'une manière automatique si un formulaire est sensible ou non en s'intéressant à tous les formulaires d'une application Web qui utilisent un cookie permanent pour l'authentification. Pour détecter cela, il envoie deux fois la même requête HTTP avec le même cookie d'authentification et il regarde si les deux réponses sont similaires. Si oui, il considère que la requête est sensible aux attaques du type CSRF.

Il s'agit d'une bonne première approche qui permet de détecter automatiquement des formulaires qui ne sont pas protégés. Cependant, cette méthode peut générer un bruit important qui empêche de déterminer quels sont les éléments réellement sensibles aux CSRF. En effet, prenons l'url : <http://www.google.fr?q=blabla>. Il s'agit d'une recherche Google sur le mot clé blabla. En effectuant deux fois cette requête avec un cookie d'authentification valide, le résultat sera évidemment deux fois le même : les sites retournés par pour le mot blabla. W3af détectera alors le formulaire

de recherche Google comme un formulaire sensible aux attaques CSRF. Ce résultat est formellement fondé, mais ce champ n'affectant en rien des données sensibles, il n'est pas utile de le signaler.

Pour détecter d'une manière efficace et automatique les failles CSRF il est nécessaire d'avoir une approche plus poussée. La méthode proposée par le logiciel Arachni produit des résultats beaucoup plus cohérents avec la réalité.

G) Arachni

Arachni est un autre framework automatisé de test de vulnérabilités web. Il est développé dans le langage Ruby par Zapotek. Le projet Arachni a démarré en septembre 2010. La dernière release datant de mars 2011 est la 0.2.2. Depuis le début du projet, Zapotek a sorti une nouvelle version environ tous les mois. La page principale du projet se trouve à l'adresse : <http://arachni.segfault.gr/news>.

Le 10 octobre 2010, Zapotek a sorti un papier intitulé « Automated detection of CSRF-worthy HTML forms ». Cette publication décrit une méthode innovante pour détecter automatiquement les formulaires sensibles aux failles CSRF. C'est cette méthode qui nous est apparue comme la plus pertinente suite aux différents logiciels testés. Nous avons donc décidé de nous en servir comme point de départ pour notre livrable.

Après avoir fait un état de l'art des différents logiciels existants pouvant servir à détecter des failles CSRF, nous allons maintenant décrire l'algorithme que nous avons choisi d'implémenter.

2) Présentation de l'algorithme

La première chose à se demander lorsque l'on veut détecter une faille web est sur quel objet HTML agit l'attaque ?

Dans le cas de failles CSRF, toute la vulnérabilité porte sur la valeur des champs grâce auxquelles l'utilisateur pourra interagir avec le site web. Ainsi, dans notre cas, les objets HTML à regarder sont les formulaires HTML, définis par la balise « <form ».

La manière la plus efficace de détecter des vulnérabilités de type CSRF est de tester deux fois la même URL et d'analyser les deux réponses du serveur web :

- ❖ Si les deux réponses sont identiques la page HTML est vulnérable
- ❖ Si les deux réponses diffèrent la page n'est pas vulnérable

Il s'agit de la méthode utilisée par W3af que nous avons présentée précédemment. Cependant, plusieurs remarques sont à ajouter à cette analyse.

La difficulté de la détection automatique de vulnérabilités CSRF réside dans le fait que le contexte de la session doit être identifié (utilisateur connecté ou non). En effet, une attaque exploitant une vulnérabilité CSRF n'a de sens que dans un contexte connecté.

De plus, aujourd'hui, la plupart des sites web ont une partie de leurs pages HTML qui change à chaque rechargement (publicité par exemple). Il est important de prendre

cette remarque en considération sans quoi l'analyse exposée au début de ce document s'avérerait complètement faussée.

Ainsi il est apparu pertinent d'avoir un algorithme qui, dans un premier temps détecte le contenu «statique» de chaque page HTML, et dans un second temps les pages qui correspondent seulement à un contexte connecté, pour pouvoir enfin analyser le contenu «statique» des pages accessibles par un utilisateur enregistré et ainsi en extraire les formulaires potentiellement sensibles aux attaques CSRF.

Dans tout l'algorithme nous avons choisi d'utiliser la méthode **rdiff** présenté par Laskos Zapotek, un des développeurs du projet Arachni.
Cette méthode est l'opposée de la méthode **diff** : elle retourne les caractères qui n'ont pas changé entre deux chaînes entrées en argument.

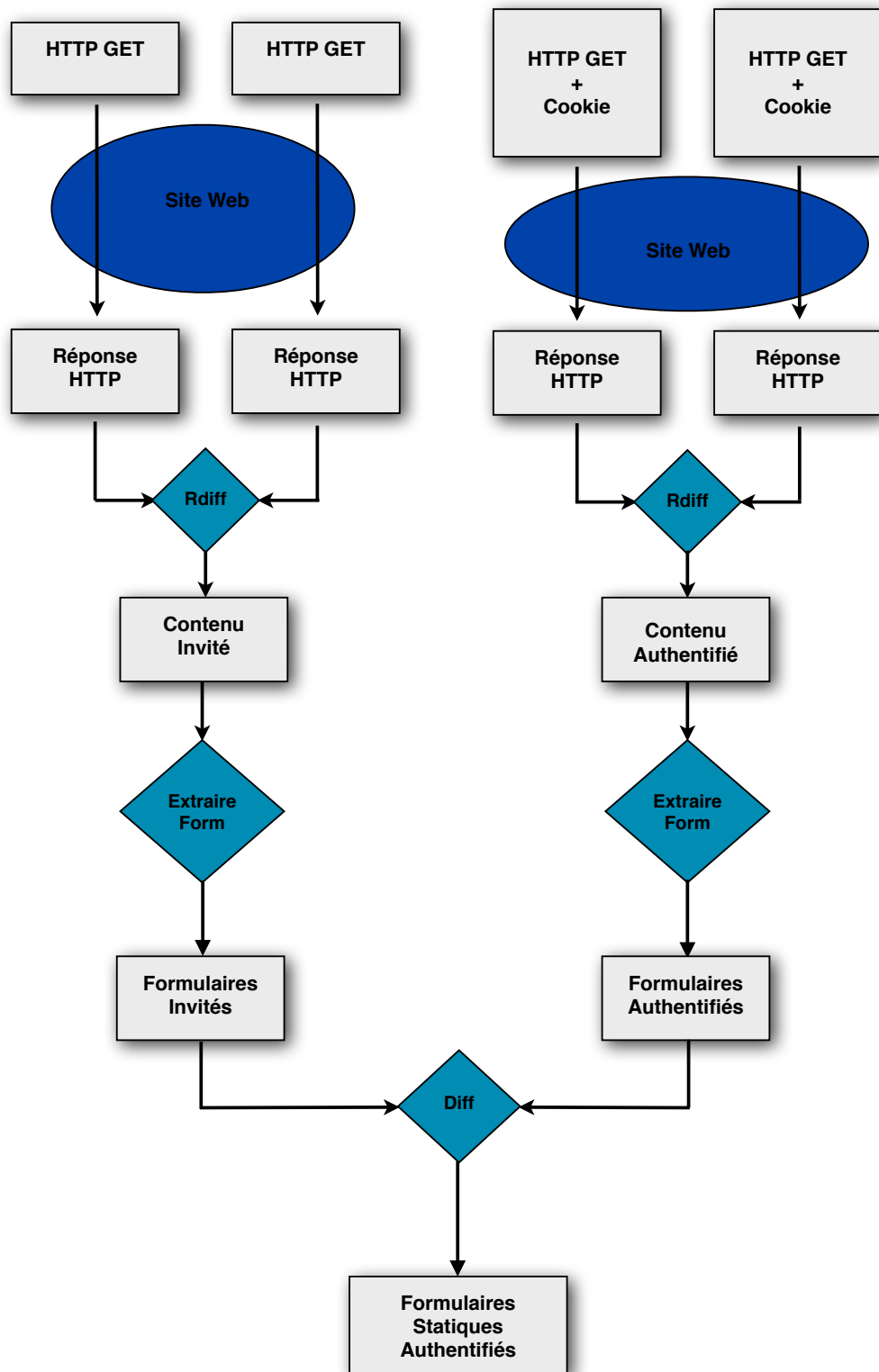


Présentation de la méthode **rdiff**

Afin de détecter le contenu statique de chaque page HTML, deux requêtes HTTP GET sont effectuées pour une même URL. Les réponses du serveur Web sont passées à la méthode **rdiff**. Le résultat de celle-ci correspond au contenu statique recherché.

Pour détecter les pages HTML qui correspondent seulement à un contexte connecté il suffit de parcourir une première fois l'ensemble du site web sans cookie d'authentification puis une seconde fois avec les cookies. Les résultats de ces requêtes sont passés à la méthode **diff**. En effet, seules les pages qui diffèrent entre un utilisateur non authentifié et un utilisateur connecté correspondent aux pages appartenant au contexte connecté.

Finalement l'algorithme final peut se représenter par la figure donnée en page suivante:



Algorithme de détection automatique de failles CSRF

Dans cette sous-partie, nous nous sommes appliqués à expliquer formellement l'algorithme de détection automatique de failles CSRF que nous avons mis au point. Nous allons maintenant nous appliquer à commenter l'implémentation de cet algorithme que nous avons effectué dans le langage de programmation Python.

3) Développement en Python et implémentation

Nous avons choisi de développer l'algorithme présenté précédemment dans le langage Python. Avant d'expliquer ce choix et de présenter l'implémentation, nous allons présenter brièvement ce langage.

A) Le langage Python

Python a vu le jour à la fin des années 1980 grâce au programmeur Guido van Rossum. Grand fan de la série télévisée Monty Python, celui-ci en a tiré le nom de son logiciel. Il faut attendre presque dix ans et la fin des années 1990 pour que Python passe en licence GPL (logiciel gratuit et open-source) dans sa version 1.6.1. En 2001, la Python Software Foundation est créée. Il s'agit d'une association à but non lucratif qui détient tout le code et les spécifications du langage depuis sa version 2.1.

En 2008 le langage prend un nouveau départ. En effet, certains défauts du langage étaient pointés du doigt par des spécialistes et empêchaient une expansion plus importante de Python. Parmi ces défauts on pouvait notamment trouver l'orientation objet avec deux types de classes ou encore la présence de nombreux éléments obsolètes et redondants dans la librairie standard. Ainsi avec Python 3.0, la Python Software Foundation a décidé de casser la compatibilité ascendante. Des programmes écrits avec la version 2.X du langage ne sont pas compatibles avec l'interpréteur Python 3.X. Cependant il existe des logiciels permettant de transformer du code de la version 2 en code compréhensible de la version 3.

Ce langage est particulièrement utilisé lors de l'écriture de script ou pour écrire des programmes dans un cadre de recherche avant de les optimiser dans un langage plus proche du système. Son approche orientée objet de la programmation lui donne un réel avantage face à d'autres langages de « scripting » comme bash.

C'est Philippe Biondi un chercheur en sécurité informatique travaillant pour EADS qui a introduit le Python dans le domaine de la sécurité avec son logiciel Scapy. Il s'agit d'une surcouche apportée à Python conçu pour générer des paquets de données d'une manière très intuitive.

La page principale de la Python Software Foundation se trouve à l'adresse :

<http://www.python.org/>. C'est à cette adresse que l'on peut trouver toute la documentation nécessaire pour développer un programme en Python.

Ce langage a été conçu dans la logique d'un langage épuré faisant fi des caractères de ponctuation que l'on peut notamment retrouver en C (';', '{', '}'). Chaque bloc est délimité par son indentation. Bien qu'un peut déroutante au début, cette syntaxe offre une vitesse de développement bien supérieure à ce que l'on peut trouver en C.

Il s'agit d'un langage orienté objet qui n'utilise donc pas de pointeur en tant que tel. Le passage de paramètre résultat à une fonction se fait en utilisant des objets. De plus, le typage n'est pas requis. Ainsi les variables suivantes:

```
a = "Blabla"  
b = "2"  
c = "3*2"
```

seront déclarées en mémoire comme des chaînes de caractères. Alors que celles-ci :

```
a_prime = 2
b_prime = 3*2
```

comme étant des entiers, la valeur de `b_prime` étant 6.

Enfin, une instance d'un objet est déclarée par le mot clé **self**.

Afin de donner au lecteur un petit aperçu d'un programme écrit en Python voici les implémentations en C et Python du célèbre « Hello World ! » :

| | |
|---|--|
| <pre># !/usr/bin/python def HelloWorld() : string = "Hello World !" print string if __init__ == "__main__" HelloWorld()</pre> | <pre>#include <stdlib.h> #include <stdio.h> void HelloWorld(); void HelloWorld(){ char * string = NULL; string = malloc(14); string = "Hello World !\n"; printf("%s", string); } int main(){ HelloWorld(); return EXIT_SUCCESS; }</pre> |
|---|--|

On peut donc voir que la vitesse de développement en Python est bien supérieure à celle en C. Cependant Python n'a pas que des avantages. En effet, il s'agit d'un langage interprété et non compilé. Ainsi le processeur va lire le programme ligne par ligne et le traduire en instruction machine à la volée. Cette spécification induit une lenteur lors de l'exécution d'un programme, lenteur qui n'apparaît évidemment pas dans le langage C qui a besoin d'être compilé avant son exécution. Pour finir, une des dernières spécificités de Python par rapport au langage C est qu'il est équipé d'un « ramasse-miettes » qui lui permet de gérer automatiquement la mémoire. Bien que très pratique, cette méthode est souvent décriée par les puristes qui aiment programmer en pouvant gérer facilement la place mémoire que prend leur programme.





Ainsi le langage Python offre de nombreux avantages pour développer rapidement un prototype pour notre algorithme de détection automatique de failles CSRF. De plus, avant ce projet nous ne connaissions pas du tout ce langage. Il nous est donc apparu intéressant de profiter de l'implémentation de l'algorithme pour apprendre une nouvelle manière de programmer. Enfin, Python possède la librairie **httplib** qui permet en quelques lignes de code d'effectuer des requêtes HTTP auprès d'un serveur web ainsi que le module **re** qui introduit la notion d'expressions régulières au langage. Comme nous allons le voir, nous avons été amenés à utiliser ces deux librairies de nombreuses fois lors de notre développement.

Avec ces différents avantages de son côté, Python nous est apparu comme un très bon langage pour notre livrable.

Nous allons maintenant voir plus en détail notre implémentation de l'algorithme.

B) L'implémentation de l'algorithme

Notre livrable se décompose en quatre modules :

-  **HTTP_request.py** : gère l'ensemble des requêtes HTTP. C'est ce module qui utilise la librairie **httplib**.
-  **comparison_method.py** : implémente les méthodes **rdiff** et **diff** nécessaire à la bonne exécution de l'algorithme.
-  **handle_html.py** : gère tous ce qui touche au langage HTML (parsage, mis en forme).
-  **main.py** : module principal qui utilise les fonctions des trois modules **HTTP_request.py**, **comparison_method.py** et **handle_html.py** pour exécuter l'algorithme de détection.

Il peut être utilisé suivant deux modes :

- Statique avec la commande :
python main.py URL Cookie

Où :

- **URL** est une chaîne de caractères représentant l'URL complète à tester (http://www.monsite.fr/index.php par exemple).
- **Cookie** est une chaîne de caractères représentant le cookie d'authentification d'un utilisateur étant connecté sur le site. Il est important de prendre le cookie d'authentification le plus récent possible. En effet, la plupart des sites utilisent des mécanismes qui expirent la session d'un utilisateur au bout d'un temps défini.

Dans ce mode là, seul l'url URL est testée. Deux requêtes HTTP GET sans cookie et deux avec cookie sont transmises à l'url. Les réponses sont analysées suivant l'algorithme (nous verront les méthodes utilisées par la suite) et les formulaires sensibles sont retournés sur la sortie standard.

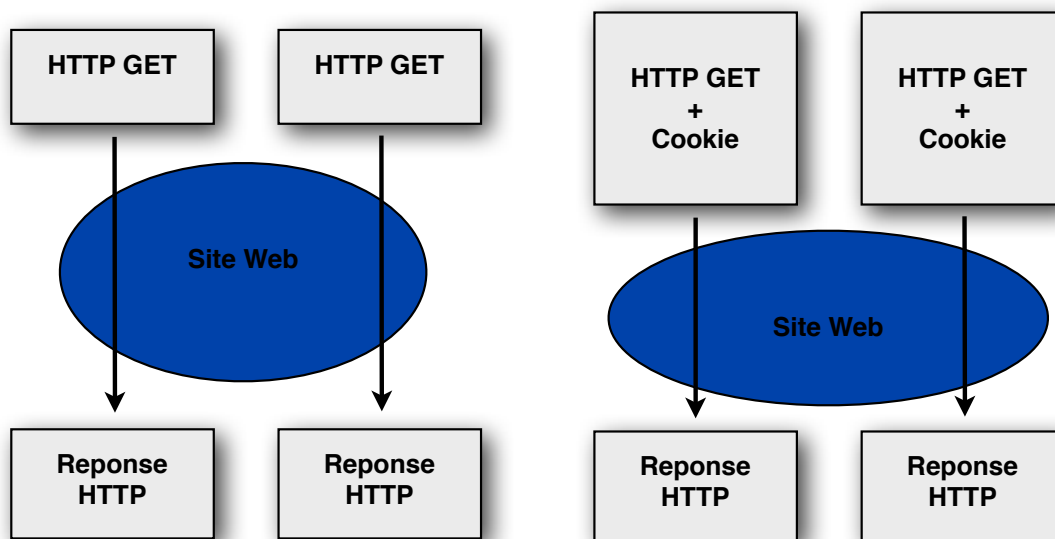
- ✓ Récuratif avec la commande :
python main.py -R URL Cookie

Dans ce mode une première étape est effectuée pour extraire tous les liens internes du site en question. Par la suite la méthode statique expliquée précédemment est appliquée à chacun de ces liens.

Nous allons maintenant nous attacher à décrire chaque module du programme afin d'expliquer au lecteur nos choix dans les méthodes de programmation utilisées.

HTTP_request.py :

Ce module gère la partie suivante du programme de détection.



Module HTTP_request.py

La première chose à faire avant de pouvoir envoyer une requête HTTP GET à une page HTML précise est de séparer de l'url passée en argument le nom du site de celui de la page.

Par exemple si l'url est la suivante : *http://www.monsite.fr/exemple.php*, le nom du site est *www.monsite.fr* et le nom de la page *exemple.php*. Pour cela, les différentes méthodes de l'objet str (une chaîne de caractères en Python) ont été très efficaces. Ainsi, l'url est passée à la fonction `url_to_host_pagename` qui retourne un dictionnaire. Il s'agit d'un autre objet du langage Python qui se comporte comme une liste si ce n'est que ces éléments ne sont pas accessibles via un index mais grâce à une chaîne de caractères. Dans notre cas le dictionnaire retourné a les caractéristiques suivantes :

```
dictionnaire["hostname"] = nom_du_site #www.monsite.fr
dictionnaire["pagename"] = nom_de_la_page #exemple.php
```

Cela permet d'avoir une structure de données contenant toutes les informations nécessaires à l'envoi d'une requête HTTP.

Nous avons aussi fait en sorte que ce module soit en mesure de gérer les redirections. En effet, dans le protocole HTTP, les messages du type *302 Redirect* ou *302 Moved temporarily* sont très fréquents. Par exemple si l'utilisateur entre l'url suivante : *http://www.monsite.fr*, seul le nom du site sera renseigné, le nom de la page HTML sera inconnu du serveur web. Ainsi celui-ci va renvoyer au client un message *302 Redirect* lui indiquant où se trouve la page d'index du site. On peut aussi voir ce type de message circuler dans le cas d'url unique générée à la volée. Il s'agit d'une des défenses mise en place pour se protéger des attaques de type CSRF : l'attaquant ne peut prédire à l'avance l'url utilisée pour modifier un formulaire sensible. Dans ce cas là, le serveur renvoi au client un message *302 Moved temporarily* en indiquant l'url unique.

Pour être en mesure de gérer ces redirections il faut aller chercher la bonne url dans le champ *location* du header HTTP. C'est ce que s'occupe de faire la fonction `handle_redirect` et plus particulièrement l'instruction :

```
new_location = [location for name, location in headers if name == "location"][0]
```

On voit ici toute la puissance du langage Python. Cette simple ligne crée une liste de tous les mots (chaînes de caractères séparée par un espace) que contient le champ *location* du header et prend le premier élément de cette liste. Ce dernier correspond à la nouvelle url renvoyée par le serveur web.

Finalement, une fois l'url découpée et les redirections effectuées, `HTTP_request.py` envoie une requête HTTP à la bonne adresse grâce à la librairie `httplib`. Le code HTML retourné est ensuite passé au module `handle_html.py`.

handle_html.py :

Ce module s'occupe de différentes étapes de l'algorithme :

- ✓ Le « passage » des réponses HTTP renvoyées par le serveur :



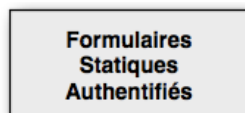
Réponses HTTP renvoyées par le serveur

- ✓ L'extraction des formulaires une fois la méthode `rdiff` appliquée :



Extraction des formulaires

- ✓ La mise en page du résultat (les formulaires potentiellement sensibles) :



Résultat de l'algorithme

Pourquoi un module complet pour gérer le langage HTML ? Il serait plus simple d'appliquer la méthode `rdiff` aux deux réponses reçues par le serveur web sans se soucier du type de donnée traitée.

Cependant, imaginons les deux pages HTML suivantes :

| | |
|--|--|
| <pre><a> <b blabla /> <c>Blibli</c> <d lala> <e>hehe</e> </d> </pre> | <pre><a> <b blablo /> <c>Blibli</c> <f lala> <e>hehe</e> </f> </pre> |
|--|--|

En passant simplement ces deux pages à la méthode **rdiff**, le résultat serait :

```
<a>
  <b blabl />
  <c>Blibli</c>
  < lala>
    <e>hehe</e>
  </>
</a>
```

alors que l'on attendrait :

```
<a>
  <c>Blibli</c>
</a>
```

Il est donc nécessaire de créer des structures de données adéquates pour pouvoir ensuite comparer deux pages non pas linéairement mais en fonction des balises HTML.

Ainsi nous avons décidé de créer une classe représentant un arbre dont le constructeur en Python est :

```
def __init__(self, value="") :
    self.value = value
    self.child = []
    self.parent = self
```

Value est une chaîne de caractères représentant la valeur du nœud et child une liste de nœuds enfants du nœud courant.

Tout nouveau nœud se verra attribuer une valeur passée en argument, sera son propre parent et n'aura pas d'enfant. Charge ensuite au développeur de lui attribuer le parent et les enfants qui conviennent. Pour cela nous avons aussi implémenté la méthode `add_child` qui permet d'ajouter un enfant à la fin de la liste des enfants du nœud concerné. Nous avons fait en sorte de pouvoir ajouter un enfant soit en passant seulement sa valeur comme argument, l'enfant créé n'aura donc pas lui-même d'enfant, soit en passant directement un nœud, ce dernier et tous ses enfants seront donc rattachés au nœud concerné. Une autre méthode de la classe nœud dont il nous paraît intéressant de parler est la fonction `is_equal`. Cette dernière prend en argument un nœud et le compare au nœud concerné.

Ainsi la ligne de code :

```
noeudA.is_equal(noeudB)
```

permet de savoir si le noeudA correspond exactement au noeudB. La relation d'ordre `noeudA == noeudB` signifie :

- La valeur de noeudA est exactement égale à la valeur de noeudB
- NoeudA et noeudB ont le même nombre d'enfants
- Chaque enfant de noeudA vérifie cette relation d'ordre avec l'enfant de noeudB placé à la même position

Cette méthode sera particulièrement utilisée dans le module **comparison_method.py** pour appliquer les méthodes **diff** et **rdiff** non pas linéairement mais à deux arbres créés à partir de codes HTML.

La fonction principale de ce module est `build_tree`. Elle permet de construire un arbre à partir d'un code HTML passé en argument sous la forme d'une chaîne de caractères. La méthode employée pour cela est assez simple au premier abord. Il suffit de prendre en compte trois types de tag :

- Le tag « ouvrant » : `<a>`
- Le tag « ouvrant – fermant » : `<a Blabla />`
- Le tag « fermant » : ``

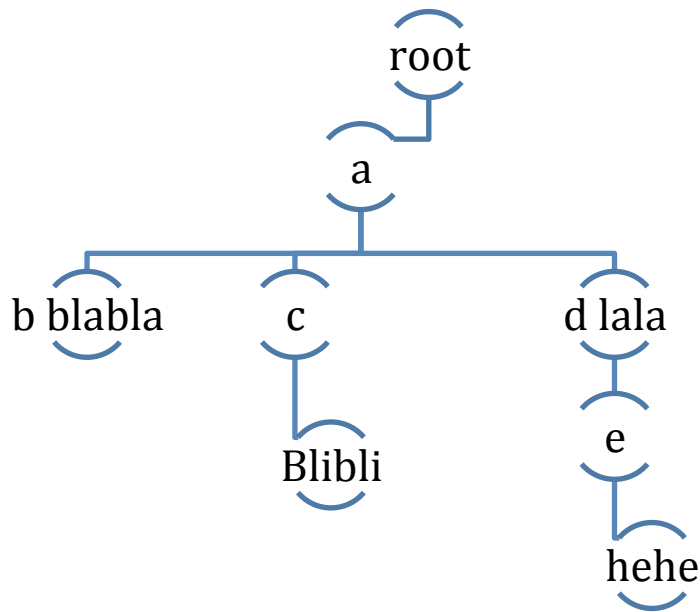
Lorsqu'un tag « ouvrant » est rencontré, un nouveau nœud est créé et tant qu'on ne rencontre pas de tag fermant tous les tags rencontrés par la suite sont ajoutés en tant qu'enfant de ce nouveau nœud. Lorsqu'un tag « ouvrant – fermant » est rencontré, un nouveau nœud est créé mais celui-ci n'aura aucun enfant.

Cependant, tout cet algorithme peut se retrouver biaisé par la présence de tag « ouvrant » n'ayant pas de tag « fermant ». Il s'agit par exemple dans le langage HTML de balises comme `
`, `` ou `<input>`. Dans ce cas là l'arbre créé se retrouvera complètement faussé puisque tous les nœuds seront ajoutés comme enfant du nœud créé à partir d'une de ces balises. Il a donc fallu implémenter la fonction `is_there_endtag` qui permet, grâce aux expressions régulières, de savoir si un tag « fermant » existe pour un tag « ouvrant » passé en argument.

Finalement grâce à ces fonctions, le code HTML suivant :

```
<a>
  <b blabla />
  <c>Blibli</c>
  <d lala>
    <e>hehe</e>
  </d>
</a>
```

se retrouvera « parsé » dans l'arbre suivant:



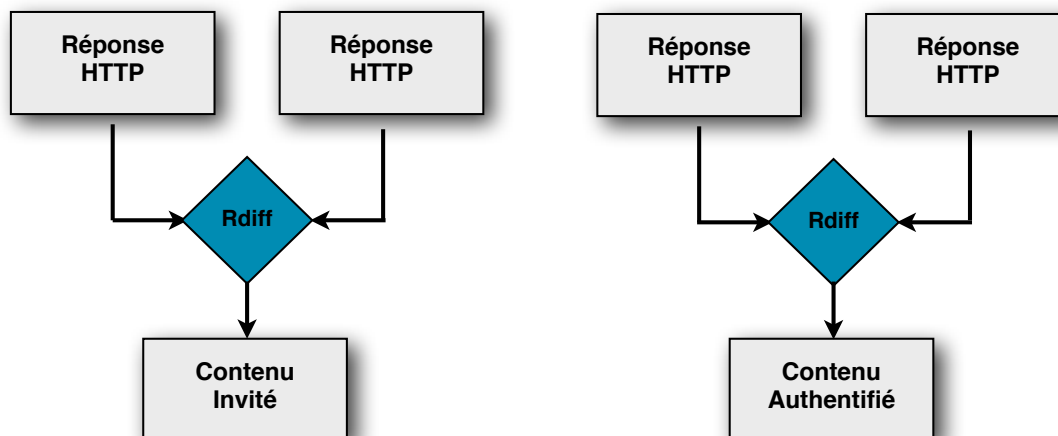
La fonction `extract_form` permet d'extraire tous les formulaires d'un arbre passé en argument. Elle retourne un arbre avec un nœud principal ayant la valeur « root » et dont tous les enfants sont les formulaires extraits. Cette fonction n'a rien de remarquable, elle se contente de parcourir un arbre à la recherche d'un nœud dont la valeur commence par « form » et d'ajouter ce dernier comme enfant du nœud root de l'arbre qui sera retourné en résultat.

La dernière étape dont s'occupe `handle_html.py` est l'impression sur la sortie standard des formulaires statiques authentifiés. Il s'agit de l'ultime étape de l'algorithme. Pour ce faire et plutôt que de faire un « passage » inverse d'un arbre vers le langage html, la fonction `print_form` se contente d'aller chercher dans le code HTML initial chaque formulaire qui a été détecté comme sensible. Elle fait appel aux expressions régulières pour détecter le début du code correspondant au formulaire et à la fonction `is_there_endtag` pour en détecter la fin. Néanmoins avant de pouvoir imprimer à l'écran le résultat de l'algorithme, les arbres HTML sont passés au module `comparison_html.py` pour s'y voir appliquer les méthodes `rdiff` et `diff`.

comparison_method.py :

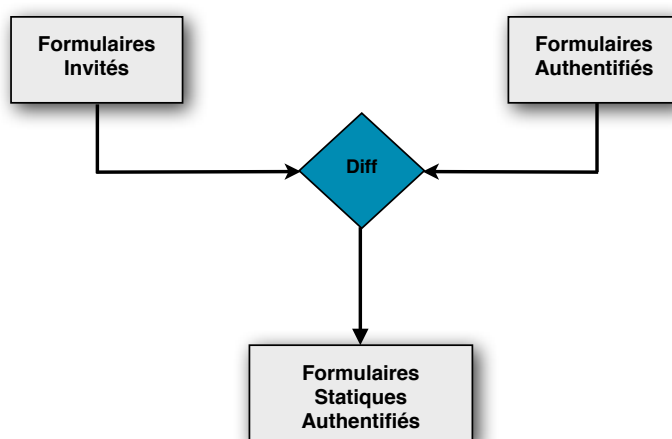
Ce module est responsable des étapes suivantes :

- ✓ Appliquer la méthode **rdiff** aux réponses du serveur web que ce soit dans un contexte authentifié ou non. Cette étape permet d'extraire le contenu statique des pages web :



Méthode rdiff appliquée aux réponses HTTP

- ✓ Appliquer la méthode **diff** aux deux arbres contenant les formulaires statiques authentifiés pour l'un et les formulaires statiques invités pour l'autre. Cette étape permet d'extraire les formulaires statiques authentifiés :



Méthode diff appliquée aux formulaires

La fonction `rdiff_tree` applique la méthode **rdiff** aux deux arbres passés en argument et retourne un arbre contenant les nœuds communs à ces deux arbres. La comparaison se fait en descendant la hiérarchie. Ainsi, si le nœud du premier arbre a une valeur différente du nœud situé au même endroit dans la hiérarchie du second arbre, on ne va pas regarder s'ils ont des enfants en commun, aucun de ces deux nœuds n'est ajouté au résultat. A titre d'exemple, reprenons les deux codes HTML mentionnés précédemment :

```

<a>
  <b blabla />
  <c>Blibli</c>
  <d lala>
    <e>hehe</e>
  </d>
</a>
  
```

```

<a>
  <b blablo />
  <c>Blibli</c>
  <f lala>
    <e>hehe</e>
  </f>
</a>
  
```

Ici, le nœud « d lala » est différent du nœud « f lala », même si leurs enfants sont identiques (« <e>hehe</e> »), aucun de ces nœuds ne sera ajouté au résultat. Finalement, `rdiff_tree` avec comme arguments les deux arbres construit à partir des codes HTML ci-dessus retournera un arbre qui correspondra au code HTML suivant :

```
<a>
  <c>Bibli</c>
</a>
```

Il s'agit bien du résultat attendu.

La fonction `diff_treeform` applique la méthode **diff** à deux arbres ne contenant que des formulaires. Celle-ci prend comme premier argument l'arbre contenant les formulaires statiques authentifiés et comme second l'arbre contenant les formulaires statiques invités. Pour qu'un formulaire statique authentifié soit ajouté à l'arbre résultat, il faut que celui-ci ne se trouve pas dans l'arbre des formulaires statiques invités. En d'autres termes, si un formulaire authentifié est égal à un formulaire invité alors il n'est pas ajouté au résultat, sinon il l'est. C'est donc dans cette fonction qu'intervient la méthode `is_equal` de la classe du module **handle_html.py**.

Le module **main.py** n'implémente pas de fonction particulière. Il se contente de faire interagir les différents modules décrits précédemment pour dérouler l'algorithme de détection en entier.

Dans toute la description de l'implémentation nous n'avons pas parlé du mode récursif du livrable. Il ne s'agit en fait que d'une surcouche qui s'occupe lors du lancement du programme d'extraire tous les liens internes du site dont l'url est passée en argument. Ensuite, le programme statique est appelé pour chaque lien trouvé. La seule difficulté de cette surcouche est de faire la différence entre les liens internes et externes. Pour cela, pour chaque lien trouvé, il faut regarder si le nom du site extrait du lien correspond au nom du site de l'url passé en argument.

Il est à noter que les différentes fonctions dont nous avons parlé dans cette sous-partie sont disponibles en annexe de ce rapport.

Nous allons maintenant voir les résultats qu'a pu produire notre livrable sur différents sites internet auprès desquels nous l'avons testé.

4) Tests et résultats du logiciel

Après avoir fini d'implémenter l'algorithme, il est très important de le tester pour voir si les résultats qu'il produit sont conformes au cahier des charges. Contrairement à une application graphique, le livrable ne dépend pas d'actions de l'utilisateur, il s'exécute de manière automatique. On pourrait donc croire que si les résultats sont concluants sur un site ils le seront sur tous les autres, chaque site internet étant développé en langage HTML. Cependant chaque site internet est aussi développé suivant une méthode bien précise, on peut donc voir apparaître de nombreux cas particuliers auxquels nous n'avons pas pensé en développant notre logiciel.

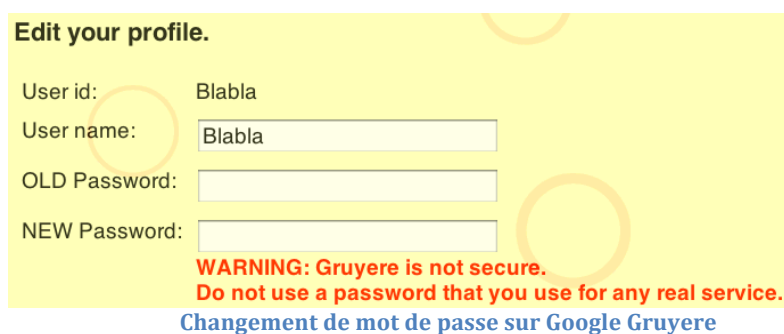
Il existe sur internet de nombreux sites pédagogiques ou « bacs à sable ». Il s'agit de sites web volontairement vulnérables à de nombreuses attaques informatiques (XSS, Buffer over Flow, SQL Injection, CSRF ...) et qui permettent à tout un chacun d'apprendre à les exécuter. Nous avons donc, dans un premier temps, testé notre détecteur de failles CSRF sur ces sites internet.

A) Google Gruyère

Google Gruyère est une application développée par Google qui est disponible directement sur internet (pas besoin d'installation préalable). Chose plutôt originale, le serveur web se trouve être développé dans le langage Python. Comme indiqué dans le nom de l'application il s'agit d'un serveur contenant une multitude de trous de sécurité.

La page principale sur projet se trouve à l'adresse suivante : <http://www.google-gruyere.appspot.com/>. Une fois inscrit, Google vous fournit un identifiant unique (ID) qui vous permet d'accéder à votre application Google Gruyère à l'adresse : <http://www.google-gruyere.appspot.com/ID>.

Il s'agit d'un site internet plutôt classique où il est possible de s'inscrire, de poster des images et des commentaires où encore de modifier ses informations personnelles. Dans la majorité des sites web, les formulaires qui sont visés par les attaques de type CSRF sont justement ceux provenant des pages où il est possible de modifier son mot de passe. En effet, si un attaquant arrive à modifier le login et le mot de passe d'un utilisateur selon son bon vouloir, il a alors accès à toute l'application. Dans le cas de la présente application, la page permettant de modifier les informations personnelles de l'utilisateur se trouve à l'adresse : <http://www.google-gruyere.appspot.com/ID/editprofile.gtl>. Celle-ci affiche le formulaire suivant pour l'utilisateur Blabla :



Edit your profile.

User id: Blabla

User name:

OLD Password:

NEW Password:

**WARNING: Gruyere is not secure.
Do not use a password that you use for any real service.**

Changement de mot de passe sur Google Gruyere

Avant de pouvoir lancer notre logiciel, il est nécessaire de récupérer un cookie d'authentification.

Ainsi, une fois l'utilisateur « Blabla » authentifié, nous lançons l'outil Wireshark qui permet de visualiser les différents paquets qui sont envoyés sur un réseau data. Nous obtenons la capture suivante :

| File Edit View Go Capture Analyze Statistics Telephony Tools Help | | | | | |
|---|----------|----------------|----------------|----------|--|
| Filter: <input type="text" value="http && ip.addr == 192.168.0.5"/> Expression... Clear Apply | | | | | |
| No. | Time | Source | Destination | Protocol | Info |
| 10 | 0.031461 | 192.168.0.5 | 209.85.227.141 | HTTP | GET /859093426533/editprofile.gtl HTTP/1.1 |
| 21 | 1.073497 | 209.85.227.141 | 192.168.0.5 | HTTP | HTTP/1.1 200 OK (text/html) |
| 23 | 1.253691 | 192.168.0.5 | 209.85.227.141 | HTTP | GET /favicon.ico HTTP/1.1 |
| 25 | 1.385849 | 209.85.227.141 | 192.168.0.5 | HTTP | HTTP/1.1 304 Not Modified |

Capture Wireshark sur Google Gruyère

En regardant plus en détail le premier paquet on peut voir les informations suivantes :

```
Hypertext Transfer Protocol
> GET /859093426533/editprofile.gtl HTTP/1.1\r\n
Host: google-gruyere.appspot.com\r\n
Connection: keep-alive\r\n
Referer: http://google-gruyere.appspot.com/859093426533/login?uid=Blabla&pw=bla\r\n
Cache-Control: max-age=0\r\n
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_6_7) AppleWebKit/534.30 (KHTML, like Gecko) Chrome/12.0.742.100 Safari/534.30\r\n
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n
Accept-Encoding: gzip,deflate,sdch\r\n
Accept-Language: fr-FR,fr;q=0.8,en-US;q=0.6,en;q=0.4\r\n
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.3\r\n
Cookie: GRUYERE=5048546|Blabla|author\r\n
\r\n
```

Détail d'une requête HTTP sur Google Gruyère

Grâce à Wireshark, on peut donc récupérer la valeur du cookie d'authentification pour l'utilisateur « Blabla » : **GRUYERE=5048546|Blabla|author**.

On peut donc lancer notre programme en mode récursif à l'adresse <http://www.google-gruyere.appspot.com/> :

```
./main.py -R http://google-gruyere.appspot.com/859093426533/
« GRUYERE=5048546|Blabla|author »
```

Le logiciel est lancé en mode récursif, il y a donc une première étape qui est effectuée afin de trouver tous les liens internes du site web. Finalement après moins d'une minute d'attente, nous avons le résultat suivant :

```
*** Aucun formulaire pour la page : www.google-gruyere.appspot.com/859093426533/ ***
*** Aucun formulaire pour la page : www.google-gruyere.appspot.com/859093426533/snippets.gtl ***
*** Aucun formulaire pour la page : www.google-gruyere.appspot.com/859093426533/upload.gtl ***

*** Pour la page : www.google-gruyere.appspot.com/859093426533/editprofile.gtl ***
<form method='get' action='/859093426533/saveprofile'>
<input type='hidden' name='action' value='update'>
<table>
  <tr><td>
    User id:
  </td><td>

    Blabla

  </td></tr>
</tr><td>
  User name:
</td><td>
  <input type='text'
    value='Blabla'
    name='name' maxlength='16'>
  </td></tr>
</tr><td>
  OLD Password:
</td><td>
  <input type='password' name='oldpw'>
  </td></tr>
</td></tr>
</tr><td>
```

```

NEW Password:
</td><td>
<input type='password' name='pw'>
<br><span style="color:red"><b>WARNING: Gruyere is not secure.<br>
Do not use a password that you use for any real service.</b></span>
</td></tr>
</table>
</form>

*** Aucun formulaire pour la page : www.google-gruyere.appspot.com/859093426533/logout ***
*** Aucun formulaire pour la page : www.google-gruyere.appspot.com/859093426533/login ***

```

Cette sortie nous indique que seule la page <http://www.google-gruyere.appspot.com/ID/editprofile.gtl> a retourné un formulaire statique apparaissant seulement dans un contexte authentifié. Il ne nous reste plus qu'à copier le code HTML retourné dans un éditeur de texte, à le sauvegarder sous le nom google_gruyere_form.html et à l'ouvrir dans un navigateur web pour avoir le résultat suivant :

```

User id:      Blabla
User name:    Blabla
OLD Password: 
NEW Password: WARNING: Gruyere is not secure.
               Do not use a password that you use for any real service.

```

Résultat du livrable sur Google Gruyère

On retrouve donc bien le formulaire de changement de mot de passe présenté précédemment. L'absence de couleur s'explique par le fait qu'ici seul le formulaire a été retourné, sans la feuille de style CSS. Mais cela ne change rien au résultat. Le résultat s'avère donc concluant. Cependant, cela ne signifie pas que le formulaire présenté en résultat du livrable est forcément sensible aux attaques de type CSRF. Il s'agit seulement d'un formulaire qui se doit d'être protégé contre ce genre d'attaque. Néanmoins, aujourd'hui très peu de sites web intègrent des mécanismes de protection contre les failles CSRF. Dans la plupart des cas les formulaires retournés par notre logiciel sont donc effectivement sensibles aux « cross site request forgery ».

La première fois que nous avons lancé le programme sur l'application Google Gruyère, nous avons un résultat nous indiquant qu'aucun formulaire statique n'apparaissait seulement dans un contexte authentifié. Il nous a fallu nous plonger à nouveau dans le code du logiciel pour comprendre notre erreur. La première chose à remarquer avant d'expliquer ce que nous avons mal fait est que lorsque nous ne sommes pas authentifiés, l'adresse <http://www.google-gruyere.appspot.com/ID/editprofile.gtl> nous retourne la page HTML suivante :

Edit your profile.

User id: <not logged in>

User name:

OLD Password:

NEW Password:

**WARNING: Gruyere is not secure.
Do not use a password that you use for any real service.**

Changement de mot de passe sur Google Gruyère en mode invité

Comme nous pouvons le remarquer il s'agit du même formulaire que lorsque l'on est authentifié. Seule la valeur des champs « User id » et « User name » a changé. Or, la première version de notre logiciel ne testait que la valeur du formulaire dans la fonction `diff_treeform` du module **comparaison_method.py** et non pas la valeur de ses champs. Ici dans les deux cas (authentifié et invité), la valeur du formulaire est :

```
<form method='get' action='/859093426533/saveprofile'>
```

Ainsi, celui-ci paraissait apparaître à la fois dans un contexte invité et authentifié et n'était donc pas ajouté au résultat lors du **diff** final.

Il nous a donc fallu modifier la fonction `diff_treeform` pour qu'elle teste toutes les valeurs du formulaire.

Dans le monde de la sécurité il existe de multiples manières de se tenir informé des nouvelles vulnérabilités découvertes sur les logiciels en circulation. En France, par exemple, le CERTA, un organisme du gouvernement, publie de nombreuses alertes sur son site web à l'adresse suivante : <http://www.certa.ssi.gouv.fr/>. Cependant une des meilleures manières de connaître les nouvelles failles informatiques reste les mailing-listes. Une des plus actives d'entre elles est Bugtraq. Cette dernière regroupe de nombreux experts en sécurité informatique tout autour du monde qui communiquent sur les problèmes de sécurité qu'ils ont pu rencontrer. Cette liste est ouverte à tous. C'est en nous y inscrivant que nous avons appris que de nombreux sites web étaient sensibles aux attaques de type CSRF. Nous avons donc décidé d'en choisir un pour y tester notre logiciel.

B) RecordPress

RecordPress est un CMS gratuit et open-source qui permet de créer rapidement son propre blog. Un CMS est un site web paramétrable dont le code peut être téléchargé sur internet. Ainsi, une personne ne voulant pas perdre du temps à écrire tout le code de son site web peut télécharger une telle application et la paramétrer selon ses désirs.

Nous avons obtenu le code de RecordPress à l'adresse suivante :

<http://www.recordpress.org/download.php>. Après l'avoir paramétré selon notre gré et obtenu le cookie d'authentification de l'utilisateur admin, nous avons lancé notre logiciel selon la ligne de commande suivante :

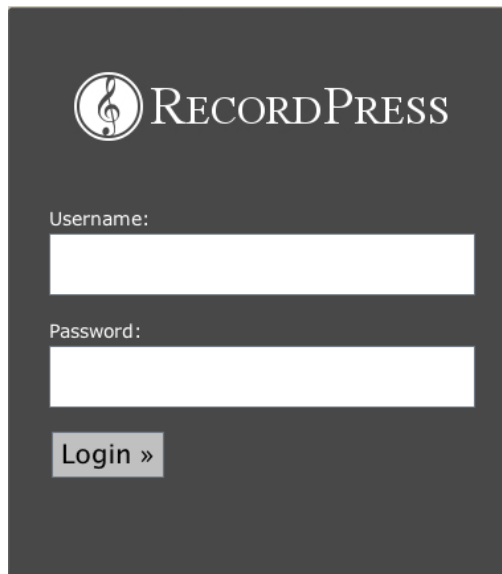
```
./main.py -R http://192.168.0.10/recordpress/  
« PHPSESSID=cfa35fcb6447738b72ffce93923d3c27 »
```

Le résultat s'est avéré un peu décevant :

```
*** Aucun formulaire pour la page : 192.168.0.10/recordpress/ ***  
*** Aucun formulaire pour la page : 192.168.0.10/recordpress/index.php ***  
*** Aucun formulaire pour la page : 192.168.0.10/recordpress/new.php ***
```

Et ceci pour toutes les pages du site web.

Après analyse des différentes pages de RecordPress, nous nous sommes aperçus que l'espace authentifié de cette application web était accessible seulement depuis la page : <http://192.168.0.10/recordpress/admin>. Cette adresse HTML nous offre la possibilité d'être authentifié via le formulaire suivant :

The image shows a login form for RecordPress. At the top, there is a logo consisting of a treble clef inside a circle, followed by the text "RECORDPRESS". Below the logo, there are two input fields: one for "Username:" and one for "Password:". Below the password field, there is a button labeled "Login »". The entire form is set against a dark gray background.

Formulaire d'authentification de RecordPress

Ainsi c'est seulement lors de la soumission de ce formulaire que l'utilisateur sera redirigé ou non vers l'espace authentifié. Le mode récursif de notre logiciel ne peut donc pas accéder directement à cet espace puisque l'adresse ne se présente pas sous la forme d'un lien du type ``.

Nous avons donc accédé manuellement à cet espace. Une fois le formulaire envoyé nous avons été redirigé à l'adresse : <http://192.168.0.10/recordpress/admin/rp.php>. Il nous suffisait donc de relancer notre programme avec cette adresse comme point de départ :

```
./main.py -R http://192.168.0.10/recordpress/admin/rp.php  
« PHPSESSID=cfa35fcb6447738b72ffce93923d3c27 »
```

Encore une fois, le résultat a été décevant :

```
*** Aucun formulaire pour la page : 192.168.0.10/recordpress/admin/rp.php ***  
*** Aucun formulaire pour la page : 192.168.0.10/recordpress/admin/rp-myprofile.php ***  
*** Aucun formulaire pour la page : 192.168.0.10/recordpress/admin/index.php?logout= ***  
*** Aucun formulaire pour la page : 192.168.0.10/recordpress/admin/rp-pages.php ***  
*** Aucun formulaire pour la page : 192.168.0.10/recordpress/admin/rp-settings.php ***  
*** Aucun formulaire pour la page : 192.168.0.10/recordpress/admin/rp-help.php ***  
*** Aucun formulaire pour la page : 192.168.0.10/recordpress/admin/rp-upload.php ***  
*** Aucun formulaire pour la page : 192.168.0.10/recordpress/admin/rp-search.php ***
```

Selon le logiciel, il n'existait donc aucun formulaire statique dans un contexte authentifié. Or, la page <http://192.168.0.10/recordpress/admin/rp-myprofile.php> est la suivante :

Name:

Enter your real name. Hmm you do have a name?

E-mail:

Enter your e-mail you use regularly!

Password:

Enter your new or existing password here. Make your password hard and please remember it since it's impossible to get it back!

Repeat password:

Please repeat your password again!

Formulaire de changement de mot de passe sur RecordPress

Il s'agit bien d'un formulaire pour modifier son mot de passe. Celui-ci devrait donc être retourné par le détecteur.

Une autre chose intéressante à remarquer est que, lorsque nous relançons le programme une seconde fois avec les mêmes paramètres, la sortie n'est plus la même :

*** Aucun formulaire pour la page : 192.168.0.10/recordpress/admin/rp.php ***

On a l'impression que le programme a été lancé en mode statique. En relançant plusieurs fois le résultat reste cette fois-ci le même.

Que s'est-il donc passé ?

La réponse est en fait assez simple. Comme on peut le voir dans les premiers résultats du logiciel, celui-ci a appelé la page

<http://192.168.0.10/recordpress/admin/index.php?logout=> :

*** Aucun formulaire pour la page : 192.168.0.10/recordpress/admin/index.php?logout= ***

Le programme a donc déclenché l'action logout, ce qui a eu pour effet d'expirer la session rattachée au cookie : **PHPSESSID=cfa35fcb6447738b72ffce93923d3c27**. Ainsi, lorsque le détecteur effectuait une requête HTTP GET vers chacune des pages appartenant au contenu authentifié du site, il était invariablement redirigé vers le formulaire d'authentification. Il ne détectait donc aucun formulaire statique authentifié.

Nous avons donc décidé de ne pas prendre en compte les liens contenant le mot clé « logout ». Nous sommes bien conscient qu'il s'agit peut être d'une mesure un peu radicale qu'il faudrait pouvoir activer ou désactiver selon les besoins du client, cependant notre logiciel n'étant qu'un prototype nous l'avons intégré directement au code.

Cette fois-ci la sortie a été beaucoup plus fournie. De nombreux formulaires ont été remontés dont celui permettant de changer son mot de passe :

```
*** Pour la page : 192.168.0.10/recordpress/admin/rp-myprofile.php ***
<form method="post" name="rpmyprofile" action="rp-myprofile.php?db" onSubmit="return
CheckForm();">

        Name:<br />
        <label><input name="f1" type="text" style="height:35px; font-size:25px;
width: 774px;" value="admin" /></label><br />
        <div class="rp-small-notify">Enter your real name. Hmm you do have a
name?</div>
        <br /><br />

        E-mail:<br />
        <label><input name="f2" type='text' style="height:35px; font-size:25px; width:
774px;" value="admin@admin.fr" /></label><br />
        <div class="rp-small-notify">Enter your e-mail you use regularly!</div>
        <br /><br />

        Password:<br />
        <label><input name="f3" type="password" style="height:35px; font-size:25px;
width: 400px;" /></label><br />
        <div class="rp-small-notify">Enter your new or existing password here. Make
your password hard and<br />
        please remember it since it's impossible to get it back!</div>
        <br /><br />

        Repeat password:<br />
        <label><input name="f4" type="password" style="height:35px; font-size:25px;
width: 400px;" /></label><br />
        <div class="rp-small-notify">Please repeate your password again!</div>
        <br /><br />
        <label><input type='submit' name='db' value='Update &raquo;,'
style='height:35px; font-size:20px; border-color:#999999;' /></label>
</form>
```

Les différents tests effectués nous ont permis de voir que notre logiciel répondait bien au cahier des charges que nous avons fixé lors de l'écriture de l'algorithme d'une manière formelle à savoir la détection de contenu authentifié statique.

Cependant ces tests nous ont aussi montré certaines limites particulièrement dans le mode récursif. Nous sommes bien conscient qu'il aurait été sans doute plus simple d'utiliser un « crawler » de contenu web existant pour ce mode. Cependant comme nous l'avons déjà dit nous avons préféré tout développer de zéro afin d'acquérir le maximum de connaissance grâce à ce projet. Des problématiques comme nous avons pu en rencontrer en testant RecordPress notamment ne seraient jamais apparu avec un « crawler » existant, ce qui aurait rendu ce projet certainement moins intéressant.

Nous allons maintenant voir les améliorations qu'il serait possible d'apporter au livrable pour que celui-ci devienne encore plus efficace.

5) Améliorations à apporter

Comme nous l'avons indiqué précédemment dans ce document, le logiciel que nous avons fourni ne permet pas de détecter directement les formulaires sensibles aux attaques de type CSRF mais « seulement » ceux statiques apparaissant dans un contexte authentifié. Bien que la détection du contexte soit la partie la plus compliquée de la détection de failles CSRF, ce n'est pas la seule. En effet, des formulaires signalés par notre programme de détection peuvent avoir une protection spéciale contre les attaques de « Cross Site Request Forgery ». Par exemple, Facebook offre ce genre de sécurité, notamment pour les formulaires permettant de déclencher l'action « j'aime ». Cette protection est offerte grâce à un champ caché du formulaire :

```
<input type="hidden" value="bc88b65186c6159880fc31b8e6a1641b" name="post_form_id" id="post_form_id" autocomplete="off">
```

Cette valeur est changée à chaque rechargement de la page. Ainsi l'amélioration majeure à apporter à notre logiciel de détection serait la capacité de détecter des jetons « anti-replay » dans les formulaires statiques authentifiés.

Une autre chose intéressante que nous avons pu remarquer lors de nos tests est que certains sites web utilisent SSL et le protocole HTTPS pour toutes les données appartenant à un contexte authentifié. Lorsqu'un utilisateur se connecte à Gmail par exemple, aucune donnée n'est échangée en clair mais toutes passent par le canal HTTPS chiffré négocié lors de la connexion. Ce chiffrement ne protège absolument pas les clients des attaques CSRF. En effet, lors de la connexion à la messagerie de Google, l'échange des clés publiques est transparent pour l'utilisateur. Ainsi si celui-ci déclenche un lien à son insu (caché dans une image ou une iframe HTTP par exemple) alors qu'il est authentifié sur l'application, la mise en place du protocole SSL se fera automatiquement et l'action sera exécutée. Cependant notre logiciel ne comprend que le protocole HTTP. Il n'est donc pas en mesure de dialoguer avec un serveur SSL ou TLS pour établir une connexion chiffrée avant de tenter de détecter des formulaires authentifiés. Ainsi, une autre amélioration qui pourrait s'avérer payante serait, pour le programme, la possibilité de parler et comprendre le protocole SSL ou TLS.

Bien sûr de nombreuses autres améliorations pourraient être apportées comme la rapidité d'exécution du programme ou encore l'augmentation de l'efficacité du « crawler » pour le mode récursif.

Cette partie du rapport nous a permis d'aborder les différents points qui touchaient au livrable de notre projet de fin d'études. Nous avons tenté d'expliquer avec le plus de précisions possibles les différentes étapes qui ont été nécessaires à la réalisation de notre projet ainsi que notre logique dans l'implémentation en Python de l'algorithme.

IV) Conclusion

Ce projet nous a permis de nous forger une première expérience dans le monde de la sécurité. Tous au long de son déroulement nous avons pu découvrir des cas concrets de failles XSS et CSRF.

Notre rapport avec l'industriel initiateur du projet s'est avéré très amical et professionnel.

Nous avons, de plus, pu découvrir la communauté de la sécurité informatique via des forums et des mailing-listes.

Alors que nous n'avions qu'une connaissance très théorique des attaques dont étaient victimes les sites web avant le début de ce projet, nous avons appris pendant ces six derniers mois comment les mettre en œuvre et se protéger contre elles. L'implémentation de l'algorithme de détection en Python nous a aussi apporté la connaissance d'un nouveau langage de programmation ainsi qu'une expérience dans le développement d'application réseau.

Dans ce rapport, nous nous sommes attachés à reprendre l'ensemble des étapes importantes de ce projet ainsi qu'à décrire de la façon la plus claire possible notre vision des failles CSRF. Cependant, il est à noter qu'une grande partie du projet n'a pas porté directement sur ce type de failles mais sur la détection du contenu authentifié d'un site internet. Il s'agit d'une problématique plus large qui peut par exemple aussi permettre de détecter quelles injections SQL peuvent s'avérer très dangereuses pour un utilisateur.

V) Annexes

Voici le code Python des différentes fonctions dont nous avons parlé dans notre rapport.

1) Module HTTP_request.py :

```
def url_to_host_pagename(url):
    """Convertie l'url complet de la page en host
    et nom de la page au sein du site
    Exemple : url = http://www.monsite.fr/exemple.php
    host : www.monsite.fr
    nom de la page : exemple.php
    Argument : - string url : adresse complete de la page
    Return : - dictionnaire :
                - Cle host : host
                - Cle pagename : nom de la page
    """
    host_pagename = {}
    url=url.lstrip("http://")

    buff_host_pagename = url.partition('/')
    #buff_host_pagename = ['www.monsite.fr', '/', 'exemple.php']

    host_pagename["host"] = buff_host_pagename[0]

    if len(buff_host_pagename[1]) != 0:
        host_pagename["pagename"] = buff_host_pagename[1]+buff_host_pagename[2]
    else:
        host_pagename["pagename"] = "/"

    return host_pagename

def handle_redirect(response, host, pagename, cookie=""):
    """Gere les redirections du type 302 Redirect ou 302 Moved temporarily
    Utile si l'utilisateur entre une url du type http://www.monsite.fr. Ici seul le host est renseigne.
    Suivant la configuration du serveur web, le client peut recevoir un message de redirection
    Arguments : - objet httpplib.HTTPResponse : reponse susceptible d'etre une redirection
                - string host : site web
                - string pagename : nom de la page
                - string cookie : cookie d'authentification
    Return : - objet httpplib.HTTPResponse : reponse une fois les redirections effectuees
    """
    while response.status == 302:
        host_pagename = {}
        headers = response.getheaders()

        #l'adresse de la redirection se trouve dans le champ location du header HTTP
        new_location = [location for name, location in headers if name == "location"][0]
        intern_or_relative = is_intern_or_relative_link(host, new_location)
        if intern_or_relative != 0:
            host_pagename["host"] = host
            host_pagename["pagename"] = get_absolute_path(new_location)
        else:
            host_pagename = url_to_host_pagename(new_location)

        response = send_request(host_pagename["host"], host_pagename["pagename"], cookie)

    return response
```

2) Module handle_html.py :

```
class Node:
    """Classe representant un arbre.
    Un arbre (ou noeud) est compose de :
        - string value : valeur du noeud
        - list child : liste des noeuds enfant
    Cette classe implemente les methodes :
        - add_child : ajoute un enfant a la fin de la liste
        - remove_child : supprime un enfant a un index precis
        - is_equal : permet de savoir si l'arbre est identique a un
                    autre
    """

    def __init__(self, value=""):
        """Constructeur de la classe. Un noeud par default n'a pas d'enfant
        et est son propre parent.
        Argument : - string value : valeur du noeud a construire
        Return : - objet Node : noeud sans enfant, etant son propre pere et ayant la valeur value
        """
        self.value = value
        self.child = []
        self.parent = self

    def add_child(self, value="", node=None):
        """Ajoute un enfant a la fin de la liste des enfants du noeud. L'ajout d'un enfant peut se faire de
        deux manieres :
            - En precisant la valeur. L'enfant cree n'aura donc pas d'enfant
            - En precisant le noeud. L'enfant cree sera le noeud passe en argument
        Arguments : - string value : valeur du noeud enfant si ajout par valeur
                    - objet Node node : le noeud a ajouter si ajout par noeud
        """
        if node == None:
            child = Node(value)
        else:
            child = node
        child.parent = self
        self.child.append(child)

    def remove_child(self, index):
        """Efface de la liste l'enfant se situant a l'index passe en argument
        Argument : - int index : index de l'enfant a effacer
        """
        del self.child[index]

    def is_equal(self, tree):
        """Permet de savoir si le noeud est identique a un autre passe
        en argument. Deux noeuds sont egaux si :
            - leurs valeurs sont egales
            - ils ont le meme nombre d'enfants
            - chaque enfant du noeud est identique a l'enfant du noeud
              passe en argument qui se situe a la meme position
        Argument : - objet Node tree : noeud a comparer
        Return : - int :
            - 0 : les noeuds ne sont pas egaux
            - 1 : les noeuds sont egaux
        """
        if self.value != tree.value or len(self.child) != len(tree.child):
            return 0
```

```

for i in range(len(self.child)):
    if self.child[i].is_equal(tree.child[i]) == 0:
        return 0
    return 1

```

```

def build_tree(string_to_tree):

```

"""Construit un arbre a partir d'un code HTML sous forme de string.
 Le code HTML est parcourut lineairement et l'arbre est
 construit suivant l'algorithme :

- Tant que le code HTML n'a pas ete parcourut entierement :
- Si le premier caractere est '<' :
- S'il s'agit d'un tag fermant (second caractere '/') : le noeud en cours devient son parent
- Sinon on ajoute le tag comme enfant au noeud courant et :
 - S'il s'agit d'un tag ouvrant et s'il existe un tag fermant correspondant au tag ouvrant, le noeud courant deviant l'enfant qui vient d'etre ajoute
 - Sinon, on ajoute le string qui se trouve entre les deux tags comme enfant du noeud courant
- On enleve le tag qui vient d'etre traite au code HTML

Argument : - string string_to_tree : code HTML a partir duquel l'arbre est construit

Return : - objet Node : l'arbre correspondant au code HTML

"""

```

tree = Node("root")

```

```

buffnode = tree

```

```

while string_to_tree != "":

```

```

    string_to_tree = string_to_tree.strip()

```

```

    #debut d'un tag

```

```

    if string_to_tree[0] == '<':

```

```

        index = string_to_tree.find('>')

```

```

        #tag fermant

```

```

        if string_to_tree[1] == '/':

```

```

            buffnode = buffnode.parent

```

```

        else:

```

```

            #tag ouvrant - fermant

```

```

            if string_to_tree[index - 1] == '/':

```

```

                buffnode.add_child(value=string_to_tree[0:index + 1])

```

```

            #tag ouvrant

```

```

            else:

```

```

                buffnode.add_child(value=string_to_tree[0:index + 1])

```

```

                value_tag = string_to_tree[1:index]

```

```

                #on verifie qu'il existe un tag fermant

```

```

                #sans cette verification cela peut decaler

```

```

                #l'ensemble de l'arbre

```

```

                if is_there_endtag(value_tag, string_to_tree[index + 1:]) != 0:

```

```

                    buffnode = buffnode.child[len(buffnode.child) - 1]

```

```

            string_to_tree = string_to_tree[index + 1:]

```

```

        #on est dans le cas d'un string entre deux tags

```

```

        #par exemple : string_to_tree = Blabla</a> (le tag <a> a ete traite au-dessus)

```

```

        else:

```

```

            index = string_to_tree.find('<')

```

```

            buffnode.add_child(value=string_to_tree[0:index])

```

```

        string_to_tree = string_to_tree[index:]

    return tree

def is_there_endtag(value, string_to_tree):
    """Permet de determiner, a partir de la valeur complete d'un tag ouvrant
    et de l'ensemble du code HTML qui suit ce tag, si il y a bien le tag fermant
    correspondant. Par exemple il arrive que les tags <br>, <li> ou <input> s'ouvre sans se fermer.
    Il faut alors leurs appliquer un traitement specifique.
    Pour determiner si il y a bien un tag fermant, on parcourt l'ensemble du code HTML a la
    Recherche d'une balise <nom_du_tag> ou </nom_du_tag>. Pour chaque balise trouvee, on
    change le score grace a la fonction move_score. Lorsque le score arrive a zero, on retourne la
    position de la balise dans le score.
    Si le score n'arrive jamais a zero, on retourne zero.
    Arguments : - string value : valeur complete du tag sans les balises '<' et '>'
                - string string_to_tree : l'ensemble du code HTML suivant le tag de valeur value
    Return : - int :
                - 0 : il n'existe pas de tag fermant correspondant
                - index : la position relative du tag fermant dans le code
    """
    index = 0
    score = 1
    #on recupere la nom du tag
    name = from_value_to_nametag(value)
    name = re.escape(name)
    #on construit la regexp pour aller chercher les balises <nom_du_tag> ou </nom_du_tag>
    #dans la suite du code
    regexp = r"<" + name + r" *?>|</" + name + r">"
    match_tag = re.search(regexp, string_to_tree)

    #on parcourt l'ensemble du code HTML
    while match_tag != None:
        #on enleve les balises '<' et '>'
        value_tag = match_tag.group(0)[1:len(match_tag.group(0)) - 1]
        #on change le score en fonction de la balise trouvee
        #ceci est utile si a l'interieur d'un paragraphe HTML se trouve
        #un autre paragraphe avec le meme nom
        #par exemple : <a>
        #                <b>
        #                <a>Blabla</a>
        #                </b>
        #                </a>
        score = move_score(value_tag, score)
        index = index + match_tag.end()
        if score == 0:
            return index
        string_to_tree = string_to_tree[match_tag.end():]
        match_tag = re.search(regexp, string_to_tree)

    return 0

def extract_form(tree_in, bufftree):
    """Extrait tous les formulaires d'un arbre construit a partir
    de la reponse a une requete HTML. Des qu'un formulaire est trouve, on
    ne descend pas plus loin dans l'arbre, on ajoute l'ensemble du formulaire
    a l'arbre qui sera retourne en resultat
    Arguments : - objet Node tree_in : arbre contenant l'ensemble du code HTML et dont
                on veut extraire les formulaires
                - objet Node bufftree : arbre auquel on ajoute chaque formulaire trouve, il sera
    """

```

```

                                retourne en resultat
Return : - objet Node : arbre contenant uniquement les formulaires
"""
for i in range(len(tree_in.child)):
    if tree_in.child[i].value.startswith("<form ") is not True:
        extract_form(tree_in.child[i], bufftree)
    else:
        bufftree.add_child(node=tree_in.child[i])

return bufftree

def print_form(tree_form, string):
    """Imprime les formulaires contenus dans l'arbre
    sur la sortie standard. Plutot que de faire la transformation
    inverse arbre - code HTML, on va chercher directement
    dans le code HTML le formulaire correspondant.
    Arguments : - objet Node tree_form : arbre ne contenant que les formulaires a imprimer
                - string string : string contenant l'ensemble du code HTML de la page web en cours
                de traitement
    """
    for i in range(len(tree_form.child)):
        size_value = len(tree_form.child[i].value)
        #on cherche le debut du formulaire dans le code HTML a partir de la valeur complete de ce
        #dernier.
        #La valeur complete est la chaine de caractere compris entre les balises '<' et '>' d'un
        #formulaire
        #exemple : value = "form method="get" action="action.php" name="Name"
        index_debut = string.find(tree_form.child[i].value)
        buff_string = string[index_debut:]
        #on se sert de is_there_endtag pour trouver la fin du formulaire dans le code HTML
        index_fin = is_there_endtag(tree_form.child[i].value[1:size_value - 1], buff_string[size_value:])
        index_fin = index_fin + size_value
        if index_fin != 0:
            print buff_string[0:index_fin]

```

3) Module comparison_method.py :

```

def rdiff_tree(treeA, treeB, bufftree):
    """Applique la methode rdiff a deux arbres
    Arguments : - objet Node treeA : premier arbre a comparer
                - objet Node treeB : second arbre a comparer
                - objet Node bufftree : arbre servant a bufferiser l'arbre qui sera retourne.
                Il est passe en argument pour que la recursivite
                puisse avoir lieu en gardant en memoire l'etat de la
                comparaison
    Return : - objet Node : Un arbre contenant tous les noeuds commun a treeA et treeB
    """
    if treeA.child != []:
        for i in range(len(treeA.child)):
            if i < len(treeB.child) and rdiff(treeA.child[i].value, treeB.child[i].value) == 1:
                bufftree.add_child(value=treeA.child[i].value)
                #la recursivite a lieu ici
                #Un enfant vient d'etre rajoute a bufftree. On relance donc la fonction
                #sur cet enfant qui est le dernier de la liste (d'ou
                #bufftree.child[len(bufftree.child) - 1])
                rdiff_tree(treeA.child[i], treeB.child[i], bufftree.child[len(bufftree.child) - 1])
        return bufftree

```



```

def diff_treeform(treeformA, treeformB):
    """Fais la difference des deux arbres passes en argument. Les deux arbres ne contiennent que
    des formulaires. Par exemple : - root
        - form1
            - champ1.1
            - champ1.2
        - form2
            - champ2.1
                - souschamp2.1.1
            - champ2.2
            - champ2.3

    Ainsi les enfants du noeud root ne sont que des formulaires.
    Chaque arbre passe en argument doit avoir un noeud principal root
    La difference se fait suivant l'algorithme :
        - Si tous les champs d'un formulaire du premier arbre sont identiques a tous les champs d'un
          formulaire du second alors ne rien faire
        - Sinon ajouter le formulaire a l'arbre qui sera retourne en resultat
    Arguments : - objet Node treeformA : arbre dont on cherche les formulaires qui ne sont pas
    dans
                                le second argument
                - objet Node treeformB : arbre qui contient les formulaires dont on ne veut pas
    Return : - objet Node : arbre contenant tous les formulaires de treeformA qui ne sont pas
              dans treeformB
    """
    bufftree = Node("root")
    for i in range(len(treeformA.child)):
        count = 0
        if len(treeformB.child) == 0:
            return treeformA
        for j in range(len(treeformB.child)):
            #si les formulaires ne sont pas strictement egaux
            if treeformA.child[i].is_equal(treeformB.child[j]) == 0:
                count = count + 1
        #si le nombre de formulaire qui ne sont pas egaux est egale
        #au nombre d'enfant de treeformB alors c'est que le formulaire
        #de treeformA ne se trouve pas dans treeformB
        if count == len(treeformB.child):
            bufftree.add_child(node=treeformA.child[i])
    return bufftree

```