UNIVERSITÉ DE SHERBROOKE
Faculté de génie
Département de génie électrique et génie informatique

RAPPORT

Sécurité web
GEI-772

Présenté à
Guy Lépine

Présenté par
Brian Compagnat – comb2301
Joel Perron-Langlois – perj2324

Sherbrooke – 3 octobre 2018

Table des matières

1 INTRODUCTION

Dans cette problématique, les étudiants ont été plongés dans un scénario dont plusieurs entreprises font face de nos jours, soit des attaques informatiques. En effet, une application faisait l'objet de différents types de vulnérabilités populaires tel qu'indiqué par l'OWASP Top 10. Dans ce rapport, il vous sera présenté ces vulnérabilités dans l'application SansSoussi ainsi que les mesures de mitigation.

2 VULNÉRABILITÉS COMMUNES

2.1 INJECTION

Description

Les attaques par injections peuvent se produire sur toute source de donnée. Une attaque est dite réussite lorsque l'utilisateur envoie des données non autorisées à un interpréteur. L'une des injections les plus communes est l'injection SQL, avec laquelle nous pourrions supprimer des renseignements dans la base de données ou bien obtenir plus d'informations qu'habituellement retourné.

Application SansSoussi

L'application SansSoussi contenait des vulnérabilités d'injection, plus particulièrement des injections SQL pour la page de commentaires (*/comments*) et celle de la recherche (*/search*) de commentaires. En effet, comme on pouvait le voir dans le fichier « HomeController.cs » les commentaires étaient reçus en arguments dans une fonction et étaient directement insérés dans la base de données à l'aide d'une requête SQL. Ce faisant, un attaquant était en mesure d'injecter une commande SQL malicieuse et s'amuser avec la base donnée. Dans ce même fichier, on voit que le contrôleur gérait de la même façon la recherche de commentaires. En effet, avec la commande suivante, on est en mesure de retourner les adresses courriel des utilisateurs du site :

```
' UNION ALL SELECT email FROM dbo.aspnet_Membership;--
```

Mitigations

Afin de mitiger les injections, en général, on recommande de séparer les commandes des données. Quant à l'application SansSoussi, notre équipe a opté pour la désinfection des entrées et l'utilisation des paramètres SQL au lieu des arguments de fonctions directement. Comme on peut le voir dans le code suivant, l'utilisation des paramètres SQL sont traités comme des valeurs littérales et non comme du code exécutable, ce qui prévient les attaques d'injection SQL.

```
SqlCommand cmd = new SqlCommand("Select Comment from Comments where UserId = '" +
      user.ProviderUserKey + "' and Comment like @searchData", _dbConnection);
SqlParameter param = new SqlParameter("@searchData",
      System.Data.SqlDbType.NVarChar, 16);
param.Value = "%"+searchData+"%";
cmd.Parameters.Add(param);
```

2.2 VIOLATION DE GESTION D'AUTHENTIFICATION ET DE SESSION

Description

Cette vulnérabilité consiste à posséder un système d'authentification faible permettant aux malfaiteurs d'usurper l'identité d'un autre usager ou même de l'administrateur. Pour ce faire, ils peuvent utiliser des dictionnaires pour faire une attaque par force brute par exemple.

Application SansSoussi

Dans l'application SansSoussi, l'authentification se faisait à l'aide d'un identifiant ainsi qu'un mot de passe. Comme le mot de passe exigé n'était que de 6 caractères minimum sans autre restriction, une attaque de force brute aurait suffi pour trouver les identifiants d'un usager. De plus, Un CSRF aurait été possible dans la page /emails puisque le cookie "username" était simplement encodé en base64 et que dans cette page si le cookie était changé pour admin, bien qu'encodé en base64, on pouvait y lire tous les emails contenus dans la BD.

Mitigations

Tout d'abord, pour palier au problème de la modification du nom d'utilisateur dans les cookies, nous avons implémenté une solution d'anti-forgery qui implémente un token anti-CSRF qui change à chaque requête. Il devient donc très difficile de changer le cookie et le token sans que le serveur s'en rende compte. Ensuite, nous avons implémenté un système d'authentification et d'autorisation avec les APIs de Google afin de déléguer l'authentification. Ainsi, l'utilisateur s'authentifiera directement sur la plateforme de Google pour que l'API nous retourne ensuite les informations de l'utilisateur ainsi qu'un token renouvelable. Pour avoir accès aux données de l'utilisateur il aura fallu demander l'autorisation à l'utilisateur par l'entremise de l'API Google plus.

2.3 EXPOSITION DE DONNÉES SENSIBLES

Description

Comme le nom le décrit bien, l'exposition des données sensibles est aussi une vulnérabilité populaire. Une exposition de ses données a souvent lieu lorsque l'information transige, c'est-à-dire, entre le serveur et le client. Un malfaiteur peut alors surveiller le réseau pour voir les informations en clair ou bien être un *man in the middle* pour se faire passer pour quelqu'un d'autre. Ainsi, l'information des usagers est à risque.

Application SansSoussi

Dans l'application, aucun protocole de sécurité n'était utilisé afin de transiger l'information entre le serveur et le client. Il était donc possible de surveiller le réseau pour obtenir de l'information sensible sur les usagers qui se connecte à l'application par l'entremise du protocole HTTP simplement.

Mitigations

Afin de sécuriser l'information qui transige sur le réseau, il est important d'utiliser un protocole de sécurité qui encryptera l'information. Pour ce faire, nous avons utilisé SSL3.0 qui nous permet de communiquer entre le client et le serveur en HTTPS. Idéalement, le protocole TLS1.2 ou bientôt TLS1.3 serait choisi, mais étant donné la version du Framework, seulement SSL3.0 était disponible.

2.4 XML EXTERNAL ENTITIES (XXE)

Description

Pour les applications basées sur les fichiers XML, cette vulnérabilité consiste à télécharger des fichiers XML directement sans vérification. Ces fichiers peuvent être malsains et donc mettre en péril la sécurité de l'application.

Application SansSoussi

Cette vulnérabilité ne s'appliquait pas à l'application.

Mitigations

Parmi les méthodes de mitigation pour ce type de vulnérabilités on y retrouve l'utilisation d'un autre format comme le JSON, la validation du côté serveur avec une liste blanche, la mise à jour des composants utilisant ce format comme SOAP.

2.5 VIOLATION DE CONTRÔLE D'ACCÈS

Description

Un viol d'accès a lieu lorsqu'un attaquant consulte des données dont il n'est pas autorisé. Par exemple, un simple utilisateur pourrait avoir accès à des fonctions privilégiées où il pourrait ajouter ou modifier des données applicatives qui sont normalement seulement accessibles par les administrateurs de l'application.

Application SansSoussi

L'application contenait une page (*/emails*) affichant les adresses courriel des utilisateurs lorsqu'on était administrateur du site. Bien que cette page vérifiait déjà si la clé d' « anti-forgery » était exacte, on peut bénéficier d'une meilleure sécurité en ne se fiant pas aux cookies afin de donner accès aux informations de la page. En effet, les courriels étaient visibles lorsque le cookie « username » était encodé en base64 et contenait « admin ». On aurait pu minimalement utiliser un encodage auquel l'utilisateur ne pourrait pas décoder.

Mitigations

Afin de mitiger les risques reliés au viol d'accès, l'utilisation de la clé d' « anti-forgery » est un bon départ. On devrait bannir toutes clés facilement devinables par un attaquant, surtout dans le cas où cette clé est utilisée afin d'avoir des renseignements confidentiels.

On devrait plutôt utiliser un chiffrage dont seulement le serveur peut en déchiffrer le contenu.

2.6 MAUVAISE CONFIGURATION SÉCURITÉ

Description

Une mauvaise configuration de sécurité est une vulnérabilité très large qui englobe beaucoup de possibilités. Parmi s'est possibilités, nous retrouvons les comptes administrateurs par défaut, les failles de sécurités connues qui n'ont pas été mises à jour, les fichiers ou dossiers non protégés, etc.

Application SansSoussi

Dans l'application SansSoussi, il y avait majoritairement deux configurations causant problème. D'abord, les versions d'Asp.net étaient disponibles dans l'entête des requêtes, ce qui donne beaucoup trop d'informations pour les malfaiteurs. Ensuite, lors d'une erreur, un *stack trace* est envoyé, mais ce *stack trace* était aussi envoyé en production. Nous voulons éviter ceci puisqu'il donne des informations quant aux erreurs et parfois il peut indiquer une faille ou une méthode pour contourner la sécurité. Un exemple serait de retourner une commande SQL n'ayant pas fonctionner. En voyant la commande, un malfaiteur peut facilement la modifier pour obtenir d'autres informations.

Mitigations

Pour enlever les versions de l'entête, il suffit d'aller le configurer dans les configurations de l'application asp.net. Pour ce qui est des *stack trace*, une autre configuration permet de les envoyer seulement lorsqu'on est local, c'est-à-dire, en développement. Bien entendu, ces configurations ont été apportées à l'application web SansSoussi.

2.7 CROSS-SITE SCRIPTING (XSS)

Description

Cette vulnérabilité consiste à permettre aux malfaiteurs d'inclure des scripts client à l'aide d'un entrée de texte ou autre sur la plateforme web. Ainsi, le nouveau script faisant maintenant partie de la page web peut désormais accéder aux cookies ou autres informations détenues par le navigateur.

Application SansSoussi

L'application était vulnérable au XSS-S. En effet, sur la page des commentaires (*/comments*) aucune désinfection d'entrée n'était effectuée. Ainsi, un attaquant était en mesure d'insérer du code malicieux, qui s'exécutait à chaque fois qu'on utilisateur téléchargeait la page. Le commentaire suivant faisait en sorte qu'une fenêtre apparaissait dans le navigateur du client.

```
<a onmouseover="alert(1)" href="#"> Salut pop-up12!</a>
```

Mitigations

L'équipe a réussi à mitiger cette vulnérabilité en insérant une validation d'entrée et éliminait toutes requêtes contenant des caractères spéciaux autres que « . , ! ? ». Par ailleurs, la longueur du commentaire ne pouvait dépasser les 115 caractères. L'utilisation de Framework gérant par *désign* les attaques XSS aurait pu augmenter la sécurité du site.

2.8 INSECURE DESERIALIZATION

Description

Cette vulnérabilité consiste à la possibilité de générer du code malsain lors d'une conversion de données sérialisées venant d'un fichier ou d'un paquet réseau en un objet reçu par un client malfaiteur.

Application SansSoussi

Cette vulnérabilité ne s'applique pas pour l'application.

Mitigations

Plusieurs moyen existe pour ce type de vulnérabilité, parmi ceux-ci, il faut restreindre le nombre de types de donnée acceptés, même obliger qu'un type sécure, sinon on peut faire une vérification d'intégrité, s'assurer de na pas posséder de privilèges non nécessaires et aussi inscrire la dé sérialisation dans un fichier de journal pour pouvoir vérifier l'abus.

2.9 UTILISATION DE COMPOSANTS AVEC DES VULNÉRABILITÉS CONNUES

Description

Cette vulnérabilité consiste à ne pas tenir à jour les différents composants de l'application. Ainsi, lorsqu'une faille de sécurité est détectée sur un de ses composants et rendue publique, les méthodes d'attaques deviennent aussi publiques et très simples d'utilisation.

Application SansSoussi

Dans cette application, le composant étant le plus désuet et nous empêchant d'utiliser les derniers protocoles de sécurité est le Framework 4.

Mitigations

La mise à jour de .NET Framework 4 vers .NET Core nous donnerait la possibilité d'utiliser TLS1.2. Présentement l'application est limitée au protocole SSL3.0 qui est toujours bon, mais le TLS1.2 est préférable. De plus, l'utilisation de .NET Core permet un développement multi-plate-forme en plus de permettre plus de performance et une extensibilité plus facile.

2.10 INSUFFICIENT LOGGING & MONITORING

Description

Une insuffisance de *logs* et de surveillance permet effectivement aux malfaiteurs de tenter plusieurs attaques sans se faire détecter. C'est pourquoi cette insuffisance consiste à une vulnérabilité importante. Par exemple, le fait de *logger* les tentatives d'authentification suivi d'un échec permet de découvrir les attaques par force brute si elles ne sont pas déjà empêchées!

Application SansSoussi

Dans cette application, il n'y a pas de système de journal d'implémenté ce qui nous permet donc pas d'observer les tentatives de connexion et les différentes erreurs qui surviennent comme des erreurs SQL qui pourraient montrer une tentative d'injection.

Mitigations

Tout d'abord il faudrait implémenter le système de journal. Ensuite, il faudrait ajouter des entrées dans le journal pour chaque tentative de connexion échouée, chaque erreur de validation, etc. Ce système de journal doit par contre rester léger pour s'y retrouver facilement et ainsi pouvoir monitorer de façon courante.

3 CONCLUSION

Bref, cette problématique a permis aux étudiants à découvrir les failles d'application web les plus répandues et comment les mitiger.