I.U.T Clermont-Ferrand

Projet n°14
OpenThermostat
Rapport

Bilan des failles de sécurité

Élèves :
Benoît Louveau
Elouan Raffray
Adrien Lenoir

Tuteur :
Sébastien Salva

Décembre 2017

Table des matières

1 Les failles de sécurité via ZapProxy **2**
1.1 **Injection SQL (Risque élevé)** 2
1.2 **Injection SQL - Contournement de l'authentification (Risque élevé)** 3
1.3 **X-Frame-Options Header Not Set (Risque moyen)** 4
1.4 **Protection XSS du navigateur Internet non activée (Risque faible** 5
1.5 **En-tête X-Content-Type-Options manquant (Risque faible** 6
1.6 **Injections Html JavaScript (Risque fort** 7

1 Les failles de sécurité via ZapProxy

1.1 Injection SQL (Risque élevé)

- **Description :** Une injection SQL peut être possible.

- **Autres informations :**
  1. Les résultats de la page ont été manipulés avec succès en utilisant les conditions booléennes [query' AND '1'='1' –] et [query' OR '1'='1' –]
  2. La valeur du paramètre en cours de modification NOT est extraite de la sortie HTML afin de comparaison.
  3. Les données n'ont PAS été retournées pour le paramètre d'origine.
  4. La vulnérabilité a été détectée en manipulant le paramètre pour récupérer avec succès plus de données qu'initialement retournées.
- **Solution(s) :**
  1. Ne faites pas confiance aux entrées du côté client, même si des mécanismes de validation sont en place côté client. En général, contrôlez du côté serveur le type de chaque donnée.
  2. Si l'application utilise JDBC, utilisez les PreparedStatement ou CallableStatement, avec les paramètres passés par ' ?'.
  3. Si l'application utilise ASP, utilisez les Objects de Commande ADO avec un typage fort et des requêtes paramétrées.
  4. Si l'utilisation de Procédure Stockées est possible, utilisez-les.
  5. Ne concaténez *pas* les chaînes de caractères dans les requêtes des procédures stockées, ou utilisez les fonctions 'exec', 'exec immediate' ou d'autre fonctions équivalentes !
  6. Ne créez pas des requêtes SQL dynamiques par simples concaténation de chaînes de caractères.
  7. Échappez toutes les données reçues du client.
  8. Appliquez une 'liste blanche' des caractères autorisés, ou une 'liste noir' des caractères interdits dans les entrées de l'utilisateur.
  9. Appliquez le principe de moindre privilège en utilisant les privilèges utilisateur minimaux sur la base de données. En particulier, évitez l'utilisation des utilisateurs 'sa' ou 'db-owner'. Ceci n'évite pas les injections SQL, mais minimise leur impact.
  10. Accordez les plus faibles droits d'accès aux bases de données nécessaires à l'application.
- **URL :**
  1. GET : http ://192.168.1.1/config ?query=query%27+AND+%271%27%3D%271%27+–+
  2. POST : http ://192.168.1.1/ ?query=query%27+AND+%271%27%3D%271%27+–+

1.2 Injection SQL - Contournement de l'authentification (Risque élevé)

- **Description :** L'injection SQL peut être possible sur une page de connexion, ce qui peut permettre de contourner le mécanisme d'authentification de l'application.

- **Solution(s) :**
  1. Ne faites pas confiance aux entrées du côté client, même si des mécanismes de validation sont en place côté client. En général, contrôlez du côté serveur le type de chaque donnée.
  2. En général, contrôlez du côté serveur le type de chaque donnée.
  3. Si l'application utilise ASP, utilisez les Objects de Commande ADO avec un typage fort et des requêtes paramétrées.
  4. Si l'application utilise JDBC, utilisez les PreparedStatement ou CallableStatement, avec les paramètres passés par ' ?'
  5. Si l'application utilise ASP, utilisez les Objects de Commande ADO avec un typage fort et des requêtes paramétrées.
  6. Si l'utilisation de Procédure Stockées est possible, utilisez-les.
  7. Ne concaténez *pas* les chaînes de caractères dans les requêtes des procédures stockées, ou utilisez les fonctions 'exec', 'exec immediate' ou d'autre fonctions équivalentes !
  8. Ne créez pas des requêtes SQL dynamiques par simples concaténation de chaînes de caractères.
  9. Échappez toutes les données reçues du client.
- **URL :**
  1. POST : http ://192.168.1.1/

1.3 X-Frame-Options Header Not Set (Risque moyen)

- **Description :** L'en-tête X-Frame-Options n'est pas incluse dans la réponse HTTP pour protéger contre les attaques de 'ClickJacking'.

- **Solution(s) :**
  1. La plupart des navigateurs Web modernes prennent en charge l'en-tête HTTP X-Frame-Options. Assurez-vous qu'il est défini sur toutes les pages Web renvoyées par votre site.
- **URL :**
  1. GET : http ://192.168.1.1
  2. GET : http ://192.168.1.1/
  3. GET : http ://192.168.1.1/config
  4. GET : http ://192.168.1.1/Dombroadcaster
  5. GET : http ://192.168.1.1/log
  6. GET : http ://192.168.1.1/planning
  7. GET : http ://192.168.1.1/sensorprint
  8. POST : http ://192.168.1.1/
  9. POST : http ://192.168.1.1/mode

1.4 Protection XSS du navigateur Internet non activée (Risque faible

- **Description :** La protection XSS du navigateur Internet n'est pas activée, ou est désactivée par la configuration de l'en-tête de réponse HTTP 'X-XSS-Protection' sur le serveur web.

- **Autres informations :**
  1. L'en-tête de réponse HTTP X-XSS-Protection autorise le serveur internet à activer ou désactiver le mécanisme de protection XSS du navigateur internet. Les valeurs suivantes tenteraient d'activer la protection :
  2. X-XSS-Protection : 1 ; mode = block
  3. X-XSS-Protection : 1 ; report = http ://www.example.com/xss
  4. Les valeurs suivantes désactiveraient cette protection :
  5. X-XSS-Protection : 0
  6. L'en-tête de réponse HTTP X-XSS-Protection est actuellement supportée par Internet Explorer, Chrome et Safari (WebKit).
  7. Notez que cette alerte n'est déclenchée que si le corps de réponse pouvait contenir une charge utile XSS (avec un type de contenu texte, d'une longueur différente de zéro).
- **Solution(s) :**
  1. Assurez-vous que le filtre XSS du navigateur internet est activé, en renseignant l'en-tête de réponse HTTP X-XSS-Protection à '1'.
- **URL :**
  1. GET : http ://192.168.1.1
  2. GET : http ://192.168.1.1/
  3. GET : http ://192.168.1.1/config
  4. GET : http ://192.168.1.1/Dombroadcaster
  5. GET : http ://192.168.1.1/log
  6. GET : http ://192.168.1.1/planning
  7. GET : http ://192.168.1.1/sensorprint
  8. POST : http ://192.168.1.1/
  9. POST : http ://192.168.1.1/mode

1.5 En-tête X-Content-Type-Options manquant (Risque faible

- **Description :** L'en-tête X-Content-Type-Options contre le sniffing MIME n'est pas renseigné à 'nosniff'. Ceci permet à de vielles versions d'Internet Explorer et de Chrome de pratiquer le sniffing MIME sur le corps de réponse, conduisant potentiellement à l'interprétation et l'affichage du contenu dans un autre type que celui déclaré. A l'heure actuelle (début 2014), les anciennes versions de Firefox utiliseront le type de contenu déclaré (au cas où un type est déterminé), plutôt qu'analyser le MIME.

- **Autres informations :**
  1. Ce problème s'applique toujours aux pages de type erreur (401, 403, 500, etc.), car ces pages sont encore souvent touchées par des problèmes d'injection, auquel cas il est encore possible que les navigateurs interprètent des pages autrement que selon leur type de contenu réel.
- **Solution(s) :**
  1. Assurez-vous que l'application ou le serveur internet renseigne l'en-tête Content-Type correctement, et que l'en-tête X-Content-Type-Options prenne la valeur 'nosniff' pour toutes les pages internet.
  2. Si possible, assurez-vous que l'utilisateur utilise un navigateur moderne et conforme aux standards, qui ne pratique pas du tout le sniffing MIME, ou qui puisse être commandé par l'application ou le serveur internet de manière à ne pas pratiquer le sniffing MIME.
- **URL :**
  1. GET : http ://192.168.1.1
  2. GET : http ://192.168.1.1/
  3. GET : http ://192.168.1.1/config
  4. GET : http ://192.168.1.1/Dombroadcaster
  5. GET : http ://192.168.1.1/log
  6. GET : http ://192.168.1.1/planning
  7. GET : http ://192.168.1.1/saveconfigfile
  8. GET : http ://192.168.1.1/sensorprint
  9. POST : http ://192.168.1.1/
  10. POST : http ://192.168.1.1/mode

1.6 Injections Html JavaScript (Risque fort

- **Description :** Tous les champs de l'application Web ne sont pas protégés. Il est donc possible d'injecter du code (HTML/JavaScript) dans ces derniers :
  - (ex : "><script>alert("hello")</script>)
  Cet exemple est basique et bénin, mais cette faille peut permettre à un utilisateur d'afficher images, texte comme il le souhaite, mais aussi exécuter des scripts JavaScript à potentiel malicieux, ou encore de récupérer des informations sur les utilisateurs (cookies phpsessionid, etc).

- **Solutions :**
  1. Il est nécessaire de protéger ces champs, une solution potentielle est le patron d'Input Guard. D'autres solutions impliquent l'utilisation de PHP pour vérifier les inputs rentrés par l'utilisateurs avec des classes comme validation et nettoyage.