Analyse d'une Attaque par Injection SQL Aveugle Basée sur le Temps

Résumé

Ce document de synthèse détaille une cyberattaque sophistiquée identifiée à travers l'analyse de fichiers journaux. L'attaque, menée depuis l'adresse IP `192.168.1.23`, est une injection SQL aveugle basée sur le temps (`time-based blind SQL injection`). L'attaquant a réussi à exfiltrer des données sensibles, spécifiquement un mot de passe de la table `membres`, en manipulant délibérément les temps de réponse du serveur.

La méthode consiste à dissimuler des charges utiles SQL malveillantes, doublement encodées (Base64 puis URL), dans le paramètre `order` d'une requête GET. Ces charges utiles convertissent les caractères du mot de passe en binaire et utilisent des instructions `SLEEP()` conditionnelles pour introduire des délais spécifiques (0, 2, 4 ou 6 secondes) en fonction de la valeur des bits. En mesurant les intervalles de temps entre les requêtes successives consignées dans les journaux, il est possible de reconstituer le mot de passe bit par bit. L'analyse des journaux confirme le caractère méthodique, automatisé et réussi de cette exfiltration furtive de données.

1. Mécanisme et Méthodologie de l'Attaque

Les journaux d'événements révèlent une attaque coordonnée visant à extraire des informations d'une base de données sans obtenir de réponse directe du serveur. L'attaquant déduit les données en mesurant les délais qu'il provoque lui-même.

Caractéristiques de l'attaque :

- **Type :** Injection SQL aveugle basée sur le temps (Time-Based Blind SQL Injection).
- **Source :** Toutes les requêtes malveillantes proviennent de l'adresse IP locale `192.168.1.23`.
- **Cible :** Le mot de passe (`password`) de l'utilisateur avec `id=1` dans la table `membres`.
- **Vecteur :** Le paramètre `order` dans les requêtes GET adressées à la page `/admin/`.

Processus d'exfiltration, étape par étape :

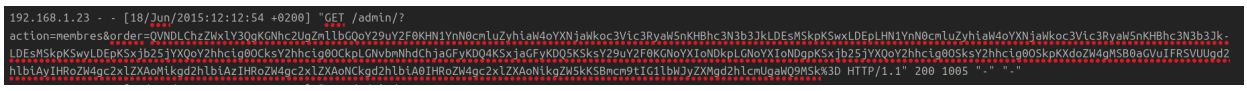
1. **Encodage de la Charge Utile :** Chaque commande SQL malveillante est d'abord encodée en Base64, puis en encodage URL. Cette double dissimulation vise à contourner les systèmes de détection simples.

2. **Extraction Bit par Bit :** L'attaquant ne vole pas le mot de passe en une seule fois. Il le reconstitue bit par bit en convertissant chaque caractère en sa valeur binaire (une suite de 7 bits pour un caractère ASCII).
3. **Manipulation du Temps de Réponse :** L'attaquant envoie une série de requêtes pour tester les bits, généralement deux à la fois. La charge utile SQL contient une instruction `sleep()` conditionnelle qui force le serveur à attendre un temps précis en fonction de la valeur des bits testés.
    - **Bits `00` :** Délai de 0 seconde.
    - **Bits `01` :** Délai de 2 secondes (`sleep(2)`).
    - **Bits `10` :** Délai de 4 secondes (`sleep(4)`).
    - **Bits `11` :** Délai de 6 secondes (`sleep(6)`).
4. **Reconstitution des Données :** L'attaquant (ou un analyste) mesure la différence temporelle entre les horodatages des requêtes dans les journaux pour déduire la valeur des bits.
5. **Assemblage du Mot de Passe :** En assemblant les séquences de bits reconstituées, l'attaquant reforme les caractères ASCII et, finalement, le mot de passe complet. La progression systématique (de `substring(password,1,1)` à `substring(password,2,1)`, etc.) confirme l'automatisation du processus.

2. Analyse Détaillée d'une Entrée de Log

Chaque ligne du fichier journal est une pièce du puzzle.

L'analyse forensique de ses composants permet de comprendre l'attaque en profondeur.

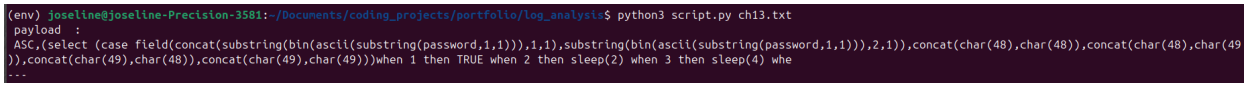


Composant	Exemple	Signification et Importance Forensique
Adresse IP Source	`192.168.1.23`	Identifie la machine à l'origine de l'attaque. C'est une adresse privée, indiquant une source sur le réseau local. Toutes les requêtes proviennent de cette unique adresse.
Ident / User	`- -`	Champs Ident et User non renseignés. Indique qu'aucune authentification HTTP n'a été utilisée.
Horodatage (Timestamp)	`[18/Jun/2015:12:12:54 +0200]`	Date, heure et fuseau horaire de la requête. **C'est l'élément clé** pour reconstituer l'attaque en calculant les deltas de temps entre les requêtes.
Requête HTTP	`"GET /admin/?action=membres&order=...%3D HTTP/1.1"`	Contient le vecteur d'attaque. Le paramètre `order` transporte la charge utile SQL doublement encodée. La présence d'une longue chaîne se terminant par `%3D` (le `=` encodé) est hautement suspecte.
Code de Réponse HTTP	`200`	Indique que le serveur a traité la requête avec succès. Cela confirme que la vulnérabilité est exploitable et que l'injection SQL n'a pas généré d'erreur côté serveur.
Taille de la Réponse	`1005`	Taille en octets de la réponse. Dans une attaque par le temps, cette valeur peut rester constante, car l'information

		n'est pas dans le contenu de la réponse mais dans son délai d'arrivée.
Referer	`"-"`	Champ non fourni. L'attaquant a potentiellement masqué cette information.
User-Agent	`"-"`	Champ non fourni. Un User-Agent vide ou atypique est un indicateur fort de l'utilisation d'un script automatisé plutôt que d'un navigateur standard.

3. La Charge Utile SQL Décortiquée

Le décodage de la valeur du paramètre `order` révèle la logique d'exfiltration. Après un décodage URL puis Base64, on obtient une requête SQL dont voici un extrait représentatif :




```
ASC,(select (case
field(concat(substring(bin(ascii(substring(password,1,1))),1,1),substr
ing(bin(ascii(substring(pass
word,1,1))),2,1)),concat(char(48),char(48)),concat(char(48),char(49)),
concat(char(49),char(48)), concat(char(49),char(49)))when 1 then TRUE
when 2 then sleep(2) when 3 then sleep(4) when 4 then sleep(6) end)
from membres where id=1)
```

Décomposition de la logique :

1. **`substring(password,1,1)` :** Extrait le premier caractère du champ `password`.
2. **`ascii(...)` :** Convertit ce caractère en sa valeur numérique ASCII.

3. **`bin(...)` :** Convertit la valeur ASCII en sa représentation binaire (ex: '1100001').
4. **`substring(...,1,1)` et `substring(...,2,1)` :** Extrait les deux premiers bits de cette chaîne binaire.
5. **`concat(...)` :** Concatène ces deux bits pour former une chaîne (`'00'`, `'01'`, `'10'` ou `'11'`).
6. **`case field(...) when ...` :** Compare la chaîne de bits obtenue à des valeurs prédéfinies et exécute une action correspondante :
    - Si `'00'` (`concat(char(48),char(48))`): La condition est `TRUE`, la réponse est immédiate.
    - Si `'01'` (`concat(char(48),char(49))`): Le serveur exécute `sleep(2)`.
    - Si `'10'` (`concat(char(49),char(48))`): Le serveur exécute `sleep(4)`.
    - Si `'11'` (`concat(char(49),char(49))`): Le serveur exécute `sleep(6)`.

4. Les Éléments les Plus Préoccupants

Bien que chaque ligne de log soit un indicateur de compromission, les schémas qui émergent de leur analyse conjointe sont particulièrement alarmants.

- **La Manipulation Délibérée du Temps :** La preuve la plus flagrante est la corrélation directe entre les requêtes et les délais observés. Un intervalle de 6 secondes entre une requête à `12:12:54` et la suivante à `12:13:00` n'est pas une coïncidence ; c'est la preuve de l'exécution d'une commande `sleep(6)` injectée, confirmant l'exfiltration de données bit par bit.
- **Le Caractère Systématique et Automatisé :** L'analyse séquentielle des logs montre une progression méthodique. Les requêtes ciblent d'abord le premier caractère (`substring(password,1,1)`), puis le deuxième (`substring(password,2,1)`), et ainsi de suite. Cette régularité prouve que l'attaque est menée par un script automatisé, capable d'exfiltrer de grandes quantités de données avec patience et précision.
- **Le Succès de l'Attaque :** Le code de réponse `200` (OK) sur toutes les requêtes malveillantes indique que le serveur a non seulement accepté, mais aussi traité avec succès la charge utile injectée. La vulnérabilité est donc confirmée et activement exploitée.

En conclusion, ce ne sont pas seulement les requêtes individuelles, mais bien *les schémas temporels et la progression séquentielle entre les entrées* qui sont les plus préoccupants. Ils détaillent comment l'attaquant exfiltre activement des données sensibles de manière furtive et automatisée.

5. Impact et Actions Correctives Recommandées

Impact de l'attaque :

- **Exfiltration de données sensibles :** L'attaquant peut extraire le contenu complet de la base de données, y compris les mots de passe des utilisateurs et toute autre information stockée dans la table `membres`.
- **Compromission de comptes :** Les mots de passe volés peuvent être utilisés pour accéder à d'autres systèmes ou pour l'usurpation d'identité.

Actions correctives suggérées :

1. **Investigation :** Corréler l'adresse IP source (`192.168.1.23`) avec les journaux DHCP, les logs d'authentification et les points de connexion réseau pour localiser la machine compromise.
2. **Confinement Immédiat :** Bloquer ou limiter le débit (`rate-limiting`) des requêtes provenant de l'IP source. Masquer temporairement la page vulnérable en attendant un correctif.
3. **Remédiation Technique :**
    - **Appliquer des requêtes paramétrées (`prepared statements`) :** C'est la défense la plus efficace contre les injections SQL, car elle sépare les instructions SQL des données.
    - **Principe du moindre privilège :** Utiliser des comptes de base de données avec des permissions restreintes (lecture seule si possible) pour l'application web.
    - **Améliorer la surveillance :** Configurer les logs pour inclure les millisecondes afin de mieux détecter les anomalies de temps de réponse. Mettre en place des règles IDS/IPS pour détecter les schémas suspects incluant des fonctions comme `sleep()`.

6. Résolution via l'Automatisation de l'Analyse

Pour reconstituer efficacement le mot de passe volé (le "flag"), une approche automatisée est nécessaire. Un script Python peut être développé pour effectuer les tâches suivantes :

1. **Lit le Fichier Journal :** Le script parcourt le fichier `ch13.txt` ligne par ligne.
2. **Extrait les Horodatages :** Il isole la partie "secondes" de chaque horodatage.
3. **Calcule les Deltas de Temps :** Il calcule la différence de temps entre chaque requête consécutive. Le script gère les passages de minute (par exemple, de 58s à 02s, le delta est de 4s).
4. **Convertit les Délais en Bits :** Sur la base des délais calculés (0, 2, 4, 6), le script les traduit en paires de bits (`00`, `01`, `10`, `11`). L'attaque utilise quatre requêtes pour extraire un caractère (trois pour 2 bits chacune, et une pour le dernier bit).
5. **Regroupe les Bits :** Le script regroupe les bits par paquets de 7 pour former un code ASCII.
6. **Convertit l'ASCII en Caractères :** Enfin, il convertit chaque code ASCII en son caractère correspondant pour reconstituer le mot de passe exfiltré.