Attacking Application Logic

Auteurs: Laurent Scherer, Nenad Rajic,
Michaël da Silva

Table des matières

Introduction

Qu’est-ce que la logique applicative/métier ?

- Toute application a une logique
- Définit par plusieurs processus qui ont chacun une logique propre
- Développer une application amène parfois à des soucis de **logic flaws**
- Exploiter des **logic flaws** de l’application

Que sont les logic flaws ?

- Logic Flaws = erreur de logique
- De simple bugs à des vulnérabilités complexes
- Pas de signature connue (presque chaque logic flaws est unique)
- Difficile à trouver et éliminer
- Sont créés à partir d’une hypothèse qui ne gère pas tous les scénarios possibles
- Out of the box

Exemples réels

1. Asking the Oracle

Fonctionnalité:

- Fonction “**RemberMe**” qui stocke dans un **cookie** des informations concernant l’utilisateur, permet de rester connecté (**chiffré avec nom, ID et données volatile (IP + timestamp)**)
- Fonction “ScreenName” qui permet de stocker le **nom de l’utilisateur** dans un cookie (**aussi chiffré**) et **l’afficher** dans un message de bienvenue

1. Asking the Oracle

Hypothèse:

- ScreenName cookie contenant des **informations moins importante** que RememberMe -> utilisation de la **même méthode de chiffrement**
- Problème: l'utilisateur peut afficher ce qu'il souhaite avec ScreenName (**modifiable**) et donc a accès à la méthode de chiffrement

Hack Steps:

- Trouver où des valeurs sont chiffrées/déchiffrées et tenter de remplacer les valeurs (créer des erreurs qui affichent les valeurs déchiffrées)
- Chercher où on peut chiffrer une donnée
- Chercher où on peut déchiffrer et afficher une donnée

Attaque:

- Placer le cookie **RememberMe** en place de **ScreenName** pour afficher son contenu :

```
Welcome, marcus | 734 | 192.168.4.282.282750184
```

- **Forger** un nouveau cookie avec **admin** en place du nom d'utilisateur et le bon ID :

```
admin | 1 | 192.168.4.282750184
```

- **Se déconnecter** de son compte et **se reconnecter** avec le **cookie forgé** dans RememberMe. Vous êtes admin !

2. Fooling a Password Change Function

Fonctionnalité:

- Fonction de **changement de mot de passe** implémenté
- Demande à l'utilisateur **son nom**, son **mot de passe existant**, son **nouveau mot de passe** et la **confirmation du nouveau mot de passe**
- Admin peut changer le mot de passe d'un utilisateur **sans spécifier l'ancien mot de passe**
- Deux cas implémentés dans **la même fonction**

La fonction:

```
String existingPassword = request.getParameter("existingPassword");
if (null == existingPassword)
{
    trace("Old password not supplied, must be an administrator");
    return true;
}
else
{
    trace("Verifying user's old password");
    ...
```

2. Fooling a Password Change Function

2. Fooling a Password Change Function

Hypothèse:

- L'interface de l'admin ne **contient pas** de champ **"old password"** -> pas de paramètre "**existingPassword**"
- La fonction devine si un utilisateur est **admin** ou non par la **présence** du paramètre **existingPassword**

Attaque:

- Remplir le form avec **l'utilisateur à attaquer**
- Avant l'envoi du POST, **supprimer** le paramètre **existingPassword**

Hack Steps:

- Tester la suppression des paramètres envoyés dans la requête (cookies, headers, URL)
- Tester un par un la suppression de paramètres dans la requête

3. Proceeding to Checkout

Fonctionnalité:

1. Parcourir le catalogue et ajouter des produits au panier.
2. Aller dans le panier et terminer la commande.
3. Entrer les infos de paiement.
4. Entrer les infos de livraison.

3. Proceeding to Checkout

Hypothèse:

- Étapes effectuées dans l'ordre
- Un utilisateur à la dernière étape a renseigné les infos de paiement

Attaque:

- **Forced Browsing**
- Aller directement de l'ajout au panier à confirmer la commande/les infos de livraison
- Produit obtenu sans payer

Hack Steps:

- Passer d'un stage de la commande à l'autre, faire une étape plusieures fois, dans le désordre, etc.
- Les "stages" peuvent utiliser différentes URL ou valeurs de paramètres
- Deviner les hypothèses des devs et les contourner
- Noter tout message intéressant

4. Breaking the Bank

Fonctionnalité:

- L’app permet aux clients d’une banque de s’enregistrer pour gérer leur compte en ligne
- Collecte nom, adresse, date de naissance
- Pas de PIN ou autre secret
- Envoie la requête au système back-end
- Envoi un email contenant une marche à suivre

4. Breaking the Bank

Infos clients stockées dans cet objet :

```
class CCustomer
{
    String firstName;
    String lastName;
    CDoB dob;
    CAddress homeAddress;
    long custNumber;
    ...
```

4. Breaking the Bank

Hypothèse:

- 3 niveaux de protection :
  - Données personnelles
  - OTP envoyé par email
  - Client doit téléphoner pour s'authentifier

Attaque:

- Même ***Data object*** utilisé pour banking + création de compte
- Détails du compte générés depuis l'ID du client

Hack Steps:

- Connexion avec des identifiants valides
- Une fois authentifié, on utilise la fonction pour s'enregistrer avec les infos d'un autre client
  - L'application écrit par dessus l'objet CCustomer avec ces données
- Retour à la fonction principale de l'app ce qui nous donne maintenant accès au compte de ce client

4. Breaking the Bank

Faille Fondamentale :

- Le même objet en DB peut être écrit de deux façons
- L'app principale de la banque permet de modifier les données après authentification
  - Les créateurs de l'app pensent le client connu
- La fonction pour s'enregistrer permet d'écrire (dans la DB) sans s'authentifier
  - Car "les clients sont connus"

5. Beating a Business Limit

Fonctionnalité:

- Le personnel financier peut transférer des fonds
- L’application empêche les transferts de plus de $10k
- Ces gros transferts nécessitent l’accord d’un Senior Manager

Le code :

```
bool CAuthCheck::RequiresApproval(int amount)
{
    if (amount <= m_apprThreshold)
        return false;
    else return true;
}
```

Toute transaction trop importante demandera qu'un manager donne son accord.

5. Beating a Business Limit

5. Beating a Business Limit

Attaque:

- Transfert d’un nombre négatif
  - tel que -$100k
- L’application ne demande pas d’accord car ≤ $10k
- Le transfert se fait dans la direction inverse

6. Escaping from Escaping

Fonctionnalité:

- Trouvé dans l'interface web d'un NIDS
- Des entrées sous le contrôle utilisateur sont envoyées à un shell
  - Le risque était compris par les devs
  - Ajout de \ (backslash) devant ces chars :
    `; | & < > ' \s \n\r`

6. Escaping from Escaping

Attaque :

- Oublie de `\` par les devs
- Envoie de **`foo\;ls`**
- Converti par l'app en **`foo\\;ls`**
- Notre `;` fonctionne

Comment se prémunir des logical flaws ?

Bonnes pratiques

- Documenter clairement le design de l’application
- Code source commenté clairement
  - Utilité et fonctionnement de chaque composant
  - Hypothèse d’utilisation / de fonctionnement pour ce qui est hors de contrôle (input utilisateur)
  - Référence du code client qui utilise le composant
- Revue de la sécurité de l’application
  - Scénarios qui violent les hypothèses du design
  - Conditions qui sont sous le contrôle de l’utilisateur
  - Gestion des comportements utilisateurs inhabituels
  - Effet de bord des dépendances/inter opérations dans le code

Conclusion

Imaginez que votre projet de plusieures semaines est dû dans les prochaines 48h, vous n’avez rien fait ; mais votre seul critère étant la fonctionnalité, vous savez que vous pourrez terminer à temps.

Imaginez encore devoir ajouter une seule fonction à une code base qui vous est inconnue.

Ou simplement devoir interagir avec des APIs mal ou peu documentées.

Attaquer la logique applicative

- Procédure de test systématique
- une réflexion originale (pensée latérale)

Quelques tests clés

- Enlever des paramètres des requêtes
- “Forced browsing”
- Envoyer des paramètres à divers endroits (de l’application)

Challenge principal

- Se mettre à la place du développeur
  - Postulats effectués
  - Raccourcis pris
  - Erreurs commises

Questions

Rajic Nenad
Scherer Laurent
da Silva Michaël