|  |
| --- |
| Revue de code 17/09/2024 - |
| Vulnerability Report |
| Thursday, October 17, 2024 |
|  |
| Contact IBC :  Non: Prénom Nom  Email: prenom.nom@Corellia.be |
|  |

Table of Contents

[1. Introduction 3](#_Toc177396537)

[1.1. Contexte et objectifs de la mission 3](#_Toc177396538)

[1.2. Périmètre de la mission 3](#_Toc177396539)

[1.3. Méthodologie 3](#_Toc177396540)

[1.4. Déroulement des travaux 3](#_Toc177396541)

[1.5. Limites 4](#_Toc177396542)

[1.6. Remise en état 4](#_Toc177396543)

[2. Summary 5](#_Toc177396544)

[2.1. Executive summary 5](#_Toc177396545)

[2.1. Vulnérabilités et recommandations 6](#_Toc177396546)

[3. Constats détaillés 7](#_Toc177396547)

# Introduction

## Contexte et objectifs de la mission

IBC demande d'effectuer Audit de sécurité. Les objectifs de ces tests étaient de :

 Identifier les vulnérabilités de l'application, évaluer les risques associés, proposer des actions correctives prioritaires et formaliser ces éléments dans un rapport clair pour les équipes techniques et commerciales.

## Périmètre de la mission

IBC nous a mandatés pour effectuer des tests de sécurité sur le périmètre suivant :

* Nous auditons uniquement la partie du code et les bibliothèques fournies par le client.

## Méthodologie

-       Phase 1 : Collecte d'informations sur le langage

-       Phase 2 : Analyse des vulnérabilités possibles et identification du plan d'attaque

-       Phase 3 : revue du code et recherche des vulnérabilités

-       Phase 4 : Consolidation et restitution des résultats.

## Déroulement des travaux

Les activités de test ont été réalisées entre 10/17/2024 et 10/17/2024.

## Limites

Cette revue de code se limiteras à une analyse statique.

## Remise en état

Sans objet!

# Summary

## Executive summary

L'audit de votre application a révélé plusieurs vulnérabilités critiques qui exposent vos systèmes à des risques majeurs de sécurité. Ces failles, dues à des pratiques de développement obsolètes et à une gestion inadéquate des informations sensibles, nécessitent des actions immédiates pour éviter toute compromission de données ou interruption de service.

Les vulnérabilités comprennent des informations d'authentification intégrées au code source (V002) et des mots de passe faibles (V004), exposant les systèmes à des accès non autorisés. Des fonctions obsolètes, telles que scanf (V001) et malloc (V003), accroissent les risques de débordement de mémoire, compromettant ainsi la stabilité des applications. Des failles supplémentaires, notamment un débordement de tampon dans sprintf (V005), une vulnérabilité de Format String (V007) et un problème de type "use-after-free" (V008), facilitent l’injection de code malveillant et la corruption de mémoire. Enfin, des vulnérabilités d'injection SQL (V006) mettent en péril l'intégrité des bases de données, nécessitant des mesures correctives pour assurer la sécurité de l'application.

## Vulnérabilités et recommandations

Les vulnérabilités suivantes ont été découvertes :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| # | Vulnérabilités | Gravité | Recommandations | Priorité |
| V004 | Injection SQL (SQLI) Mot de passe Faible (audit de code) | High | L'audit de votre application a révélé plusieurs vulnérabilités critiques qui exposent vos systèmes à des risques majeurs de sécurité. Ces failles, dues à des pratiques de développement obsolètes et à une gestion inadéquate des informations sensibles, nécessitent des actions immédiates pour éviter toute compromission de données ou interruption de service.  Les vulnérabilités comprennent des informations d'authentification intégrées au code source (V002) et des mots de passe faibles (V004), exposant les systèmes à des accès non autorisés. Des fonctions obsolètes, telles que scanf (V001) et malloc (V003), accroissent les risques de débordement de mémoire, compromettant ainsi la stabilité des applications. Des failles supplémentaires, notamment un débordement de tampon dans sprintf (V005), une vulnérabilité de Format String (V007) et un problème de type "use-after-free" (V008), facilitent l’injection de code malveillant et la corruption de mémoire. Enfin, des vulnérabilités d'injection SQL (V006) mettent en péril l'intégrité des bases de données, nécessitant des mesures correctives pour assurer la sécurité de l'application. | 2 |
| V001 | Buffer Overflow dans sprintf (audit de code) | High | Filtrer et de gérer la taille des entrées pour éviter les dépassements de tampon, ainsi que d'utiliser une gestion dynamique de la mémoire. | 2 |
| V002 | Format string vulnerability (audit de code) | High | Filtrer les données utilisateurs. | 2 |
| V005 | Utilisation de la Fonction Malloc (obsolete) (audit de code) | High | Nous vous recommandons de contrôler les pointeurs et d’utiliser des fonctions sécurisées. | 2 |
| V006 | Utilisation de la Fonction scanf (obsolete) (audit de code) | High | Définir une longueur maximale pour les entrées scanf ou d'utiliser fgets pour limiter la taille du buffer, ainsi que de valider les entrées selon les formats attendus. | 3 |
| V003 | Informations d'authentification sont présentes dans le code d'une application (audit de code) | High | stocker les informations d'authentification dans des gestionnaires de secrets sécurisés et de les référencer dynamiquement dans le code, plutôt que de les inclure directement. | 3 |

# Constats détaillés

Les détails suivants présentent les vulnérabilités identifiées dans le périmètre, accompagnés d'une estimation de l'impact de l'exploitation de la vulnérabilité, de la facilité d'exploitation de la vulnérabilité et de sa criticité. Sont également référencés les risques métier associés à la vulnérabilité dans le contexte de votre environnement, ainsi que les recommandations pour remédier à la vulnérabilité et réduire les risques.

|  |  |  |
| --- | --- | --- |
| V004 | Injection SQL (SQLI) Mot de passe Faible (audit de code) | Gravité |
| High |
| Description | L'injection SQL dans un code source en C représente une faille critique de sécurité. Cette vulnérabilité survient lorsque des données utilisateur sont insérées dans des requêtes SQL sans filtrage ni validation appropriés. Par cette méthode, un attaquant peut insérer des commandes SQL malveillantes dans les champs de saisie de l’application. Cela modifie le comportement des requêtes et peut permettre l’accès non autorisé à des informations sensibles. Les conséquences peuvent inclure la modification ou la suppression de tables et, dans certains cas, le contrôle total de la base de données. Cette faille provient souvent de l’utilisation de concaténations de chaînes pour former des requêtes SQL, rendant le code vulnérable aux manipulations. En somme, l’injection SQL compromet la confidentialité, l’intégrité et la disponibilité des données de l’application. | |
| Périmètre |  | |

|  |  |  |
| --- | --- | --- |
| Recommandation | | |
| E004 | L'audit de votre application a révélé plusieurs vulnérabilités critiques qui exposent vos systèmes à des risques majeurs de sécurité. Ces failles, dues à des pratiques de développement obsolètes et à une gestion inadéquate des informations sensibles, nécessitent des actions immédiates pour éviter toute compromission de données ou interruption de service.  Les vulnérabilités comprennent des informations d'authentification intégrées au code source (V002) et des mots de passe faibles (V004), exposant les systèmes à des accès non autorisés. Des fonctions obsolètes, telles que scanf (V001) et malloc (V003), accroissent les risques de débordement de mémoire, compromettant ainsi la stabilité des applications. Des failles supplémentaires, notamment un débordement de tampon dans sprintf (V005), une vulnérabilité de Format String (V007) et un problème de type "use-after-free" (V008), facilitent l’injection de code malveillant et la corruption de mémoire. Enfin, des vulnérabilités d'injection SQL (V006) mettent en péril l'intégrité des bases de données, nécessitant des mesures correctives pour assurer la sécurité de l'application. | Priorité |
| 2 |
| Remédiation | Pour remédier aux injections SQL dans le code source en C, il est essentiel de remplacer les concaténations de chaînes par des requêtes paramétrées, en utilisant une bibliothèque prenant en charge cette fonctionnalité. Si les requêtes paramétrées ne sont pas possibles, toutes les entrées utilisateur doivent être soigneusement filtrées pour éliminer les caractères spéciaux susceptibles de compromettre la requête. Il est également recommandé de valider strictement les entrées pour garantir qu'elles respectent les formats attendus. Par ailleurs, les permissions de la base de données doivent être restreintes au minimum nécessaire, limitant les impacts en cas de vulnérabilité. Enfin, des audits et tests de sécurité réguliers permettront de vérifier que le code reste résilient face aux risques d’injection SQL. | |

**Preuves de concept:**

Fonction1

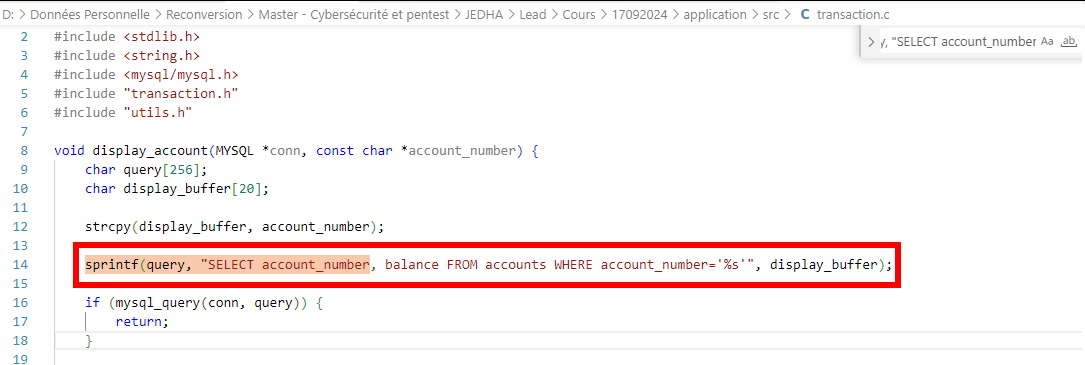


Image 1 – La fonction display\_account construit une requête en insérant directement la valeur via sprintf

Fonction2

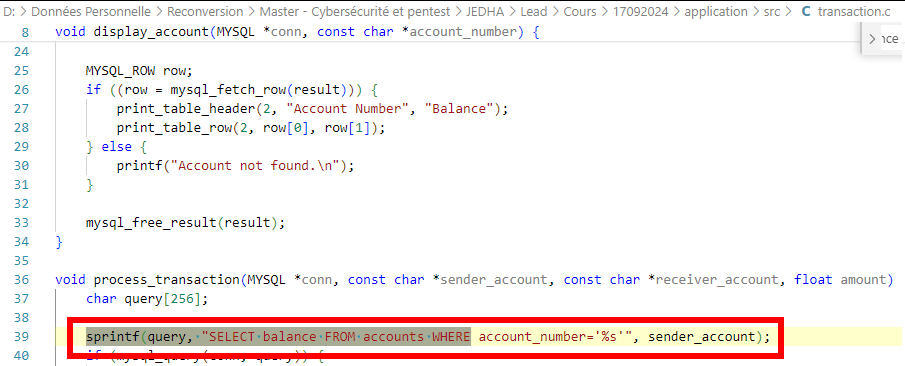


Image 1 – La fonction process\_transaction construit plusieurs requêtes SQL en insérant directement les valeurs des paramètres sender\_account et receiver\_account :

Fonction3

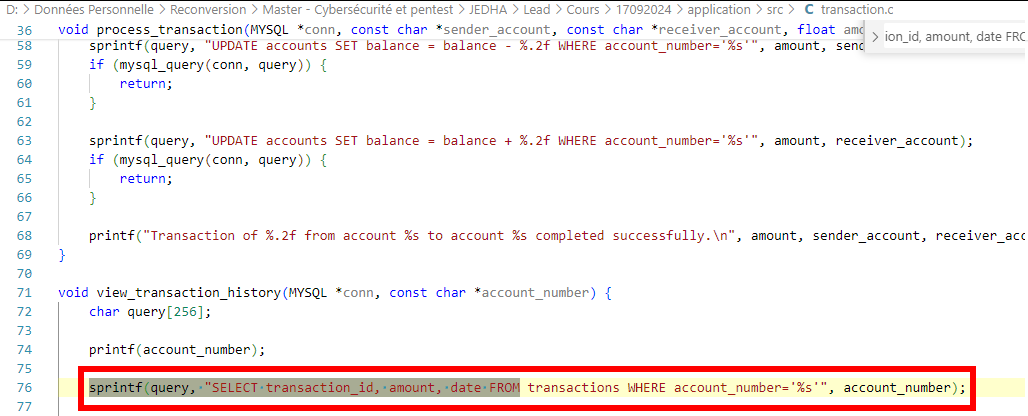


Image 1 – La fonction view\_transaction\_history insère directement le paramètre account\_number dans la requête SQL :

Fonction4

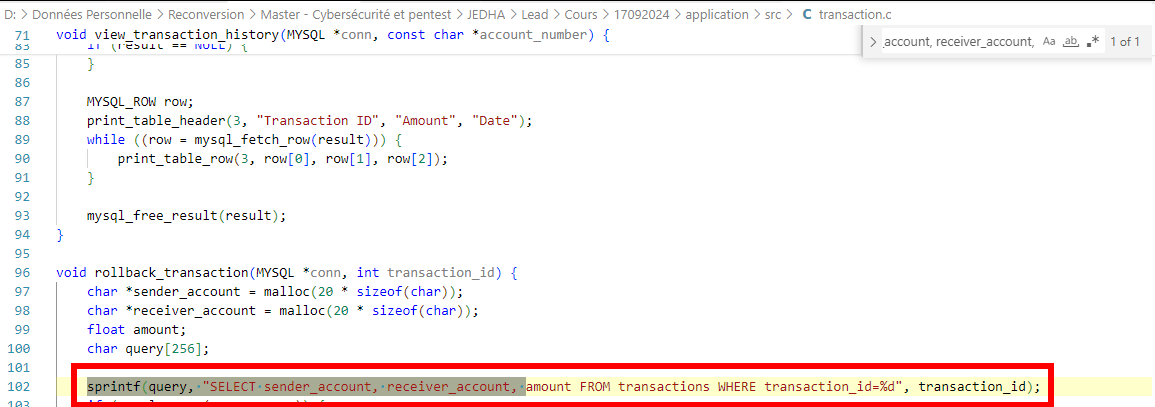


Image 1 – La fonction rollback\_transaction utilise des paramètres directement dans les requêtes SQL sans échappement, y compris le transaction\_id :

|  |  |  |
| --- | --- | --- |
| V001 | Buffer Overflow dans sprintf (audit de code) | Gravité |
| High |
| Description | La vulnérabilité de Buffer Overflow dans sprintf en C survient lorsque cette fonction écrit plus de données qu’un buffer ne peut en contenir. Cela se produit lorsque sprintf est utilisé sans limite de longueur, ce qui permet aux données de déborder dans la mémoire adjacente. Un attaquant peut exploiter cette faille pour injecter du code malveillant ou altérer le comportement de l'application, avec des conséquences graves comme des crashs ou un accès non autorisé. L'absence de contrôle sur la taille des données est la cause principale de cette vulnérabilité. Pour la corriger, il est recommandé d’utiliser snprintf, qui impose une limite au nombre de caractères écrits dans le buffer. | |
| Périmètre |  | |

|  |  |  |
| --- | --- | --- |
| Recommandation | | |
| E001 | Filtrer et de gérer la taille des entrées pour éviter les dépassements de tampon, ainsi que d'utiliser une gestion dynamique de la mémoire. | Priorité |
| 2 |
| Remédiation | Pour remédier au risque de Buffer Overflow dans sprintf, il est recommandé de remplacer cette fonction par snprintf, qui impose une limite explicite sur le nombre de caractères écrits dans le buffer, empêchant ainsi tout débordement. En spécifiant la taille maximale du buffer, snprintf assure que les données excédentaires sont coupées, ce qui réduit le risque de corruption de la mémoire adjacente. Par ailleurs, il est crucial de toujours valider et contrôler la taille des entrées utilisateur avant de les utiliser dans des fonctions de formatage. Une gestion stricte de la mémoire, comme l’allocation dynamique pour les buffers en fonction des besoins, peut également réduire les risques. Enfin, des tests réguliers et une revue de code permettront de détecter et de corriger rapidement tout usage potentiellement risqué de fonctions de formatage dans le code. correctifs. | |

**Preuves de concept:**

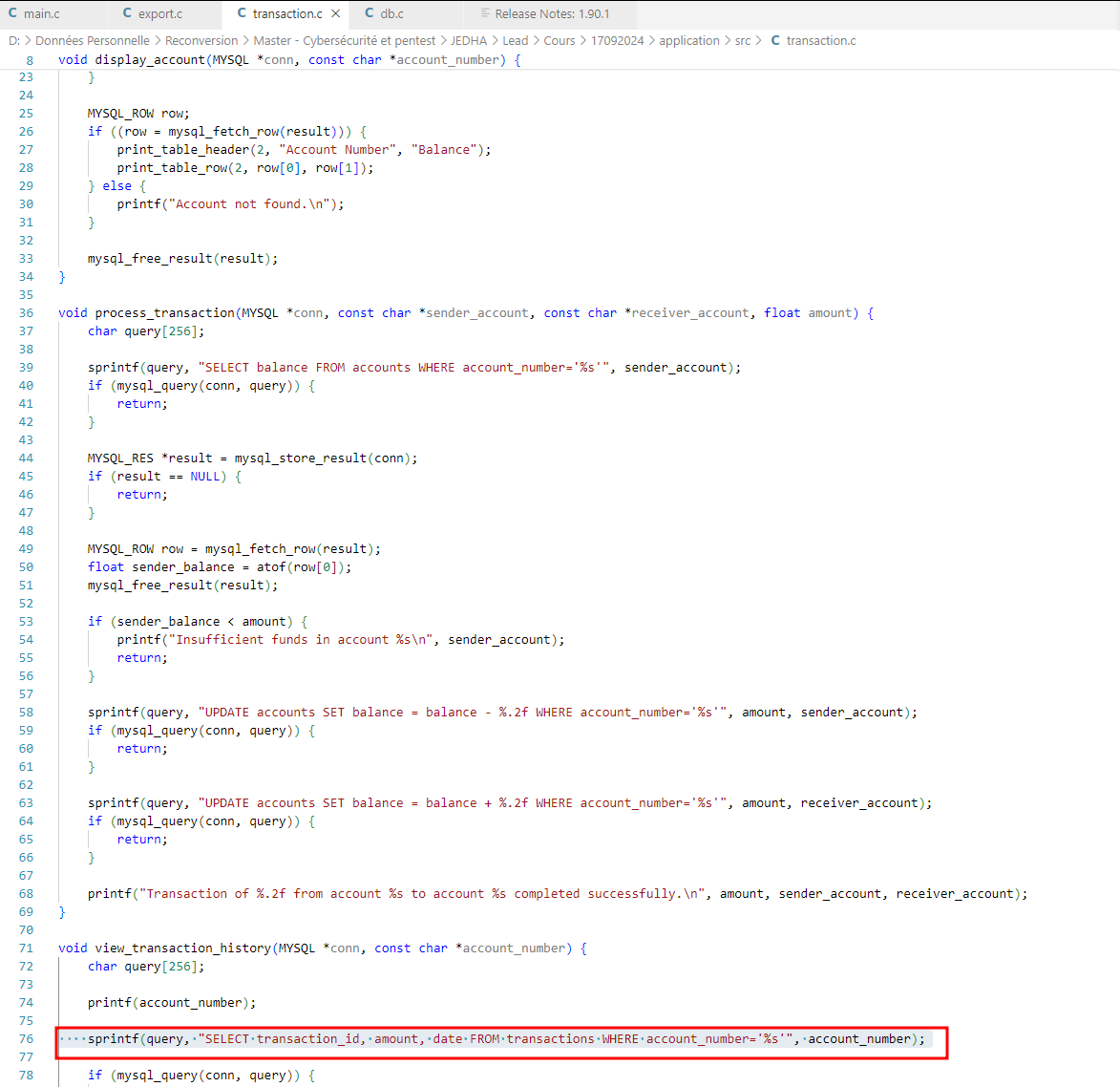


Image 1 – Buffer Overflow dans sprintf

|  |  |  |
| --- | --- | --- |
| V002 | Format string vulnerability (audit de code) | Gravité |
| High |
| Description | Une **vulnérabilité de format de chaîne** survient lorsque des entrées utilisateur sont incorrectement utilisées dans des fonctions de formatage de chaîne (comme printf en C), permettant à un attaquant de manipuler la mémoire du programme. Cela peut conduire à la lecture de données sensibles, l'exécution de code arbitraire, ou le crash du programme. | |
| Périmètre |  | |

|  |  |  |
| --- | --- | --- |
| Recommandation | | |
| E002 | Filtrer les données utilisateurs. | Priorité |
| 2 |
| Remédiation | Pour remédier à une vulnérabilité de Format String dans fprintf, assurez-vous que les chaînes de format sont toujours définies statiquement et n’incluent pas de données utilisateur non filtrées. Les entrées utilisateur doivent être passées comme paramètres sécurisés et non insérées directement dans le format. Une validation stricte des entrées est recommandée pour garantir leur sécurité. Enfin, un audit régulier du code permet de détecter et corriger toute utilisation risquée de fonctions de formatage. | |

**Preuves de concept:**

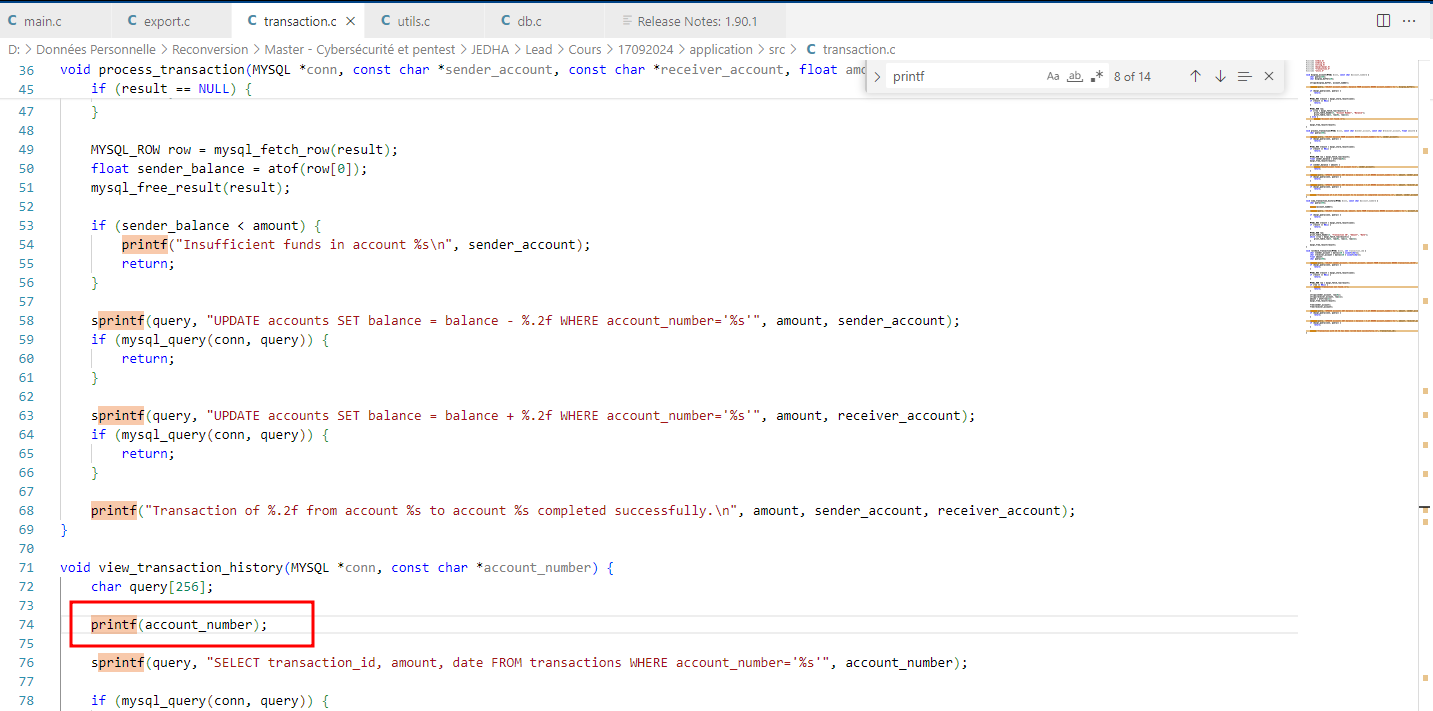


Image 1 – image.png

|  |  |  |
| --- | --- | --- |
| V005 | Utilisation de la Fonction Malloc (obsolete) (audit de code) | Gravité |
| High |
| Description | L’utilisation non sécurisée de malloc en C peut créer des vulnérabilités de sécurité en raison du risque de fuite de mémoire, de corruption ou d'accès à une mémoire non allouée. Si malloc échoue et n’est pas vérifié (retourne NULL), cela peut entraîner des erreurs critiques exploitables. Sans suivi des tailles d’allocation ou libération de la mémoire, ces risques augmentent. | |
| Périmètre |  | |

|  |  |  |
| --- | --- | --- |
| Recommandation | | |
| E005 | Nous vous recommandons de contrôler les pointeurs et d’utiliser des fonctions sécurisées. | Priorité |
| 2 |
| Remédiation | Pour remédier aux vulnérabilités de malloc, vérifiez toujours que l’allocation a réussi en contrôlant si le pointeur retourné est NULL. En cas d'échec, gérez cette situation de manière adéquate, par exemple en affichant un message d'erreur et en interrompant le processus. Si possible, utilisez calloc à la place de malloc, car calloc initialise la mémoire, réduisant ainsi le risque d’accès à des données résiduelles. Il est également crucial de libérer chaque bloc de mémoire lorsque celui-ci n’est plus nécessaire, afin d’éviter les fuites de mémoire. Une gestion rigoureuse des allocations et libérations renforce la sécurité et la stabilité du code. | |

**Preuves de concept:**

Malloc 1



Image 1 – Buffer overwritte : export.c

Malloc 2 et 3

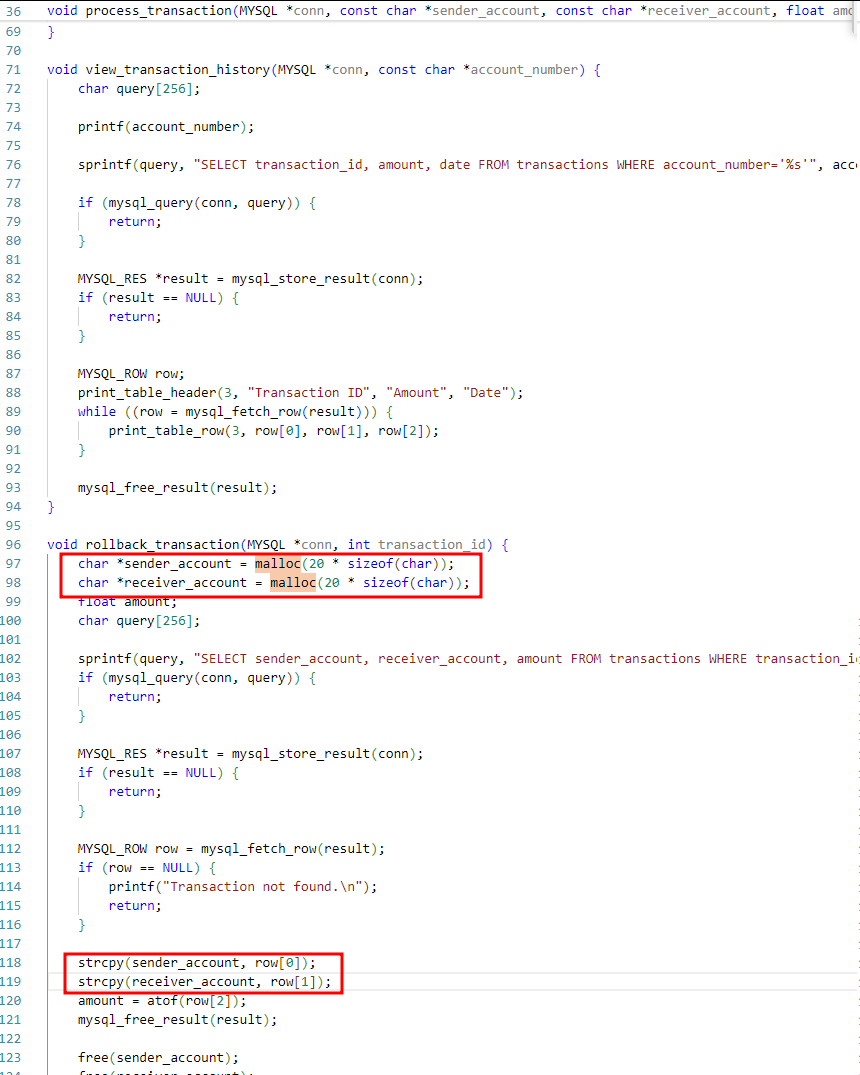


Image 1 – Buffer overwritte : Transaction.c

|  |  |  |
| --- | --- | --- |
| V006 | Utilisation de la Fonction scanf (obsolete) (audit de code) | Gravité |
| High |
| Description | La fonction scanf en C est vulnérable si elle est utilisée sans précautions, car elle ne limite pas par défaut le nombre de caractères lus, ce qui peut entraîner un débordement de tampon. Cela permet à des données trop volumineuses de corrompre la mémoire, compromettant la stabilité du programme et introduisant des risques de sécurité. Des attaques utilisant cette faille peuvent causer des crashs, des fuites d'informations ou une exécution de code non autorisée. | |
| Périmètre |  | |

|  |  |  |
| --- | --- | --- |
| Recommandation | | |
| E006 | Définir une longueur maximale pour les entrées scanf ou d'utiliser fgets pour limiter la taille du buffer, ainsi que de valider les entrées selon les formats attendus. | Priorité |
| 3 |
| Remédiation | Pour remédier aux vulnérabilités liées à l'utilisation de scanf, il est recommandé de toujours définir une longueur maximale pour les entrées dans les chaînes de format. Cela empêche les débordements de tampon en évitant de lire plus de données que le buffer ne peut en contenir. Une autre solution consiste à utiliser des alternatives plus sûres comme fgets, qui permet de spécifier explicitement la taille du buffer et limite la saisie de l’utilisateur en conséquence. De plus, validez les entrées utilisateur pour s'assurer qu'elles respectent les formats attendus avant de les utiliser dans le programme. Ces pratiques de sécurité permettent de réduire les risques d’exploitation de débordement de tampon et renforcent la résilience du code face aux erreurs de saisie. | |

**Preuves de concept:**

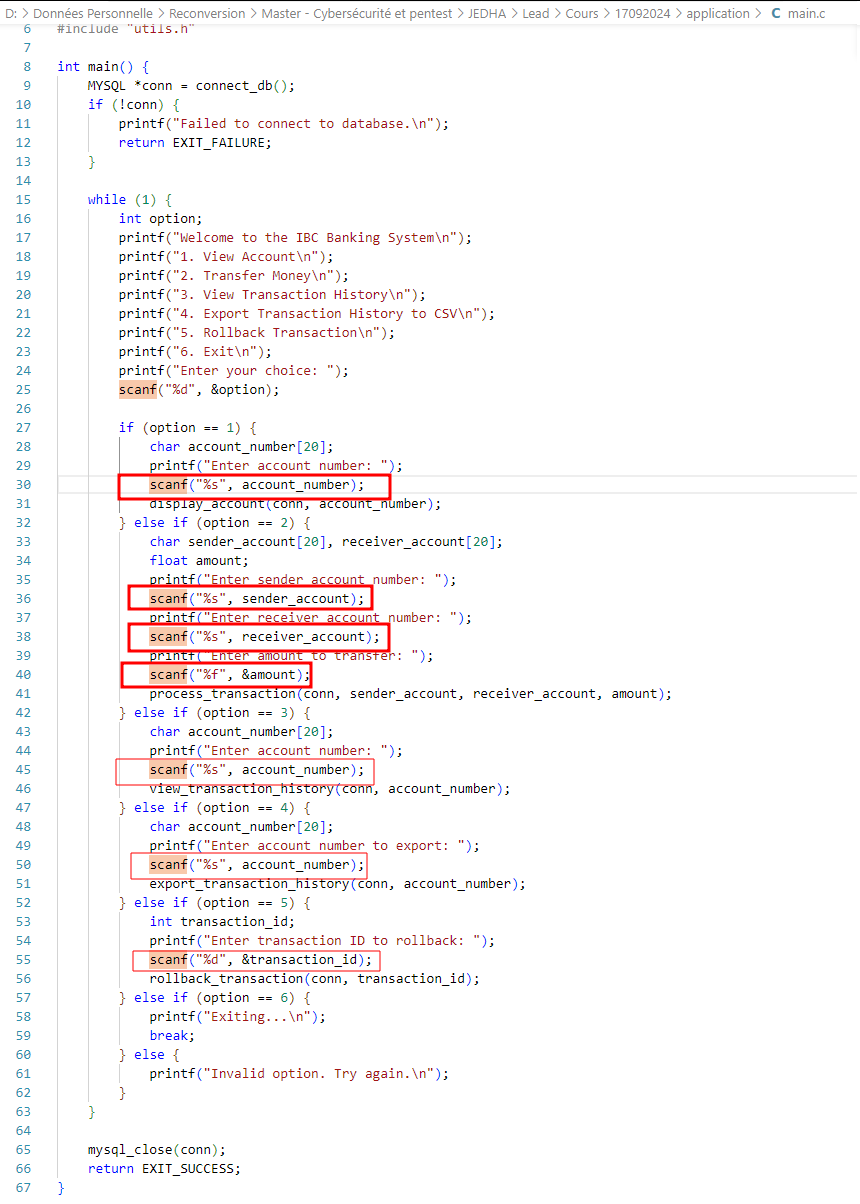


Image 1 – Utilisation de scanf dans main.c

|  |  |  |
| --- | --- | --- |
| V003 | Informations d'authentification sont présentes dans le code d'une application (audit de code) | Gravité |
| High |
| Description | Les informations d'authentification sont présentes en clair dans le code. Lors de la compilation, ces informations sont importées dans le binaire et peuvent être retrouvées en cas de décompilation. | |
| Périmètre |  | |

|  |  |  |
| --- | --- | --- |
| Recommandation | | |
| E003 | stocker les informations d'authentification dans des gestionnaires de secrets sécurisés et de les référencer dynamiquement dans le code, plutôt que de les inclure directement. | Priorité |
| 3 |
| Remédiation | Lorsque des informations d'authentification (identifiants, mots de passe, clés API) sont présentes dans le code source, elles deviennent vulnérables à une exposition non autorisée, surtout si le code est partagé ou stocké dans des dépôts accessibles. Cela peut permettre à des attaquants d’accéder directement aux systèmes ou bases de données, exposant l’application à des risques élevés de compromission. Pour atténuer ce risque, il est recommandé de stocker ces informations dans des gestionnaires de secrets sécurisés et de les référencer dynamiquement dans le code. | |

**Preuves de concept:**

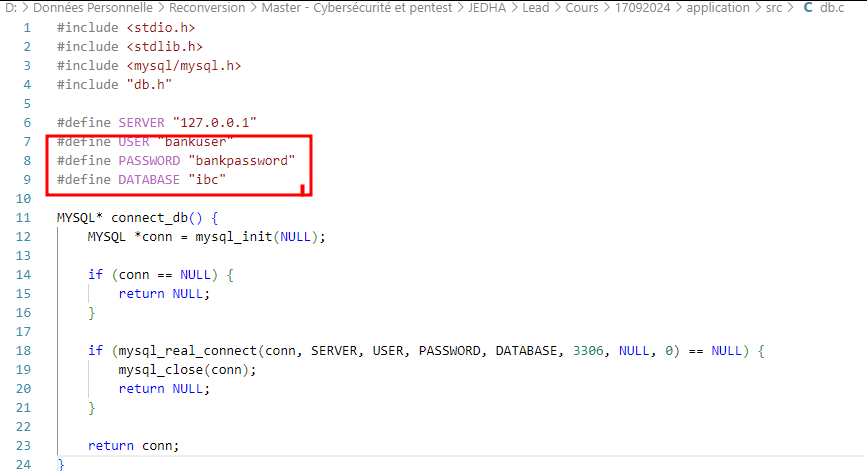


Image 1 – information d'authentification ecrite dans le code