Muhamad Hidayat Follow
Feb 19 · 2 min read

OWASP API Security (Offensive Prespective) : Insufficient Logging & Monitoring #10

Five Reasons You Need Log Monitoring | Scalyr (sentinelone.com)

Assalamualaikum Wr Wb.

Apa itu Insufficient Logging & Monitoring?

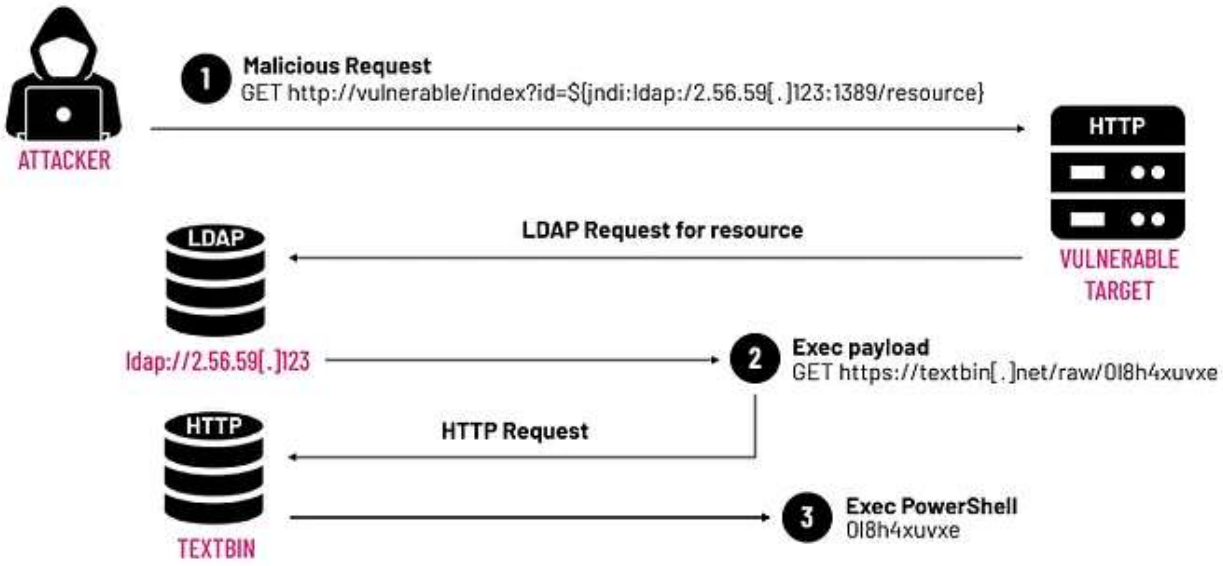
Kurangnya Logging & Monitoring system yang memungkinkan Threat Actor melakukan serangan, serta penyerangan mungkin saja terjadi tanpa terdeteksi.

Beberapa hal yang menyebabkan terjadinya Insufficient Logging & Monitoring:

- Log dan monitoring tidak memproduksi log dengan set Level log yang benar, menyebabkan banyak informasi yang tidak di sertakan pada log.
- Integritas sebuah log, yang nantinya memungkinkan Threat Actor memanfaatkan untuk melakukan serangan (Log Injection).
- Log tidak memonitoring berkelanjutan.

Berikut salah satu contoh penyerangan terkait Insufficient Logging & Monitoring yang umum terjadi pada REST API:

Log4j — Apache Log4j Security Vulnerabilities



https://blog.checkpoint.com/wp-content/uploads/2021/12/log4j-figure3.png

Vulnerability “Log4Shell” adalah celah keamanan yang ditemukan pada beberapa versi dari perpustakaan log4j, termasuk pada **JDBC Appender.** Vulnerability ini memungkinkan penyerang untuk melakukan eksekusi kode jarak jauh (remote code execution) dengan memanfaatkan sebuah fitur pada log4j yang disebut dengan **“JNDI Lookup”.**

JNDI Lookup digunakan dalam **JDBC Appender** untuk mencari data source dalam environment yang tersedia, dan ini dapat dimanfaatkan oleh penyerang untuk

memasukkan payload khusus yang akan dieksekusi oleh log4j. Dalam kasus Log4Shell, penyerang dapat memasukkan payload berupa perintah untuk mengambil alih kontrol server target.

Berikut adalah contoh konfigurasi sebelum dan setelah patching vulnerability Log4Shell pada JDBC Appender dan JNDI Lookup:

Contoh kondisi sebelum fixing:

```
// Konfigurasi JDBC Appender dengan JNDI Lookup
log4j.appender.myJDBC=org.apache.log4j.jdbc.JDBCAppender
log4j.appender.myJDBC.URL=jdbc:mysql://localhost:3306/mydatabase
log4j.appender.myJDBC.DataSource=java:comp/env/jdbc/myDataSource
log4j.appender.myJDBC.driver=com.mysql.jdbc.Driver
log4j.appender.myJDBC.user=myusername
log4j.appender.myJDBC.password=mypassword
log4j.appender.myJDBC.sql=INSERT INTO log_table (timestamp, logger, level, mess

// Konfigurasi JNDI Lookup
java.naming.factory.initial=org.apache.naming.java.javaURLContextFactory
java.naming.factory.url.pkgs=org.apache.naming
java.naming.provider.url=http://localhost:8080/
```

Pada konfigurasi di atas, perhatikan bahwa JDBC Appender dihubungkan dengan sumber daya data JNDI `java:comp/env/jdbc/myDataSource`, yang memungkinkan log4j mencari sumber daya data di environment yang tersedia. Hal ini memungkinkan penyerang memanfaatkan vulnerability Log4Shell untuk melakukan eksekusi kode jarak jauh dengan menginjeksi payload berbahaya ke dalam environment yang tersedia.

Kondisi setelah fixing:

```
log4j.appender.myJDBC.driver=com.mysql.jdbc.Driver
log4j.appender.myJDBC.user=myusername
log4j.appender.myJDBC.password=mypassword
log4j.appender.myJDBC.sql=INSERT INTO log_table (timestamp, logger, level, mess

// Konfigurasi JNDI Lookup setelah patch
java.naming.factory.initial=org.apache.naming.java.javaURLContextFactory
java.naming.factory.url.pkgs=org.apache.naming
java.naming.provider.url=http://localhost:8080/
```

Perhatikan bahwa pada konfigurasi setelah patch, JNDI Lookup menggunakan prefix `java:/` pada sumber daya data JNDI, yang membatasi pencarian hanya pada namespace `java:` . Selain itu, vulnerability Log4Shell pada library log4j juga telah diperbaiki, sehingga memastikan bahwa aplikasi aman dari serangan yang memanfaatkan celah keamanan ini.

Dalam konfigurasi JNDI, terdapat perbedaan antara penggunaan prefix `java:/` dan `java:` . Prefix `java:/` menandakan bahwa sumber daya data hanya akan dicari pada