2 ИЮНЯ 2026, МОСКВА, ЛОФТ ГОЭЛРО

БЕКОН LUNTRY '26

ЕДИНСТВЕННАЯ КОНФЕРЕНЦИЯ ПО БЕЗОПАСНОСТИ КОНТЕЙНЕРОВ И КОНТЕЙНЕРНЫХ СРЕД

Бронебойный Java образ

Максим Лебедев | Axiom JDK

БЕКОН'26

КОНФЕРЕНЦИЯ ПО БЕЗОПАСНОСТИ КОНТЕЙНЕРОВ И КОНТЕЙНЕРНЫХ СРЕД



Бронебойный Java образ

Максим Лебедев

Developer Advocate

О себе

- Более 10 лет пишу на Java, прошёл путь от монолитов до микросервисов
- Ведущий разработчик на разных проектах
- Сейчас в Axiom JDK пытаюсь разобраться как работает JVM внутри

Цель и план доклада

Разберем по шагам как сделать безопасный java контейнер

- Частые сценарии атак и эксплуатации уязвимостей
- Безопасность в JDK
- Как снизить CVE в JDK и java-приложениях
- Компромиссы

В итоге сформируем чек-лист из 12 шагов для создания доверенного и защищенного образа с минимальной JDK

Громкие уязвимости и атаки Java-стека 2024-2026

CVE-2026-0848 — CVSS 10.0

Подмена JAR-файлов

Это классический пример подмены байт-кода на этапе загрузки. Уязвимость заключается в том, что компонент динамически загружает JAR-файл, но не проверяет его подпись или целостность. Злоумышленник может подменить этот файл, и JVM загрузит вредоносный код.

CVE-2026-33701 — CVSS 9.8

Атаки через Java-агенты

Уязвимость в агенте OpenTelemetry Java позволяет атакующему, имеющему сетевой доступ к JMX/RMI порту, выполнить произвольный код через небезопасную десериализацию

CVE-2026-29000 — CVSS 9.1

Библиотека pac4j-jwt

Уязвимость в библиотеке позволяла злоумышленнику, зная только открытый RSA-ключ сервера, подделать токен JWT и получить максимальные права в обход аутентификации

Cross-Ecosystem Attacks

Shai-Hulud v2

Червь, изначально распространявшийся в экосистеме npm, проник в Maven Central через автоматический инструмент mvnpm, который конвертирует npm-пакеты в Maven-артефакты

DependencyHijacking

MavenGate

Атака нацелена на проекты, использующие заброшенные библиотеки. Проблема усугубляется тем, что большинство приложений не проверяют цифровые подписи библиотек

Typescquatting

Поддельный Jackson

Злоумышленники зарегистрировали пакет **org**.fasterxml.jackson.core:jackson-databind, который отличается от легитимного (**com**.fasterxml.jackson.core) всего одним элементом пространства имен.
Такой трюк позволил вредоносному пакету попасть в Maven Central.

Три наиболее частых сценария атак и эксплуатации уязвимостей в Java

По анализу БДУ ФСТЭК, Oracle CPU и публичных баз CVE

01

Десериализация

Наиболее повторяемый сценарий: небезопасная обработка входных объектов и XML/JSON-представлений, ведущая к отказу в обслуживании или удалённому выполнению кода.

Где встречается

Apache Commons Collections, Jackson, XStream, SnakeYAML — классические векторы. RCE через цепочки gadget-классов остаётся актуальным в 2024–2026.

02

Криптография

Ошибки в валидации подписи и криптографических процедурах опасны тем, что ломают доверенную логику без явного отказа сервиса.

Где встречается

Bouncy Castle, JWT-библиотеки, проверка X.509-сертификатов. Bypass проверок может оставаться незамеченным месяцами, в отличие от очевидных DoS.

03

Сетевые интерфейсы

JMX, RMI, API plugins и middleware-компоненты увеличивают площадь атаки там, где Java пересекается с администрированием и интеграцией.

Где встречается

Особенно опасны там, где runtime экспонируется на сеть: серверы приложений, шины интеграции, плагины API-шлюзов с динамической загрузкой кода.

Откуда берутся CVE в Java-контейнерах?



Базовый образ

Alpine, Ubuntu, Debian
Сотни CVE без SLA
Нет гарантий обновлений



JDK

Квартальные обновления
CVE в неиспользуемых
модулях JVM



Maven-зависимости

Транзитивные пакеты
Log4Shell и подобные
Сложно отследить

Почему обычная Java в контейнере это риск?

📦 Гигантский размер

Стандартный JDK весит 300–350 МБ — замедляет развёртывание, увеличивает поверхность атаки и затраты на хранение .

⚙️ JVM не знать о том, что она в контейнере

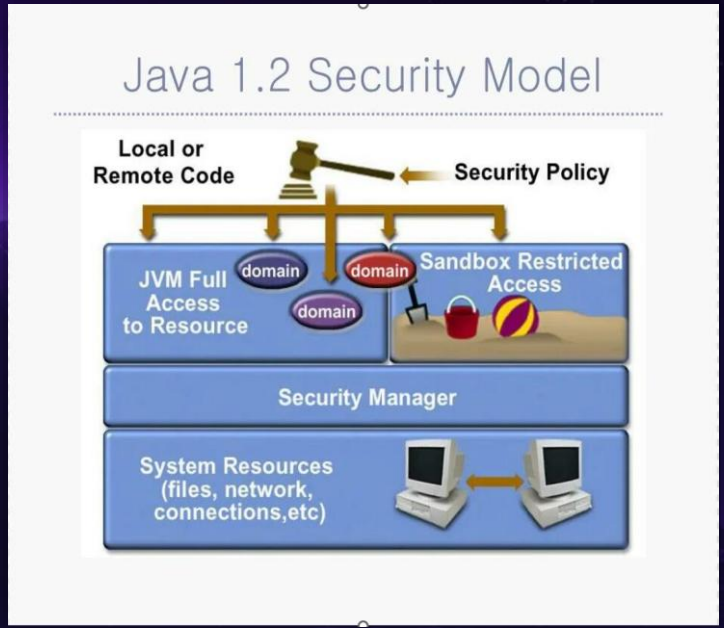
Без специальных флагов JVM может видеть ресурсы хоста → OOMKilled и троттлинг CPU.

📊 Функции безопасности

Тяжелая и непонятная настройка

Как Java защищалась раньше?

Был такой механизм — **SecurityManager**.
Он позволял задавать политики: может ли код читать файл, открывать сокет, выполнять системные вызовы.
А теперь?





Шок-контент

Из Java убрали SecurityManager

Java 17
2021

@Deprecated(forRemoval=true)
JEP 411

Java 24
2025

Удалён окончательно
JEP 486

И что же теперь — Java небезопасна?

НЕТ

Это признание реальности



В мире контейнеров за безопасность отвечают ОС и оркестратор: Linux capabilities, seccomp, namespaces



JVM больше не нужно изобретать велосипед — это делает инфраструктура



Всё решается на уровне управления capabilities контейнера

Безопасность платформы

Усиление целостности JDK: JEP 403 — Strongly Encapsulate JDK Internals

JAR Signing

Управление криптографией (keytool, JCA/JCE)

Serialization Filters – JEP 290, JEP 415

Java Platform Module System – JEP 261

https://docs.oracle.com/javase/8/docs/technotes/tools/unix/jarsigner.html
https://www.java.com/en/jre-jdk-cryptoroadmap.html
https://docs.oracle.com/en/java/javase/25/security/java-security-overview1.html#GUID-2EF91196-D468-4D0F-8FDC-DA2BEA165D10



Как снизить CVE в JDK

Ключевая идея: CVE живут в модулях JDK

Неиспользуемый модуль
→ удаляем
→ CVE не эксплуатируемы

Выбор инструмента

```
$ osv-scanner scan image --serve eclipse-temurin:25-jre

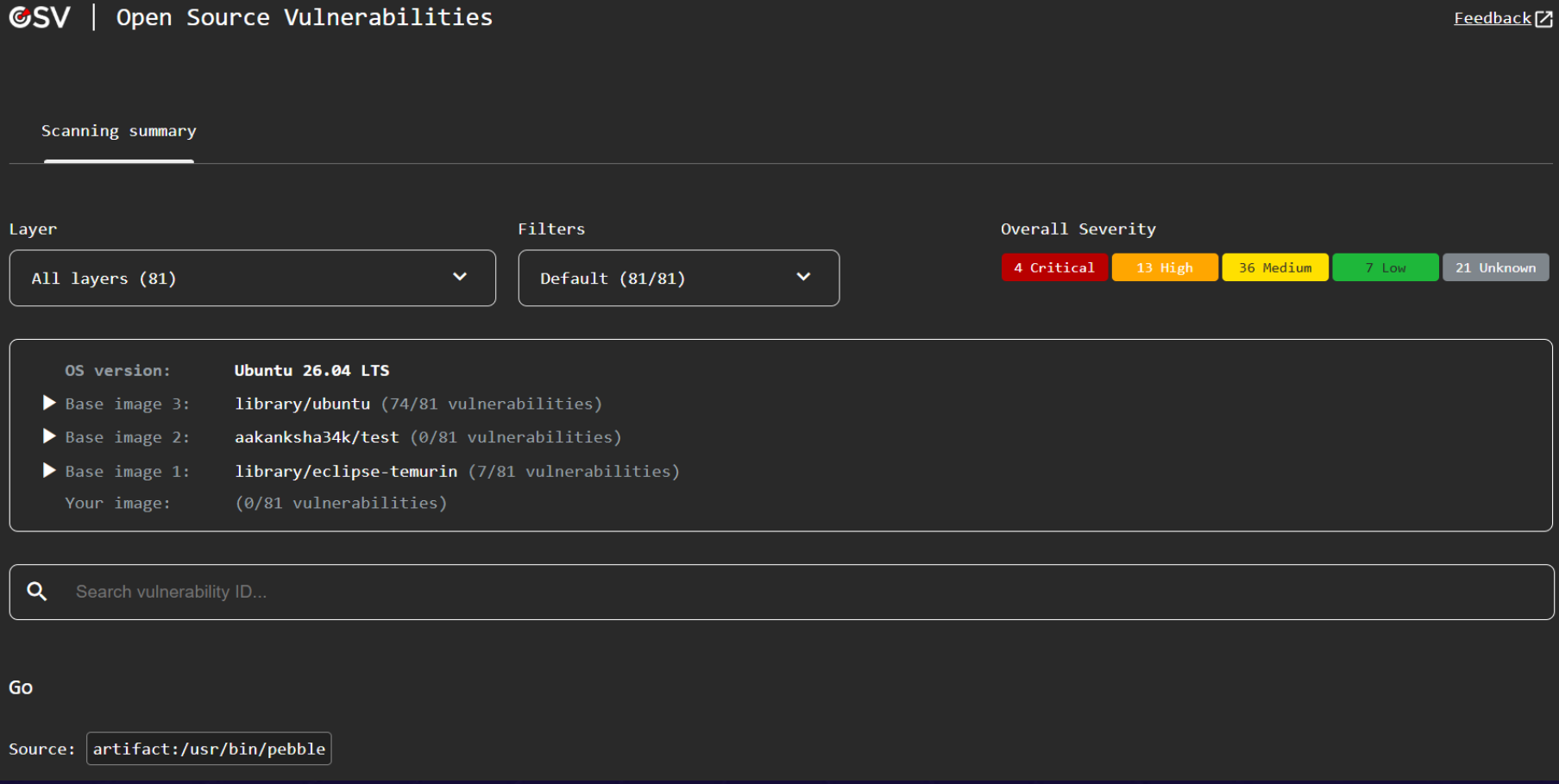
Scanning image "eclipse-temurin:25-jre"
Starting filesystem walk for root:

Container Scanning Result (Ubuntu 26.04 LTS) (Based on "library/eclipse-temurin" image):
Total 18 packages affected by 81 known vulnerabilities (4 Critical, 13 High, 36 Medium, 7 Low, 21 Unknown) from 2 ecosystems.
20 vulnerabilities can be fixed.

Serving HTML report at http://localhost:8000/
```

https://google.github.io/osv-scanner/

https://github.com/nes-examples/demo-scanners/tree/main


OSV | Open Source Vulnerabilities
Feedback
Scanning summary
Layer
All layers (81)
Filters
Default (81/81)
Overall Severity
4 Critical
13 High
36 Medium
7 Low
21 Unknown
OS version: Ubuntu 26.04 LTS
Base image 3: library/ubuntu (74/81 vulnerabilities)
Base image 2: aakanksha34k/test (0/81 vulnerabilities)
Base image 1: library/eclipse-temurin (7/81 vulnerabilities)
Your image: (0/81 vulnerabilities)
Search vulnerability ID...
Go
Source: artifact:/usr/bin/pebble

jlink как инструмент снижения CVE

```
$ jlink --add-modules java.base,…
        --compress=zip-9 --strip-debug
        --no-man-pages   --no-header-files
        --output /jre
```

Уровни сжатия (JDK 21+)

`zip-0`	~106 МБ	Без сжатия
`zip-6`	~69 МБ	По умолчанию
`zip-9`	~55 МБ	Максимум

Важные флаги

`--add-modules`	Список корневых модулей
`--strip-debug`	Удаляет отладочную информацию
`--no-man-pages`	Удаляет справочные страницы
`--no-headers-files`	Исключает заголовочные файлы

https://openjdk.org/jeps/282

Удаляем неиспользуемые модули

```
$ java --list-modules

java.base@21.0.10
java.compiler@21.0.10
java.datatransfer@21.0.10
java.desktop@21.0.10
…

$ cat module-info.java

module java.base {
   exports java.io;
   exports java.lang;

   requires java.sql;
}
```

```
FROM eclipse-temurin:25-jre AS build

RUN jlink \
   --add-modules java.base \
   --strip-debug \
   --compress zip-9 \
   --no-header-files \
   --no-man-pages \
   --output /myjre

FROM ubuntu:26.04

ENV JAVA_HOME /user/java/jdk25
ENV PATH $JAVA_HOME/bin:$PATH
COPY --from=build /myjre $JAVA_HOME

CMD ["jshell"]

$ docker build –t java_unsafe:v1 –f Dockerfile_jlink_ubuntu .
```

https://docs.oracle.com/en/java/javase/25/docs/api/

Сканируем temurin

$ osv-scanner scan image --format vertical eclipse-temurin:25-jre

Container Scanning Result (Ubuntu 26.04 LTS) (Based on "library/eclipse-temurin" image):
Total 18 packages affected by 81 known vulnerabilities (4 Critical, 13 High, 36 Medium, 7 Low, 21 Unknown) from 2 ecosystems.
20 vulnerabilities can be fixed.

CRITICAL	4
HIGH	13
MEDIUM	36
LOW	7
UNSPECIFIED	21

81 vulnerabilities

$ docker scout cves eclipse-temurin:25-jdk

size **150 MB**

Сканируем наш кастомный образ

$ osv-scanner scan image --format vertical java_unsafe:v1

Container Scanning Result (Ubuntu 26.04 LTS) (Based on "aakanksha34k/test" image):
Total 17 packages affected by 74 known vulnerabilities (3 Critical, 11 High, 32 Medium, 7 Low, 21 Unknown) from 2 ecosystems.
20 vulnerabilities can be fixed.

CRITICAL	3
HIGH	11
MEDIUM	32
LOW	7
UNSPECIFIED	21

74 vulnerabilities

$ docker scout cves java_unsafe:v1

size **64 MB**

Сравнение образов

eclipse-temurin:25-jre

150 MB

81 vulnerabilities

CRITICAL	4
HIGH	13
MEDIUM	36
LOW	7
UNSPECIFIED	21

java_unsafe:v1

64 MB

74 vulnerabilities

CRITICAL	3
HIGH	11
MEDIUM	32
LOW	7
UNSPECIFIED	21

eclipse-temurin vs java_unsafe

- 86 MB

- 7 vuln

Сравнение базовых образов

Образ	CVE	libc	Подписи	SBOM
Alpine	5-10	musl	-	-
Axiom Linux	0	musl + glibc	✓	✓
Chainguard	0	glibc	✓	✓
Ubuntu	80+	glibc	-	-

По состоянию на 2025 год

Поменяем базовый образ

Было

```
FROM eclipse-temurin:25-jre AS build

RUN jlink \
    --add-modules java.base \
    --strip-debug \
    --compress zip-9 \
    --no-header-files \
    --no-man-pages \
    --output /myjre

FROM ubuntu:26.04
```

```
$ docker build –t java_unsafe:v1 –f Dockerfile_jlink_ubuntu .
```

Стало

```
FROM eclipse-temurin:25-jre AS build

RUN jlink \
    --add-modules java.base \
    --strip-debug \
    --compress zip-9 \
    --no-header-files \
    --no-man-pages \
    --output /myjre

FROM axiom-linux-base:25-musl
```

```
$ docker build –t java_trusted:v1 –f Dockerfile_jlink_trusted .
```

Сравнение образов

eclipse-temurin:25-jre	java_trusted:v1
150 MB	**27 MB**
81 vulnerabilities	**0 vulnerabilities**
CRITICAL 4	No issues found
HIGH 13	
MEDIUM 36	
LOW 7	
UNSPECIFIED 21	

eclipse-temurin vs java_trusted

- 123 MB - 81 vuln

Сканируем axiom-linux-base

```
$ osv-scanner scan image --format vertical axiom-linux-base:25-musl

Scanning image "axiom-linux-base:25-musl"
Starting filesystem walk for root:
End status: 84 dirs visited, 464 inodes visited, 45 Extract calls, 5.002376ms elapsed, 5.00248ms wall time

No issues found

$ docker scout cves cr.int.axiomjdk.ru/axiom-linux-25/axiom-linux-base:25-musl

  Target              | axiom-linux-base:25-musl
  vulnerabilities |    0C     0H     0M     0L
  size                | 4.0 MB
  packages          | 21

## Packages and Vulnerabilities

  No vulnerable packages detected
```

Собираем контейнер с java-приложением

Состав нашего приложения

app.jar
-> BOOT-INF - зависимости
-> META-INF - мета
-> org - наш код

```
FROM maven:3.9.16-eclipse-temurin-25 AS build

WORKDIR /app

COPY pom.xml .

RUN mvn dependency:go-offline -B

COPY src src

RUN mvn package -DskipTests

FROM eclipse-temurin:25-jre

WORKDIR /app

ENV LANG=ru_RU.UTF-8
ENV LC_ALL=ru_RU.UTF-8

COPY --from=build /app/target/*.jar app.jar

ENTRYPOINT ["java", "-jar", "app.jar"]


$ docker build –t my_app:v1 –f Dockerfile_app .
```

Сканируем my_app

$ osv-scanner scan image --format vertical my_app:v1

Container Scanning Result (Ubuntu 26.04 LTS) (Based on "library/eclipse-temurin" image):
Total 20 packages affected by 88 known vulnerabilities (4 Critical, 15 High, 39 Medium, 8 Low, 22 Unknown) from 3 ecosystems.
27 vulnerabilities can be fixed.

OSV-SCANNER		DOCKER SCOUT	
CRITICAL	4	CRITICAL	12
HIGH	15	HIGH	18
MEDIUM	39	MEDIUM	15
LOW	8	LOW	3
UNSPECIFIED	22	UNSPECIFIED	1

137 vulnerabilities

$ docker scout cves my_app:v1

size **179 MB**



jdeps как инструмент для определения зависимостей

```
$ jdeps --ignore-missing-deps –q
  --recursive
  --multi-release 25
  --print-module-deps
  --class-path 'BOOT-INF/lib/*’
  app.jar > deps.info

$ cat deps.info
java.base
```

Образ с нужными модулями

```
FROM maven:3.9.16-eclipse-temurin-25 AS build
RUN mkdir /usr/src/project
COPY . /usr/src/project
WORKDIR /usr/src/project

RUN mvn package -DskipTests
RUN jdeps --ignore-missing-deps -q  \
    --recursive  \
    --multi-release 25  \
    --print-module-deps  \
    --class-path 'BOOT-INF/lib/*'  \
    target/app.jar > deps.info

RUN jlink \
    --add-modules $(cat deps.info) \
    --strip-debug \
    --compress zip-9 \
    --no-header-files \
    --no-man-pages \
    --output /myjre
```

```
FROM axiom-linux-base:25-musl

ENV JAVA_HOME /user/java/jdk25
ENV PATH $JAVA_HOME/bin:$PATH
COPY --from=build /myjre $JAVA_HOME

RUN mkdir /app

COPY --from=build /usr/src/project/target/app.jar /app/
WORKDIR /app

ENTRYPOINT ["java", "-jar", "app.jar"]

$ docker build -t my_app_custom:v1 .
```

Сканируем my_app_custom

$ osv-scanner scan image --format vertical my_app_custom:v1

Container Scanning Result (Axiom Linux 25 LTS (musl)) (Based on "zontak/agirunner-sdk-typescript" image):
Total 2 packages affected by 7 known vulnerabilities (0 Critical, 2 High, 3 Medium, 1 Low, 1 Unknown) from 1 ecosystem.
7 vulnerabilities can be fixed.

OSV-SCANNER		DOCKER SCOUT	
CRITICAL	0	CRITICAL	11
HIGH	2	HIGH	12
MEDIUM	3	MEDIUM	5
LOW	2	LOW	2
UNSPECIFIED	1	UNSPECIFIED	1

39 vulnerabilities

$ docker scout cves my_app_custom:v1

size **89 MB**

Сравнение образов

my_app:v1	my_app_custom:v1
179 MB	**89 MB**
137 vulnerabilities	**39 vulnerabilities**

my_app vs my_app_custom

- 90 MB

- 98 vuln

Native Image Kit — альтернативный путь

Преимущества

AOT-компиляция оставляет в бинарнике только реально используемые классы и методы

AOT исключает JIT из рантайма -> снижение риска эксплуатации багов компилятора

Динамическая загрузка невозможна без явной конфигурации -> защита от внедрения вредоносного кода

SBOM встраивается прямо в бинарник

Ограничения

⚠ Рефлексия (нужен tracing-агент)

⚠ Динамические прокси

⚠ Java-сериализация

⚠ Не подходит для легаси-монолитов

Настройки JVM для контейнера

-XX:+UseContainerSupport

Всё ещё актуален. Начиная с Java 10 контейнерная поддержка включена по умолчанию

-XX:ActiveProcessorCount

Тоже актуален. Используется для ограничения числа CPU, видимых JVM.

-XX:InitialRAMPercentage

Определяет начальный heap как процент RAM

$$Xms = RAM \times \frac{InitialRAMPercentage}{100}$$

-XX:MaxRAMPercentage

JVM использует 75% памяти контейнера, а не всей машины

$$Xmx = RAM \times \frac{MaxRAMPercentage}{100}$$

-XX:MinRAMPercentage

Нужен если у вас маленький объем RAM

```
small memory system:
    Xmx = RAM * MinRAMPercentage
```

Мы используем тысячи Open Source библиотек, но откуда мы знаем, что они не содержат уязвимостей или вредоносного кода?

Software Bill of Materials (SBOM)

Инвентаризация компонентов — основа проверяемой цепочки поставки

Типы SBOM:

Design, Source, Build, Analysed, Deployed, Runtime

Стандартные форматы:

- CycloneDX (CDX)
- SPDX
- Другие форматы

SBOM превращает контейнер из «чёрного ящика» в проверяемый артефакт.

Знакомство с SBOM

```
$ mvn install
...
Downloaded from central:
https://repo1.maven.org/.../log4j-api-2.25.3.jar
...
[INFO] Installing .../target/spring-petclinic-4.0.0-SNAPSHOT.jar to ...
[INFO] Installing .../target/classes/META-INF/sbom/application.cdx.json to
        ...spring-petclinic-4.0.0-SNAPSHOT-cyclonedx.json
[INFO] BUILD SUCCESS
```

SBOM автоматически генерируется плагином cyclonedx-maven-plugin

Снижаем количество уязвимостей

$ osv-scanner --sbom=sbom.cdx.json
Total 9 packages affected by 32 known vulnerabilities
(10 Critical, 14 High, 5 Medium, 2 Low)

OSV URL	CVSS	ECOSYSTEM	PACKAGE	VERSION	FIXED VERSION	SOURCE
https://osv.dev/GHSA-563x-q5rq-57qp	7.5	Maven	org.apache.tomcat.embed:tomcat-embed-core	11.0.18	11.0.20	sbom.cdx.json
https://osv.dev/GHSA-5m62-pw8w-7w9f	9.1	Maven	org.apache.tomcat.embed:tomcat-embed-core	11.0.18	11.0.22	sbom.cdx.json
https://osv.dev/GHSA-5mp6-jrq3-r938	7.5	Maven	org.apache.tomcat.embed:tomcat-embed-core	11.0.18	11.0.22	sbom.cdx.json
https://osv.dev/GHSA-69cc-cv78-qc8g	7.5	Maven	org.apache.tomcat.embed:tomcat-embed-core	11.0.18	11.0.20	sbom.cdx.json
https://osv.dev/GHSA-8mc5-53m5-3qj2	6.9	Maven	org.apache.tomcat.embed:tomcat-embed-core	11.0.18	11.0.20	sbom.cdx.json
https://osv.dev/GHSA-9m3c-qcxr-9x87	6.9	Maven	org.apache.tomcat.embed:tomcat-embed-core	11.0.18	11.0.20	sbom.cdx.json
https://osv.dev/GHSA-9m89-8frq-c98c	3.7	Maven	org.apache.tomcat.embed:tomcat-embed-core	11.0.18	11.0.22	sbom.cdx.json
https://osv.dev/GHSA-fv25-8x9x-gqjg	7.3	Maven	org.apache.tomcat.embed:tomcat-embed-core	11.0.18	11.0.22	sbom.cdx.json

$ osv-scanner --sbom= sbom.fixed.cdx.json
Total 1 package affected by 7 known vulnerabilities
(3 Critical, 3 High, 0 Medium, 1 Low)

Верификация библиотек

```
$ mvn org.simplify4u.plugins:pgpverify-maven-plugin:show \
  -Dartifact=org.apache.logging.log4j:log4j-api:2.25.2

Artifact:
        groupId:     org.apache.logging.log4j
        artifactId:  log4j-api
        type:        jar
        version:     2.25.2

PGP signature:
        version:     4
        algorithm:   SHA512 with RSA (Encrypt or Sign)
        keyId:       0x077E8893A6DCC33DD4A4D5B256E73BA9A0B592D0
        create date: Thu Sep 18 21:35:37 MSK 2025
        status:      valid

PGP key:
        version:     4
        algorithm:   RSA (Encrypt or Sign)
        bits:        4096
        fingerprint: 0x077E8893A6DCC33DD4A4D5B256E73BA9A0B592D0
        create date: Tue Jan 10 11:25:28 MSK 2023
        uids:        [ASF Logging Services RM <private@logging.apache.org>]
```

Верификация базового образа и встроенного SBOM

```
$ cosign verify --key cosign-axiom.key 25-trusted-axiom-runtime-container-pro:jdk-all-25-musl

Verification for 25-trusted-axiom-runtime-container-pro:jdk-all-25-musl @sha256:bf4691b25802...

The following checks were performed:
  - The cosign claims were validated
  - Existence of the claims in the transparency log was verified offline
  - The signatures were verified against the specified public key

[{"critical":{"identity":{...},"image":{...},"type":"..."},"optional":null}]

---

$ IMG='25-trusted-axiom-runtime-container-pro:jdk-all-25-musl'
$ cosign verify-attestation --key cosign-axiom.key --type cyclonedx $IMG
```

📦 SBOM встроен в образ как аттестация cosign

🔒 Верифицируется вместе с самим образом

📋 Возвращает полный SBOM в формате CycloneDX

https://github.com/sigstore/cosign

Харденинг образа

1	**Удалить shell (/bin/sh)** Атака невозможна без интерактивного доступа
2	**Удалить package manager (apk, apt)** Нет возможности установить вредоносный пакет
3	**Запускать от non-root** Ограничение привилегий по умолчанию
4	**Read-only rootfs** Запись в ФС заблокирована на уровне ОС
5	**Drop capabilities** Можно оставить NET_BIND_SERVICE при необходимости
6	AOT + jlink Максимальный харденинг

Сложности и компромиссы

Сложность	Как смягчить
Отладка без shell	Отдельный debug-образ с shell только для разработки
Read-only rootfs	Смонтировать tmpfs в /tmp
Сложность jlink	jdeps для анализа зависимостей + tracing-агент NIK
CVE в транзитивных зависимостях	SBOM + Dependabot + регулярный аудит зависимостей

Цепочка поставки: доверие и прозрачность



SBOM

```
mvn org.cyclonedx:cyclonedx-maven-plugin:makeBom
docker buildx build --sbom=true
```

Генерируйте в CI/CD — полный список зависимостей и компонентов образа.



Подписи (Cosign / Sigstore)

```
cosign sign myimg:version
cosign verify myimg:version
```

Подписывайте образы — убеждайтесь в целостности перед деплоем.



VEX-фильтрация ложных CVE

```
trivy --vex trivy.vex.json
osv-scanner --config=osv-scanner.toml
```

Исключайте неактуальные уязвимости, которые не применимы к вашему runtime.
VEX даёт возможность явно указать, эксплуатируема ли уязвимость в конкретном контексте



Сканер (CI/CD)

```
trivy sbom myapp.sbom.cdx.json
osv-scanner --sbom=myapp.sbom.cdx.json
```

Сканируем образы, SBOM и зависимости

Итоговый чек-лист

1	Выбрать минимальный базовый образ с подписями и SBOM
2	Сканировать базовый образ
3	Собрать кастомный JRE через jlink
4	Настроить JVM
5	Убедиться: Java ≥ 12
6	Сгенерировать SBOM
7	Сгенерировать VEX
8	Использовать multi-stage билды
9	Харденинг итогового образа
10	Подписать образ
11	Настроить сканирование CVE в CI/CD с VEX
12	Автоматически пересобирать после обновлений

БЕКОН'26

КОНФЕРЕНЦИЯ ПО БЕЗОПАСНОСТИ КОНТЕЙНЕРОВ И КОНТЕЙНЕРНЫХ СРЕД

AXIOM JDK



@axiomjdkpro

@lebmax010