1. Giới thiệu

JEP 411: *Deprecate the Security Manager for Removal* là một đề xuất quan trọng nhằm đưa cơ chế bảo mật truyền thống Security Manager (SM) vào trạng thái không khuyến nghị sử dụng để loại bỏ khỏi nền tảng Java trong các bản phát hành tương lai. Được giới thiệu từ Java 1.0, SM từng là một phần cốt lõi trong mô hình bảo mật của Java, nhưng đã trở nên lỗi thời và kém hiệu quả trước các yêu cầu bảo mật hiện đại.

2. Phân tích Nội dung

2.1. Thông tin JEP

Số JEP: 41
Tên chính thức: Deprecate the Security Manager for Removal
Loại: Feature JEP
Phiên bản được triển khai: JDK 17
Liên kết: Đồng bộ với JEP 398 (Applet API deprecate) và tiếp nối JEP 486 (loại bỏ hoàn toàn ở JDK 24).

2.2. Bối cảnh và Động lực (Motivation)

JEP 411 được đề xuất vì Security Manager là một cơ chế lỗi thời, kém hiệu quả và là gánh nặng lớn cho sự phát triển của nền tảng Java. SM được thiết kế cho kỷ nguyên Applet, nhằm bảo vệ khỏi mã độc từ xa, nhưng vai trò đó đã không còn. SM không thể giải quyết các vấn đề bảo mật quan trọng ngày nay như lỗ hổng liên quan đến *speculative-execution* hoặc các cuộc tấn công qua deserialization. Ngoài ra, SM tạo ra một gánh nặng bảo trì lớn, làm tăng đáng kể sự phức tạp và chi phí phát triển nền tảng. Hiệu suất kém do thuật toán kiểm soát truy cập phức tạp cũng là lý do chính khiến nó luôn bị vô hiệu hóa theo mặc định. Cuối cùng, mô hình lập trình khó khăn của SM đã khiến nó hiếm khi được sử dụng trong các môi trường sản xuất. JEP 411 có mục tiêu chuẩn bị cho các nhà phát triển về việc loại bỏ SM và cảnh báo người dùng nếu ứng dụng của họ đang dựa vào cơ chế này. JEP này không cung cấp một giải pháp thay thế toàn diện mà chỉ đánh giá liệu có cần các API hoặc cơ chế mới cho các trường hợp sử dụng hẹp mà SM từng đảm nhiệm (ví dụ: chặn System::exit).

2.3. Các Thay đổi Kỹ thuật

JEP 411 đã thiết lập một lộ trình loại bỏ SM một cách an toàn. Trong **JDK 17**, các API liên quan đến SM đã được đánh dấu là

@Deprecated(forRemoval=true), báo hiệu rõ ràng cho các nhà phát triển về việc chúng sẽ bị loại bỏ trong tương lai. JVM cũng được cấu hình để phát ra một cảnh báo không thể triệt tiêu khi SM được kích hoạt. Từ **JDK 18**, quá trình giảm chức năng tiếp tục với việc thay đổi giá trị mặc định của thuộc tính hệ thống java.security.manager thành **disallow**. Điều này có nghĩa là các ứng dụng cố gắng cài đặt SM động sẽ thất bại, trừ khi người dùng cấp quyền rõ ràng qua dòng lệnh. Các bản phát hành trong tương lai sẽ tiếp tục giảm chức năng của các API liên quan và loại bỏ các kiểm tra quyền khỏi các API của Java SE.

2.4. Phân tích Tác động

Đối với Nền tảng Java: JEP 411 cho phép nền tảng Java tiến lên, loại bỏ gánh nặng bảo trì lỗi thời và cải thiện hiệu suất tổng thể. Nó tạo tiền đề để củng cố bảo mật ở các cấp độ nền tảng thấp hơn, như hệ thống module (JPMS).

Đối với Lập trình viên: Đề xuất này buộc cộng đồng phải ngừng sử dụng một cơ chế đã lỗi thời. Các ứng dụng cũ dựa vào SM sẽ cần phải được cập nhật, trong khi các ứng dụng mới có thể khai thác các giải pháp bảo mật hiện đại hơn, thường là ở cấp độ bên ngoài tiến trình (ví dụ: containers) hoặc các API chuyên dụng hơn.

2.5. Hướng giải quyết và Các API Thay thế

Việc loại bỏ Security Manager (SM) không có nghĩa là nền tảng Java trở nên kém an toàn. Thay vào đó, nó thúc đẩy một cách tiếp cận bảo mật hiện đại và đa tầng, chuyển từ cơ chế kiểm soát nội bộ lỗi thời sang các giải pháp bên ngoài hiệu quả hơn. Hướng giải pháp chính được đề xuất là thay thế các chức năng của SM bằng các cơ chế chuyên biệt và mạnh mẽ hơn.

Đối với các trường hợp sử dụng SM để giám sát tài nguyên hoặc ghi nhật ký các hoạt động, giải pháp thay thế là sử dụng **JDK Flight Recorder (JFR)**. JFR cung cấp một cơ chế hiệu suất cao để thu thập dữ liệu chẩn đoán và cấu hình, cho phép các nhà phát triển và quản trị viên hệ thống có được cái nhìn sâu sắc về các hoạt động của ứng dụng, bao gồm cả quyền truy cập vào file hệ thống và mạng, mà không phải chịu chi phí hiệu suất đáng kể như SM.

Đối với các trường hợp sử dụng hẹp như chặn `System::exit` trong các môi trường như IDE hoặc framework, hướng giải pháp là sử dụng **Java Agents**. Java Agents có khả năng can thiệp vào mã bytecode của ứng dụng, cho phép

kiểm soát các lệnh gọi phương thức cụ thể một cách linh hoạt và có mục tiêu hơn nhiều. Các API chuyên dụng cho những trường hợp sử dụng này cũng có thể được phát triển trong tương lai.

Đối với việc bảo mật tương tác với mã native, JEP 411 khuyến khích chuyển sang **Foreign Function & Memory API (JEP 454)**. API này cung cấp một cách an toàn và hiệu quả để gọi các hàm ngoại và truy cập bộ nhớ ngoài JVM mà không cần đến các kiểm tra quyền phức tạp của SM, từ đó loại bỏ sự phụ thuộc vào JNI và SM.

2.6. Ví dụ minh họa (Tình huống cảnh báo và vô hiệu hóa)

1. Cảnh báo khi enable SM tại startup (JDK 17)

```
// File: LegacyApp.java

public class LegacyApp {

    public static void main(String[] args) {

        System.out.println("Ứng dụng đang chạy với Security Manager...");

        System.getProperty("user.home"); // Cần permission

    }

}
```

Chạy lệnh:

```
java -Djava.security.manager LegacyApp
```

Kết quả (JDK 17):

```
WARNING: A command line option has enabled the Security Manager

WARNING: The Security Manager is deprecated and will be removed in a future release
```

Ứng dụng đang chạy với Security Manager...

/home/user

2. Cảnh báo khi cài đặt động (JDK 17)

java// File: DynamicSM.java

```
public class DynamicSM {
    public static void main(String[] args) {
        System.setSecurityManager(new SecurityManager());
        System.out.println("SM đã được cài đặt động!");
    }
}
```

Chạy lệnh:

java DynamicSM

Kết quả:

textWARNING: A terminally deprecated method in java.lang.System has been called

WARNING: System::setSecurityManager has been called by DynamicSM

WARNING: System::setSecurityManager will be removed in a future release

SM đã được cài đặt động!

3. Lỗi khi cài đặt động (JDK 18+)

java DynamicSM

Kết quả:

textException in thread "main" java.lang.UnsupportedOperationException:

```
The Security Manager is deprecated and will be removed in a future release
    at java.lang.System.setSecurityManager(System.java:XXX)
```

Opt-in để chạy (JDK 18+):

```
java -Djava.security.manager=allow DynamicSM
```

4. Thay thế: Chặn System::exit bằng Java Agent (hiện đại)

```
// File: ExitBlockerAgent.java
public class ExitBlockerAgent {
    public static void premain(String args, Instrumentation inst) {
        inst.addTransformer((loader, name, clazz, pd, data) -> {
            if (name.equals("java/lang/System")) {
                // Chặn System.exit
                throw new SecurityException("System.exit() bị chặn bởi agent!");
            }
            return data;
        });
    }
}
```

Chạy với agent:

```
java -javaagent:ExitBlockerAgent.jar MyApp
```

→ Không cần SM, linh hoạt, hiệu năng cao.

3. Kết luận

JEP 411 đánh dấu một bước ngoặt trong chiến lược bảo mật của nền tảng Java. Bằng cách loại bỏ một cơ chế lỗi thời, nó giải phóng nền tảng khỏi các ràng buộc về hiệu suất và bảo trì, đồng thời thúc đẩy việc sử dụng các giải pháp bảo

mật hiện đại hơn. Sự thay đổi này không chỉ là một quyết định kỹ thuật mà còn phản ánh sự thay đổi triết lý trong cách tiếp cận bảo mật của hệ sinh thái Java.