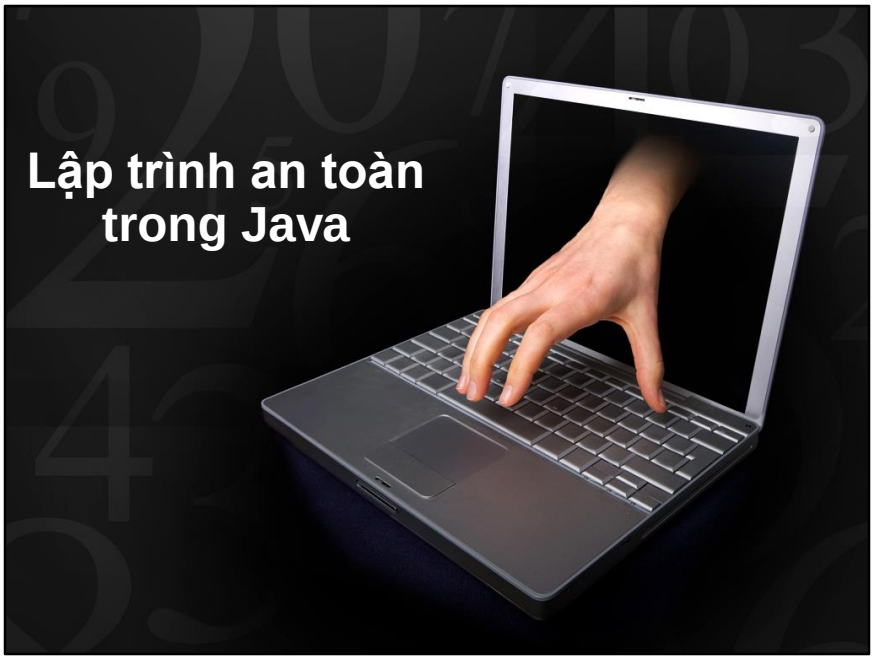

Nội dung

- Xác thực input
- Tính biến đổi

Làm gì để bảo vệ ứng dụng:
* Khi phương thức nhận dữ liệu từ bên ngoài?
* Trước những obj có thể bị thay đổi nội dung trong lúc thực thi hoặc sau khi khởi tạo obj?

6. Xác thực input

Xác thực input từ bên ngoài là một phần quan trọng của bảo mật.

6.1 – Xác thực Input

Thông qua việc truyền tham số ở phương thức, những input độc hại có thể gây ra vấn đề.

→ Phải xác thực input từ nguồn không tin cậy trước khi sử dụng.

Ví dụ:

- Tràn giá trị số nguyên → pass kiểm tra → cấp phát hết tài nguyên
- Tấn công traversal (quét) thư mục bằng cách thêm dãy "../" trước tên tập tin.

+ Input: method arguments hay external streams.
+ Nên tách biệt những đặc trưng dễ sử dụng khỏi giao diện lập trình.

Lưu ý:
Xác thực phải thực hiện ở mọi bản sao của input cần bảo vệ.

6.2 – Xác thực output của đối tượng không tin cậy

Tổng quát, nên xác thực những tham số phương thức nhưng không return giá trị.

Đối với upcall, giá trị được return phải được xác thực.

Upcall: (A call from a lower-level subsystem)
Lời gọi một phương thức của code ở mức cao hơn.

6.3 – Định nghĩa wrapper che những phương thức native

Java code được kiểm tra lúc runtime và có khả năng chống những tấn công như tràn bộ đệm.

Native code không được kiểm tra và phương thức native cũng không có khả năng chống lại tấn công.

Kiểm tra lúc runtime: type, array bound và việc sử dụng thư viện.

Java Native Interface (JNI) là một framework hỗ trợ native code để khắc phục vấn đề quản lý bộ nhớ và ràng buộc hiệu suất.
Native code có thể gọi và được gọi bởi chương trình được viết bằng ngôn ngữ khác: C/C++, assembly.
Ví dụ: gọi các phương thức gọi hay gửi tin nhắn Android.

Main.java:

```
public class Main {
    public native int intMethod(int i);
    public static void main(String[] args) {
        System.loadLibrary("Main");
        System.out.println(new Main().intMethod(2));
    }
}
```

Main.c:

```
#include <jni.h>
#include "Main.h"

JNIEXPORT jint JNICALL Java_Main_intMethod(
    JNIEnv *env, jobject obj, jint i) {
  return i * i;
}
```

Compile and run:

```
javac Main.java
javah -jni Main
gcc -shared -fpic -o libMain.so -I${JAVA_HOME}/include \
  -I${JAVA_HOME}/include/linux Main.c
java -Djava.library.path=. Main
```


cuu duong than cong . com

6.3 – Định nghĩa wrapper che những phương thức native

Biện pháp bảo vệ:

Không khai báo phương thức native ***public***.

Lợi ích:

Wrapper có thể thực hiện xác thực input trước khi gọi các phương thức native

Không khai báo phương thức native ***public,*** nên khai báo ***private*** và đưa ra các chức năng thông qua phương thức wrapper public dựa trên Java.

6.3 – Định nghĩa wrapper che những phương thức native

```
public final class NativeMethodWrapper {
        // private native method
        private native void nativeOperation(byte[] data, int offset, int len);

        // wrapper method performs checks
        public void doOperation(byte[] data, int offset, int len) {
                // copy mutable input
                data = data.clone();

                // validate input
                // Note offset+len would be subject to integer overflow.
                // For instance if offset = 1 and len = Integer.MAX_VALUE,
                //   then offset+len == Integer.MIN_VALUE which is lower
                //   than data.length.
                // Further, loops of the form
                //       for (int i=offset; i<offset+len; ++i) { ... }
                //   would not throw an exception or cause native code to crash.

                if (offset < 0 || len < 0 || offset > data.length - len) {
                        throw new IllegalArgumentException();
                }

                nativeOperation(data, offset, len);
        }
}
```


cuu duong than cong . com

7. Tính biến đổi (Mutability)

Tính biến đổi tưởng chừng vô hại, có thể gây ra nhiều vấn đề an ninh.

Mutable và Immutable?

- Mutable: trạng thái thay đổi được
- Immutable: trạng thái không thay đổi được

An toàn thông tin - UIT 11

Các đối tượng Mutable và Immutable: đều là những Class bạn tạo ra hay được cung cấp sẵn trong Java.
Phân biệt: không dựa vào tên Class, tính thừa kế hay những cấu trúc đối tượng phức tạp mà **dựa vào đặc điểm cũng như hành vi** của đối tượng đó bằng cặp phương thức **Get/Set.**

Mutable Object: khi khởi tạo 1 đối tượng, tức ta có 1 tham chiếu tới 1 thể hiện của 1 lớp, thì trạng thái của đối tượng có thể thay đổi được sau khi việc khởi tạo đối tượng thành công.
Trạng thái: thường là các trường thông tin mà đối tượng đó nắm giữ. Ví dụ: tên, tuổi của 1 đối tượng sinh viên chẳng hạn.

Immutable Object: khi khởi tạo 1 đối tượng thì trạng thái của tối tượng đó không thể thay đổi được sau khi việc khởi tạo đối tượng thành công.
Điều này có nghĩa là, bạn chỉ có thể **get** mà không thể **set.**

```
public class MutableClass {

    private String first_name;
    private String last_name;

    public MutableClass(String first_name, String last_name) {
        this.first_name = first_name;
        this.last_name = last_name;
    }

    //Mutator
    public String getFirstName() {
        return first_name;
    }

    public void setFirstName(String first_name) {
        this.first_name = first_name;
    }

    public String getLastName() {
        return last_name;
    }

    public void setLastName(String last_name) {
        this.last_name = last_name;
    }
}
```

```
public final class ImmutableClass {

    private String first_name;
    private String last_name;
    private Date dateOfBirth;

    public ImmutableClass(String first_name, String last_name, Date dob) {
        this.first_name = first_name;
        this.last_name = last_name;
        dateOfBirth = dob;
    }

    public String getFirstName() {
        return first_name;
    }

    public String getLastName() {
        return last_name;
    }

    public Date getDateOfBirth() {
        return dateOfBirth;
    }
}
```


cuu duong than cong . com

7.1 – Xây dựng Immutable Class

- Khai báo private cho tất cả các trường thuộc tính
- Không cung cấp các phương thức set
- Đảm bảo phương thức không thể bị override
- Nếu trường thuộc tính không phải là kiểu dữ liệu nguyên thủy hay Immutable → sao chép đối tượng dựa trên tham số truyền vào

Ưu điểm:

- Không thay đổi giá trị khi chia sẻ → bảo mật cho ứng dụng.
- Tạo ra những class bất biến → tránh các vấn đề thay đổi giá trị từ mã nguồn bên ngoài.

Nhược điểm: hiệu suất (cần khởi tạo nhiêu đối tượng hơn) → tốn không gian lưu trữ.

→ Cân nhắc khi xây dựng class.

+ Thiết kế tất cả các trường thuộc tính với phạm vi private.
+ Không cung cấp các phương thức set cho các trường thuộc tính
+ Đảm bảo rằng phương thức không thể bị override bằng cách: đặt final class (mạnh) hoặc đặt final method (yếu).
+ Nếu 1 trường thuộc tính không phải là kiểu dữ liệu nguyên thủy hay không phải Immutable, sao chép đối tượng dựa trên tham số truyền vào thay vì phụ thuộc vào nó.
+ Cung cấp bộ khởi tạo để xây dựng các đối tượng.

```java
public class StudentRunner {
    public static void main(String[] args) {
        Date myDOB = new Date();
        ImmutableClass student = new ImmutableClass("Manh", "Do", myDOB);
    }
}
```


cuu duong than cong . com

```java
public class StudentRunner {

    public static void main(String[] args) {
        Date myDOB = new Date();

        ImmutableClass student = new ImmutableClass("Manh", "Do", myDOB);

        System.out.println(student.getDateOfBirth());

        myDOB.setMonth(myDOB.getMonth() + 2);

        System.out.println(student.getDateOfBirth());

}
```


Khi chạy chương trình, thời gian bị thay đổi do Date là mutable class.

7.2 – Tạo ra những bản sao khi đưa ra output

- Đối với đối tượng có thể thay đổi trạng thái, khi return tham chiếu đến nó thì mã nguồn bên ngoài sẽ có thể thay đổi nó.
- Sao chép đối tượng và return bản sao.

Nếu một phương thức return một tham chiếu đến một đối tượng có thể thay đổi trạng thái nội bộ, thì mã nguồn bên ngoài cũng có thể thay đổi nó.

Trừ khi, có ý định chia sẻ trạng thái, không thì sao chép những đối tượng có thể thay đổi và return về bản sao.

7.2 – Tạo ra những bản sao khi đưa ra output

```
public class CopyOutput {
    private final java.util.Date date;
    ...
    public java.util.Date getDate() {
        return (java.util.Date)date.clone();
    }
}
```


Để tạo ra một bản sao của 1 mutable obj tin cậy, gọi bộ khởi tạo sao chép hoặc phương thức clone.

Hàm clone() có thể bị ghi đè nếu có đủ quyền.

7.3 – Sử dụng bản sao khi input là đối tượng có thể biến đổi

Mutable objects có thể thay đổi sau và ngay cả trong lúc thực thi.

Nếu phương thức không được chỉ định rõ để thao tác trực tiếp trên input có thể thay đổi, hãy tạo ra một bản sao của đầu vào đó và thao tác trên bản sao.

Input là mutable Obj hoặc subclassable.
Mutable objects có thể được thay đổi sau và ngay cả trong lúc thực thi một phương thức hoặc gọi hàm khởi tạo.
Loại như Subclassable có thể bị mở rộng, dẫn đến xử lý không chính xác, không nhất quán hoặc độc hại.

```java
public final class ImmutableClass {

    private String first_name;
    private String last_name;
    private Date dateOfBirth;

    public ImmutableClass(String first_name, String last_name, Date dob) {
        this.first_name = first_name;
        this.last_name = last_name;
        dateOfBirth = new Date(dob.getTime()); //Sao chép ra 1 đối tượng mới hơn là tham
    }

    public String getFirstName() {
        return first_name;
    }

    public String getLastName() {
        return last_name;
    }

    public Date getDateOfBirth() {
        return new Date(dateOfBirth.getTime()); //The same ^^
    }
}
```


cuu duong than cong . com

7.3 – Sử dụng bản sao khi input là đối tượng có thể biến đổi

```
public final class CopyMutableInput {
    private final Date date;
    // java.util.Date is mutable
    public CopyMutableInput(Date date) {
        // create copy
        this.date = new Date(date.getTime());
    }
}
```


Để tạo ra một bản sao của 1 mutable obj không tin cậy, gọi hàm khởi tạo sao chép hoặc phương thức tạo.

7.3 – Sử dụng bản sao khi input là đối tượng có thể biến đổi

```
public final class CopyCookie {
      // java.net.HttpCookie is mutable
      public void copyMutableInput(HttpCookie
cookie) {
            // create copy
            cookie = (HttpCookie)cookie.clone();
            // HttpCookie is final

            // perform logic (including relevant security
checks)
            // on copy
            doLogic(cookie);
      }
}
```


Trong những trường hợp hiếm, có thể an toàn khi gọi phương thức clone() trên thể hiện của chính nó.

Gọi HttpCookie.clone() an toàn vì nó không thể được ghi đè.

Date cũng cung cấp một phương thức clone public, nhưng vì phương thức này có thể bị ghi đè, nên nó có thể được tin cậy chỉ khi đối tượng Date là từ một nguồn đáng tin cậy.
Một số lớp, chẳng hạn như java.io.File, là subclassable nhưng chúng không bị thay đổi.

7.4 – Cung cấp hàm sao chép cho mutable class

- Khi thiết kế class mà giá trị có thể thay đổi, cần cung cấp phương thức để tạo ra những bản sao an toàn.
- Có thể là phương thức tạo static, constructor sao chép, hoặc bằng cách thực thi phương thức sao chép public (cho các lớp **final**).

Khi thiết kế class giá trị có thể thay đổi, cần cung cấp cách để tạo ra những bản sao an toàn của thể hiện.

Điều này cho phép các thể hiện của class đó được truyền một cách an toàn hoặc được trả về từ những phương thức trong những class khác.

Chức năng này có thể được cung cấp bởi phương thức tạo static, constructor sao chép hoặc bằng cách thực thi phương thức sao chép public (cho các lớp final).

7.5 – Đừng tin so sánh bằng khi những đối tượng tham chiếu input có khả năng bị ghi đè

- Phương thức có thể ghi đè có thể không xử lý như mong đợi.
- Object.**equals** có thể được ghi đè để return true cho các đối tượng khác nhau

identity: a variable holds the **same** instance as another variable.
equality: two *distinct* objects can be used interchangeably. they often have the same id.

7.6 – Xử lý giá trị truyền vào input đối tượng không tin cậy như output

```
private final byte[] data;
public void writeTo(OutputStream out)
                throws IOException {
        // Copy (clone) private mutable data before sending.
        out.write(data.clone());
}
```


Khi được truyền cho đối tượng không tin cậy thì áp dụng như cách đối tượng được đưa ra làm output → cần phải được sao chép.

7.7 – Xử lý output từ đối tượng không tin cậy như đầu vào

```
private final Date start;
private Date end;

public void endWith(Event event) throws IOException {
        Date end = new Date(event.getDate().getTime());
        if (end.before(start)) {
                throw new IllegalArgumentException("...");
        }
        this.end = end;
}
```


Xử lý kết quả được trả về từ đối tượng không tin cậy như khi xử lý input, cần sao chép và xác thực.

7.8 – Định nghĩa những phương thức wrapper che trạng thái nội bộ có thể được chỉnh sửa

- Nếu trạng thái nội bộ của class phải được công khai truy cập và sửa đổi, khai báo private và cho phép truy cập vào nó thông qua các phương thức wrapper public.
- Nếu trạng thái chỉ có ý định để được truy cập bởi subclass, khai báo private và cho phép truy cập thông qua các phương thức wrapper protected.
- Những phương thức Wrapper cho phép xác thực đầu vào trước khi gán giá trị mới.

cuu duong than cong . com

7.8 – Định nghĩa những phương thức wrapper che trạng thái nội bộ có thể được chỉnh sửa

```
public final class WrappedState {
        // private immutable object
        private String state;

        // wrapper method
        public String getState() {
                return state;
        }

        // wrapper method
        public void setState(final String newState) {
                this.state = requireValidation(newState);
        }

        private static String requireValidation(final String state) {
                if (...) {
                        throw new IllegalArgumentException("...");
                }
                return state;
        }
}
```


Phương thức wrapper cho phép kiểm tra trước khi gán giá trị.
Tạo ra những bản sao để tăng cường bảo vệ trong getState và setState nếu trạng thái nội bộ là có thể thay đổi.

7.9 – Tạo các trường public static final

Các trường public static non-final có thể được truy cập và sửa đổi.

Luôn khai báo những trường public static với final.

Các trường public static non-final có thể được truy cập và sửa đổi bởi các Caller. Do không có thứ gì để bảo vệ chống lại việc truy cập, hiệu chỉnh và những giá trị mới được thiết lập mà không được xác thực.

Các trường với subclassable types có thể được thiết lập đến objects với những thực thi độc hại.

7.9 – Tạo các trường public static final

```
public class Files {
    public static final String separator = "/";
    public static final String pathSeparator = ":";
}
```


cuu duong than cong . com

7.9 – Tạo các trường public static final

- Nếu dùng interface thay vì class, thì các từ public static final có thể bỏ qua để cải thiện khả năng đọc. Vì constant = public, static và final.
- Khai báo protected static cũng bị các vấn đề tương tự như public

Hằng số = final public static.
Khai báo protected static cũng bị các vấn đề tương tự như public mà còn có xu hướng yêu cầu thiết kế phức tạp.

Tóm tắt

- **Xác thực đầu vào**
  - Xác thực đầu vào chương trình bất kể từ đâu
  - Xác thực output từ những đối tượng không được tin cậy
  - Định nghĩa wrapper để tránh truy cập trực tiếp vào phương thức native

Tóm tắt

- **Tính biến đổi**
  - Xây dựng class bất biến
  - Tạo bản sao khi dùng đối tượng biến đổi làm output hoặc truyền vào input cho đối tượng không tin cậy
  - Tạo bản sao để sử dụng khi input là đối tượng có thể biến đổi
  - Cung cấp hàm sao chép khi viết mutable class
  - Phương thức equal dùng để so sánh có thể bị ghi đè và trả về kết quả sai
  - Xử lý output của đối tượng không tin cậy như input
  - Định nghĩa phương thức wrapper để che giấu khi cần thao tác với những thuộc tính riêng
  - Xem xét để đặt final cho trường có khai báo là public static

cuu duong than cong . com