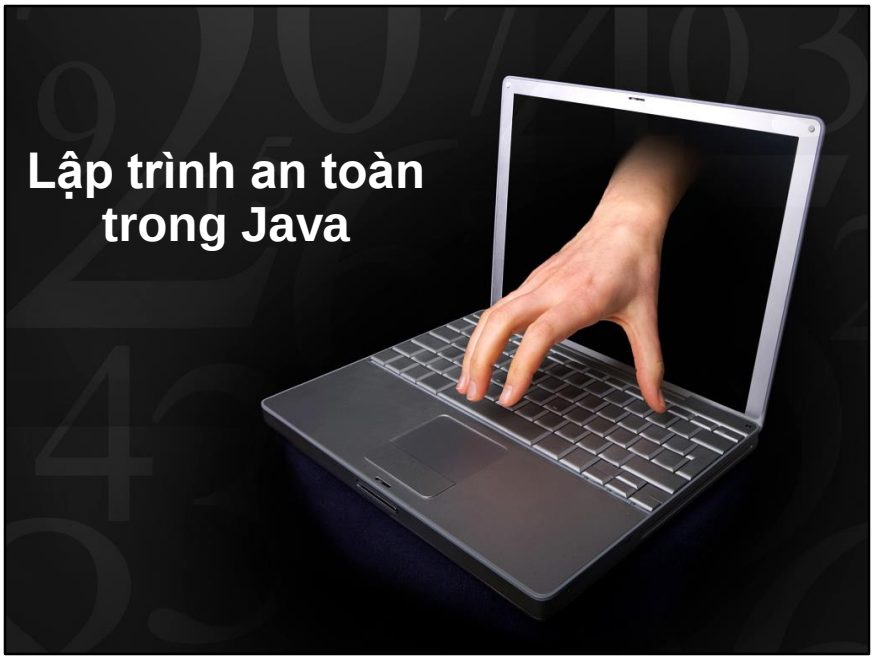


Nội dung

- Khả năng truy cập
- Khả năng mở rộng

cuu duong than cong . com

4. Injection và Inclusion

Lỗi định dạng văn bản → thay đổi quyền kiểm soát

cuu duong than cong . com

5 – Khả năng truy cập và mở rộng

Nhiệm vụ bảo mật hệ thống sẽ dễ dàng hơn khi giảm thiểu “attack surface”

Accessibility and Extensibility

Nhiệm vụ của việc bảo mật hệ thống được thực hiện dễ dàng hơn bằng cách giảm các "tấn công bề mặt" của mã nguồn.

Attack surface

- Khai thác tất cả các lỗ hỏng (“attack vectors”) để attacker có thể truy cập vào dữ liệu hoặc lấy dữ liệu từ hệ thống
- Được chia thành 3 loại:
  - Network attack surface
  - Software attack surface
  - Physical attack surface

Attack surface (tấn công bề mặt):
Khai thác tất cả các lỗ hỏng khác nhau (“attack vectors”) để một người không có quyền có thể truy cập vào dữ liệu hoặc lấy dữ liệu từ hệ thống.

Attack vector: trường input, chụp màn hình, cài phần mềm mở port, dịch vụ mới,…

Attack surface of an Application

- Tất cả đường dẫn dữ liệu/lệnh
- Mã nguồn bảo vệ các đường dẫn
  - Kết nối tài nguyên
  - Xác thực
  - Cấp quyền
  - Log hoạt động
  - Xác thực và mã hóa dữ liệu

- Tất cả đường dẫn dữ liệu/câu lệnh đi vào hoặc đi ra của ứng dụng.
- Mã nguồn bảo vệ các đường dẫn: kết nối tài nguyên, xác thực, cấp quyền, log hoạt động, xác thực và mã hóa dữ liệu.

Attack surface of an Application

- Dữ liệu có giá trị:
  - Dữ liệu bí mật, Key
  - Tài sản trí tuệ, dữ liệu thương mại quan trọng
  - Dữ liệu cá nhân, PII
- Mã nguồn bảo vệ các dữ liệu:
  - Mã hóa và checksum
  - Kiểm tra truy cập
  - Quản lý hoạt động bảo mật và toàn vẹn dữ liệu

Tất cả dữ liệu có giá trị dùng trong ứng dụng và mã nguồn bảo vệ.
PII – Personally Identifiable Information or SPI (Sensitive Personal Information): dùng để nhận diện, liên lạc, xác định vị trí một người.

Biện pháp giảm thiểu Attack Surface

- Hạn chế ứng dụng, dịch vụ không cần thiết
- Hạn chế những điểm truy cập có sẵn đối với người dùng không tin cậy
- Loại những dịch vụ được yêu cầu bởi những những người dùng ít liên quan

Chiến lược cơ bản để giảm thiểu.

- Giảm running code – code thực thi: ứng dụng, dịch vụ, những function không cần thiết.
- Hạn chế những điểm truy cập có sẵn đối với người dùng không tin cậy: FTP.
- Hạn chế đối với những mã nguồn có ít quyền.

Có thể dùng Firewall, data policies, rule, hệ thống phòng chống,…

5.1 – Hạn chế khả năng truy cập class, interface, phương thức và trường

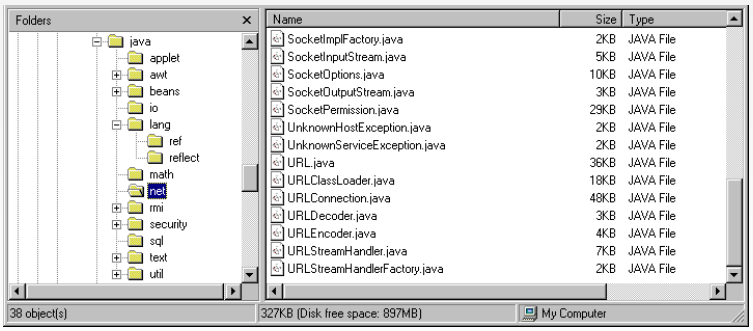
- Package chứa một nhóm class và interface có liên quan.
- Khai báo publish nếu:
  - ✓ Class
  - ✓ Interface

là một phần của API được publish.

Ngược lại, khai báo package-private (default)

Application programming interface - API:
Là một tập định nghĩa chương trình con, giao thức, tool dùng cho việc xây dựng phần mềm.
Tổng quát, một tập các method được định nghĩa một cách rõ ràng, giúp giao tiếp giữa các thành phần của phần mềm.

Package



Package giúp tổ chức file vào các thư mục khác nhau → dễ dàng duy trì, tổ chức, tăng tính tương tác với thành viên khác.

Tránh sự nhập nhằng khi có cùng tên class.

Trong Java có rất nhiều package được phân theo chức năng:

- java.util: chứa các lớp tiện ích.
- java.io: chứa các lớp input/output dữ liệu.
- java.lang: chứa các lớp thường dùng…

Đặc tả truy xuất

- Định nghĩa khả năng cho phép truy xuất đến các thành viên của class.
- Có 4 loại đặc tả:
  - **private**: chỉ được phép sử dụng nội bộ trong class
  - **public**: công khai hoàn toàn
  - **{default}**:
    - ✓ Là public đối với các lớp truy xuất cùng gói
    - ✓ Là private với các lớp truy xuất khác gói.
  - **protected**: tương tự {default} nhưng cho phép kế thừa dù lớp con và cha khác gói.
- Mức độ che dấu tăng dần theo chiều mũi tên

public → protected → {default} → private

Đặc tả truy xuất: xác định có hay không 1 class khác có thể sử dụng 1 trường hay gọi 1 hàm cụ thể.
Có 2 mức độ:
+ Top-level: class, interface. Sử dụng 2 loại đặc tả: public, default.
+ Member-level: member.

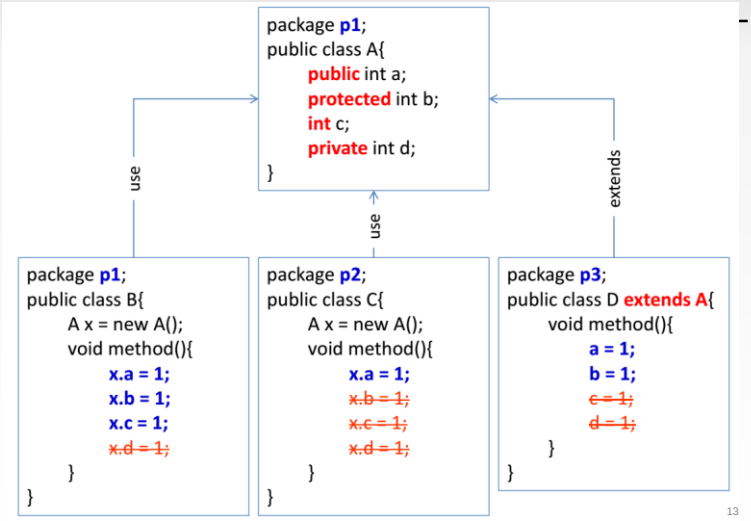
Đặc tả truy xuất ảnh hưởng theo 2 cách:
+ Khi sử dụng class từ nguồn khác, Java platform.
+ Khi viết 1 class mới, quyết định thành phần nên có.

Mức độ truy xuất đến thành viên

Modifier	Class	Package	Subclass	World
public	Y	Y	Y	Y
protected	Y	Y	Y	N
no modifier	Y	Y	N	N
private	Y	N	N	N

- Cột 1: Một class bản thân nó có thể truy xuất đến member
- Cột 2: Classes trong cùng Package
- Cột 3: Subclass của class được khai báo bên ngoài package
- Cột 4: All classes

Ví dụ: Đặc tả truy xuất



13

cuu duong than cong . com

5.1 – Hạn chế khả năng truy cập class, interface, phương thức và trường

- Tương tự, khai báo publish hoặc protected:
  - Member: nested class, method, field
  - Constructor

Ngược lại, khai báo private hoặc package-private → tránh lộ sự thực thi

Chú ý:
Thành phần của interface luôn luôn public

cuu duong than cong . com

Nested class

- Một class nằm trong một class khác

```
class OuterClass {
        ...
        class NestedClass {
                ...
        }
}
```

- Có 2 loại:
  - Static
  - Non-Static (inner classes)

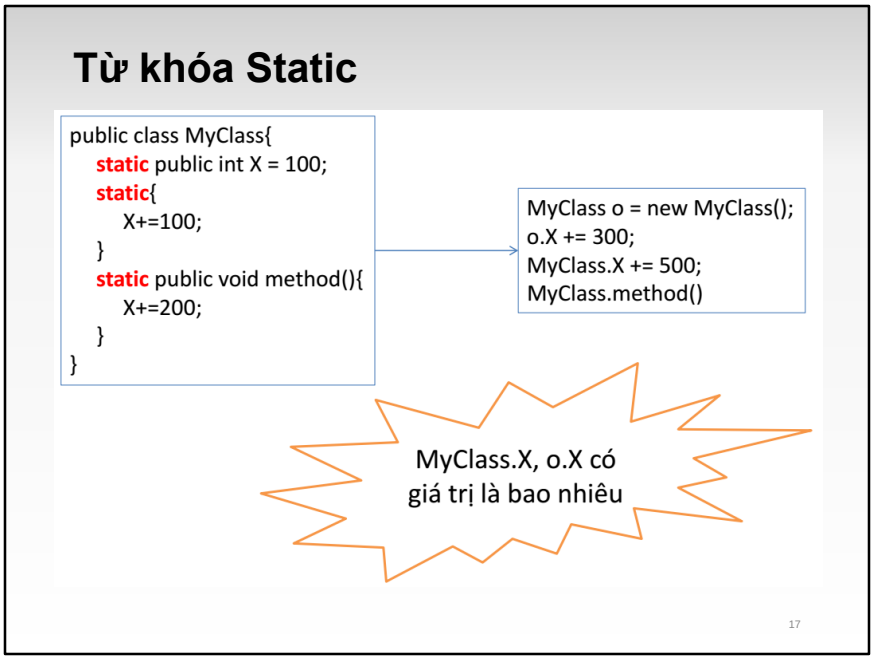
Nested class: là member của 1 class phía ngoài, cũng có 4 loại đặc tả truy xuất. Có 2 loại:

- **Non-static**: có thể truy cập đến members khác dù là khai báo private.
- **Static**: Không thể truy cập đến members khác.

Từ khóa Static

- Khối static {} sẽ **chạy trước** khi tạo đối tượng hoặc truy xuất bất kỳ thành viên static khác.
- Thành viên static của class được sử dụng **độc lập** với các đối tượng được tạo ra từ class đó.
- Có thể truy cập đến một thành viên static thông qua **tên class** mà không cần tham chiếu đến một đối tượng cụ thể.
- Trường static là dữ liệu **dùng chung** cho tất cả các đối tượng được tạo ra từ class đó.
- Trong khối và phương thức static **chỉ được truy cập đến các thành viên static** khác mà không được phép truy cập đến thành viên thông thường của class.

16



Giá trị cuối cùng là: 1200 cho cả MyClass.X và o.X.

Tại sao sử dụng Nested class

- Cách hợp lý để nhóm các class chỉ được sử dụng trong một chỗ duy nhất
- Tăng tính đóng gói
- Giúp code dễ đọc và dễ duy trì

- Nếu 1 class chỉ được dùng chỉ ở 1 class khác thì đây là cách hợp lý để nhúng nó vào trong class đó và giữ chúng cùng nhau. Nested class như "helper class" làm cho package được bố trí hợp lý hơn.
- Xét 2 class ở top-level: A và B. B cần truy xuất đến member private của A, ẩn B trong A → member của A có thể private, B có thể truy cập chúng và ẩn với bên ngoài.

5.1 – Hạn chế khả năng truy cập class, interface, phương thức và trường

- Class được nạp bởi loader khác không có quyền truy cập package-private đến class khác dù cùng tên package.
- Class cùng package được nạp bởi cùng class loader phải chia sẻ cùng một chứng chỉ số hoặc không có một chứng chỉ nào cho tất cả
- Trong JVM, class loader chịu trách nhiệm định nghĩa package

Class Loader

- Là một đối tượng chịu trách nhiệm nạp các class vào JVM
- Phân loại:
  - Bootstrap Class Loader: nạp thư viện lõi (/jre/lib)
  - Extensions Class Loader: nạp thư viện mở rộng (/jre/lib/ext)
  - System Class Loader: nạp thư viện trong java.class.path (CLASSPATH)

Java Classloader là một phần của JRE làm nhiệm vụ tự động nạp các class Java vào JVM.

Đây là 1 lớp abstract, cố gắng xác định và tạo dữ liệu để định nghĩa 1 class. Biến name thành class file từ hệ thống.

2 thư viện không thể cùng nạp 1 thư viện thứ 3 nhưng nhờ class loader thì có thể.

(tên gói, tên lớp, class loader) khi một class được nạp vào JVM.

5.2 – Hạn chế khả năng truy cập package

Container có thể giấu đi mã nguồn thực thi bằng cách thêm thuộc tính bảo mật package.access

→ giúp ngăn chặn những class không tin cậy từ những loader class khác liên kết đến

Container: code để quản lý hoặc chứa code khác (ở mức tin cậy thấp hơn).

Lưu ý: Hãy chắc rằng những package không bị truy cập trước khi thuộc tính này được thiết lập.

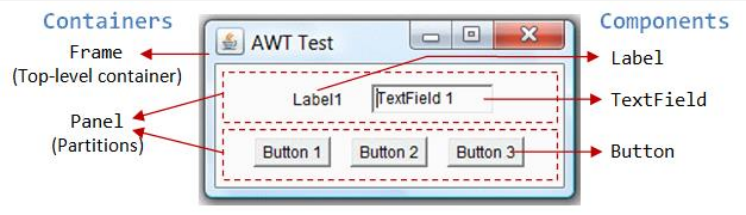
Programming GUI

- Có 2 tập API:
  - AWT Package
  - Swing Package (lớn hơn, mới hơn)
- Thường sử dụng:
  - java.awt: component, container, layout, custom graphic
  - java.awt.event: event, event listener interface, event listener apdater

AWT chứa 12 package với 370 class.
Swing lớn hơn, với 18 package, 737 classes - JDK 1.8
Thường chỉ dùng 2 package - java.awt and java.awt.event.

- java.awt package chứa các class đồ họa core AWT:
  GUI Component class: Button, TextField, và Label
  GUI Container class: Frame và Panel
  Layout manager: FlowLayout, BorderLayout và GridLayout,
  Custom graphics class: Graphic, Color and Font.
- java.awt.event package cung cấp xử lý sự kiện:
  Event class: ActionEvent, MouseEvent, KeyEvent và WindowEvent,
  Event Listener Interface: ActionListener, MouseListener, KeyListener và WindowListener
  Event Listener Adapter class: MouseAdapter, KeyAdapter và WindowAdapter.

GUI – Container




```
Panel pnl = new Panel();
Button btn = new Button("Press");
pnl.add(btn);
```

An toàn thông tin - UIT 23

Component: thành phần cơ bản của GUI.
Container: Frame (Form) và Panel.
+ Giữ thành phần khác trong 1 layout cụ thể (FlowLayout or GridLayout).
+ Có thể chứa container khác.

5.2 – Hạn chế khả năng truy cập package

```
private static final String PACKAGE_ACCESS_KEY =
"package.access";
static {
        String packageAccess = java.security.Security.getProperty(
             PACKAGE_ACCESS_KEY
        );
        java.security.Security.setProperty(
                PACKAGE_ACCESS_KEY,
                (
                        (packageAccess == null ||
                        packageAccess.trim().isEmpty()) ? "" :
                        (packageAccess + ",")
                )  + "xx.example.product.implementation."
        );
}
```

An toàn thông tin - UIT 24

Thêm thuộc tính bảo mật: package.access
Mã nguồn chỉ nên xuất hiện 1 lần trong hệ thống.
Code này giúp chặn Reflection API.

Reflection API

- Thường dùng bởi những chương trình muốn có khả năng xem xét và chỉnh sửa hình vi runtime của ứng dụng đang chạy trong JVM.
- Đặc trưng cấp cao nên chỉ được sử dụng bởi developer có kiến thức sâu về nguyên tắc nền tảng của ngôn ngữ.
- Kỹ thuật cho phép ứng dụng thực hiện những hành động mà bằng những cách thông thường khác không thể làm.

Hạn chế

- Không còn tối ưu:
- Hạn chế bảo mật: cho phép chạy mà không có Security Manager.
- Bộc lộ nội tại: cho phép truy cập vào private field, method,…

5.3 – Tách biệt mã nguồn không liên quan

- Container nên tách biệt mã nguồn không liên quan
- Vì mã nguồn không tin cậy thường được cho quyền truy cập đến mã nguồn gốc

Ví dụ, Java Plugin load những applet không liên quan vào trong những thể hiện nạp class riêng biệt và chạy chúng trong những group luồng riêng.

Applet là chương trình **java** có thể được nhúng vào các trang HTML và có thể chạy được trên các trình duyệt có bật **java.**

5.3 – Tách biệt mã nguồn không liên quan

- Mutable static và ngoại lệ thường bị vi phạm, cho phép mã nguồn bất kỳ sử dụng (trực tiếp hoặc gián tiếp).
- Mã thư viện thường có thể sử dụng an toàn bởi mã nguồn ít tin cậy. Mã thư viện yêu cầu mức tin cậy ≤ mã nguồn gọi nó để tránh vi phạm tính toàn vẹn. Container nên chắc rằng mã tin cậy ít hơn không thể thay thế mã thư viện tin cậy hơn và không truy cập được package-private

→ Sử dụng thể hiện Class Loader riêng

Mã thư viện có thể được viết cẩn thận để nó có thể sử dụng một cách an toàn bằng mã ít tin cậy hơn. Thư viện đòi hỏi một mức độ tin cậy tối thiểu bằng mã nguồn nó được sử dụng để không vi phạm sự toàn vẹn của mã nguồn khách hàng. Những container cần đảm bảo rằng mã ít tin cậy hơn là không thể thay thế mã thư viện đáng tin cậy hơn và không có quyền truy cập package-private.

5.4 – Hạn chế để lộ thể hiện của ClassLoader

Truy cập thể hiện ClassLoader cho phép những hoạt động không mong đợi:

- Truy cập đến những class mà mã ngoài thường không thể truy cập
- Truy vấn thông tin trong các URL tài nguyên (bị hạn chế)
- Những trạng thái xác thực có thể được bật và tắt.
- Những thể hiện có thể được đưa đến subclass. Subclass ClassLoader thường có những phương thức không mong muốn.

Mỗi class sẽ sử dụng classloader của riêng mình để load những class khác. Thế nên nếu ClassA.class tham chiếu đến ClassB.class thì Class B cần ở trên classpath của classloader ClassA hoặc parent của nó.

Thread context classloader là classloader hiện tại cho thread hiện tại.
1 Obj có thể được tạo ra từ 1 class trong ClassLoaderC và sau đó truyền đến 1 thread thuộc sở hữu của ClassLoaderD.
Trong trường hợp này, Obj cần sử dụng Thread.currentThread().getContextClassLoader() một cách trực tiếp nếu nó muốn load những tài nguyên mà không có sẵn trong classloader của mình.

5.5 – Hạn chế khả năng mở rộng class và phương thức

Thiết kế class và phương thức cho phép kế thừa hoặc khai báo chúng final.

Vì class hay phương thức non-final có thể bị ghi đè.

cuu duong than cong . com

Định nghĩa hằng

- Trong Java có 3 loại hằng:
  - **Lớp hằng:** lớp không cho phép kế thừa
  - **Phương thức hằng**: phương thức không cho phép ghi đè
  - **Biến hằng**: biến không cho phép thay đổi giá trị
- Sử dụng từ khóa final để định nghĩa hằng

```
final public class MyFinalClass{...}
```

```
public class MyClass{
    final public double PI = 3.14
    final public void method(){...}
}
```

29

Định nghĩa Hằng

A

```
final public class Parent{…}
```

```
public class Child extends Parent{
…
}
```

B

```
public class MyClass{
    final int PI = 3.14;
    public void method(){
        PI = 3.1475;
    }
}
```

C

```
public class Parent{
    final public void method(){…}
}
```

```
public class Child extends Parent{
    public void method(){…}
}
```

D

```
public class Parent{
    public void method(){…}
}
```

```
public class Child extends Parent{
    public void method(){…}
}
```

30

cuu duong than cong . com

5.5 – Hạn chế khả năng mở rộng class và phương thức

```
// Unsubclassable class with composed behavior.
public final class SensitiveClass {
        private final Behavior behavior;

        // Hide constructor.
        private SensitiveClass(Behavior behavior) {
                this.behavior = behavior;
        }

        // Guarded construction.
        public static SensitiveClass newSensitiveClass(
                Behavior behavior ) {
                // ... validate any arguments ...

                // ... perform security checks ...

                return new SensitiveClass(behavior);
        }
}
```


cuu duong than cong . com

5.5 – Hạn chế khả năng mở rộng class và phương thức

- Subclass độc hại ghi đè phương thức Object.finalize có thể phục hồi các đối tượng dù bộ khởi tạo đưa ra ngoại lệ.
- Từ JDK 6, ngoại lệ được đưa ra trước khi bộ xây dựng java.lang.Object thoát ra, giúp ngăn chặn các finalizer tránh bị gọi.
- Thực hiện bằng cách chèn một lời gọi phương thức như một tham số đến lời gọi bộ khởi tạo.

Finalizer:

java.lang.Object.finalize() được gọi bởi cơ chế thu gom rác trên 1 Object khi cơ chế này xác định không có tham chiếu đến đối tượng.

1 Subclass ghi đè phương thức này để quyết định cách xử lý tài nguyên hệ thống hoặc thực hiện các kiểu làm sạch khác.

5.5 – Hạn chế khả năng mở rộng class và phương thức

```
public class NonFinal {
        // sole accessible constructor
        public NonFinal() {
                this(securityManagerCheck());
        }

        private NonFinal(Void ignored) {
                // ...
        }

        private static Void securityManagerCheck() {
                SecurityManager sm = System.getSecurityManager();
                if (sm != null) {
                        sm.checkPermission(...);
                }
                return null;
        }
}
```


Security Manager:

- Class này cho phép ứng dụng thực thi chính sách bảo mật.
- Cho phép ứng dụng xác định hoạt động và có hay không nó đang cố gắng truy cập vào phạm vi bảo mật cho phép hoạt động được thực hiện, trước khi thực hiện 1 hoạt động có khả năng không an toàn hoặc nhạy cảm.
- Ứng dụng có thể cho phép hoặc không cho phép hoạt động đó.
- Chứa nhiều phương thức với tên bắt đầu bằng từ Check…
- Những phương thức này có thể được gọi bởi nhiều phương thức khác trong thư viện Java trước khi thực hiện hoạt động có tiềm năng gây hại.

checkPermission(java.security.Permission) xác định có cấp phép cho 1 yêu cầu truy cập đòi 1 quyền đặc biệt hay từ chối nó.

5.5 – Hạn chế khả năng mở rộng class và phương thức

Khi xác thực class type của một đối tượng, không so sánh thể hiện class chỉ sử dụng tên lớp (qua Class.getName)

Khi xác thực class type của 1 đối tượng bằng cách kiểm tra thể hiện java.lang.Class thuộc đối tượng đó, không so sánh thể hiện class chỉ sử dụng tên lớp (qua Class.getName), vì thể hiện được xem xét kĩ bởi cả **class name cũng như bộ nạp class.**

Dùng this.getClass().getCanonicalName() để get tên class đầy đủ.

5.6 – Hiểu cách superclass có thể ảnh hưởng đến hành vi của subclass

Subclass không có khả năng duy trì sự kiểm soát tuyệt đối trên hành vi của mình.

- Superclass thay đổi việc thực thi một phương thức kế thừa bằng cách không cho ghi đè.
- Superclass đưa ra các phương thức mới.

→ phá vỡ các giả định được thực hiện trong subclass và dẫn đến lỗ hổng bảo mật khó thấy.

Một superclass có thể ảnh hưởng đến hành vi của subclass bằng cách thay đổi việc thực thi một phương thức kế thừa không được ghi đè.
Nếu một subclass ghi đè tất cả các phương thức kế thừa, superclass vẫn có thể ảnh hưởng đến hành vi của subclass bằng cách đưa ra các phương thức mới. Những thay đổi đó trên superclass có thể vô tình phá vỡ các giả định được thực hiện trong subclass và dẫn đến lỗ hổng bảo mật khó thấy.

5.6 – Hiểu cách superclass có thể ảnh hưởng đến hành vi của subclass

Class Hierarchy	Inherited Methods
java.util.Hashtable	put(key, val) remove(key)
^ | extends |	
java.util.Properties	
^ | extends |	

java.security.Provider	put(key, val) //SecurityManager
	remove(key) // checks for these
	// methods

An toàn thông tin - UIT 36

Lớp Hashtable được cải tiến trong JDK 1.2 bằng cách thêm một phương thức mới, entrySet, hỗ trợ xóa bỏ những entry từ Hashtable.
Lớp Provider không được cập nhật để ghi đè phương thức mới này. Sự thiếu sót này cho phép người tấn công bỏ qua kiểm tra SecurityManager bắt buộc trong Provider.remove và xóa những obj của Provider bằng cách gọi phương thức Hashtable.entrySet.

Những subclass độc hại có thể thực thi java.lang.Cloneable. Thực thi giao diện này ảnh hưởng đến hành vi của subclass. Một bản sao của một đối tượng nạn nhân có thể được tạo ra. Các bản sao sẽ là một bản sao nông. Các khóa nội tại và các trường của hai đối tượng sẽ khác nhau, nhưng các đối tượng được tham chiếu sẽ như nhau. Điều này cho phép kẻ xấu làm xáo trộn trạng thái thể hiện của class bị tấn công.

Tóm tắt

- Hạn chế khả năng truy cập:
  - Top-level: class, interface
  - Member-level: property, method, nested-class
- dựa vào đặc tả truy xuất.
- Hạn chế khả năng truy cập package
- Tách mã nguồn không liên quan
- Hạn chế để lộ thể hiện của Class Loader
- Hạn chế khả năng mở rộng class và phương thức khi thiết kế
- Hiểu cách class cha ảnh hưởng tới hành vi của class con khi kế thừa

cuu duong than cong . com