6

Lab

Bảo mật
Lập trình Java

Thực hành Bảo mật web và ứng dụng

GVTH: Ung Văn Giàu

Học kỳ I – Năm học 2017-2018

Lưu hành nội bộ

A. TỔNG QUAN

1. Giới thiệu

Java là một ngôn ngữ lập trình cấp cao, hướng đối tượng, bảo mật và mạnh mẽ. Khác với phần lớn ngôn ngữ lập trình thông thường, thay vì biên dịch mã nguồn thành mã máy hoặc thông dịch mã nguồn khi chạy, Java được thiết kế để biên dịch mã nguồn thành bytecode, bytecode sau đó sẽ được thực thi. Máy ảo Java chứa môi trường thực thi được thiết kế bảo mật với nhiều tính năng như tách biệt các package chứa class, bộ xác thực bytecode giúp kiểm tra mã nguồn không hợp lệ truy cập vào đối tượng, cơ chế quản lý bảo mật giúp xác định các tài nguyên mà một class có thể truy cập,…

Mặc dù vậy, các ứng dụng Java vẫn có thể bị tấn công vì chúng có thể nhận giá trị truyền vào từ bên ngoài hay tương tác với những hệ thống con phức tạp.

2. Mục tiêu

Giúp sinh viên có được kiến thức cũng như kỹ năng xử lý dữ liệu nhạy cảm, những đặc trưng ngôn ngữ có thể bị bỏ sót tạo ra các lỗ hổng bảo mật.

3. Môi trường và cấu hình

a) Môi trường

Sử dụng Netbeans để viết mã nguồn. Bạn có thể sử dụng trên môi trường Windows hoặc Linux.

Lưu ý: chỉ cần tải phiên bản Java SE, không nhất thiết phải tải bản full.

Link tải Netbeans: https://netbeans.org/downloads/

b) Cấu hình

Mặc định Netbeans đã có sẵn Java SE Development Kit (JDK). Nếu Netbeans IDE có yêu cầu bản cập nhật JDK 8 mới nhất thì bạn vào link sau để cập nhật.

http://www.oracle.com/technetwork/java/javase/downloads/jdk-netbeans-jsp-142931.html

Tham khảo chi tiết tài liệu tham khảo [1].

B. THỰC HÀNH

1. Hạn chế vòng đời của dữ liệu nhạy cảm

Dữ liệu nhạy cảm trong bộ nhớ có thể tạo ra những lỗ hỏng cho việc tấn công. Người tấn công có thể thực thi mã độc trên cùng hệ thống với ứng dụng và có thể truy cập vào dữ liệu nếu ứng dụng:

- Sử dụng đối tượng để lưu dữ liệu nhạy cảm, nội dung không được làm sạch hoặc không được thu gom sau khi sử dụng.
- Có những trang bộ nhớ có thể được swap lên đĩa cứng khi được yêu cầu bởi hệ điều hành.
- Giữ dữ liệu nhạy cảm trong buffer (bản sao của dữ liệu trong cache hệ điều hành hay bộ nhớ).
- Để lộ dữ liệu nhạy cảm qua log, debug,…

Để hạn chế lộ dữ liệu nhạy cảm, lập trình viên phải tối thiểu vòng đời của nó.

Đoạn mã sau nhận tên đăng nhập và mật khẩu từ console và lưu mật khẩu như một đối tượng String. Điều này sẽ dẫn đến việc mật khẩu sẽ được giữ cho đến khi cơ chế thu dọn rác thu hồi lại bộ nhớ của đối tượng này.

```
class Password {
    public static void main (String args[]) throws IOException {
        Console c = System.console();
        if (c == null) {
            System.err.println("No console.");
            System.exit(1);
        }
        String username = c.readLine("Enter your user name: ");
        String password = c.readLine("Enter your password: ");
        if (!verify(username, password)) {
            throw new SecurityException("Invalid Credentials");
        }
        // ...
    }

    // Dummy verify method, always returns true
    private static final boolean verify(String username, String password) {
        return true;
    }
```

```
}
```

Yêu cầu: Sửa lại đoạn mã trên để xóa đi mật khẩu ngay sau khi sử dụng xong. Giải thích tại sao bảo mật hơn.

Gợi ý:

- **Bước 1:** dùng hàm readPassword() của Console để đọc mật khẩu.
- **Bước 2:** xóa mật khẩu bằng cách thiết lập các giá trị trong chuỗi mật khẩu về rỗng.
- **Giải thích**: dựa vào hàm và cách giải quyết vấn đề.

2. Không sử dụng phương thức clone() để sao chép những tham số không tin cậy được truyền vào

Sử dụng phương thức clone() không đúng cách có thể cho phép người tấn công khai thác các lỗ hổng, bằng cách cung cấp các tham số nhìn có vẻ bình thường nhưng sau đó các tham số này có thể trả về những giá trị không mong đợi. Do đó, những đối tượng này có thể vượt qua sự kiểm tra bảo mật và kiểm tra xác thực tính hợp lệ. Khi một class có thể được truyền như tham số của một phương thức, hãy đối xử với tham số theo cách của tham số không tin cậy và không sử dụng phương thức clone() được cung cấp bởi class. Cũng như không sử dụng phương thức clone() của những class không final để tạo ra những bản sao an toàn.

Đoạn mã sau định nghĩa phương thức validateValue() để xác thực tính hợp lệ của giá trị thời gian:


```
private Boolean validateValue(long time) {
    // Perform validation
    return true; // If the time is valid
}

private void storeDateInDB(java.util.Date date) throws SQLException {
    final java.util.Date copy = (java.util.Date)date.clone();
    if (validateValue(copy.getTime())) {
        Connection con =
            DriverManager.getConnection(
                "jdbc:microsoft:sqlserver://<HOST>:1433",
                "<UID>", "<PWD>"
            );
        PreparedStatement pstmt = con.prepareStatement("UPDATE ACCESSDB
SET TIME = ?");
        pstmt.setLong(1, copy.getTime());
```

```
            // ...
        }
    }
```

Phương thức storeDateInDB() chấp nhận một tham số date không tin cậy và cố gắng tạo ra một bản sao an toàn sử dụng phương thức clone(). Điều này cho phép người tấn công chiếm lấy quyền điều khiển của chương trình bằng cách tạo ra một class date độc từ cách mở rộng class Date. Nếu mã của người tấn công chạy với cùng quyền với phương thức storeDateInDB() thì người tấn công chỉ đơn thuần nhúng mã độc vào bên trong hàm clone() của họ:

```
class MaliciousDate extends java.util.Date {
    @Override
    public MaliciousDate clone() {
        // malicious code goes here
    }
}
```

Tương tự, bằng cách mở rộng class Date và với quyền thực thi thấp hơn cho tham số date độc, người tấn công vẫn có thể vượt qua xác thực tính hợp lệ và vẫn có thể làm hỏng phần còn lại của chương trình. Bạn hãy viết một đoạn mã nguồn để thực hiện nhiệm vụ phá hoại này.

Hướng dẫn 1:

Kế thừa class Date và chỉnh sửa phương thức getTime() sao cho lần đầu tiên được gọi thì trả về kết quả đúng để vượt qua xác thực nhưng lần sau khi gọi để lấy giá trị lưu vào cơ sở dữ liệu thì sẽ trả về giá trị sai.

Bạn đã thấy được các người tấn công dựa vào phương thức clone() để vượt qua kiểm tra tính hợp lệ và kiểm tra bảo mật. Nhiệm vụ kế tiếp của bạn là sửa lại đoạn chương trình đầu tiên để chương trình hoạt động an toàn.

Hướng dẫn 2:

Tìm hiểu tài liệu để biết nhiệm vụ của hàm clone().

Nghĩ cách khác để thực hiện nhiệm vụ đó mà không sử dụng hàm clone().

3. Không dựa trên những phương thức có thể bị ghi đè (overridden) bởi mã không tin cậy

Mã không tin cậy có thể tận dụng những API được cung cấp bởi mã tin cậy để ghi đè những phương thức như Object.equals(), Object.Code() và Thread.run(). Những phương thức này trở thành những mục tiêu có giá trị bởi vì chúng có thể được sử dụng ngoài kịch bản và có thể tương tác với những thành phần (component) trong cách thức không dễ dàng để nhận thấy.

Với khả năng ghi đè, người tấn công có thể sử dụng mã không tin cậy để thu thập những không tin nhạy cảm, chạy (run) mã tùy ý hay khởi chạy một cuộc tấn công từ chối dịch vụ.

Mã nguồn bên dưới trình bày một class LicenseManager giữ licensMap. Map lưu giữ giá trị license với khóa (key) là LicenseType.

```
public class LicenseManager {
    Map<LicenseType, String> licenseMap = new HashMap<LicenseType, String>();
    public LicenseManager() {
            LicenseType type = new LicenseType();
            type.setType("demo-license-key");
            licenseMap.put(type, "ABC-DEF-PQR-XYZ");
    }
    public Object getLicenseKey(LicenseType licenseType) {
            return licenseMap.get(licenseType);
    }
    public void setLicenseKey(LicenseType licenseType, String licenseKey) {
            licenseMap.put(licenseType, licenseKey);
    }
}

class LicenseType {
    private String type;
    public String getType() {
            return type;
    }
    public void setType(String type) {
            this.type = type;
    }

    @Override
    public int hashCode() {
            int res = 17;
            res = res * 31 + type == null ? 0 : type.hashCode();
```

```
            return res;
        }

        @Override
        public boolean equals(Object arg) {
            if (arg == null || !(arg instanceof LicenseType)) {
                return false;
            }
            if (type.equals(((LicenseType) arg).getType())) {
                return true;
            }
            return false;
        }
    }
```

Hàm khởi tạo LicenseMap khởi tạo licenseMap với một khóa giấy phép demo cần phải giữ bí mật. Khóa giấy phép ở đây được mã cứng cho mục đích minh họa. Thực tế, khóa này được đọc từ một tập tin cấu hình bên ngoài và được lưu trữ dưới dạng đã được mã hóa. Class LicenseType cho phép ghi đè cho các phương thức equals() và hashCode().

Khả năng được ghi đè tạo ra những lỗ hổng để người tấn công mở rộng class LicenseType và ghi đè phương thức equals() và hashCode():

```
    public class CraftedLicenseType extends LicenseType {
        private static int guessedHashCode = 0;
        @Override
        public int hashCode() {
            // Returns a new hashCode to test every time get() is called
            guessedHashCode++;
            return guessedHashCode;
        }

        @Override
        public boolean equals(Object arg) {
            // Always returns true
            return true;
```

```
    }
}
```

Nhiệm vụ: viết một đoạn mã để in ra license.

```
public class DemoClient {
    public static void main(String[] args) {
        // Nội dung mã nguồn cần viết
    }
}
```

Hướng dẫn:

Mã cần viết sẽ duyệt qua tất cả trường hợp có thể của hash code sử dụng CraftedLicenseType cho đến khi nó khớp với hash code của demo license key được lưu trong class LicenseManager.

Có nhiều giải pháp để giải quyết vấn đề này:

- Sử dụng IdentityHashMap thay thế cho HashMap để lưu thông tin giấy phép

```
public class LicenseManager {
    Map<LicenseType, String> licenseMap = new IdentityHashMap<LicenseType, String>();
    // ...
}
```

Nhiệm vụ 1: trình bày lý thuyết về IdentityHashMap và so sánh với HashMap.

Hướng dẫn: xem xét cách lưu trữ, tham chiếu và cách so sánh giữa 2 key.

- Sử dụng kiến thức khi khai báo class để giới hạn phạm vi, khả năng của class

Nhiệm vụ 2: viết mã nguồn khai báo class LicenseType để xử lý vấn đề ghi đè.

Hướng dẫn: xem lại lý thuyết khai báo class.

4. Cẩn thận khi cho phép hàm khởi tạo đưa ra ngoại lệ

Một đối tượng được khởi tạo một phần nếu hàm khởi tạo đã bắt đầu xây dựng đối tượng nhưng không hoàn thành. Miễn là đối tượng không được khởi tạo hoàn thành thì nó phải được ẩn với những class khác.

Những class khác có thể truy cập vào những đối tượng được khởi tạo một phần từ những luồng (thread) đang chạy cùng lúc.

Mã dưới đây định nghĩa hàm khởi tạo cho class BankOperations và xác thực SSN sử dụng phương thức performSSNVerification(). Phương thức này giả định rằng người tấn công không biết chính xác SSN nên kết quả xác thực là false.

```
public class BankOperations {
```

```
    public BankOperations() {
        if (!performSSNVerification()) {
            throw new SecurityException("Access Denied!");
        }
    }
    private boolean performSSNVerification() {
        return false;
        // Returns true if data entered is valid, else false.
        // Assume that the attacker always enters an invalid SSN.
    }
    public void greet() {
        System.out.println("Welcome user! You may now use all the features.");
    }
}
public class Storage {
    private static BankOperations bop;
    public static void store(BankOperations bo) {
        // Only store if it is initialized
        if (bop == null) {
            if (bo == null) {
                System.out.println("Invalid object!");
                System.exit(1);
            }
            bop = bo;
        }
    }
}
public class UserApp {
    public static void main(String[] args) {
        BankOperations bo;
        try {
            bo = new BankOperations();
```

```
            } catch (SecurityException ex) { bo = null; }
            Storage.store(bo);
            System.out.println("Proceed with normal logic");
        }
    }
```

Hàm khởi tạo ném ra SecurityException khi xác thực SSN không đúng. Class UserApp sẽ bắt được ngoại lệ này và hiển thị một thông báo "Access Denied". Tuy nhiên, chính chỗ này làm cho chương trình độc có thể gọi những phương thức của class BankOperations được khởi tạo một phần.

Mục đích của người tấn công là bắt được tham chiếu đến đối tượng được khởi tạo một phần của class BankOperations. Nếu subclass độc bắt được SecurityException được ném ra bởi hàm khởi tạo BankOperations, thì nó không thể khai thác lỗ hỏng mã nguồn. Thay vào đó, người tấn công có thể khai thác mã này bằng cách mở rộng class BankOperations và ghi đè phương thức finalize().

Khi hàm khởi tạo ném ra một ngoại lệ, bộ thu dọn rác chờ để thu gom tham chiếu của đối tượng. Tuy nhiên, đối tượng không thể bị thu dọn cho đến sau khi bộ finalizer thực thi hoàn thành. Bộ finalizer của người tấn công sẽ giữ và lưu tham chiếu bằng cách sử dụng keyword this. Do đó, người tấn công có thể gọi bất kỳ phương thức trên class cơ sở bằng cách sử dụng tham chiếu vừa cướp được. Tấn công này thậm chí có thể vượt qua sự kiểm tra của cơ chế quản lý bảo mật.

```
public class Interceptor extends BankOperations {
    private static Interceptor stealInstance = null;
    public static Interceptor get() {
        try {
            new Interceptor();
        } catch (Exception ex) {/* ignore exception */}
        try {
            synchronized (Interceptor.class) {
                while (stealInstance == null) {
                    System.gc();
                    Interceptor.class.wait(10);
                }
            }
        } catch (InterruptedException ex) { return null; }
        return stealInstance;
```

```
    }
    public void finalize() {
        synchronized (Interceptor.class) {
            stealInstance = this;
            Interceptor.class.notify();
        }
        System.out.println("Stole the instance in finalize of " + this);
    }
    public class AttackerApp { // Invoke class and gain access
        // to the restrictive features
        public static void main(String[] args) {
            Interceptor i = Interceptor.get(); // stolen instance
            // Can store the stolen object even though this should have printed
            // "Invalid Object!"
            Storage.store(i);
            // Now invoke any instance method of BankOperations class
            i.greet();
        }
        UserApp.main(args); // Invoke the original UserApp
    }
}
```

Nhiệm vụ: dựa vào kiến thức đã học trên lớp sửa lại class BankOperations để người tấn công không thể khai thác. Lưu ý: vẫn giữ lại việc đưa ra ngoại lệ thông báo khi xác thực SSN sai.

Hướng dẫn:

Có 3 hướng giải quyết:

- Giải quyết vấn đề khi khai báo class
- Giải quyết khi viết phương thức finalize()
- Đặt cờ khởi tạo

C. YÊU CẦU

- Sinh viên tìm hiểu và thực hành theo hướng dẫn.
- Nộp báo cáo kết quả gồm chi tiết những việc bạn đã thực hiện, quan sát thấy và mã nguồn; giải thích cho quan sát (nếu có).
- Sinh viên báo cáo kết quả thực hiện và nộp bài.

Báo cáo:

- File .PDF. Tập trung vào nội dung, không mô tả lý thuyết.
- Đặt tên theo định dạng: [Mã lớp]-LabX_MSSV1-Tên SV.

Ví dụ: [NT101.H11.1]-Lab1_14520000-NguyenVanA.

- Nếu báo cáo có nhiều file, nén tất cả file vào file .ZIP với cùng tên file báo cáo.
- Nộp file báo cáo trên theo thời gian đã thống nhất tại courses.uit.edu.vn.

Đánh giá: Sinh viên hiểu và tự thực hiện được bài thực hành. Khuyến khích:

- Chuẩn bị tốt và đóng góp tích cực tại lớp.
- Có nội dung mở rộng, ứng dụng trong kịch bản phức tạp hơn, có đóng góp xây dựng bài thực hành.

Bài sao chép, trễ, … sẽ được xử lý tùy mức độ vi phạm.

D. THAM KHẢO

[1] NetBeans IDE 8.2 Installation Instructions, https://netbeans.org/community/releases/82/install.html
[2] Object cloning in Java, https://www.javatpoint.com/object-cloning
[3] The try Block, https://docs.oracle.com/javase/tutorial/essential/exceptions/try.html

HẾT