Javaセキュアコーディングセミナー東京
第4回
メソッドとセキュリティ
演習解説

2012年12月16日(日)
JPCERTコーディネーションセンター
脆弱性解析チーム
熊谷 裕志
戸田 洋三

Hands-on Exercise

- finalizer 攻撃を阻止しよう
- 他のクラスから、htに入っているキー「1」のエントリを消せるか？

Hands-on Exercise(1)

finalizer 攻撃を阻止しよう

サンプルアプリケーション(1/2)

```
public class LicenseManager {
  public LicenseManager() {
    if (!licenseValidation()) {
      throw new SecurityException("License Invalid!");
    }
  }
  private boolean licenseValidation() {
    // ライセンスファイルをリードしてチェックし、ライセンスが正当ならtrueを返す
    return false;
  }
}

public class SecuritySystem {
  private static LicenseManager licenseManager = null;
  public static void register(LicenseManager lm) {
    // licenseManagerが初期化されていない場合のみ登録
    if (licenseManager == null) {
      if (lm == null) {
        System.out.println("License Manager invalid!");
        System.exit(1);
      }
      licenseManager = lm;
    }
  }
}
```

Heinz M. Kabutz. *Exceptional Constructors - Ressurecting the dead.* Java Specialists' Newsletter. 2001

サンプルアプリケーション(2/2)

```
public class Application {
  public static void main(String[] args) {
    LicenseManager lm;
    try {
      lm = new LicenseManager();
    } catch(SecurityException ex) { lm = null; }

    SecuritySystem.register(lm);
    System.out.println("Now let's get things started");
  }
}
```

```
% ls *.java
Application.java    LicenseManager.java    SecuritySystem.java
% javac *.java
% java Application
License Manager invalid!
%
```

ファイナライザー攻撃を行うコード(1/2)

攻撃コード

```
public class LicenseManagerInterceptor extends LicenseManager {
  private static LicenseManagerInterceptor instance = null;
  public static LicenseManagerInterceptor make() {
    try {
      new LicenseManagerInterceptor();
    } catch(Exception ex) {} // 例外を無視
    try {
      synchronized(LicenseManagerInterceptor.class) {
        while (instance == null) {
          System.gc();
          LicenseManagerInterceptor.class.wait(100);
        }
      }
    } catch(InterruptedException ex) {
      return null;
    }
    return instance;
  }
  public void finalize() {
    System.out.println("In finalize of " + this);
    synchronized(LicenseManagerInterceptor.class) {
      instance = this;
      LicenseManagerInterceptor.class.notify();
    }
  }
  public LicenseManagerInterceptor() {
    System.out.println("Created LicenseManagerInterceptor");
  }
}
```

ファイナライザー攻撃を行うコード(2/2)

攻撃コード

```
public class AttackerApp {
  public static void main(String[] args) {
    LicenseManagerInterceptor lm = LicenseManagerInterceptor.make();
    SecuritySystem.register(lm);
    // now we call the other application
    Application.main(args);
  }
}
```

```
% ls
Application.class    LicenseManager.class    SecuritySystem.class
AttackerApp.java   LicenseManagerInterceptor.java
% javac *.java
% java AttakerApp
In finalize of LicenseManagerInterceptor@7dcb3cd
Now let's get things started
%
```

ファイナライザ攻撃対策

- **finalize()**メソッドを上書きされないように定義
- 重要なインスタンスは、初期化の完了を必ず確認
- サブクラス化による悪用を防ぐために、クラスを**final**宣言する

サンプルコードを
修正してみよう!

ファイナライザ攻撃対策

解説編

- **finalize()**メソッドを上書きされないように定義
- 重要なインスタンスは、初期化の完了を必ず確認
- サブクラス化による悪用を防ぐために、クラスを**final**宣言する

- **finalize()**をサブクラスで上書きされないよう **final** 宣言つきで定義
- **LicenseManager** クラスを **final** 宣言

LicenseManager の修正

解説編

```
public final class LicenseManager {
  public LicenseManager() {
    if (!licenseValidation()) {
      throw new SecurityException("License Invalid!");
    }
  }
  private boolean licenseValidation() {
    // ライセンスファイルをリードしてチェックし、ライセンスが正当ならtrueを返す
    return false;
  }

  @Override
  protected final void finalize(){}

}
```

final宣言を追加

final宣言つきで**finalize()** メソッドを定義

修正の効果

解説編

この修正により、攻撃コードはコンパイルできなくなる。

```
% javac AttackerApp.java LicenseManagerInterceptor.java
LicenseManagerInterceptor.java:1: error: cannot inherit from final LicenseManager
public class LicenseManagerInterceptor extends LicenseManager {
                                               ^
LicenseManagerInterceptor.java:19: error: finalize() in LicenseManagerInterceptor cannot
override finalize() in LicenseManager
  public void finalize() {
              ^
  overridden method is final
2 errors
%
```

ファイナライザ攻撃対策

解説編

- **finalize()**メソッドを上書きされないように定義
- **重要なインスタンスは、初期化の完了を必ず確認**
- サブクラス化による悪用を防ぐために、クラスを**final**宣言する

LicenseManager のコンストラクタが正常終了したことを確認できるようにする

初期化完了フラグを設ける

LicenseManager の修正

解説編

```
public class LicenseManager {

  boolean init = false;

  public LicenseManager() {
    if (!licenseValidation()) {
      throw new SecurityException("License Invalid!");
    }
    init = true;
  }
  private boolean licenseValidation() {
    // ライセンスファイルをリードしてチェックし、ライセ
     す
    return false;
  }

}
```

初期化完了フラグ

コンストラクタが正常終了する
最後に初期化完了フラグの値を
変更する

SecuritySystem の修正

解説編

```
public class SecuritySystem {
  private static LicenseManager licenseManager = null;
  public static void register(LicenseManager lm) {
    // licenseManagerが初期化されていない場合のみ登録
    if (licenseManager == null) {
      if (lm == null) {
        System.out.println("License Manager invalid!");
        System.exit(1);
      }
      if (lm.init == false) {
        System.out.println("incomplete License Manager!");
        System.exit(2);
      }
      licenseManager = lm;
    }
  }
}
```

渡されたインスタンスが正しく初期化されたものかどうかチェックする

修正の効果

解説編

この修正により、初期化完了していないインスタンスを検出できるようになる。

```
% java AttackerApp
In finalize of LicenseManagerInterceptor@2aa05bc3
incomplete License Manager!
%
```

しかしこれでは不十分!

攻撃コードで init の値を操作してしまえば...

攻撃コード(改)

解説編

攻撃コード

```
public class AttackerApp {
  public static void main(String[] args) {
    LicenseManagerInterceptor lm = LicenseManagerInterceptor.make();
    lm.init = true;
    SecuritySystem.register(lm);
    // now we call the other application
    Application.main(args);
  }
}
```

init の値を操作

```
% javac AttackerApp.java
% java AttackerApp
In finalize of LicenseManagerInterceptor@2aa05bc3
Now let's get things started
%
```

攻撃成功!

初期化完了フラグを改変されないようにする必要がある

LicenseManager をさらに修正

解説編

```
public class LicenseManager {

  private boolean init = false;

  public boolean is_init(){ return init; }

  public LicenseManager() {
    if (!licenseValidation()) {
      throw new SecurityException("License Invalid!");
    }
    init = true;
  }
  private boolean licenseValidation() {
    // ライセンスファイルをリードしてチェックし、ライセンスが正当ならtrueを返す
    return false;
  }

}
```

private 宣言する

init の値を調べるメソッドを追加

SecuritySystem をさらに修正

解説編

```
public class SecuritySystem {
  private static LicenseManager licenseManager = null;
  public static void register(LicenseManager lm) {
    // licenseManagerが初期化されていない場合のみ登録
    if (licenseManager == null) {
      if (lm == null) {
        System.out.println("License Manager invalid!");
        System.exit(1);
      }
      if (lm.is_init() == false) {
        System.out.println("incomplete License Manager!");
        System.exit(2);
      }
      licenseManager = lm;
    }
  }
}
```

初期化完了のチェックはメソッドを呼び出す形に変更

さらなる修正の効果

解説編

AttackerApp から初期化フラグを改変できなくなる。

```
% javac AttackerApp.java
AttackerApp.java:4: error: init has private access in LicenseManager
    lm.init = true;
       ^
1 error
%
```

Hands-on Exercise(2)

他のクラスから、htに入っているキー「1」のエントリを消せるか？

Hands-on Exercise(2)
他のクラスから、htに入っているキー「1」のエントリを消せるか？

問題のコード

```
import java.util.Hashtable;
import java.security.AccessController;
import java.security.SecurityPermission;

public final class SensitiveHash {
  private Hashtable<Integer,String> ht = new Hashtable<Integer,String>();

  SensitiveHash() {
    ht.put(1, "one");
    ht.put(2, "two");
    ht.put(3, "three");
  }

  void removeEntry(Object key) {
    check("removeKeyPermission");
    ht.remove(key);
  }

  public void showEntries() {
    System.out.println("" + ht.get(1));
    System.out.println("" + ht.get(2));
    System.out.println("" + ht.get(3));
  }

  private void check(String directive) {
    SecurityPermission sp = new SecurityPermission(directive);
    AccessController.checkPermission(sp);
  }
}
```

例えば

```
SensitiveHash sh = new SensitiveHash();
sh.removeEntry(1);
sh.showEntries();
```

セキュリティチェックされているため
エントリーを消すことはできない

Hands-on Exercise(2)
他のクラスから、htに入っているキー「1」のエントリを消せるか？

やってみる

Hands-on Exercise(2)
他のクラスから、htに入っているキー「1」のエントリを消せるか？

```
class AttackHash extends SensitiveHash {
  void removeEntry(Object key) {
    ht.remove(key);
  }
}
```

SensitiveHashクラスはfinal宣言されているのでサブクラスは作れない

```
import java.util.Hashtable;

public class Attack {
  public static void main(String[] args){
    SensitiveHash sh = new SensitiveHash();
    sh.removeEntry(1);
    sh.showEntries();
  }
}
```

removeKeyPermissionが許可されていないのでremoveEntryは実行できない

Hands-on Exercise(2)
他のクラスから、htに入っているキー「1」のエントリを消せるか？

リフレクションを使えば
エントリーを消すことができる

Hands-on Exercise(2)
他のクラスから、htに入っているキー「1」のエントリを消せるか？

```
import java.util.Hashtable;
import java.lang.reflect.Field;
import java.lang.reflect.Method;

public class AttackAns {
  public static void main(String[] args){

    try {
      Class<SensitiveHash> c = SensitiveHash.class;
      SensitiveHash sh = new SensitiveHash();

      Field field = c.getDeclaredField("ht");
      field.setAccessible(true);

      Hashtable hh = (Hashtable)field.get(sh);
      hh.remove(1);

      sh.showEntries();

    } catch (Exception ex) {
      ex.printStackTrace(System.out);
    }
  }
}
```

リフレクションを使ってprivate フィールドにアクセスする

Hashtable#removeで削除

Hands-on Exercise(2)
他のクラスから、htに入っているキー「1」のエントリを消せるか？

問題のコード

```
import java.util.Hashtable;
import java.security.AccessController;
import java.security.SecurityPermission;

public final class SensitiveHash {
  private Hashtable<Integer,String> ht = new Hashtable<Integer,String>();

  SensitiveHash() {
    ht.put(1, "one");
    ht.put(2, "two");
    ht.put(3, "three");
  }

  void removeEntry(Object key) {
    check("removeKeyPermission");
    ht.remove(key);
  }

  public void showEntries() {
    System.out.println("" + ht.get(1));
    System.out.println("" + ht.get(2));
    System.out.println("" + ht.get(3));
  }

  private void check(String directive) {
    SecurityPermission sp = new SecurityPermission(directive);
    AccessController.checkPermission(sp);
  }
}
```

「removeKeyPermission」の権限があるかどうかのチェックのみ

セキュリティマネージャは有効になっていない

ということで
リフレクションが使えて
エントリーを消すことができる