「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

JBoss Application Server におけるディレクトリトラバーサルの脆弱性

CVE-2006-5750

JVNDB-2006-002376

JBossとは

- JavaEEのアプリケーションサーバ
- JBossの名のもとに40以上のさまざまなプロジェクトが存在し、 JBoss.orgコミュニティによって開発、運営されている。

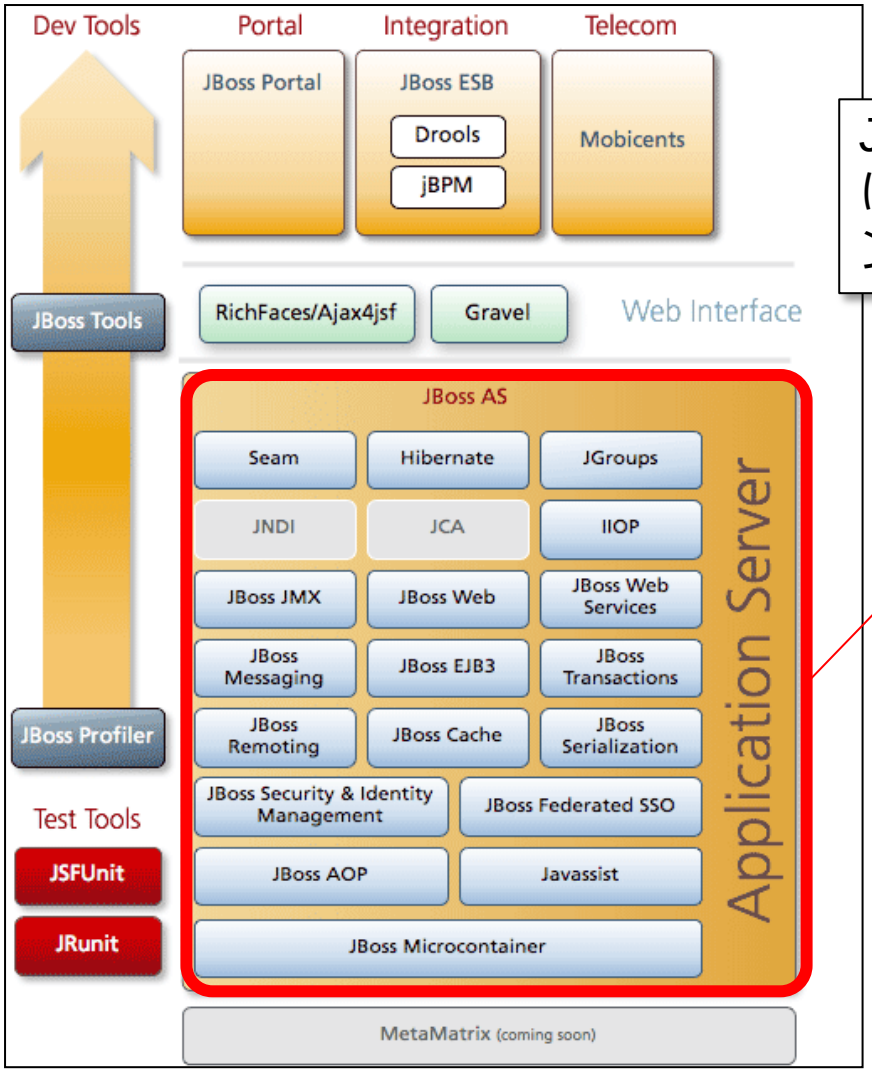
JBoss Application Serverとは

- Javaで記述されたサーバサイドアプリケーションを動作させるための基盤を提供する。

- JBossの様々なソフト群の中核になるソフトであることから、このソフトを指して単に「JBoss」と呼ぶ場合もある。

JBoss Application Serverとは

JBossプロジェクト　ポートフォリオ

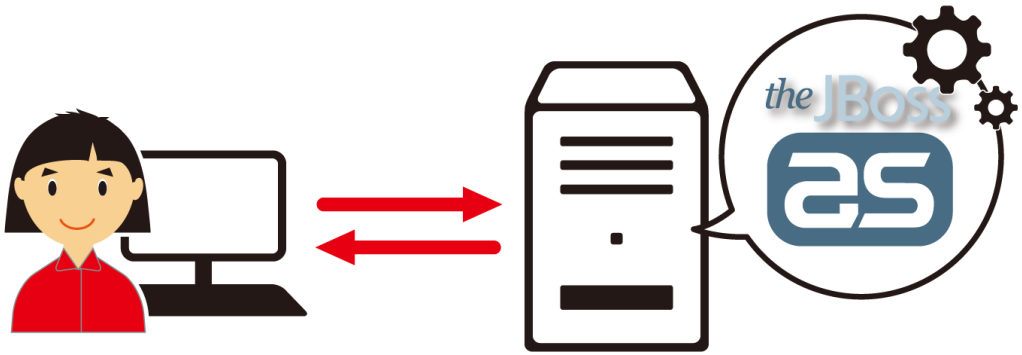


JBoss Application Server 内に”JBoss”と名の付く複数のコンポーネントが存在している。

※JBoss Application Server 5.0.0 Administration And Development Guideから引用
http://docs.jboss.org/jbossas/docs/Server_Configuration_Guide/beta500/html-single/index.html

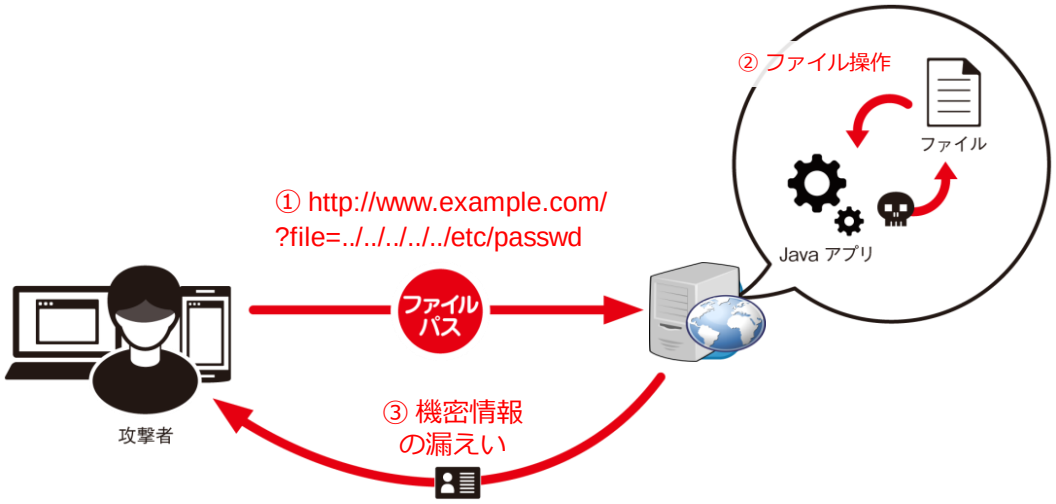
脆弱性の概要

- JBoss Application Serverには、JMXコンソールという管理機能が存在する。
- JMXコンソールは、外部からサーバのシャットダウンやファイルのアップロードなどの管理機能を提供している。
- その中のファイル操作機能においてディレクトリトラバーサルの脆弱性が存在する。



ディレクトリトラバーサルとは

- パスの値を不正に操作されることで、想定外のファイルやディレクトリに対して操作が行われてしまう攻撃。
- ファイル操作（読み出し/変更/削除等）におけるパスの値を、外部から受け取った入力を元に動的に生成するアプリケーションで発生する。



脆弱性が悪用された場合のリスク

■機密情報の漏えい
- アプリケーションが動作するサーバ上のファイルの内容が読み出され、機密情報が漏えいする可能性があります。

■システムやデータの破壊・改ざん
- アプリケーションが動作するサーバ上のファイルが改ざん・削除され、アプリケーションの誤動作、停止などサービス提供に影響が及ぶ可能性があります。

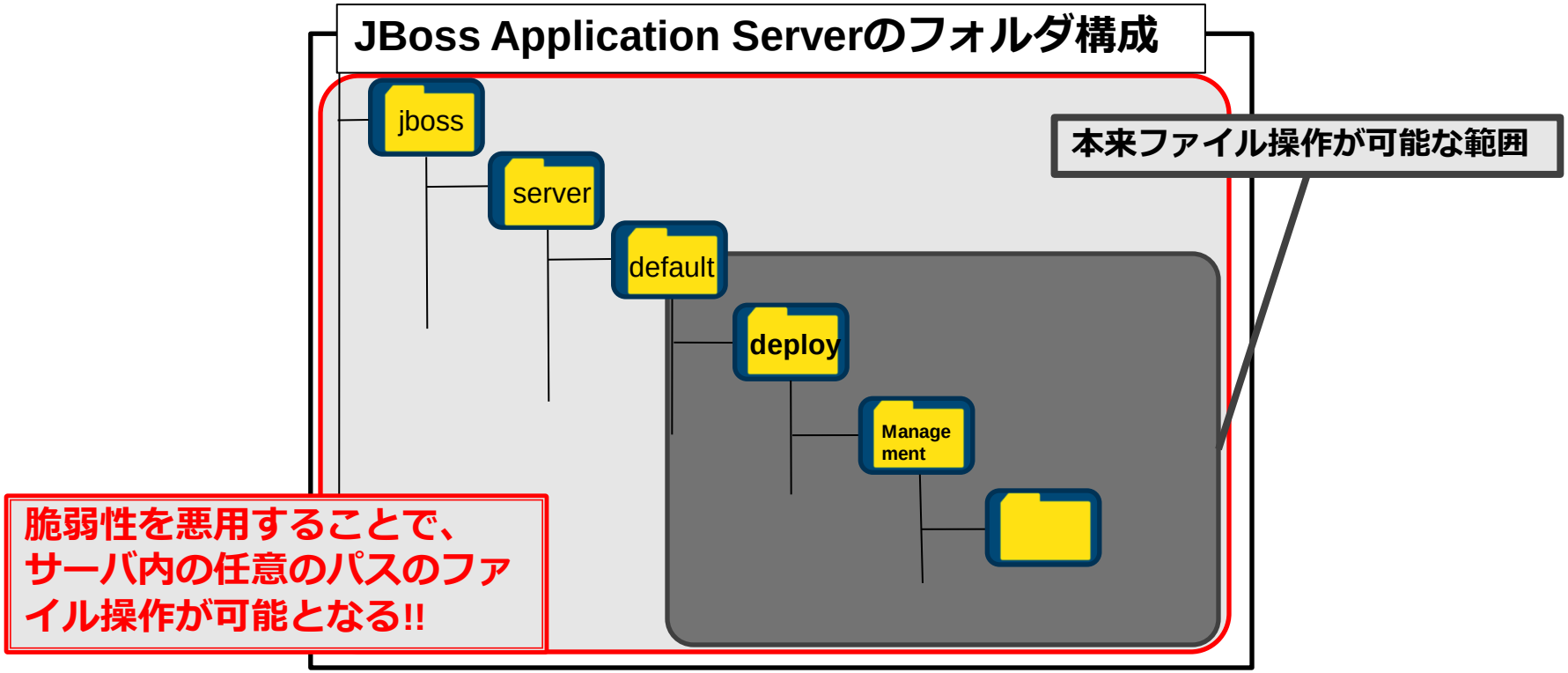
JMXコンソールとは

- JMXコンソールは、サーバの実行状況モニタリングや設定の変更などを行う機能
- JMXコンソールのファイル操作機能では以下の操作が可能
  - ファイル作成
  - ファイル削除
  - ファイル存在確認

- 本来はアプリケーションのプログラムが配置されているパス配下のファイルに対してのみ、上記の操作が可能である。

JMXコンソールの悪用

—JMXコンソールから不正なリクエストを送信するだけで脆弱性が悪用でき、アプリケーションが稼働するサーバ内の任意のファイル作成、削除、存在確認を行うことが可能となる。



JMXコンソールの悪用

■JMXコンソールを通じてファイル作成/削除/存在確認機能を悪用することでこんなことが可能!!

- **ファイル作成機能/ファイル削除機能**
  - ◆設定ファイルの削除/作成による上書き
    ⇒セキュリティ設定の変更、ユーザーの追加等
  - ◆不正なバイナリの配置
    ⇒ウイルスやバックドアの配置等
  - ◆サーバ上で稼働するサービスへの攻撃
    ⇒Webコンテンツの改ざん等
- **ファイル存在確認機能**
  ⇒ディレクトリ構造を調べることで、ＯＳバージョンの特定等

JMXコンソールにおけるファイル作成

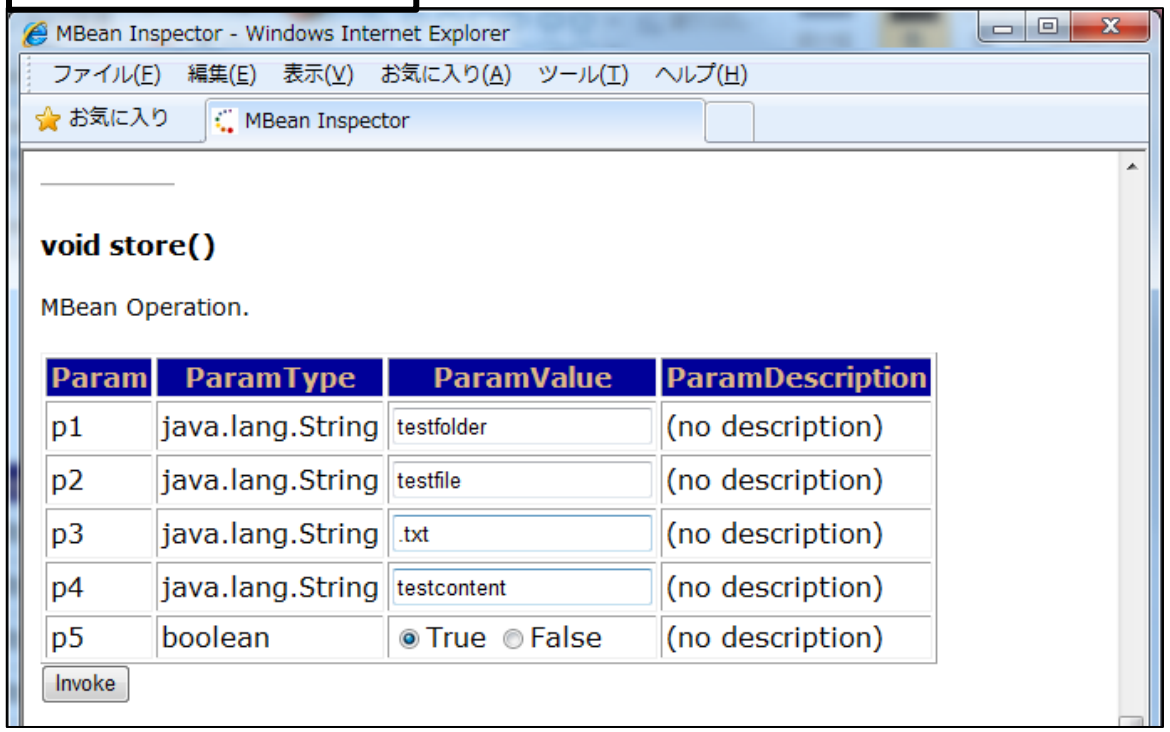
JMXコンソールを利用したファイル作成時の処理フローは以下のようになる。

① クライアントからリクエストが送信される
② アプリケーション(Jboss Application Server)がリクエストを受信し、フォルダ名、ファイル名、拡張子、ファイルデータの情報を取り出す
③ DeploymentFileRepositoryクラスのstoreメソッドでファイルを作成する
④ 結果を含むレスポンスがクライアントへ送信される

①クライアントからリクエストが送信される

ファイル「./testfolder/testfile.txt」作成リクエストがどのように処理されるか見てみよう。

JMXコンソール



①クライアントからリクエストが送信される

HTTPリクエスト

```
POST /jmx-console/HtmlAdaptor HTTP/1.1
Host: localhost:8080
:
action=invokeOp&name=jboss.admin%3Aservice%3DDeploymentFileRepository&
methodIndex=5&arg0=testfolder&arg1=testfile&arg2=.txt&arg3=testcontent&ar
g4=True
```

arg0：作成するファイルの上位フォルダ名
arg1：ファイル名
arg2：ファイル拡張子
arg3：ファイルの内容

上記HTTPリクエストを送信するためのHTML

```
<form action='/jmx-console/HtmlAdaptor' method='POST'>
:
<input type='hidden' name='arg0' value='testfolder'>
<input type='hidden' name='arg1' value='testfile'>
<input type='hidden' name='arg2' value='.txt'>
<input type='hidden' name='arg3' value='testcontent'>
:
</form>
```

②アプリケーションがリクエストを受信し、コピー処理を開始

ファイル作成はDeploymentFileRepositoryクラスの
storeメソッドで行われる。

HTTPリクエスト

POST /jmx-console/HtmlAdaptor HTTP/1.1
Host: localhost:8080
:
…&**arg0**=**testfolder**&**arg1**=**testfile**&**arg2**=**.txt**&**a**
rg3=**testcontent**&arg4=True

経由するメソッド

HtmlAdaptorServlet.doPost()
HtmlAdaptorServlet.processRequest()
HtmlAdaptorServlet.invokeOp()
Server.invokeOp()
Server.invokeOpByName()
MBeanServerImpl.invoke()
XMBean(AbstractMBeanInvoker).invoke()
Invocation.invoke()
Invocation.dispatch()
ReflectedDispatcher.invoke()
Method.invoke()
DelegatingMethodAccessorImpl.invoke()
NativeMethodAccessorImpl.invoke()
NativeMethodAccessorImpl.invoke0()
DeploymentFileRepository.store()

DeploymentFileRepository.java

public class
DeploymentFileRepository…
public void store(…

③DeploymentFileRepositoryクラスのstoreメソッドでファイルを作成

DeploymentFileRepository.java

```
public class DeploymentFileRepository extends ServiceMBeanSupport
    implements DeploymentFileRepositoryMBean
{
        :
        public void store(String folder, String name, String fileExtension, String data,
                                boolean noHotDeploy) throws IOException
        {
                :
```

arg0:”testfolder”

arg1:”testfile”

arg2:”.txt”

arg3:”testcontent”

storeメソッドの第1引数(folder)にリクエストのarg0、第2引数(name)にarg1、第3引数(fileExtension)にはarg2、第4引数(data)にはarg3の値が渡される。

③DeploymentFileRepositoryクラスのstoreメソッドでファイルを作成

DeploymentFileRepository.java

```
public class DeploymentFileRepository …{
        :
        public void store(String folder, String name, String fileExtension,
                                String data, boolean noHotDeploy) throws IOException
        {
                log.debug("store called");
                File dir = new File(base, folder);
                log.debug("respository folder: " + dir.toString());
                log.debug("absolute: " + dir.getAbsolutePath());
                if (!dir.exists())
                {
                   if (!dir.mkdirs())
                   {
                      throw new RuntimeException("Failed to create directory: " +
                                                        dir.toString());
                   }
                }
```

testfolder

③DeploymentFileRepositoryクラスのstoreメソッドでファイルを作成

DeploymentFileRepository.java

```
public class DeploymentFileRepository …{
    :
    public void store(String folder, String name, String fileExtension,
                      String data, boolean noHotDeploy) throws IOException
    {
        log.debug("store called");
        File dir = new File(base, folder);
        log.debug("respository folder: " + dir.toString());
        log.debug("absolute: " + dir.getAbsolutePath());
        if (!dir.mkdirs())
        {
            throw new RuntimeExc
        }
    }
```

testfolder

./deploy/management/testfolder

./deploy/management

引数folderを使用してFileオブジェクトを作成する。
baseはjmxコンソール上で設定されたコンテンツ保存のディレクトリパスのFileオブジェクトであり、デフォルトでは「./deploy/management」となる。
そのため、ここで作成されるFileオブジェクトのパスは「./deploy/management/testfolder」となる。

③DeploymentFileRepositoryクラスのstoreメソッドでファイルを作成

DeploymentFileRepository.java

```
public class DeploymentFileRepository …{
    :
    public void store(String folder, String name, String fileExtension,
                      String data, boolean noHotDeploy) throws IOException
    {
        log.debug("store called");
        File dir = new File(base, folder);
        log.debug("respository folder: " + dir.toString());
        log.debug("absolute: " + dir.getAbsolutePath());
        if (!dir.exists())
        {
            if (!dir.mkdirs())
            {
                throw new RuntimeException("Failed to create directory: " +
                                                    dir.toString());
            }
        }
```

./deploy/management/testfolder

「./deploy/management/testfolder」が作成される

③DeploymentFileRepositoryクラスのstoreメソッドでファイルを作成

DeploymentFileRepository.java

```
public class DeploymentFileRepository …{
    :
    public void store(String folder, String name, String fileExtension,
                      String data, boolean noHotDeploy) throws IOException
    {
        :
                if (!dir.mkdirs())
        :
        String filename = name.replace(' ', '_') + fileExtension;
        File file = new File(dir, filename);
        File tmpfile = new File(dir, filename + ".tmp");
        PrintWriter writer = new PrintWriter(new FileOutputStream(tmpfile));
        writer.write(data);
        writer.close();
        :
```

testfile

.txt

第2引数と第3引数を使ってfilenameを作成する。ここでは「testfile.txt」となる。

③DeploymentFileRepositoryクラスのstoreメソッドでファイルを作成

DeploymentFileRepository.java

```
public class DeploymentFileRepository …{
        :
        public void store(String folder, String name, String fileExtension,
                                              n noHotDeploy) throws IOException

                          if (!dir.r
          :
          String filename = name.replace(' ', '_') + fileExtension;
          File file = new File(dir, filename);
          File tmpfile = new File(dir, filename + ".tmp");
          PrintWriter writer = new PrintWriter(new FileOutputStream(tmpfile));
          writer.write(data);
          writer.close();
          :
```

./deploy/management/testfolder/testfile.txt

./deploy/management/testfolder

testfile.txt

Fileオブジェクト
「./deploy/management/testfolder /testfile.txt」
を作成。

③DeploymentFileRepositoryクラスのstoreメソッドでファイルを作成

DeploymentFileRepository.java

```
public class DeploymentFileRepository …{
        :
        public void store(String folder, String name, String fileExtension,
                                                                    otDeploy) throws IOException

                        if (
          :
          Strin   ename = na   eplace(' ', '_'              sion;
          File file = new File(dir,   ename);
          File tmpfile = new File(dir, filename + ".tmp");
          PrintWriter writer = new PrintWriter(new FileOutputStream(tmpfile));
          writer.write(data);
          writer.close();
          :
```

./deploy/management/testfolder /testfile.txt.tmp

./deploy/management/testfolder

testfile.txt

一時ファイル用のFileオブジェクト
「 ./deploy/management/testfolder /testfile.txt .tmp」
を作成する。

③DeploymentFileRepositoryクラスのstoreメソッドでファイルを作成

DeploymentFileRepository.java

```
public class DeploymentFileRepository …{
    :
    public void store(String folder, String name, String fileExtension,
                      String data, boolean noHotDeploy) throws IOException
    {
        :
                if (!dir.mkdirs
        :
        String filename = name.replace(' ', '_') + fileExtension;
        File file = new File(dir, filename);
        File tmpfile = new File(dir, filename + ".tmp");
        PrintWriter writer = new PrintWriter(new FileOutputStream(tmpfile));
        writer.write(data);
        writer.close();
        :
```

testcontent

一時ファイル
「 ./deploy/management/testfolder /testfile.txt .tmp」
に第4引数dataの値を書きこむ。

./deploy/management/testfolder /testfile.txt.tmp

③DeploymentFileRepositoryクラスのstoreメソッドでファイルを作成

DeploymentFileRepository.java

```
public class DeploymentFileRepository …{
      :
      public void store(String folder, String name, String fileExtension,
                        String data, boolean noHotDeploy) throws IOException
      {
            :
            File file = new File(dir, filename);
            File tmpfile = new File(dir, filename + ".tmp");
            PrintWriter writer = new PrintWriter(new FileOutputStream(tmpfile));
            writer.write(data);
            writer.close();
            if (file.exists() && noHotDeploy)
            {
                  :
                  file.delete();
            }
            if (!tmpfile.renameTo(file))
            {
```

Fileオブジェクト
「./deploy/management/testfolder /testfile.txt」

「./deploy/management/testfolder /testfile.txt」
が既に存在していたら削除する。

③DeploymentFileRepositoryクラスのstoreメソッドでファイルを作成

DeploymentFileRepository.java

```
public class DeploymentFileRepository …{
                                                    String fileExtension,
                                    String data
    {
            :
        File file = new File(dir, filename);
        File tmpfile = new File(dir, filename + ".tmp");
        PrintWriter writer = new PrintWriter(new FileOutputStream(tmpfile));
        writer.write(data);
        writer.close();
        if (file.exists() && noHotDeploy)
        {
                :
            file.delete();
        }
        if (!tmpfile.renameTo(file))
        {
```

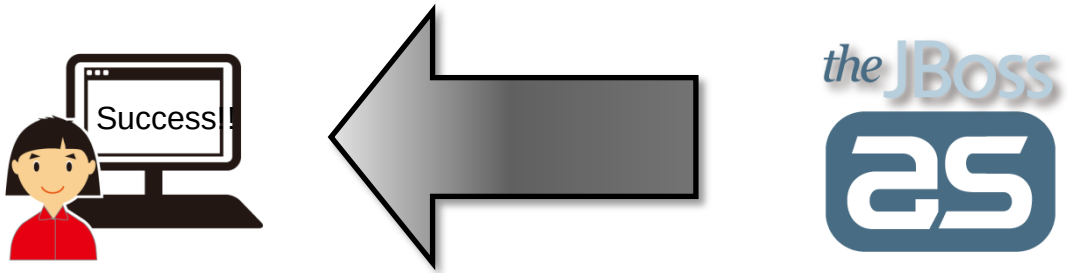
Fileオブジェクト
「./deploy/management/testfolder /testfile.txt」

Fileオブジェクト
「./deploy/management/testfolder/testfile.txt.tmp」

一時ファイル
「./deploy/management/testfolder /testfile.txt.tmp」を
「./deploy/management/testfolder /testfile.txt」に
変更して処理が完了。

④結果を含むレスポンスがクライアントへ送信される。

ファイル作成処理が終了し、結果をレスポンスとしてクライアントへ送信する。



攻撃コード

HTTPリクエスト

POST /jmx-console/HtmlAdaptor HTTP/1.1
Host: localhost:8080
:
action=invokeOp&name=jboss.admin%3Aservice%3DDeploymentFileRepository&methodIndex=5&**arg0**=**..¥..¥..¥..¥testfolder**&**arg1**=**testfile**&**arg2**=**.txt**&**arg3**=**testcontent**&arg4=True

arg0 : 作成するファイルの上位フォルダ名
arg1 : ファイル名
arg2 : ファイル拡張子
arg3 : ファイルの内容

■攻撃コードのポイント
パラメータarg0の値として上位ディレクトリを示す値を含んだ「..¥..¥..¥..¥testfolder」が指定されている。

攻撃コード

攻撃コードのHTTPリクエストを送信するためのHTML

```
<form action='/jmx-console/HtmlAdaptor' method='POST'>
:
<input type='hidden' name='arg0' value='..¥..¥..¥..¥testfolder'>
<input type='hidden' name='arg1' value='testfile'>
<input type='hidden' name='arg2' value='.txt'>
<input type='hidden' name='arg3' value='testcontent'>
:
</form>
```

arg0 : 作成するファイルの上位フォルダ名
arg1 : ファイル名
arg2 : ファイル拡張子
arg3 : ファイルの内容

攻撃コードが実行された際の処理

JMXコンソールでのファイル作成時の処理フロー

① クライアントからリクエストが送信される
② アプリケーション(Jboss Application Server)がリクエストを受信し、フォルダ名、ファイル名、拡張子、ファイルデータの情報を取り出す
③ DeploymentFileRepositoryクラスのstoreメソッドでファイルを作成する
④ 結果を含むレスポンスがクライアントへ送信される

攻撃コードが実行された際に、③の処理でディレクトリトラバーサル攻撃が成立する。

攻撃コード実行時

③DeploymentFileRepositoryクラスのstoreメソッドでファイル作成

DeploymentFileRepository.java

```
public class DeploymentFileRepository extends ServiceMBeanSupport
    implements DeploymentFileRepositoryMBean
{
        :
        public void store(String folder, String name, String fileExtension, String data,
                                boolean noHotDeploy) throws IOException
        {
                :
```

arg0:”..¥..¥..¥..¥testfolder”

arg1:”testfile”

arg2:”.txt”

arg3:”testcontent”

storeメソッドの第1引数(folder)にリクエストのarg0、第2引数(name)にarg1、第3引数(fileExtension)にはarg2、第4引数(data)にはarg3の値が渡される。

攻撃コード実行時

③DeploymentFileRepositoryクラスのstoreメソッドでファイル作成

DeploymentFileRepository.java

..¥..¥..¥..¥testfolder

```
public class DeploymentFileRepository …{
      :
      public void store(String folder, String name, String fileExtension,
                        String data, boolean noHotDeploy) throws IOException
      {
            log.debug("store called");
            File dir = new File(base, folder);
            log.debug("respository folder: " + dir.toString());
            log.debug("absolute: " + dir.getAbsolutePath());
            if (!dir.exists())
            {
               if (!dir.mkdirs())
               {
                  throw new RuntimeException("Failed to create directory: " +
                                                dir.toString());
               }
            }
```

攻撃コード実行時

③DeploymentFileRepositoryクラスのstoreメソッドでファイル作成

DeploymentFileRepository.java

```
public class DeploymentFileRepository …{
        :
        public void store(String folder, String n    e, String fileExtension,
                          String data, boolean    oHotDeploy) throws IOException
        {
            log.debug("store called");
            File dir = new File(base, folder);
            log.debug("resposito   folder: " + dir.toString());
            log.debug("absolute    dir.getAbsolutePath());
            if (!dir.exists())
            {
               if (!dir.mkdirs())
               {
                  throw new RuntimeExcep

               }
            }
```

..¥..¥..¥..¥testfolder

./deploy/management

引数folderを使用して作成したFileオブジェクトは
「 ./deploy/management/../../../../testfolder」
となる。
これはフォルダdeployの2階層上に位置する
「testfolder」フォルダを意味する。

攻撃コード実行時

③DeploymentFileRepositoryクラスのstoreメソッドでファイル作成

DeploymentFileRepository.java

```
public class DeploymentFileRepository …{
    :
    public void store(String folder, String name, String fileExtension,
                      String data, boolean noHotDeploy) throws IOException
    {
        log.debug("store called");
        File dir = new File(base, folder);
        log.debug("respository folder: " + dir.toString());
        log.debug("absolute: " + dir.getAbsolutePath());
        if (!dir.exists())
        {
            if (!dir.mkdirs())
            {
                throw new RuntimeException("Failed to create directory: " +
                                                    dir.toString());
            }
        }
```

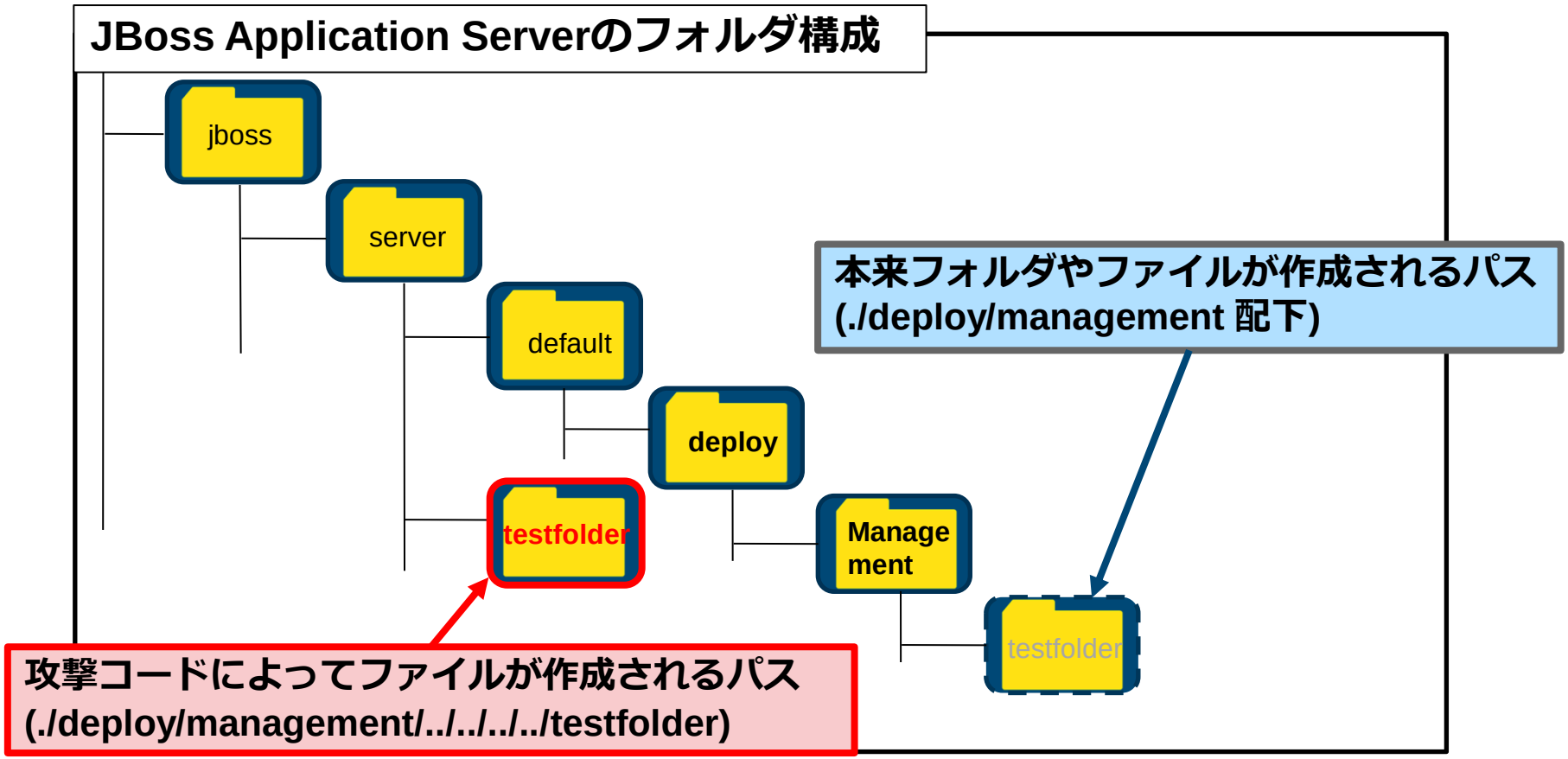
「./deploy/management/../../../../testfolder」というFileオブジェクト

「./deploy/management/../../../../testfolder」が作成されてしまう。

攻撃コード実行時

③DeploymentFileRepositoryクラスのstoreメソッドでファイル作成

その後は正常処理と同様の処理が実行され、下記の場所にファイルが作成されてしまう。本来は ./deploy/management 配下にフォルダ、ファイルが作成されるはずが、攻撃コードでパスを操作することで任意のパスにファイルを作成することが可能となる。



JMXコンソールにはファイル作成以外にも、次の3つの機能に同様の脆弱性が存在する。

1.ファイル削除(removeメソッド)

```
public void remove(String folder, String name, String fileExtension)
{
   File dir = new File(base, folder);
   String filename = name.replace(' ', '_') + fileExtension;
   File file = new File(dir, filename);
   file.delete();
}
```

2.ファイル存在確認(isStoredメソッド)

```
public boolean isStored(String folder, String name, String fileExtension)
{
   File dir = new File(base, folder);
   String filename = name.replace(' ', '_') + fileExtension;
   File file = new File(dir, filename);
   return file.exists();
}
```

3.ファイル保存パス設定(setBaseDirメソッド)

```
public void setBaseDir(String baseDir)
{
   this.baseDir = baseDir;
   this.base = new File(serverHome, baseDir);
}
```

問題点

- **今回のアプリケーションにおける具体的な問題点**

  引数として渡されたパスの値を検証せずに処理を行っていた。

以下のコーディングガイドに違反している

「 MET00-J. メソッドの引数を検証する」

「 IDS02-J. パス名は検証する前に正規化する」

問題点

- **問題点に対してどうすべきだったか**
  - ・引数の値を使って構成したパスが、想定しているディレクトリの下にあることを検証した上で、ファイルを作成すべきであった。
  - ・パスの検証のためには、パスの正規化処理が必要。

修正版コード

脆弱性はバージョン4.2.0で修正されている。

JMXコンソールでのファイル作成時の処理フロー

① クライアントからリクエストが送信される
② アプリケーション(Jboss Application Server)がリクエストを受信し、フォルダ名、ファイル名、拡張子、ファイルデータの情報を取り出す
③ DeploymentFileRepositoryクラスのstoreメソッドでファイルを作成する
④ 結果を含むレスポンスがクライアントへ送信される

③の処理を行うコードが修正されている。

修正前/修正後の処理比較

③DeploymentFileRepositoryクラスのstoreメソッドでファイル作成

DeploymentFileRepository.java (修正前)

```
public class DeploymentFileRepository …{
        :
        public void store(String folder, String name, String fileExtension, String data, boolean
        noHotDeploy) throws IOException
        {
            log.debug("store called");
            File dir = new File(base, folder);
            log.debug("respository folder: " + dir.toString());
              :
```

DeploymentFileRepository.java (修正後)

```
public class DeploymentFileRepository …{
        :
        public void store(String folder, String name, String fileExtension, String data, boolean
        noHotDeploy) throws IOException
        {
            log.debug("store called");
            File dir = getFile(base, folder);
            log.debug("respository folder: " + dir.toString());
              :
```

getFileメソッドを使用するようにコードが変更されている。

修正前/修正後の処理比較

③DeploymentFileRepositoryクラスのstoreメソッドでファイル作成

getFileメソッド

./deploy/management

storeメソッドの第1引数
(フォルダ名)

```
private File getFile(File parent, String child) throws IOException
{
    File childFile = new File(parent, child);

    if (childFile.getCanonicalPath().indexOf(parent.getCanonicalPath()) != 0)
        throw new IllegalArgumentException(
        "child '" + child + "' should be a child of parent '" + parent + "'"
        );

    return childFile;
}
```

新規作成するフォルダの
Fileオブジェクトを作成

getCanonicalPathメソッドで正規化をした後に、
indexOfメソッドを使用して parentがchildFileの
一部に含まれているか検証している。

※ getCanonicalPath メソッドを使用すると
「/parent/child1/child2/../../child0」→「 /parent/child0 」
というように正規化される。

修正前/修正後の処理比較

③DeploymentFileRepositoryクラスのstoreメソッドでファイル作成

■修正版コードが攻撃を受けるとどうなるか

getFileメソッド内でgetCanonicalPathメソッドにより
parent :
「./deploy/management」→
「/jboss/server/default/deploy/management」
childFile :
「 ./deploy/management/../../../../testfolder」→
「 /jboss/server/testfolder」
と正規化される。

パスを検証するコードが、childFileのパスにparentが含まれていないことを検知し、エラーとして処理する。

その他の修正

- 下記のメソッドも getFile メソッドを利用したパスの正規化と検証を行うように修正されている。
  - ファイル削除(removeメソッド)
  - ファイル存在確認(isStoredメソッド)
  - ファイル保存パス設定(setBaseDirメソッド)

まとめ

■ この脆弱性から学べるプログラミングの注意点
- アプリケーションの処理内容や扱うデータ構造に応じて、引数が適切な値であることを検証すべき。
- 今回のケースでは、ファイル/ディレクトリ操作で扱うパス名を細工されることで、ディレクトリトラバーサルの脆弱性につながった。

■ 上記への対策
- 引数のパス名を**正規化**してから(IDS02-J)、想定した範囲に収まっていることを確認する。

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
Jboss Application Server におけるディレクトリトラバーサルの脆弱性
https://www.jpcert.or.jp/securecoding/2012/No.05_JBoss.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