社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2008－CSEC－41（19）
2008／5／23

Java Plug-in および Java Web Start における隠れエントリクラスの脅威

兒島 尚* 中田 正弘*

*（株）富士通研究所

本論文では、Java Plug-in および Java Web Start において、アプレットまたはアプリケーションとして攻撃者に悪用される可能性があるにも関わらず、開発者が存在を意識していないクラスの危険性を示す。我々はこれらを隠れエントリクラスとよぶ。隠れエントリクラスは開発者による対策が施されていない可能性が高く、実行されるだけで深刻な被害を及ぼす恐れがある。我々は Java Plug-in および Java Web Start の実行環境に脆弱な隠れエントリクラスが存在することを発見した。これらは様々な環境に存在する恐れがあり対策は急務である。本論文ではその脅威と対策について説明する。

Dangerousness of Unnoticed Entry Classes in Java Plug-in / Web Start

Hisashi Kojima* Masahiro Nakada*

* Fujitsu Laboratories Ltd.

In this paper, we show dangerousness of unnoticed entry classes in Java Plug-in / Web Start. Unnoticed entry classes mean classes which are executable as applets or applications but unnoticed by developers. Such classes are likely to cause serious security breaches because usually they are not fortified at all. We discovered some vulnerable entry classes in Java Plug-in / Web Start implementations. Unnoticed entry classes may exist in various environments and there is a pressing need to deal with this issue. In this paper, we describe its threats and countermeasures.

1. はじめに

Java Plug-in および Java Web Start（以降、JPI/JWS）[1]は Java Runtime Environment（以降、JRE）上の RIA（Rich Internet Application）プラットフォームである。ここで、RIA とは Web ブラウザを通じて高機能なユーザーインターフェイスを提供するアプリケーションの総称であり、自動的にダウンロードされて実行されるため、配布コストが小さいという利点がある。JPI/JWS は Java プログラムであり、クラスの集合で構成され、JAR ファイルというアーカイブ形式で配布されることが一般的である。JPI はよく知られた技術である Java アプレットを主要な Web ブラウザで動作させるためのプラグインである。一方、JWS は JPI とよく似た技術であり、Web ページの所定のリンクをクリックすることで実行されるが、アプレットと異なり Web ブラウザ上ではなく独立したアプリケーションとして動作するという違いがある。JPI/JWS は Java の普及度の高さから広く利用されており、電子政府などの重要なシステムでも採用されている。

JPI/JWS の特徴のひとつがサンドボックス機構であり[2]、ダウンロードされたアプレットおよび JWS アプリケーションの動作は厳しく制限される。これにより悪意のあるプログラムからの攻撃を防ぐことができるが、必要な動作まで禁止されてしまうという問題がある。そこで、電子署名や配備場所などに基づき、信頼されたプログラムについては制限を緩和する仕組みが用意されている。例えば、電子署名付きアプレットを配布することにより、利用者の同意が得られれば、制限された動作を実行できるようになる。

この仕組みでは、信頼されないプログラムは依然として厳しい制限を受けるため、単独での攻撃は難しい。しかし、信頼されたプログラムを構成する JAR ファイルと、攻撃者の JAR ファイルを組み合わせることにより、信頼されたプログラムの機能を悪用する攻撃手法がある。我々はこれを再構成攻撃とよび、様々な手法が存在することを過去に示した[3]。特に、クラス置き換えという手法は強力であり、セキュリティ専門家ではない一般の開発者が適切に対策を施すことは難しい。そこで我々は、特別なクラスローダーで保護対象プログラムをカプセル化して再構成攻撃を防ぐ、Coziletと称するツールを作成した。

ただし、前述の論文では、アプレットまたは JWS アプリケーションとして実行されることを開発者が意図したクラスだけが攻撃対象として想定されていた。しかし、開発者が意図しなくとも、実行可能であればすべて攻撃対象になりうる。ここでは実行可能なクラスをエントリクラスとよび、開発者が意図したものを正規エントリクラス、開発者が意図しないものを隠れエントリクラスとよぶ。隠れエントリクラスは攻撃されることを想定していない可能性が高く、クラス置き換えなどの高度な攻撃手法を用いずとも、実行されるだけで深

刻な被害を引き起こす恐れがある。この脅威に対しては、一般のアプレット及びJWSアプリケーションだけでなく、JPI/JWS、ミドルウェア、そしてライブラリなど、多くの環境が影響を受ける。我々はJPI/JWSにおいて実際に脆弱な隠れエントリクラスが存在することを発見しており、他の環境にも存在する可能性は高い。この問題は情報セキュリティ早期警戒パートナーシップ[8]のもとで2006年10月にIPA[9]に報告され、2008年4月に対応が完了している。本論文ではその詳細について説明する。

本論文の構成は次の通りである。まず、第2章で隠れエントリクラスについて説明する。次に、第3章で隠れエントリクラスの脅威について説明し、第4章で対策を示す。最後に、第5章で関連研究を示し、第6章でまとめを述べる。

2. 隠れエントリクラス

ここではまず、隠れエントリクラスの定義やその特徴について説明する。

2.1. エントリクラス

既に述べたように、エントリクラスとはアプレットまたはJWSアプリケーションとして実行可能なクラスのことである。以下に定義を示す。

- アプレット: java.applet.Appletクラスのサブクラス。ここではアプレットクラスとよぶ。
- **JWS** アプリケーション: public static void main(String[])メソッドが定義されたクラス。ここではメインクラスとよぶ。

アプレットの場合はHTMLページのapplet要素のcode属性に、JWSアプリケーションの場合はJNLPファイル[12]のapplication-desc要素のmain-class属性にそれぞれエントリクラスを記述することで実行される。またJWSでは、application-desc要素の代わりにapplet-desc要素を記述することでアプレットを実行させることもできる。図 1と図 2に例を示す。

```
<html><body>
  :
  <applet
    code="foo.MyApplet"
    width="400" heigth="400"
    archive="foo.jar">
  <param name="cmd" value="install">
  </applet>
  :
</body></html>
```

図 1 アプレットのHTMLページの例

```
<?xml version="1.0"
      encoding="UTF-8"?>
<jnlp codebase="http://foo...com/">
  :
<resources>
  <j2se version="1.3+"/>
  <jar href="bar.jar"/>
</resources>
<!-- アプレットなら applet-desc -->
<application-desc
  main-class="bar.MyApp">
  <argument>install</argument>
</application-desc>
</jnlp>
```

図 2 JWSアプリケーションのJNLPファイルの例

攻撃者がHTMLページやJNLPファイルを作成した場合でも、参照されているクラスファイルが信頼されたものならば、プログラムとしては信頼されたものとして扱われる。JNLPファイルの場合には電子署名して改ざんを検出することが可能だが、この機能はオプションであり、多くのJWSアプリケーションでは使われていない可能性が高い。よって、適切なHTMLページやJNLPファイルを用意することにより、攻撃者が望むエントリクラスをアプレットまたはJWSアプリケーションとして実行させることが可能である。

2.2. クラスパス

ただし、全てのエントリクラスが実行可能ではなく、クラスパスに含まれている必要がある。クラスパスとは、クラスの実体であるクラスファイルの検索場所や検索順位を定義する概念である。クラスのロード要求があると、JREはクラスパスを参照してクラスファイルを検索する。以下に標準的なクラスパスを示す。

- システムクラスパス: 全てのアプレットおよびJWSアプリケーションにおいてデフォルトで検索対象となるクラスパス。JPI/JWS同梱クラスやミドルウェア同梱クラスなどが含まれる。
- プログラム毎クラスパス: 個々のアプレットおよびJWSアプリケーションに固有のクラスパス。アプレットにおけるHTMLページのarchive属性、JWSアプリケーションにおけるJNLPファイルのjar要素に記述されたパスが該当する。プログラム固有クラスや、プログラムが利用しているライブラリ同梱クラスなどが含まれる。

これらのクラスパスに含まれるエントリクラスは全て攻撃者に悪用される可能性がある。また、クラスパスにはURLを記述できるので、リモートのクラスも実行可能である。さらに、プログラムがアプレットとJWSアプリケーションのどちらとして配布されているかではなく、クラスパスに含まれているかどうかが重要であることに注意してほしい。例えば、アプレットとして配布

された JAR ファイルに隠れメインクラスが含まれていれば、JWS アプリケーションとして悪用される可能性がある。

2.3. JWSafeによる隠れエントリクラスの検出

隠れエントリクラスを探し出すには、クラスパスに含まれる全てのクラスについて、エントリクラスかどうか判定する必要がある。我々はこの判定を自動化するために、JWSafeと称する調査ツールを作成した。本ツールはJARファイルを入力とし、含まれる全てのクラスについてエントリクラスに該当するものを列挙する。本ツールはJRE上で動作し、アプレットクラスの判定にはリフレクションを、メインクラスの判定にはApache BCEL[5]を利用している。本ツールの出力例を図 3に示す。

```
% java -jar jws.jar find_main foo.jar
command: find_main
JAR files:
foo.jar

[foo.jar]
com/example/app/MyApp.class
com/example/app/Test1.class
com/example/app/Test2.class
```

図 3 JWSafe の出力例

上記の例は foo.jar という JAR ファイルについて、メインクラスの検出を行っている。ここでは、MyApp.class、Test1.class、Test2.class の 3 個のメインクラスが検出されている。MyApp.class が正規メインクラスの場合、Test1.class と Test2.class が隠れメインクラスである。本ツールにより隠れエントリクラスを容易に探し出すことができる。

2.4. サンドボックス機構による制限

ただし、隠れエントリクラスも通常のクラスと同様にサンドボックス機構による制限を受ける。よって、実際には、検出されたエントリクラスのうち、信頼されたクラスのみが攻撃対象となる。しかし、システムクラスパスに含まれるクラスは全て信頼されたものとみなされ、また、プログラム毎クラスパスに含まれるクラスも、プログラムが信頼されたものである場合には制限は緩和される。よって、検出されたエントリクラスが攻撃対象である可能性は高い。

2.5. アプレットクラスとメインクラスの違い

ここで、アプレットクラスとメインクラスの違いについて述べておく。両者は、アプレットとして実行されるか、JWS アプリケーションとして実行されるかの違いしかなく、攻撃対象となった場合の脅威は同様である。しかし、アプレットクラスの定義はアプレットに特化しており、作成者が明示的に作成しない限りは該当しにくいのに対し、メインクラスの定義は一般的な Java アプリケーションと同様であるため、作成者が明示的に作成しなくても悪用される恐れがある。例えば、デバッグ目的のテストプログラムは、通常は Java アプリケーションとして作成されるため、隠れエントリクラスとなってしまう可能性が高い。よって、隠れメインクラスの方が隠れアプレットクラスよりも脅威が大きいといえる。

3. 隠れエントリクラスの脅威

ここでは隠れエントリクラスの脅威について説明する。我々は脅威の大きさや対策の責任者の違いにより、隠れエントリクラスを JPI/JWS 同梱クラス、ミドルウェア同梱クラス、ライブラリ同梱クラス、プログラム固有クラスの 4 種類に分類した。ここではそれぞれについて説明する。

3.1. JPI/JWS同梱クラス

JPI/JWS 同梱クラスは JPI/JWS に標準で存在するクラスである。システムクラスであるためサンドボックス機構による制限を受けない。また、JPI/JWS をインストールしている全ての利用者が影響を受けるので、影響範囲が広い。もし攻撃が可能な隠れエントリクラスが存在した場合、すなわち JPI/JWS 自体の脆弱性を意味し、提供元の Sun 社などが責任を持って修正する必要がある。我々は JPI/JWS の隠れエントリクラスを調査し、それぞれに深刻な脆弱性が存在することを発見した(ただし最新版の JPI/JWS では修正済みである)。ここでは JPI/JWS の隠れエントリクラスと我々が発見した脆弱性について説明する。

3.1.1. JPIの隠れエントリクラスと脆弱性

第 2.5 節で述べたように、JPI の隠れエントリクラスは少なく、JWSafe で検出されたのは数個程度であった。しかし、2 個の脆弱な隠れエントリクラスが含まれていた。

- **ControlPanelAppletクラス[7]**: JPIの設定を変更するためのWindows用のコントロールパネルのアプレット版である。プロキシ設定や証明書ストアの情報など、利用者の設定情報が漏えいする脆弱性が存在した。
- **XSLTProcessorAppletクラス[10]**: JRE同梱のApache Xalanライブラリ[6]に含まれるクラスであり、XSLTプロセッサのアプレット版である。任意のリモートコード実行やローカルファイルの漏えいなどの脆弱性が存在した。

ただし、上記の脆弱性は最新版の JPI では修正済みである。我々の知る限り、他の脆弱な隠れエントリ

クラスは発見されていない。

3.1.2. JWSの隠れエントリクラスと脆弱性

JWSの仕様によれば、jar要素で明示的に指定された、プログラム毎クラスパスに含まれるクラスだけがエントリクラスとして実行可能とされている[12]。よって、JWS同梱のエントリクラスは攻撃対象にはならないことになる。しかし我々は、JWSの実装に脆弱性があり、JWS同梱のクラスがエントリクラスとして実行できることを発見した[11]。

JWS では、エントリクラスがプログラム毎クラスパスに含まれているかどうかのチェックをクラスファイルの存在の有無で行っていた。エントリクラスはチェックを通過した場合に限りクラスローダーによってロードされる。しかし、クラスローダーはプログラム毎クラスパスよりもシステムクラスパスを優先してクラスの検索を行うため、同名のクラスが両方のパスに含まれていた場合はJWS 同梱クラスが実行されてしまう。我々は、攻撃対象のJWS 同梱クラスと同名のダミークラスファイルをプログラム毎クラスパスに含めることで、チェックを回避してJWS 同梱クラスが実行されることを確認した。

JWS 同梱クラスの隠れエントリクラスは数多く存在する。以下に例を挙げる。

- **jar コマンドクラス**: jar コマンドの実体である。ローカルファイルが改ざんされる。
- **XSLT プロセッサクラス**: Apache Xalan ライブラリ同梱のXSLTプロセッサである。任意のリモートコードが実行される。
- **XSLTコンパイラクラス**: Apache Xalanライブラリ同梱の XSLT コンパイラである。任意のリモートコードが実行される。

ただし、上記の脆弱性は最新版の JWS では修正済みであり、JWS 同梱クラスは隠れエントリクラスとして実行できないため、隠れエントリクラスの脅威はない。

以上のことから、我々の知る限り、現在ではJPI/JWS 同梱クラスに対する隠れエントリクラスの脅威はないと考えられる。

3.2. ミドルウェア同梱クラス

ミドルウェア同梱クラスは、JPI/JWS同梱クラスと同様にシステムクラスパスに含まれ、サンドボックス機構による制限を受けないので、隠れエントリクラスの脅威は大きい。ミドルウェア同梱クラスの隠れエントリクラスについては、ミドルウェアがインストールされた環境が影響を受けるため、個々のミドルウェアの提供元が責任を持って対処する必要がある。ただし、現在では第3.1.2節で述べたJWSの脆弱性が修正されているため、システムクラスパスに含まれるミドルウェア同梱クラスがJWSで実行されることはない。よって、隠れアプレットクラスだけに注意すればよい。

3.3. ライブラリ同梱クラス

ライブラリ同梱クラスは、アプレットや JWS アプリケーションなどのプログラムが利用しているサードパーティ製ライブラリに含まれるクラスである。プログラムと共に配布され、プログラム毎クラスパスに含まれる。ライブラリ同梱クラスは通常はプログラムと同等の信頼度で実行され、プログラムが信頼されたものである場合、ライブラリ同梱クラスもサンドボックス機構による制限を受けないことが多い。しかし、ライブラリ同梱クラスは、利用者であるプログラム作成者が内部を熟知していないため、隠れエントリクラスが存在する可能性が高い。例えば、Apache Xalan ライブラリには以下の危険性の高いエントリクラスが含まれている。

- **XSLTProcessorAppletクラス**: 第3.1.1節と同様のアプレットクラス。利用者の設定情報が漏えいする恐れがある。
- **XSLTプロセッサクラス**: 第3.1.2節と同様のメインクラス。任意のリモートコードが実行される恐れがある。

ライブラリ同梱クラスに含まれる隠れエントリクラスについては、利用者であるプログラム作成者が注意を払う必要がある。もし、ライブラリ同梱クラスが攻撃された場合には、ライブラリ作成者ではなくプログラム作成者が責任を問われると考えられる。ただし、ライブラリ作成者が隠れエントリクラスのリストを提供することは有用である。

3.4. プログラム固有クラス

プログラム固有クラスは、アプレットやJWSアプリケーションの作成者が新規に開発したものである。通常、1 つのプログラムには 1 つの正規エントリクラスがあればよいが、デバッグ目的などで開発時に利用していたエントリクラスの消去を忘れて、配布時にも残ってしまう場合が考えられる。こうした隠れエントリクラスはプログラム作成者が責任を持って対処する必要がある。

以上のように、隠れエントリクラスは幅広い環境に存在する。我々はJPI/JWS同梱クラスの隠れエントリクラスに対処したが、ミドルウェア同梱クラス、ライブラリ同梱クラス、プログラム固有クラスについては、それぞれの提供元が注意を払う必要がある。第4章ではそれらへの対策について説明する。

4. 対策

4.1. 基本的な対策

基本的な対策は、隠れエントリクラスを検出して無効化することである。以下に流れを示す。

(1) **サンドボックス制限を超えるクラスを特定**: 対象のミドルウェア、ライブラリ、プログラムに含まれる JAR ファイルのうち、サンドボックス制限を超える可能性のあるものを全て列挙する。

(2) **全ての隠れエントリクラスを検出**: 列挙されたJARファイルに含まれるクラスを全てチェックする。JWSafe を使って自動的に検出することができる。

(3) **可能ならば隠れエントリクラスを無効化**: 基本的に隠れエントリクラスは不要であり、可能ならば単純に削除すればよい。他のクラスからの参照などの理由で削除できない場合は、非エントリクラスに変更する方法が考えられる。例えば、エントリクラスや main()関数のアクセス範囲を public からデフォルトアクセスに変更すれば、同一パッケージ以外から不可視になるのでエントリクラスとして実行できなくなる。

(4) **さもなければ適切な対策を実施**: どうしても無効化できない隠れエントリクラスや、プログラムの実行に必要な正規エントリクラスについては、様々な攻撃を考慮して適切な対策を施す必要がある。ここでは、第 1 章で述べた再構成攻撃などの高度な攻撃まで考慮する必要があることに注意してほしい。

我々の経験では、多くの隠れエントリクラスは不要であり、削除などにより容易に無効化できると考えられる。隠れエントリクラスは可能な限り無効化し、実際に配布されるものを最小限にすることが重要である。

4.2. JWSafe の対策支援機能

電子署名付きの隠れメインクラスに限っては、JWSafe の対策支援機能を利用することもできる。本機能は、JWS が提供する JNLP ファイルの署名検証機能を利用するものであり、攻撃者による JNLP ファイルの作成を防ぐことで隠れメインクラスを保護する。JNLP ファイルの署名検証機能は JWS 標準の機能であり、本来は特にツールを作成する必要はない。しかし、JWS の署名検証機能はバイパスされる恐れがあるため、本ツールはより安全な方法を提供している。

JNLP ファイルの署名は、署名付き JAR ファイルに JNLP-INF/APPLICATION.JNLP というエントリとして追加することで実現されている。ここでは JNLP エントリとよぶ。JWS はこのエントリと実際に読み込まれた JNLP ファイルを比較し完全性を検証する。もし JNLP エントリが存在しなければ JNLP ファイルへの署名がないものとみなされる。しかし、プログラムが複数の JAR ファイルで構成されていた場合は、JNLP ファイルで最初に指定された JAR ファイルのみがチェックされ、残りの JAR ファイルはチェックされない。通常、JNLP エントリは最初に指定された JAR ファイルにしか含まれないので、JAR ファイルの指定順序を変えることにより、JNLP ファイルへの署名検証をバイパスできる[4]。

この問題に対処するため、JWSafe は以下の機能を提供する。

- **アプレット向け**: 全 JAR ファイルに空の JNLP エントリを追加する。これらの JAR ファイルは JWS アプリケーションとして実行できなくなる。
- **JWS アプリケーション向け**: 指定された JNLP ファイルを全 JAR ファイルに JNLP エントリとして追加する。これにより、指定順序を入れ替えても署名を無効化できない。また、この機能は第 2.1 節で述べた JWS で動作するアプレットにも有効である。

本機能は隠れメインクラスが対象であり、隠れアプレットクラスは保護できない。しかし、第 2.5 節で述べたように隠れメインクラスの脅威はより大きいため、有用な対策支援機能といえる。

4.3. メインクラスの定義の変更

これは JWS の仕様に対する提案である。第 2.5 節で述べたように、隠れメインクラスが存在しやすい理由は、一般的な Java アプリケーションと定義が同じだからである。そこで、定義を変更すれば、隠れアプレットクラスと同程度に少なくなると考えられる。例えば、現状ではメインクラスに必要なメソッドの名前は main()だが、これを jwsmain()などに変更すればよい。ただし、互換性や利便性の問題があるため、実現性については考慮する必要があるだろう。

4.4. JPI のシステムクラスの実行禁止

これは JPI の仕様に対する提案である。第 3.1 節で述べたように、JWS ではシステムクラスの実行が禁止されているが、JPI では禁止されていないため、システムクラスが危険にさらされている。そこで、JPI も JWS と同様にシステムクラスの実行を禁止すればよい。ただし、これも互換性や利便性の考慮は必要である。

5. 関連研究

我々の知る限り、アプレットや JWS アプリケーションが悪用される危険性に関する研究は存在するが、隠れエントリクラスに焦点を当てた研究は存在しない。ただし、注目すべき技術としては、Java Community Process で標準化が進められている JSR277: Java Module System がある[13]。これは Java のプログラムをモジュール化して、配布する際の依存関係の問題などを解消することを目的としている。ここで提案されている JAM(JAva Module)という配布形式では、我々の開発した Cozilet が提供する配布形式と同様に、個々の JAR ファイルを再構成できないようになっており、再構成攻撃の防止に効果がある。よって、この技術が JPI/JWS に採用されれば、隠れエントリクラスへの攻撃は難しくなる可能性がある。ただし、もしこの技術が採用されたとしても、互換性のため既存の JPI/JWS は維持されると考えられるので、隠れエントリクラスの脅威は依然として残る可能性が高いことに注意してほしい。

6. まとめ

本論文では、Java Plug-in および Java Web Start において、アプレットまたはアプリケーションとして攻撃者に悪用される可能性があるにも関わらず、開発者が存在を意識していないクラス、すなわち隠れエントリクラスの危険性を示した。隠れエントリクラスは攻撃への対策が施されていない可能性が高く、実行されるだけで深刻な被害を及ぼす恐れがある。隠れエントリクラスは脅威の大きさと対策の責任者の違いにより、JPI/JWS 同梱クラス、ミドルウェア同梱クラス、ライブラリ同梱クラス、プログラム固有クラスの 4 種類に分類できる。我々は JPI/JWS 同梱クラスの隠れエントリクラスについて複数の脆弱性を発見し、残りのクラスについても調査・対策の手順を示した。今後の課題としては、JSR277 に基づいた、JPI/JWS の実行環境側での対策の検討が考えられる。

謝辞

本件に対応してくださった JPCERT/CC と IPA の脆弱性取り扱いチーム、Sun Microsystems, Inc.の脆弱性対応チーム、そして(株)富士通の脆弱性対応チームの皆様に感謝いたします。

参考文献

[1] Sun Microsystems, Inc., “Java Deployment Overview”, http://java.sun.com/javase/6/docs/technotes/guides/deployment/deployment-guide/overview.html
[2] Sun Microsystems, Inc., “Java 2 Platform Security Architecture”, http://java.sun.com/javase/6/docs/technotes/guides/security/spec/security-spec.doc.html
[3] 兒島 尚、金谷 延幸、「Java アプレット保護ツール Cozilet の Java Web Start 及び署名なしプログラムへの適用」、情報処理学会論文誌、Vol.48、No.9
[4] 兒島 尚、鳥居 悟、「信頼された Java Web Start アプリケーションの悪用に対する一考察」、研究報告「コンピュータセキュリティ」、2006-CSEC-033
[5] Apache, “Apache BCEL”, http://jakarta.apache.org/bcel/
[6] Apache, “Apache Xalan”, http://xalan.apache.org/
[7] Sun BugID 6502030, http://java.sun.com/j2se/1.3/ReleaseNotes.html, http://java.sun.com/j2se/1.4.2/ReleaseNotes.html
[8] 情報セキュリティ早期警戒パートナーシップ、http://jvn.jp/
[9] 情報処理推進機構(IPA) セキュリティセンター、http://www.ipa.go.jp/security/vuln/report/index.html
[10] JVN#04032535, http://jvn.jp/jp/JVN%2304032535/
[11] JVN#44724673, http://jvn.jp/jp/JVN%2344724673/
[12] Java Community Process, “JSR 56: Java Network Launching Protocol and API”, http://jcp.org/en/jsr/detail?id=56
[13] Java Community Process, “JSR 277: Java Module System”, http://jcp.org/en/jsr/detail?id=277