

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

兒島 尚* 鳥居 悟*

* (株)富士通研究所

RIA (Rich Internet Application) とよばれる Web アプリケーションのクライアント側技術が広く利用されているが、信頼されたプログラムが攻撃者に悪用されるという問題がある。我々は過去に、RIA の一つである Java アプレットについて、再構成攻撃という攻撃により信頼された Java アプレットが悪用される危険性を指摘した。この攻撃は開発者の注意だけでは対策が難しい。本論文では、Java アプレットに類似の RIA である Java Web Start について、この攻撃の可能性を考察する。

A Study of Abuse of Trusted Java Web Start Applications

Hisashi Kojima* Satoru Torii*

* Fujitsu Laboratories Ltd.

Client-side technologies of Web applications, called RIA (Rich Internet Application), are widely used, but an attacker may abuse trusted RIA components. In the past, we showed that a Java applet which is a kind of RIA was vulnerable to an attack which we called a malicious recomposition attack. It is difficult for developers to prevent this attack only through their careful design and programming. In this paper, we study a possibility of the attack to Java Web Start which is also a kind of RIA and a similar technology to Java applets.

1. はじめに

近年、RIA (Rich Internet Application) とよばれる技術が広く利用されている。RIA とは、Web アプリケーションにおいて、従来の HTML ベースのユーザーインターフェイスの操作性や表現力を向上させるための技術の総称である。例えば、Java アプレット、Java Web Start[1]、ActiveX コントロールなどであり、最近では Ajax (Asynchronous Javascript And XML) もその一つとして挙げられる。

RIA は自動的にダウンロードされて実行されるという特徴があり、事前に実行環境の準備が必要な場合もあるが、利用者への負担が少ないのが利点である。しかし、安全のため、基本的にプログラムには厳しいアクセス制限を課し、電子署名などにより出所が保証された場合に限りアクセス制限を緩める方式が採られている。特に、電子署名がある場合、実行時に署名者を利用者に提示してアクセス制限の解除を尋ねる方式があり、事前にセキュリティ設定などを変更する必要がないのではしばしば利用されている。

しかし、電子署名されたプログラムは攻撃者に再利用されてその機能を悪用される恐れがある。我々は過去に、署名付き Java アプレットを構成する署名付き JAR ファイルを、攻撃者の用意した JAR ファイルと組み合わせることで、攻撃者が署名付き Java ア

プレットの機能を悪用できることを示した[4]。我々はこの攻撃を再構成攻撃とよぶ。これは Java アプレットだけではなく、再利用可能な署名付き部品を利用した RIA 一般における問題である。また、サーバ側の脆弱性とは異なり、被害の拡大を抑えるのが難しいという特徴がある。なぜなら、サーバ側の脆弱性についてはサーバを停止すれば被害の拡大を防げるが、RIA では攻撃者が入手した署名付き部品を自由に再配布できるからである。再構成攻撃への対策は急務であるといえる。

今回、我々は RIA の一つとして Java Web Start に注目し再構成攻撃の可能性について考察した。Java Web Start は Java アプレットに類似の技術であり、Java アプレットが Web ブラウザ上で HTML ページに埋め込まれて動作するのに対し、Java Web Start アプリケーションはデスクトップアプリケーションとして単体で動作する。本論文ではその考察結果について報告する。

本論文の構成は次の通りである。まず、第 2 章では Java アプレットを例に再構成攻撃について説明する。次に、第 3 章では Java Web Start の概要について説明する。そして第 4 章で Java Web Start に対する再構成攻撃の可能性について考察し、第 5 章で対策の方向性を示す。最後に、第 6 章で関連研究を示し、第 7 章でまとめを述べる。

2. 再構成攻撃

まず再構成攻撃とはどのような攻撃なのかを Java アプレットの場合を例に説明する[4]。

再構成攻撃とは、信頼されたプログラムの部品を攻撃者の部品と組み合わせることで、信頼されたプログラムの機能を悪用する攻撃である。Java アプレットの場合、部品は署名付き JAR ファイルであり、組み合わせ方法を決定するのは HTML ページのアプレットタグである。HTML ページは誰でも用意でき、攻撃者は攻撃対象の署名付き JAR ファイルを入手し、何処かに用意した Web サイト上で自らが作成した JAR ファイルと組み合わせることができる。

ここでは再構成攻撃の典型的な 2 つの手法、特権コードの悪用、クラス置き換えについて説明する。

2.1. 特権コードの悪用

1 つ目は攻撃対象の JAR ファイルをライブラリとして利用し、特権コードを悪用する手法である。攻撃対象の JAR ファイルをライブラリとして利用することは容易である。例えば、次のような署名付きアプレットがあったとする。

```
<applet code="app.PurchaseApplet"
archive="purchase.jar" ...
```

ここで、攻撃者は攻撃対象の JAR ファイル purchase.jar と自らの JAR ファイル evil.jar を次のように組み合わせれば、信頼されたクラスのメソッドやフィールドにアクセスできる。(EvilApplet は攻撃者のクラス)

```
<applet code="EvilApplet"
archive="evil.jar,purchase.jar" ...
```

ただし、Java にはスタック検査による呼び出し元のチェック機能があり、攻撃者のクラスから信頼されたクラスのメソッドを単純に呼び出しただけでは、このチェック機能によって拒否されてしまう[4]。

そこで攻撃者は特権コードを狙う。特権コードとは前述のチェック機能をバイパスするための仕組みであり、信頼されたプログラムの機能を信頼されていないプログラムにも限定的に利用させるために利用される。以下に例を示す。

```
public String loadConf() {
    Reader r = // 特権コード開始
    (Reader)AccessController.doPrivileged(
        new PrivilegedAction() {
            public Object run() {
                try {
                    // ファイルダイアログを開く
                    :
                    // 選択されたファイルを開く
```

```
        return new FileReader(name);
    } catch (Exception e) {
        e.printStackTrace();
        return null;
    } } } ; // 特権コード終了
// ファイルを読み込み文字列として返す
:
}
```

loadConf() はファイルダイアログをユーザーに表示し、選択されたファイルを読み込んで文字列として返すメソッドである。ファイル読み込みという危険な操作を実行する箇所が特権コードになっている。この例ではファイル名はユーザーに選択されるが、もしファイル名がメソッドの引数で与えられていた場合は、このメソッドを呼び出せば誰でも任意のファイルを読み込めることになる。

特権コードの安全性はコードの開発者に依存しており、前述のような脆弱性のある特権コードを実装してしまう恐れがある。攻撃者はそのような脆弱な特権コードを探し出して悪用するのである。

2.2. クラス置き換え

2 つ目は攻撃対象の一部のクラスを攻撃者のクラスに置き換える手法である。前述の特権コードを悪用する方法では、対象のプログラムが特権コードを実装している必要があるが、多くのプログラムでは特権コードを実装していないので、悪用の可能性はそれほど高くはない。しかし、ここで説明する手法は特権コードを必要とせず、より成功する可能性が高い。

この手法はアプレットのクラスのロード方法を利用する。アプレットでは、複数の JAR ファイルに同じパスのファイルが存在する場合、archive 属性で先に指定された JAR ファイルを優先する。これを利用して、攻撃者は攻撃対象の JAR ファイルに含まれるファイルと同じパスのファイルを作成して、JAR ファイルに含めて archive 属性で先に指定することで、ロードされるファイルを置き換えることができる。

ここで、2.1 節の purchase.jar が次のような構成だったとする。

```
purchase.jar
+- app/PurchaseApplet.class
+- util/MyFileDialog.class
```

ここで攻撃者は MyFileDialog.class の置き換え用のクラスファイルを作成し、evil.jar に含める。

```
evil.jar
+- util/MyFileDialog.class
```

そして、次のように evil.jar を purchase.jar より先に指定する。

```
<applet code="app.PurchaseApplet"
archive="evil.jar,purchase.jar" ...
```

これで `MyFileDialog.class` は攻撃者のものに置き換わって実行される。ただし、置き換わったクラスは署名付きクラスとしては扱われないので、そのクラスから直接危険な動作を行うことはスタック検査によって禁止される。そこで、攻撃者はユーティリティクラスなどの、直接危険な動作は行わないが、様々な動作に影響を与えるようなクラスを置き換える。そうした構成のプログラムは多いと考えられ、攻撃が成功する可能性は高いといえる。

ただし、Java には *same package same signer* とよばれる保護機能があり[4]、あるパッケージのクラスは全て同一の署名者に署名されている必要がある。よって、対象の署名付きアプレットが単一のパッケージで構成されている場合は置き換えできない。しかし、複数のパッケージで構成されている場合、全てのパッケージが同一の署名者である必要はないので、ユーティリティクラスが含まれるパッケージだけ置き換えることが可能である。また、この保護機能はクラスのみが対象でリソースは対象外である。よって、この保護機能の下でも攻撃は十分に成功するといえる。

以上のように、再構成攻撃には特権コードの悪用とクラス置き換えの 2 種類の手法があり、特にクラス置き換えの脅威は大きいといえる。この攻撃への対策は開発者の注意だけでは難しいと考えられ、我々は[4]において対策技術を提案し実装している。

3. Java Web Start の概要

Java Web Start に対する再構成攻撃について考察する前に、まず Java Web Start の概要について説明する。Java Web Start[1]は Java 実行環境で動作する RIA プラットフォームの一つである。Java アプレットと同様に、Java Web Start アプリケーション(以降、JWS アプリケーションとよぶ)が配備された Web サイトに利用者がアクセスすると、自動的にダウンロードされて実行される。Java アプレットが Web ブラウザ上で HTML ページに埋め込まれて実行されるが、JWS アプリケーションは単体でデスクトップアプリケーションとして実行される。

3.1. JWS アプリケーションの構成

JWS アプリケーションは主に JAR ファイルによって構成される。JAR ファイルにはクラスファイルや、画像や音声などのリソースファイルを含めることができる。JAR ファイルの構成や最初に行うべきクラスなどのアプリケーションの実行に必要な情報は、JNLP ファイルというファイルに記述される[2]。

JNLP ファイルは XML 形式のファイルであり、

JAR ファイルなどと共に Web サイトに配備される。JNLP ファイルは大きく分けると、名前や作成者などの情報(<information>)、セキュリティの情報(<security>)、必要な JAR ファイルなどの情報(<resources>)、そして JWS アプリケーションの性質の情報(<application-desc>)などで構成される。以下に例を示す。

```
----- purchase.jnlp -----
<?xml version="1.0" encoding="UTF-8"?>
<jnlp codebase="...">
<information>
  <title>MyApp</title>
  <vendor>Hisashi</vendor>
</information>
<security><all-permissions/></security>
<resources>
  <j2se version="1.5"/>
  <jar href="purchase.jar"/>
</resources>
<application-desc
  main-class="app.PurchaseApp"/>
</jnlp>
```

上記の例では、これがアプリケーション型で、`MyApp` という名前を持ち、`Hisashi` に作成され、完全なアクセス権を必要とし、J2SE のバージョン 1.5 を対象とし、`purchase.jar` という単一の JAR ファイルによって構成され、`app.PurchaseApp` というクラスが最初に行うべきメインクラスであることを示している。以降では簡単のため<information>と<j2se>は省略するものとする。

3.2. アプリケーションとエクステンション

JWS アプリケーションには大きく分けてアプリケーション型とエクステンション型の 2 種類の性質がある。性質は排他的であり、一つの JWS アプリケーションは一つの性質しか持つことができない。

アプリケーション型は単体で実行可能である。JNLP ファイルで<application-desc>を記述するとデスクトップアプリケーションとして、<applet-desc>を記述するとアプレットとして扱われる。

一方、エクステンション型は基本的に単体では実行されず、アプリケーション型からの利用を意図したものである。JNLP ファイルで<component-desc>を記述すると単純なライブラリとして扱われ、<installer-desc>を記述するとインストーラ付きのライブラリとして扱われる。

アプリケーションからエクステンションを参照する場合には、<resources>に<extension>を記述する。エクステンションのセキュリティコンテキストはアプリケーションと独立しており、例えばアプリケーションには署名するがエクステンションには署名しないとい

ったことが可能である。

3.3. Java Web Start のセキュリティ

Java Web Start のセキュリティは Java 実行環境のアクセス制御機構にしたがっており、出所が信頼できない JWS アプリケーションには基本的に厳しいアクセス制限が課せられる。アクセス制限を緩めるには次の 3 通りの方法がある。

- 1) 利用者のマシンにあるポリシーファイル(アクセス制御リスト)を変更する。
- 2) 実行環境に標準で用意されている JNLP サービスを呼び出し利用者に許可を求める。
- 3) JWS アプリケーションに電子署名し、Web サイトに配備された JNLP ファイルに必要なアクセス権を記述する。

1)は利用者の設定を予め変更しておく必要があるが、2)及び 3)は実行時に利用者に許可を求めることができる。ここではよく利用される 2)及び 3)について説明する。

2)の方法では、JWS アプリケーションは予め定義された API を利用して JNLP サービスとよばれる機能呼び出せる。これは JWS アプリケーションの出所に関わらず利用できる機能で、ファイルダイアログを表示して利用者が選択したファイルを読み書きするサービスや、印刷サービスなどが用意されている。サービスが呼び出されると、利用者に許可を求めるダイアログが表示される。

3)の方法では、JWS アプリケーションの署名の正当性が検証されると、署名者の情報と要求されているアクセス権が利用者に表示され、実行の許可を求める。利用者が許可すると要求されたアクセス権が与えられて実行される。アクセス権には完全なアクセス権(<all-permissions/>)と部分的なアクセス権(<j2ee-application-client-permissions/>)の 2 種類があり、JNLP ファイルに<security>を記述することで指定される。

既に述べたように 2)は JWS アプリケーションの出所に関わらず利用可能であり、悪用の危険を考慮すべきなのは 3)の方法を採用している場合である。よって、本論文では 3)の方法を採用する場合を前提とする。

4. Java Web Start に対する再構成攻撃

ここまで、再構成攻撃とは何か、Java Web Start とは何かについて説明してきた。ここでは Java Web Start に対する再構成攻撃がどのように達成できるのかを示す。攻撃手法は Java アプレットの場合と基本的には変わらないが、署名の方法が異なり、開発者が望めば JNLP ファイルの改ざんを署名で保護できる点が大きな違いである。ここではまず

JNLP ファイルに署名がない場合の攻撃について説明し、次いで JNLP ファイルに署名がある場合でも攻撃が成功するかどうかを示す。ここでは Java Web Start 1.5.0_06 を対象環境としている。

4.1. JNLP ファイルが署名されていない場合

まず、JNLP ファイルが署名されていない場合について説明する。既に述べたように JNLP ファイルへの署名は必須ではないので、多くの JWS アプリケーションが該当すると考えられる。

Java Web Start の場合でも、攻撃手法は Java アプレットの場合と基本的には変わらない。最も単純な方法は、Java アプレットで HTML ページを用意すると同様に、攻撃対象の JAR ファイルと攻撃者の JAR ファイルを組み合わせるための JNLP ファイルを用意することである。以下は 2.1 節で述べたような特権コードを悪用する例である。

```
----- evil.jnlp -----
<?xml version="1.0" encoding="UTF-8"?>
<jnlp codebase="...">
<security><all-permissions/></security>
<resources>
  <jar href="evil.jar"/>
  <jar href="purchase.jar"/>
</resources>
<application-desc main-class="EvilApp"/>
</jnlp>
```

しかし、Java アプレットと異なり、Java Web Start で署名付きアプリケーションと扱われるためには、JNLP ファイルから<jar>で参照された JAR ファイルが全て同一の署名者に署名されている必要がある。上記の例では署名付きではない攻撃者の JAR ファイル evil.jar が混在しており、署名付きとして扱われないため攻撃は成功しない。

そこで、攻撃者は<extension>を利用する。3.2 節で述べたように、エクステンションとして参照する場合、セキュリティコンテキストは独立して扱われる。以下に例を示す。

```
----- evil.jnlp -----
<?xml version="1.0" encoding="UTF-8"?>
<jnlp codebase="...">
<resources>
  <jar href="evil.jar"/>
  <extension href="purchase.jnlp"/>
</resources>
<application-desc main-class="EvilApp"/>
</jnlp>

----- purchase.jnlp -----
<?xml version="1.0" encoding="UTF-8"?>
<jnlp codebase="...">
<security><all-permissions/></security>
<resources>
  <jar href="purchase.jar"/>
```

```
</resources>
<component-desc/>
</jnlp>
```

evil.jnlp、purchase.jnlp のいずれも攻撃者が用意したものである。evil.jnlp はアプリケーション型として、purchase.jnlp はエクステンション型として定義されており、evil.jnlp はエクステンションとして purchase.jnlp を参照している。purchase.jnlp は署名付きの JAR ファイルで構成されており、署名付きとして扱われるので攻撃者に悪用の機会がある。

また、2.2 節で述べたクラス置き換え攻撃を行うには次のようにすればよい。

```
----- purchase.jnlp -----
<?xml version="1.0" encoding="UTF-8"?>
<jnlp codebase="...">
<security><all-permissions/></security>
<resources>
  <extension href="evil.jnlp"/>
  <jar href="purchase.jar"/>
</resources>
<application-desc
  main-class="app.PurchaseApp"/>
</jnlp>

----- evil.jnlp -----
<?xml version="1.0" encoding="UTF-8"?>
<jnlp codebase="...">
<resources>
  <jar href="evil.jar"/>
</resources>
<component-desc/>
</jnlp>
```

先の例とは逆に、purchase.jnlp がアプリケーション型で、evil.jnlp がエクステンション型になっている。purchase.jnlp から evil.jnlp をエクステンションとして参照しているが、purchase.jar よりも先に参照されていることに注意してほしい。こうすることで evil.jar が purchase.jar よりも優先され、クラスやリソースを置き換えることができる。

以上のように、JNLP ファイルが署名されていない場合は、特権コードの悪用、クラス置き換えのいずれの攻撃も可能である。

4.2. JNLP ファイルが署名されている場合

JNLP ファイルが署名されている場合、JNLP ファイルが改ざんされていると JWS アプリケーションが実行されないで、前述の攻撃は難しいということになる。しかし、Java Web Start における JNLP ファイルの署名の検証方法では、JNLP ファイルの署名を無効化できる場合がある。

JNLP ファイルへの署名検証は、署名付き JAR ファイルに JNLP-INF/APPLICATION.JNLP という名前

のエントリを格納し、実際の JNLP ファイルと比較することで行われる。エントリがなければ JNLP ファイルは署名されていないとみなす。このエントリは以下の手順で検索される。

- 1) アプリケーション型ならば、メインクラスを含む JAR ファイルが存在すればそのファイルから探す。
- 2) メインクラスを含む JAR ファイルが存在しない、もしくはエクステンション型の場合は、JNLP ファイルで最初に指定された JAR ファイルから探す。

この方法によれば、攻撃対象の JWS アプリケーションが単一の JAR ファイルだけで構成されている場合は、JNLP ファイルの署名を無効化できない。しかし、複数の JAR ファイルで構成されている場合、攻撃者はエクステンション型として定義された JNLP ファイルを用意し、JNLP-INF/APPLICATION.JNLP を含まない JAR ファイルを最初に指定することにより、JNLP ファイルへの署名を無効化できる。

以下に例を示す。まず、アプリケーション型として定義され、2 つの JAR ファイルで構成された署名付き JWS アプリケーションがあるとする。JNLP-INF/APPLICATION.JNLP は最初に指定されている purchase1.jar にのみ格納されている。

```
----- purchase.jnlp -----
<?xml version="1.0" encoding="UTF-8"?>
<jnlp codebase="...">
<security><all-permissions/></security>
<resources>
  <jar href="purchase1.jar"/>
  <jar href="purchase2.jar"/>
</resources>
<application-desc
  main-class="app.PurchaseApp"/>
</jnlp>
```

ここで、攻撃者は purchase.jnlp を変更してエクステンション型にし、purchase2.jar を purchase1.jar より先に指定する。

```
----- purchase.jnlp -----
<?xml version="1.0" encoding="UTF-8"?>
<jnlp codebase="...">
<security><all-permissions/></security>
<resources>
  <jar href="purchase2.jar"/>
  <jar href="purchase1.jar"/>
</resources>
<component-desc/>
</jnlp>
```

すると、最初に指定された purchase2.jar については JNLP-INF/APPLICATION.JNLP の検索が行

われるが、`purchase1.jar` については検索が行われない。よって、JWSアプリケーション自体の署名は有効なままだが、`purchase.jnlp` は署名されていないものとして扱われる。そしてアプリケーション型のものをエクステンション型として参照することで、特権コードを悪用するなどの攻撃が可能になる。

ただし、クラス置き換え攻撃はできない。この攻撃を達成するためには署名付きクラスをメインクラスとして実行する必要があり、2種類の方法が考えられるが、Java Web Start の仕様により JWS アプリケーションが実行されないからである。1つ目は、`purchase.jnlp` をアプリケーション型として再定義する方法である。しかし、アプリケーション型の場合、`JNLP-INF/APPLICATION.JNLP` はメインクラスのある JAR ファイルから検索されるので、参照する順番を変えても署名は無効化されず、この方法は成功しない。2つ目は、`purchase.jnlp` をエクステンション型として再定義してメインクラスをそこからロードする方法である。しかし、メインクラスは`<jar>`により直接 JNLP ファイルで参照された JAR ファイルに存在しなければならないので、これも成功しない。

以上で述べたように、JNLP ファイルが署名されていても、複数の JAR ファイルで構成されている場合は、エクステンション型に再定義することで特権コードを悪用する攻撃が可能である。もちろん、JWS アプリケーションがそもそもエクステンション型であればそのまま攻撃できる。ただし、クラス置き換え攻撃はできない。

5. 対策の方向性

4章で Java Web Start に対する再構成攻撃の可能性について検討し、JWS アプリケーションの種類によっては攻撃が可能であることを示した。ここでは対策の方向性について考えてみる。

まず、開発者側で実践可能な対策について考える。第一の対策は、JNLP ファイルに署名することである。これによってクラス置き換え攻撃を防ぐことができる。しかし、既に述べたように、JWS アプリケーションが複数の JAR ファイルで構成されている場合は JNLP ファイルの署名は無効化される恐れがある。そこで第二の対策として、全ての JAR ファイルに `JNLP-INF/APPLICATION.JNLP` を含めるようにする。これによって JAR ファイルの順番に関わらず JNLP ファイルの署名が検証され、攻撃者は JNLP ファイルを変更できなくなる。

一方、Java Web Start の実行環境側で考えられる対策は、JWS アプリケーションの構成の変更である。現状の構成では JNLP ファイルの署名の有無が判定しにくい。そこで、JNLP ファイルや JAR ファイル群など、必要なファイルを単一の JAR ファイルに格納し、まとめて署名するような構成が考えられる。

そしてこの JAR ファイルだけを配備するようにすれば、攻撃者は再構成できない。

ただし、ここでの対策は JNLP ファイルを保護するものであり、JAR ファイル自身の悪用を保護するわけではないことに注意してほしい。JAR ファイルは Java 実行環境で共通の形式であり、Java アプレットの一部として悪用されることも考えられる。例えば、特定の JWS アプリケーションでのみ JAR ファイルが有効になるよう強制する仕組みがあった方がより安全になるだろう。

6. 関連研究

ここで関連研究について述べておく。Java Web Start の安全性については[2]で議論されているが、本論文で述べたような再構成攻撃については議論されていない。また、Java Web Start の実行環境自体の脆弱性はこれまでに幾つか指摘されているが[3]、再構成攻撃が原因の脆弱性は指摘されていない。このように Java Web Start に対する再構成攻撃は現状ではあまり議論されていないが、無視できない問題であるというのが我々の主張である。

7. まとめ

本論文では Java Web Start に対する再構成攻撃の可能性について考察した。再構成攻撃とは、信頼されたプログラムの部品を攻撃者が自らの部品と組み合わせる悪用する攻撃である。この問題は事後対策が難しいという特徴があり、事前対策が欠かせない。我々はまず、特権コードの悪用、クラス置き換えという2種類の攻撃手法を紹介し、JWS アプリケーションの種類によっては攻撃が可能であることを示した。そして、開発者側、実行環境側で実践可能な対策の方向性について検討した。

今後の課題としては、実行環境側での対策の実現方法の検討、JAR ファイルの JWS アプリケーションへの束縛手法の検討などが挙げられる。

参考文献

- [1] Sun Microsystems, Inc., "Java Web Start", <http://java.sun.com/j2se/1.5.0/ja/docs/ja/guide/javaws/index.html>
- [2] "JSR 56: Java Network Launching Protocol and API", <http://jcp.org/en/jsr/detail?id=56>
- [3] Sun Alert Notification 57740, 101748, 102170, etc
- [4] Hisashi Kojima, Ikuya Morikawa, Yuko Nakayama, and Yuji Yamaoka, "Cozilet: Transparent Encapsulation to Prevent Abuse of Trusted Applets", ACSAC'04, Tucson, December 2004.