



RIA 的安全挑战 1

通过沙箱实现安全保护 2

安全发展 5

安全性：共同的责任 7

Adobe® AIR™ HTML 安全模型建立在开发人员在开发 HTML 和 Ajax 应用程序过程中的反馈和对固有安全问题不断深入了解的基础之上。与基于 Web 的 RIA 所不同的是 AIR 应用程序提供了对台式机的访问，因此为 AIR 开发丰富的 Internet 应用程序提出了新的挑战。这意味着 AIR 应用程序可以读取和写入文件，绘制到屏幕，以及与网络进行通信等。

安全性是 Adobe、用户、系统管理员和应用程序开发人员关注的焦点。因此，Adobe AIR 包含一组安全性规则和控制，以保护用户和应用程序开发人员的利益。本白皮书介绍了开发基于 HTML 的 AIR 应用程序过程中的一些安全注意事项。

有关 AIR 安全性的详细信息，请参阅《Adobe AIR 安全性》白皮书，发布于 http://www.adobe.com/go/learn_air_security_wp_cn。

RIA 的安全挑战

无论是 HTML 桌面应用程序还是 HTML Web 应用程序，它们所面临的一个主要威胁是会导致恶意代码执行的注入攻击，如跨站点脚本攻击 (XSS)。代码通常会通过一些常见矢量来注入，例如：URL 处理 (javascript: 和其它危险方案)，eval() 和为 DOM 元素 (如 innerHTML 和 outerHTML) 分配外部 HTML 内容。如果加载操作通过 HTTP 执行，恶意代码还可能注入到可信赖的服务 (如 Google 地图) 中。

如果开发人员认为应用程序使用的内容无害，而事实上这些内容却包含恶意代码，那么这些漏洞就会暴露出来。例如，某个基于 HTML 的 AIR 应用程序从在线博客提取注释并将其显示出来，它可能会允许用户编写如下注释：

```
<a href="#" onclick="var f=new air.File('c:\\something.txt'); f.deleteFile();">I'm cool!</a>
```

如果用户单击此链接，文件可能会被删除。损坏可能远不止于此，安全漏洞在基于浏览器的 HTML 和 Ajax 应用程序中是很常见的。

一种假设为，通过 JavaScript 对象表示法 (JSON) 加载的恶意内容所带来的损坏可以受到现有域沙箱限制的限制，这种假设是不正确的。细粒度 DOM 沙箱边界 (如 frame 或 iframe) 中的所有数据都在此沙箱中运行，无论其来源如何。即使是粒度最小的帧定义的 DOM 沙箱边界，也只能在内容被置于不同的原始域时成功运行。此外，常见的设计和实现模式会促进使用这些危险的做法。

对于传统的 Ajax 应用程序，由于浏览器中运行的应用程序经过沙箱处理，因此在某种程度上是可接受的。浏览器应用程序与用户的计算机隔离且受同源策略的约束，具有独特的对称性。

有权访问功能丰富的 API 的丰富 Internet 应用程序提出了独特的挑战。即使一个简单的应用程序，如果编码欠佳，也可能包含将导致严重损坏用户计算机的注入弱点。对于任何 RIA 平台，恶意攻击可能造成的损坏与其运行时提供的值成正比。RIA 平台中 API 的功能越强大、越丰富，风险就越大。

有关与这些问题相关的信息和资源，请参阅：

- <http://www.fortifysoftware.com>
- <http://arstechnica.com>
- <http://www.openajax.org>

通过沙箱实现安全保护

为降低这些风险，AIR 现在提供了一个全面的安全体系结构，用于定义 AIR 应用程序中每个内部文件和外部文件的权限。根据文件的来源授予其相应的权限，并将权限分配到称为沙箱的逻辑安全组中。运行时使用这些沙箱来定义对数据的访问和可执行操作：

AIR API 为安全地将不断发展的浏览器安全模型与桌面风格的安全模型桥接在一起提供了一种安全方式。较之典型的基于浏览器的应用程序，AIR 沙箱桥模型功能更广泛、风险更小。

- 应用程序沙箱中的代码可以访问 AIR API，在页面加载后动态生成代码的能力受限，并且无法通过 `script` 标签加载内容。
- 所有其它沙箱（称为非应用程序沙箱）中的代码可以对本地受信任 HTML 从本质上模拟典型的浏览器的限制和行为。

应用程序沙箱

应用程序沙箱的主要功能是支持 AIR API 与受信任目录中的文件之间的交互。隔离非应用程序沙箱中的代码是通过在加载初始页面后禁用 JavaScript 分析器来维护的，因此 `eval()` 这样的代码会引发异常。允许设置 `innerHTML`，但是其中包含的任何脚本都将被忽略。这样无需将 AIR API 直接向外部加载的数据公开即可提供广泛的功能。

开发人员可以使用 `app:/` URL 架构引用应用程序沙箱中受信任目录。甚至可以将应用程序目录中的一些文件放置到非应用程序沙箱中，前提是：使用 `frame` 或 `iframe` 的 `AIR sandboxRoot` 和 `documentRoot` 属性将这些文件加载到 `frame` 或 `iframe` 中。AIR 安装程序中包括的所有文件都会自动安装到 `app:/` 下，从而成为应用程序沙箱的一部分。应用程序运行时，这些文件会被授予一整套 AIR 应用程序权限，包括与本地文件系统交互。`app:/` 外部的内容并非应用程序沙箱的组成部分，因此应按浏览器中的内容对待。

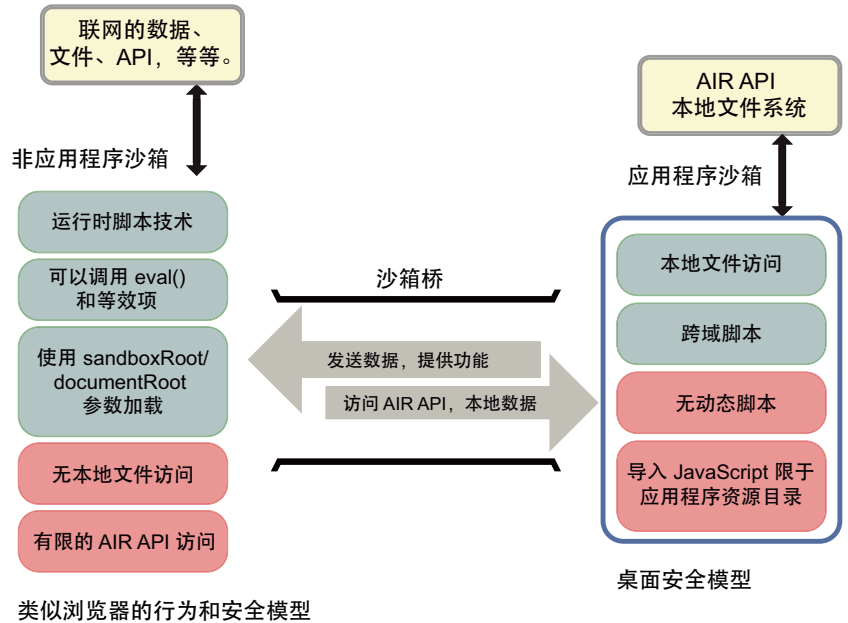
非应用程序沙箱

非应用程序沙箱的主要功能是限制对本地文件和 AIR API 的访问，只有得到开发人员的明确许可才能对其进行访问。从网络或 Internet 位置加载的内容将自动分配给此沙箱。非应用程序沙箱可以使用 `eval()` 函数，通过 `innerHTML` 赋值执行代码等。非应用程序沙箱中的代码受同源策略的约束，默认情况下，无法通过 `XMLHttpRequest` 执行跨域数据加载。但是，非应用程序沙箱中的代码无法直接执行 AIR API。

鼓励开发人员编写在应用程序沙箱中运行的代码，即使对此沙箱的一些限制使其行为方式与浏览器模型相比略有不同。

沙箱桥

最简单的情况是，使用网络内容的基于 HTML 的 AIR 应用程序包含根用户界面 (UI) 文件、应用程序下载的数据和文件，以及一些应用程序文件（在应用程序沙箱中）。但是，如果外部内容和一些 UI 文件在非应用程序沙箱中运行，应用程序文件在应用程序沙箱中运行，那么非应用程序沙箱内容与应用程序沙箱内容之间将如何进行交互？



AIR 解决方案为沙箱桥 API，可实现不同沙箱之间的间接通信。开发人员可以编写调用 AIR API 的函数，然后在沙箱桥上公开这些“中间”函数。这些桥函数实质上“位于桥上”，等待非应用程序沙箱中的代码调用它们。

此体系结构实现了两个目标。首先，非应用程序沙箱内容不能直接访问 AIR API，应用程序内容得到保护，防止未知和可能存在恶意的外部数据访问。其次，只有那些开发人员明确选定通过桥机制公开的 API 才能公开。

虽然沙箱桥并不能保证安全性，但是开发人员可以通过控制向不受信任的内容公开 AIR API 的方式来降低风险。这非常类似于客户端—服务器模型，服务器永远不应信任客户端，而客户端永远不应信任从第三方加载的数据。同样，开发人员不应使用或信任服务器上加载的代码，这些代码可能会危及服务器的安全，进而获得对操作系统的访问权。因此，对于浏览器和 AIR 沙箱桥中的客户端—服务器模型，应始终保护原始系统 API（如 `writeFile()` 和 `readFile()`）以防受到攻击。

开发人员必须显式指出桥接沙箱。

Adobe AIR 安全性在设计上考虑到了 Web 应用程序开发社区。Adobe 支持有助于降低安全风险的标准和方法。包括 OpenAjax Alliance 在内的多个组织正进一步帮助宣扬这些最佳做法：

- 隔离 frame 或 iframe 中不受信任的内容，利用同源策略使攻击者难以访问整个 DOM 树。
- 对于加载到不同域的帧中的数据，应给定一个唯一的 JavaScript 执行上下文和 DOM 树。
- 不要动态生成和执行代码。
- 只可插入受信任的来源的 HTML 内容。
- 安全地使用 JSON。

AIR 为开发人员提供了一些遵循这些最佳做法的机制，从而增强了基于 HTML 的应用程序的安全性。有关详细信息，请参阅 <http://www.openajax.org/whitepapers/Ajax%20and%20Mashup%20Security.html>。

沙箱权限摘要

功能	应用程序沙箱	其它 (非应用程序) 沙箱
是否能直接访问 AIR API?	是	否
是否能通过桥访问使用 AIR API 的应用程序沙箱功能?	不可用	是
是否能加载 <code><script src='http://www.example.com/some_code.js'></code> 这样的远程脚本?	否	是
默认情况下，是否能执行跨域请求 (XHR)	是	否
是否支持在 load 事件之后将字符串作为代码动态加载： eval() 函数，setTimeout('string', millis)， javascript: URL，通过 innerHTML 插入的元素 (如 onclick='myClick()') 上的属性处理函数等。	否	是
Ajax 框架是否能不经更改而发挥作用?	一些框架	是

安全发展

AIR 安全模型旨在向后与旧式 Ajax 模型兼容，同时降低向不受开发人员控制的数据公开系统 API 所带来的风险。在浏览器客户端—服务器模型中，浏览器负责处理大量演示和一些动态第三方数据加载。服务器处理大多数敏感操作并提供操作系统风格的设施，如存储、密码系统等。通过提供两个通过间接公开的函数通信的沙箱，Adobe AIR 减小了攻击者访问最终用户计算机和数据的几率。

利用 Ajax 框架

由于非应用程序沙箱是有效的 HTML 浏览器沙箱，因此 Ajax 开发人员会发现将现有应用程序和模式移植到 AIR 相对简单。例如，您可以从 Web 提取现有的填字游戏，然后将其整个插入到非应用程序沙箱帧中。

运行在非应用程序沙箱中的 Ajax 框架应按照其在 Web 浏览器中的运行方式运行，而一些依赖 `eval()` 和跨域代码加载的框架版本可能无法完全发挥作用。很大程度上取决于给定框架对 `eval()` 和类似行为的依赖程度，以及开发人员使用框架的哪些部分。

这些限制不会阻止将 `eval()` 与 JSON 对象文本一起使用。这样便可以将应用程序内容与 JSON JavaScript 库一起使用。但是，会限制应用程序沙箱内容使用重载的 JSON 代码（通过事件处理函数）。对于其它 Ajax 框架和 JavaScript 代码库，应检查框架或库中的代码在动态生成的代码受限制时是否起作用。如果不起作用，则需要非应用程序安全沙箱中包括使用框架或库的所有内容。Adobe 维护着一个文档，其中列出了 AIR 应用程序沙箱的已知 Ajax 框架支持：

http://www.adobe.com/go/airappsandboxframeworks_cn

迁移应用程序

如果要提取简单的填字游戏并实现 `saveGame()` 和 `loadGame()` 这样的函数，应如何操作？要完成此操作需要访问 AIR API，您可以在应用程序沙箱中实现相关的 API。然后，应用程序沙箱中的函数和数据将通过沙箱桥向非应用程序沙箱公开。概括地说，此流程包括以下步骤：

1. 将原始根内容文件重命名为 `someName.htm`。
2. 创建一个名为 `myRoot.htm` 的新文件，并添加指向 `someName.htm` 的 `IFRAME` 或 `FRAME`。
3. 编写调用 AIR API 的函数。不直接调用 API 会更安全。
4. 创建用于向非应用程序沙箱公开应用程序沙箱功能的 `parentSandboxBridge`。

当然，两个沙箱之间的通信可以是双向的：`parentSandboxBridge` 和 `childSandboxBridge` 支持从两个方向遍历桥。这让开发人员能够更好地控制应用程序和应用程序环境之间的内容流。

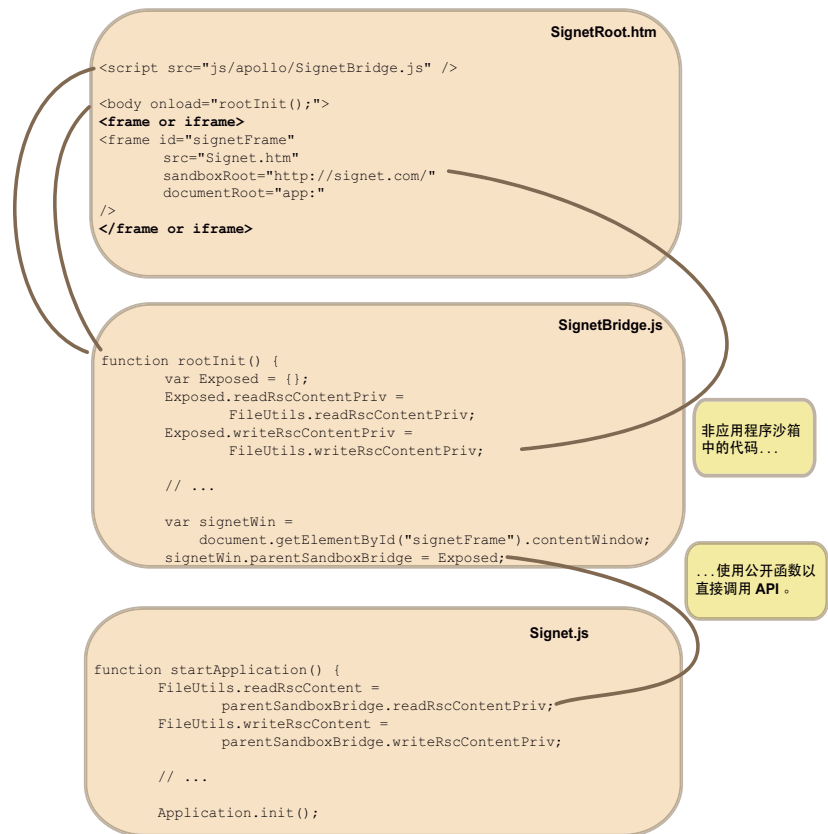
实际使用案例

Signet（面向 del.icio.us 用户的书签管理程序）是一款基于 HTML 的 AIR 应用程序，它通过以下方式实现 AIR 安全模型：

1. 首先，创建 SignetRoot.htm，让其包含 id 为 signetFrame 的帧和三个标识可用域的必需属性。
2. 在 SignetRoot.htm 中，onload 调用 SignetBridge.js，后者包含所有公开的函数。
3. 在 SignetBridge.js 中，通过 parentSandboxBridge 向 signetFrame 公开这些函数。
4. 在 Signet.js 中，公开的函数可进行直接 AIR API 调用。例如，writeRscContentPriv 调用 writeRscContent。

虽然此方法比较简单，但其功效非常强大：可以为两个沙箱定义远程和本地域，在装载过程中调用桥，借助中间函数和桥 API 间接练习 API。

有关最佳做法的列表，请参考 AIR 开发人员文档。其中，这些最佳做法包括仅在绝对必要时才使用外部文件，不将来自网络的数据用于某些操作等。



安全性：共同的责任

最后，RIA 安全性与应用程序生命周期中涉及到的每个人息息相关。最终用户相信开发人员能够遵循最佳做法和标准。

除了推荐的开发方法之外，AIR 应用程序的安全性从在用户计算机上安装开始：

- 无法修改初始安装工作流程。
- 多个应用程序共享流畅而一致的安装过程，该过程在所有 Web 浏览器和操作系统中都是一致的。
- 安装由运行时管理，安装的应用程序无法对其进行操作。
- 必须对安装程序文件进行数字签名，以使用户可以验证代码的来源和确定应用程序的访问权限。
- 运行时提供内存管理，可以最大程度地减少缓冲区溢出和内存损坏之类的漏洞。

虽然对任何安全问题作出响应最终会增加开发人员的负担，但是，AIR 安全模型尝试减小编写和维护安全代码的复杂性。