ADOBE® AIR® Security

Contents

AIR Security Overview

Security is a key concern of Adobe, users, system administrators, and application developers. For this reason, Adobe® AIR® includes a set of security rules and controls to safeguard the user and application developer. This white paper presents the security considerations in using and developing applications for Adobe AIR.

Although the AIR security model is an evolution of the security model for SWF content running in Flash® Player and HTML content running in the browser, the security contract is different from the security contract applied to content in a browser. This contract offers developers a secure means of broader functionality for rich experiences with freedoms that would be inappropriate for a browser-based application.

AIR applications run under the same operating system security constraints of other, native applications on a given computing device. In general, these constraints allow for broad access to operating system capabilities such as reading and writing files, drawing to the screen, and communicating with the network. Operating system restrictions that apply to native applications, such as user-specific privileges, equally apply to AIR applications.

AIR applications are written using either compiled bytecode (SWF content) or interpreted script (JavaScript, HTML) so that memory management is provided by the runtime. This minimizes the chances of AIR applications being affected by vulnerabilities related to memory management, such as buffer overflows and memory corruption. These are some of the most common vulnerabilities affecting desktop applications written in native code.

***Note:** This white paper discusses security-related issues in Adobe AIR. The following developer documentation provides technical details on developing secure AIR applications and considerations in using the AIR APIs:*

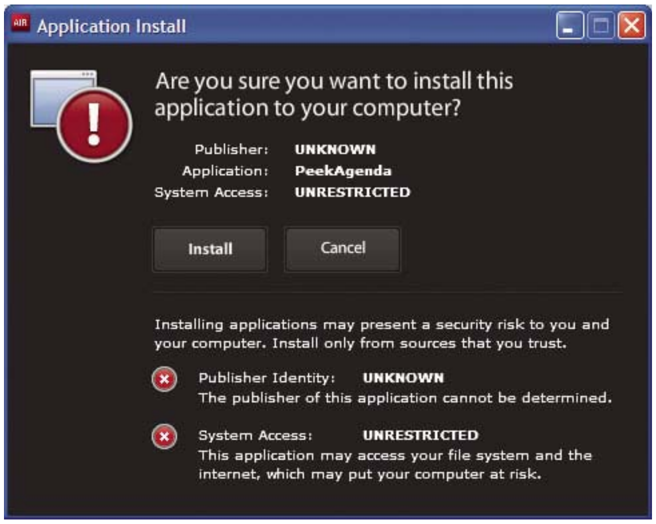
- For ActionScript (Flash and Flex) developers, see AIR Security in the ActionScript 3.0 Developer's Guide
- For Ajax developers, see AIR Security in the HTML Developer's Guide for Adobe AIR

Installing and updating desktop applications

Desktop AIR applications can be distributed via AIR installer files which use the `air` extension. When Adobe AIR is installed and an AIR file is opened, the runtime administers and manages the application installation process.

***Note:** Developers can specify a version, and application name, and a publisher source, but the initial application installation workflow itself cannot be modified. This restriction is advantageous for users because all AIR applications share a secure, streamlined, and consistent installation procedure administered by Adobe AIR. If application customization is necessary, it can be provided when the application is first executed.*

The default application installer provides the user with security-related information. AIR displays the publisher name during installation when the AIR application has been signed with a certificate that is trusted, or which chains to a certificate that is trusted on the installation computer. Otherwise the publisher name is displayed as “Unknown.” This lets the user make an informed decision whether to install the application or not:



Application Install

Are you sure you want to install this application to your computer?

Publisher: UNKNOWN
Application: PeekAgenda
System Access: UNRESTRICTED

Installing applications may present a security risk to you and your computer. Install only from sources that you trust.

Publisher Identity: UNKNOWN
The publisher of this application cannot be determined.

System Access: UNRESTRICTED
This application may access your file system and the internet, which may put your computer at risk.

AIR applications first require the runtime to be installed on a user’s computer, just as SWF files first require the Flash Player browser plug-in to be installed.

The runtime can be installed in two ways: using the seamless install feature or via a manual installation.

- The seamless install feature provides developers with a streamlined installation experience for users who do not have Adobe AIR installed yet. In the seamless install method, the developer embeds a SWF file in a web page, and that SWF file presents the name of the AIR application for installation. When a user clicks in the SWF file to install the application, the SWF file checks for the presence of the runtime. If the runtime cannot be detected it is installed, and the runtime is activated immediately with the installation process for the developer's application. The user is provided with the option to cancel installation.
- Alternatively, the user can manually download and install the runtime before installing an AIR file. The developer can then distribute an AIR file by different means (for example, via e-mail or an HTML link on a web site). When the AIR file is opened, the runtime is activated and begins to process the application installation.

The AIR security model allows users to decide whether to install an AIR application. The AIR installer provides several improvements over native application install technologies that make this trust decision easier for users:

- The runtime provides a consistent installation experience on all operating systems, even when an AIR application is installed from a link in a web browser. Most native application install experiences depend upon the browser or other application to provide security information, if it is provided at all.
- The AIR application installer identifies the source of the application (or, if the source cannot be verified, the installer makes this clear) and it provides information about the privileges that are available to the application if the user allows the installation to proceed.
- The runtime administers the installation process of an AIR application. An AIR application cannot manipulate the installation process the runtime uses.

In general, users should not install any application (including an AIR application) that comes from a source that they do not trust, or that cannot be verified. The burden of proof on security for native applications is equally true for AIR applications as it is for other installable applications.

AIR 2 adds support for extended desktop AIR applications. These applications are installed using native installer files:

- DMG files on Mac OS
- EXE files on Windows
- RPM or DEB packages on Linux

When you create a native installer, you no longer gain the benefits of the standard AIR installation and update model. You become responsible for the install experience just as you would with a native application created with any other technology.

AIR 3 adds support for captive runtime bundles. In this deployment model, your application no longer uses the shared runtime installed on a user's computer. Instead, your application contains its own, private copy of the AIR runtime. In this model, you are responsible for the install and update experience. Furthermore, since the AIR runtime used by your application is never updated by Adobe, you are also responsible for updating your application whenever applicable security fixes to the runtime are published.

Updating AIR applications

Development and deployment of software updates are one of the biggest security challenges facing native code applications. An installed AIR application can check a remote location for an update AIR file. If an update is appropriate, the AIR file is downloaded, installed, and the application restarts. The developer documentation provides details on using this method not only to provide new functionality but also respond to potential security vulnerabilities.

AIR 2 adds support for extended desktop AIR applications. These applications are installed and also updated using native installer files:

- DMG files on Mac OS
- EXE files on Windows
- RPM or DEB packages on Linux

The built-in AIR Updater class and the AIR update framework do not support updating AIR applications installed via a native installer. (There are open source projects are available that do support this.)

More Help topics

RIASpace: Native Application Updater project

Removing an AIR application

Removing an AIR application removes all files in the application directory. However, it does not remove files that the application may have written outside the application directory. Removing AIR applications does not revert changes the AIR application has made to files outside the application directory.

Installing and updating mobile applications

Mobile AIR applications can be distributed as native packages for the supported platforms. On Android, the package format is an APK file; on iOS, the package format is an IPA file. Users can download and install mobile AIR applications via the normal means supported by the platform. For example, via the Market on Android and the App Store on iOS. Installation of AIR applications is subject to the same restrictions as any other application on the platform.

On Android, the AIR runtime is installed separately and is activated whenever an AIR for Android app is launched.

On iOS devices, such as the iPhone, the AIR runtime is not installed separately; each AIR app on iOS is a self-contained application.

In general, users should not install any application (including an AIR application) that comes from a source that they do not trust, or that cannot be verified. The burden of proof on security for native applications is equally true for AIR applications as it is for other installable applications.

AIR 3 adds support for captive runtime bundles on Android. In this deployment model, your application no longer uses the shared runtime installed on a user's device. Instead, your application contains its own, private copy of the AIR runtime. In this model, you are responsible for the install and update experience. Furthermore, since the AIR runtime used by your application is never updated by Adobe, you are also responsible for updating your application whenever applicable security fixes to the runtime are published. Note that the deployment model used on iOS has always used a captive runtime.

Updating mobile AIR applications

Development and deployment of software updates are one of the biggest security challenges facing native code applications. AIR applications on mobile devices can use the native platform update mechanism. On Android, this mechanism is the Android Market. On iOS, this mechanism is the Apple iTunes App Store.

Adobe AIR updates

Periodically, Adobe updates Adobe AIR with new features and fixes.

Desktop AIR updates

On desktop operating systems, the Automatic Notification and Update feature allows Adobe to automatically notify users when an updated version of Adobe AIR is available.

Updates to Adobe AIR ensure that Adobe AIR works properly and may contain important changes to security. Adobe recommends that users update to the latest version of Adobe AIR whenever a new version is available, especially when a security update is mentioned.

By default, when an AIR application is launched, the runtime checks if an update is available. It performs this check if it has been more than two weeks since the last update check. If an update is available, AIR downloads the update in the background.

Users can disable the auto-update capability by using the AIR SettingsManager application. The AIR SettingsManager application is available for download at
http://airdownload.adobe.com/air/applications/SettingsManager/SettingsManager.air.

The normal installation process for Adobe AIR includes connecting to http://airinstall.adobe.com to send basic information about the installation environment such as operating system version and language. This information is only transmitted once per installation and it allows Adobe to confirm that the installation was successful. No personally identifiable information is collected or transmitted.

Mobile AIR updates

On Android, updates to the AIR runtime are distributed through the Android Market.

On iOS, the runtime is bundled with each application. Application developers can rebuild their apps using an updated SDK to incorporate bug fixes, security updates, and new runtime features.

Code Signing

Adobe AIR requires all AIR applications to be digitally signed. Code signing is a process of digitally signing code to ensure integrity of software and the identity of the publisher. Developers can sign AIR applications with a certificate issued by a Certification Authority (CA) or by constructing a self-signed certificate.

Digitally signing AIR files with a certificate issued by a recognized certificate authority (CA) provides significant assurance to users that the application they are installing has not been accidentally or maliciously altered. Digitally signing AIR files with a certificate issued by a recognized certificate authority (CA) identifies the developer as the signer (publisher). AIR recognizes code signing certificates issued by the Verisign and Thawte certificate authorities. The AIR application installer displays the publisher name during installation when the developer has signed the AIR file with a Verisign or Thawte certificate.

The AIR application installer displays the publisher name during installation when the AIR application has been signed with a certificate that is trusted, or which chains to a certificate that is trusted on the installation computer. The Certification Authority (CA) verifies the publisher's or developer's identity using established verification processes before issuing a high assurance certificate.

Developers can also sign AIR applications using a self-signed certificate; one that they create themselves. However, the AIR application installer presents these applications as originating from an unverified publisher.

When an AIR file is signed, a digital signature is included in the installation file. The signature includes a digest of the package, which is used to verify that the AIR file has not been altered since it was signed, and it includes information about the signing certificate, which is used to verify the publisher identity.

AIR uses the public key infrastructure (PKI) supported through the operating system's certificate store. The computer on which an AIR application is installed must either directly trust the certificate used to sign the AIR application, or it must trust a chain of certificates linking the certificate to a trusted certificate authority in order for the publisher information to be verified.

If an AIR file is signed with a certificate that does not chain to one of the trusted root certificates (and normally this includes all self-signed certificates), then the publisher information cannot be verified. While AIR can determine that the AIR file has not been altered since it was signed, there is no way to verify who actually created and signed the file.

Details on the code signing process and accepted certificate formats are provided in the developer documentation.

Code signing with desktop native installers

When you package your application as a native installer, you can optionally apply a native code signature. Native code signing is supported on Windows only. See MSDN: Introduction to Code Signing.

Code signing on mobile platforms

On mobile platforms, AIR applications are signed according to the platform conventions and requirements. Developers sign their applications using tools from the AIR SDK and a certificate that meets the requirements of the mobile platform. The installation of mobile AIR apps is handled by the device operating system, not the AIR runtime. Thus AIR does not validate the application signature or certificate holder identity.

More Help topics

Signing AIR applications

Security sandboxes

AIR provides a comprehensive security architecture that defines permissions for each file in an AIR application. This includes both those files installed with the application and other files loaded by the application. Permissions are granted to files according to their origin, and are assigned to logical security groupings called sandboxes.

Files installed with the application are in a directory known as the application directory, and as such, they are, by default, placed in a security sandbox — known as the application sandbox — that has access to all AIR APIs. This includes APIs that would pose a great security risk if made available to content from sources other than the application resource directory (in other words, files that are not installed with the application).

The AIR security model of sandboxes is composed of the Flash Player security model with the addition of the application sandbox. Files that are not in the application sandbox have security restrictions like those specified by the Flash Player security model.

The runtime uses these security sandboxes to define the range of data that a file may access and the operations it may execute. To maintain local security, the files in each sandbox are isolated from the files of other sandboxes. For example, a SWF file loaded into an AIR application from an external Internet URL is placed into the remote sandbox, and does not by default have permission to script into files that reside in the application directory, which are assigned to the application sandbox.

Note: *On iOS, the execution of downloaded code is not permitted.*

Privileges of content in the application sandbox

When an application is installed, all files included within an AIR installer file are installed onto the user's computer into an application directory. All files within the application directory tree are assigned to the application sandbox when the application is run. Content in the application sandbox is allowed the full privileges available to an AIR application, including interaction with the local file system.

Many AIR applications use only these locally installed files to run the application. However, AIR applications are not restricted to just the files within the application directory — they can load any type of file from any source. This includes files on the user's computer as well as files from external sources, such as those on a local network or from the Internet. File type has no impact on security restrictions; loaded HTML files have the same security privileges as loaded SWF files from the same source. (However, content in the application sandbox is restricted from loading JavaScript files from outside that sandbox. Details are provided in the developer documentation.)

Content in the application security sandbox has access to AIR APIs that content in other sandboxes is prevented from using. For example, only content in the application security sandbox can read and write to the local file system.

Some JavaScript techniques exist for dynamically transforming strings into executable code. Content in the application security sandbox can only use these techniques while code is loading from application URLs. Using these techniques within the application sandbox would pose a security risk. For example, an application could inadvertently execute a string loaded from a network sandbox, and that string may contain malicious code, such as code to delete or alter files on the user’s computer or to report back the contents of a local file to an untrusted network domain. Details are provided in the developer documentation.

Note: *In mobile AIR applications, HTML and JavaScript cannot be loaded into the application sandbox. Mobile AIR applications display such content using the system web control. This control has the same security considerations as the default system web browser.*

Privileges of content in non-application sandboxes

Files loaded from a network or Internet location are assigned to a non-application sandbox. Such content behaves with the same set of privileges and restrictions as SWF content running in a web browser (in Flash Player) or HTML content running in a web browser.

Unlike content in the application security sandbox, HTML code in a non-application security sandbox *can* use JavaScript methods to execute dynamically generated code at any time.

Code in a non-application sandbox does not have access to the privileged AIR APIs that provide application functionality.

Details are provided in the developer documentation.

Accessing the file system

Applications running in a web browser have only limited interaction with the user's local file system. Web browsers implement security policies that ensure that a user's computer cannot be compromised as a result of loading web content. For example, SWF files running through Flash Player in a browser cannot directly interact with files already on a user's computer. Shared objects can be written to a user's computer for the purpose of maintaining user preferences and other data, but this is the limit of file system interaction. Because AIR applications are natively installed, they have a different security contract with the end user. This contract between the application and the end user is made at install time just like native applications, and it includes the capability for the application to read and write across the local file system.

This freedom comes with a higher degree of responsibility for developers. Accidental application security gaps jeopardize not only the functionality of the application, but also the integrity of the user’s computer. The developer documentation includes AIR security information that addresses best practices.

Unless there are administrator restrictions applied to the user’s computer, AIR applications are privileged to write to any location on the user's hard drive. However, developers are encouraged to use the user- and application-specific application storage directory that the runtime provides for each application. The AIR API provides convenient methods for developers to read and write data in the application storage directory. The runtime also provides an encrypted local data storage area unique to each application and user. This allows applications to save and retrieve data that is stored on the user’s local hard drive in an encrypted format that cannot be deciphered by other applications or users. A separate encrypted local store is used for each AIR application, and each AIR application uses a separate encrypted local store for each user. Applications may use the encrypted local store to store information that must be secured, such as login credentials for web services. AIR uses DPAPI on Windows and KeyChain on Mac OS to associate encrypted local stores to each user. The encrypted local store uses AES-CBC 128-bit encryption.

In Adobe AIR 2, applications can open files with the default application registered for the file type. For example, applications can open an mp3 file with the default application used to open mp3 files. AIR prevents applications from opening files that contain certain file types. These file types can potentially execute code when opened. An example is an EXE file on Windows. The restricted file types are listed in the ActionScript 3.0 Reference for the Flash Platform. However, extended desktop AIR applications, which are installed with native installers, can open files of any type. (For information on extended desktop applications, see “Communicating with native processes” on page 8.)

Communicating with native processes

As of Adobe AIR 2, desktop AIR applications can run and communicate with other native processes via the command line. For example, an AIR application can run a process and communicate with it via the standard input and output streams.

To communicate with native processes, the developer packages an AIR application to be installed via a native installer. The file type of native installer is specific to the operating system for which it is created:

- It is a DMG file on Mac OS.
- It is an EXE file on Windows.
- It is an RPM or DEB package on Linux.

These applications are known as extended desktop profile applications.

When packaging these applications, the developer signs the application with a code-signing certificate. The same kinds of certificates are used as those used in signing a standard desktop AIR application.

The native process API can run any executable on the user's system. The AIR documentation provides developers with guidance on using the native process API securely. Developers are warned to take care when constructing and executing commands. Applications should validate data that is sent to a native process.

AIR on Windows prevents extended desktop applications from running .bat files directly. Command-line arguments to a .bat file could contain potential malicious injections of extra characters. These injections could cause the cmd.exe application (which executes .bat files) to execute harmful or insecure applications.

Working securely with untrusted content

Content not assigned to the application sandbox can provide additional scripting functionality to an AIR application, but only if it meets the security criteria of the runtime. This section explains the AIR security contract with non-application content.

AIR applications restrict scripting access for non-application content more stringently than the Flash Player browser plug-in restricts scripting access for untrusted content. For example, in Flash Player in the browser, a SWF file can call the `System.allowDomain()` method to grant scripting access to any SWF content loaded from a specified domain. Calls to this method are not permitted for content in the AIR application security sandbox, since it would grant unreasonable access to the non-application file into the user's file system.

AIR applications that script between application and non-application content have more complex security arrangements. Files that are not in the application sandbox are only allowed to access the properties and methods of files in the application sandbox through the use of a *sandbox bridge*. A sandbox bridge acts as a gateway between application content and non-application content, providing explicit interaction between the two files. When used correctly, sandbox bridges provide an extra layer of security, restricting non-application content from accessing object references that are part of application content.

The benefit of sandbox bridges is best illustrated through example. Suppose an AIR music store application wants to provide an API to advertisers who want to create their own SWF files, with which the store application can then communicate. The store wants to provide advertisers with methods to look up artists and CDs from the store, but also wants to isolate some methods and properties from the third-party SWF file for security reasons.

A sandbox bridge can provide this functionality. By default, content loaded externally into an AIR application at runtime does not have access to any methods or properties in the main application. With a custom sandbox bridge implementation, a developer can provide services to the remote content without exposing these methods or properties. The sandbox bridge provides a limited pathway between trusted and untrusted content.

For full details on using sandbox bridges, see:

- For ActionScript (Flash and Flex) developers, see AIR Security in the ActionScript 3.0 Developer's Guide.
- For Ajax developers, see AIR Security in the HTML Developer's Guide for Adobe AIR.

Security on Android devices

On Android, as on all computing devices, AIR conforms to the native security model. At the same time, AIR maintains its own security rules, which are intended to make it easy for developers to write secure, Internet-connected applications.

Since AIR applications on Android use the Android package format, installation falls under the Android security model. The AIR application installer is not used.

The Android security model has three primary aspects:

- Permissions
- Application signatures
- Application user IDs

Android permissions

Many features of Android are protected by the operating system permission mechanism. In order to use a protected feature, the AIR application descriptor must declare that the application requires the necessary permission. When a user attempts to install the app, the Android operating system displays all requested permissions to the user before the installation proceeds.

Most AIR applications will need to specify Android permissions in the application descriptor. By default, no permissions are included. The following permissions are required for protected Android features exposed through the AIR runtime:

ACCESS_COARSE_LOCATION Allows the application to access WIFI and cellular network location data through the Geolocation class.

ACCESS_FINE_LOCATION Allows the application to access GPS data through the Geolocation class.

ACCESS_NETWORK_STATE and ACCESS_WIFI_STATE Allows the application to access network information through the NetworkInfo class.

CAMERA Allows the application to access the camera.

INTERNET Allows the application to make network requests. Also allows remote debugging.

READ_PHONE_STATE Allows the AIR runtime to mute audio when an incoming call occurs.

RECORD_AUDIO Allows the application to access the microphone.

WAKE_LOCK and DISABLE_KEYGUARD Allows the application to prevent the device from going to sleep using the SystemIdleMode class settings.

WRITE_EXTERNAL_STORAGE Allows the application to write to the external memory card on the device.

Application signatures

All application packages created for the Android platform must be signed. Since AIR applications on Android are packaged in the native Android APK format, they are signed in accordance to Android conventions rather than AIR conventions. While Android and AIR use code signing in a similar fashion, there are significant differences:

- On Android, the signature verifies that the private key is in possession of the developer, but is not used to verify the identity of the developer.
- For apps submitted to the Android market, the certificate must be valid for at least 25 years.
- Android does not support migrating the package signature to another certificate. If an update is signed by a different certificate, then users must uninstall the original app before they can install the updated app.
- Two apps signed with the same certificate can specify a shared ID that permits them to access each others cache and data files. (Such sharing is not facilitated by AIR.)

Application user IDs

Android uses a Linux kernel. Every installed app is assigned a Linux-type user ID that determines its permissions for such operations as file access. Files in the application, application storage, and temporary directories are protected from access by file system permissions. Files written to external storage (in other words, the SD card) can be read, modified, and deleted by other applications, or by the user, when the SD card is mounted as a mass storage device on a computer.

Cookies received with Internet requests are not shared between AIR applications.

More Help topics

Android: Security and Permissions

Application installation

By default, AIR applications on Android use a shared runtime library that is maintained and updated by Adobe. Starting in AIR 3, you can bundle your application with a “captive” runtime. An application installed with a captive runtime uses that version of the runtime — not the shared AIR runtime, which might also exist on the device. A captive runtime is not automatically updated when a new version of the AIR runtime is published.

Important: *When you use a captive runtime, you assume the responsibility for updating that runtime when relevant security updates are published by Adobe.*

Encrypted data on Android

AIR applications on Android can use the encryption options available in the built-in SQL database to save encrypted data.

The EncryptedLocalStore class can be used to save data, but that data is not encrypted. Instead, the Android security model relies on the application user ID to protect the data from other applications. Applications that use a shared user ID and which are signed with the same code signing certificate use the same encrypted local store.

Important: *On a rooted phone, any application running with root privileges can access the files of any other application. Thus, data stored using the encrypted local store is not secure on a rooted device.*

Security on iOS devices

On iOS AIR conforms to the native security model. At the same time, AIR maintains its own security rules, which are intended to make it easy for developers to write secure, Internet-connected applications.

Since AIR applications on iOS use the iOS package format, installation falls under the iOS security model. The AIR application installer is not used. Furthermore, a separate AIR runtime is not used on iOS devices. Every AIR application contains all the code needed to function.

Important: *Since the runtime code is included as part of your application, it is not automatically updated when Adobe releases a new version of AIR. It is your responsibility to update your application when relevant security updates are published by Adobe.*

Application signatures

All application packages created for the iOS platform must be signed. Since AIR applications on iOS are packaged in the native iOS IPA format, they are signed in accordance with iOS requirements rather than AIR requirements. While iOS and AIR use code signing in a similar fashion, there are significant differences:

- On iOS, the certificate used to sign an application must be issued by Apple; Certificates from other certificate authorities cannot be used.
- On iOS, Apple-issued distribution certificates are typically valid for one year.

HTML security

HTML content has different security considerations than SWF-based content, primarily due to the ability of JavaScript to create dynamically generated code. Dynamically generated code, such as that which is made when calling the `eval()` function, could pose a security risk if allowed within the application sandbox. For example, an application could inadvertently execute a string loaded from a network sandbox, and that string may contain malicious code, such as code to delete or alter files on the user's computer or to report back the contents of a local file to an untrusted network domain.

Ways to generate dynamic code include the following:

- Calling the `eval()` function.
- Setting `innerHTML` properties or calling DOM functions to insert script tags to load a script outside the resource directly.
- Setting `innerHTML` properties or calling DOM functions to insert script tags that have in-line code (rather than loading a script via the src).
- Setting the `src` for `script` tags for content in the application sandbox to a file that is not in the application resource directory.
- Using the `javascript` URL scheme (as in `href="javascript:alert('Test')"`).

Code in the application security sandbox can only use these methods while content is loading from application directory. This prevents code in the application sandbox, which has access to the full AIR APIs, from executing scripts from potentially untrusted sources.

Content from non-application security sandboxes can generate dynamic code using these methods. However, they do not have direct access to the AIR APIs. The AIR sandbox bridge feature provides means for code in non-application security sandboxes to interact with code in the application sandbox in ways that are limited and decided by the application code.

AIR applications can generate HTML content from string variables (rather than loading them from files or network sources). However, by default, HTML content generated by strings is not given application sandbox privileges. This prevents the application from inadvertently granting application access to string content obtained from potentially unsafe internet sources.

Note: *On mobile devices, AIR uses the web control provided by the host operating system. Content running in this control does not have access to the AIR APIs and is never loaded or executed in the application security sandbox.*

For details on HTML security, see “AIR Security” in the developer documentation:

- For ActionScript (Flash and Flex) developers, see AIR Security in the ActionScript 3.0 Developer’s Guide.
- For Ajax developers, see AIR Security in the HTML Developer’s Guide for Adobe AIR.

  Also, see the Adobe AIR HTML Security white paper.

Other security considerations

Although AIR applications are built using web technologies, it is important for developers to note that they are not working within the browser security model. This means that it is possible to build AIR applications that can do harm to the local system, either intentionally or unintentionally. AIR attempts to minimize this risk, but there are still ways where vulnerabilities can be introduced. This section covers important potential insecurities. The developer documentation provides best practices for building applications that avoid these risks.

Risk from importing files into the application security sandbox

Content the application sandbox have the full privileges of the runtime. Developers are advised to consider the following:

- Include a file in an AIR file (in the installed application) only if it is necessary.
- Include a scripting file in an AIR file (in the installed application) only if its behavior is fully understood and trusted.
- Do not use data from a network source as parameters to methods of the AIR API that may lead to code execution.

  Adobe AIR protects content in the application sandbox from using data from a network source as code, which could lead to inadvertent execution of malicious code. This includes use of the ActionScript `Loader.loadBytes()` method and the JavaScript `eval()` function

Risk from using an external source to determine paths

An AIR application can be compromised when using external data or content. For this reason, applications must take special care when using data from the network or file system. The onus of trust is ultimately upon to the developer and the network connections they make, but loading foreign data is inherently risky, and should not be used for input into sensitive operations. Developers are advised against the following:

- Using data from a network source to determine a filename
- Using data from a network source to construct a URL that the application uses to send private information or to launch other applications

Risk from using, storing, or transmitting unsecured credentials

Storing user credentials on the user's local file system inherently introduces the risk that these credentials may be compromised. Developers are advised to consider the following:

- If credentials must be stored locally, encrypt the credentials when writing to the local file system. Adobe AIR provides an encrypted storage unique to each installed application, which is described in detail in the developer documentation.
- Do not transmit unencrypted user credentials to a network source unless that source is trusted and the transmission protocol uses Transport Layer Security (TLS), in other words https: or the SecureSocket class.
- Never specify a default password in credential creation — let users create their own. Users who leave the default expose their credentials to an attacker that already knows the default password.

Risk from receiving or transmitting remote data

When information travels over the Internet, it is subject to eavesdropping and alteration. You can use the Transport Layer Security (TLS) or the older Secure Socket Layer (SSL) protocols to encrypt communication between a server and a client system. Use the HTTPS protocol to protect Hypertext Protocol (HTTP) communication. Use the SecureSocket class to protect TCP socket communications. In both cases, the server providing the data must use TLS or SSL in order for your AIR application to send or receive the data securely. The SecureSocket class was added in AIR 2 and can be used by content running in the AIR application sandbox. Developers are advised to consider the following:

- Provide access to sensitive data hosted by a server using the most recent version of the TLS protocol supported. (At this time, AIR supports TLS version 1 and SSL version 4.)
- Use HTTPS to transmit or receive sensitive data using the HTTP protocol.
- Use the SecureSocket class to transmit or receive sensitive data using a TCP socket.

Risk from a downgrade attack

During the application installation process, the runtime checks to ensure that a version of the application is not currently installed. If an application is already installed, the runtime compares the version string of the existing application against the version that is being installed. If this string is different, the user can choose to upgrade their installation. The runtime cannot guarantee that the newly installed version is newer than the older version, only that it is different. An attacker can distribute an older version to the user to circumvent a security weakness. This risk can be mitigated, and the developer documentation provides best practices for implementing version schemes and update checks to avoid risks.