



WHITE PAPER

Adobe Flash Player 10 Security  
Flash Player 10.0.12

November 2008

Copyright © 2006-2008 Adobe Systems Incorporated. All rights reserved.

The information contained in this document represents the current view of Adobe on the issue discussed as of the date of publication. Because Adobe must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Adobe, and Adobe cannot guarantee the accuracy of any information presented after the date of publication.

This white paper is for information purposes only. ADOBE MAKES NO WARRANTIES, EXPRESS OR IMPLIED, IN THIS DOCUMENT.

Adobe may have patents, patent applications, trademark, copyright or other intellectual property rights covering the subject matter of this document. Except as expressly provided in any written license agreement from Adobe, the furnishing of this document does not give you any license to these patents, trademarks, copyrights or other intellectual property.

Adobe, the Adobe Logo, Adobe AIR, Macromedia, Flash, Flash Lite, and Flex are trademarks or registered trademarks of Adobe Systems Incorporated in the United States and/or other countries. The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

Adobe Systems Incorporated  
345 Park Avenue  
San Jose, CA 95110  
(408) 536-6000

# Contents

Introduction .....	5
About this document.....	5
Intended audience .....	5
About the Flash Player client runtime .....	6
About security.....	6
Other sources of information.....	7
The Flash Player security environment .....	9
Stakeholders .....	9
Administrative user (of a particular client computer) and the user institution.....	10
User (of a particular computer and programs) .....	10
Website administrator.....	10
Author (developer of a Flash application) .....	10
Overview of permission controls .....	11
Sources for potential risk .....	12
Innocent bugs.....	12
Other stakeholders .....	12
Internet providers.....	12
Flash Player security architecture.....	13
Basic sandbox security model .....	13
Domain of origin .....	14
Default permissions.....	14
Accessing data in another sandbox.....	16
Permissions for specific domains .....	17
Server interaction .....	20
Port blocking.....	20
Content-disposition headers .....	20
POST commands and file uploads .....	21
User interaction.....	21
User interaction required.....	21
User interaction limited.....	21
Interpreters and byte code .....	22
Background.....	22
Flash Player and byte codes.....	22
Code isolation .....	23
Disk, memory, and processor protections.....	23
Disk storage protections.....	23
Memory usage protections and processor quotas.....	24

Permission controls.....	24
Administrative user controls.....	25
The mms.cfg file.....	25
Global Flash Player Trust directory.....	26
User controls.....	26
Settings Manager.....	27
Settings UI.....	29
Runtime dialog boxes.....	31
User Flash Player Trust directory.....	33
Website controls (policy files).....	33
URL policy files.....	34
Header sending permissions.....	37
Socket policy files.....	37
Policy file logging.....	39
Developer controls.....	39
Permission mechanisms.....	40
Security.allowDomain().....	41
Security.loadPolicyFile().....	42
Security.exactSettings.....	42
Security.sandboxType.....	42
The LocalConnection class.....	43
Options when publishing.....	44
ActiveX control and browser plug-in APIs.....	44
Hierarchy of local file security controls.....	45
Loading into the local-trusted sandbox.....	45
Loading into the local-with-networking sandbox.....	45
The default setting: local-with-file-system.....	45
Flash Player integration with native applications.....	46
Deployment of the Flash Player runtime.....	46
Browser plug-ins and ActiveX controls.....	46
Authoring player.....	47
Stand-alone player and Flash projector.....	47
Other distributions.....	48
Deployment of Flash applications.....	48
SWF files.....	48
Executable projector files.....	48

Other security-related information .....	49
Network protocols .....	49
AMF .....	49
SMB.....	49
RTMP .....	49
HTTP.....	49
HTTPS .....	50
TCP sockets .....	50
SSL (Secure Sockets Layer) utilization .....	51
Basic SSL—browser plug-ins.....	51
Projector files .....	51

# Introduction

Security is a key concern of Adobe®, users, website administrators, and content developers. For this reason, Adobe Flash® Player 10 includes a set of security rules and controls to safeguard the user, website administrator, and content developer. This white paper provides an overview of the Flash Player security model.

## About this document

This document focuses on the security-relevant features of the Flash Player client runtime, including those introduced in earlier versions of the product. While not attempting to distinguish between versions, some references are included where changes in the security model or potential operation of applications designed and implemented in earlier versions of Flash Player may significantly differ from the target Flash Player environment described here. Unless otherwise noted, however, this document assumes that the target platform for your development is Flash Player 10.0.12 running content that uses ActionScript® 3.0.

If you have already implemented applications or web sites for an earlier version of Flash Player, the table at the end of this section (see “Other sources of information” on page 7) lists documents that can provide information on potential implementation differences between your current Flash Player implementations and Flash Player 10 implementations. That is, applications you published for an earlier version of Flash Player may not run as expected if the end user is running Flash Player 10; refer to the additional resources listed to learn how to address these potential issues.

As you may know, Flash Player 9 introduced an entirely new ActionScript Virtual Machine, AVM2. AVM2 runs a much faster, ECMAScript standards-compliant language: ActionScript 3.0. AVM2 itself introduced some new architecture to this environment due to its enhanced byte code interpretation model. AVM1 remains available to run ActionScript 1.0 and 2.0 content, but this document doesn't address implementation features specific to AVM1, ActionScript 1.0, or ActionScript 2.0.

There are no distinctions in the runtime security model between applications created using different development tools, such as Adobe AIR™, Adobe Flex™ Builder, the Adobe Flex SDK, or Adobe Flash. There are differences in the runtime security model between applications running in Flash Player and those running in Adobe AIR. This document doesn't address the security environment of Adobe AIR only that of Flash Player.

## Intended audience

This paper is intended for the following audiences:

- Enterprise architects who are evaluating or need to better understand the security model of the Flash Platform
- IT managers and system administrators who are interested in the security of Flash applications in their network environment
- Website administrators who deploy Flash applications from their sites
- Developers (including programmers and other authors) who design and publish Flash applications

This document assumes that the reader is familiar with Flash and ActionScript, as well as with their related terms, authoring tools, and environments.

# About the Flash Player client runtime

Adobe Flash Player runs Flash applications (also referred to as SWF files). Flash Player content is delivered as a series of instructions in binary format to Flash Player over web protocols in the precisely described SWF (.swf) file format (described at [www.adobe.com/devnet/swf/](http://www.adobe.com/devnet/swf/)). The SWF files themselves are typically hosted on a server, and downloaded to and displayed on the client computer when requested. SWF files consist of multimedia content (vectors, bitmaps, sound, and video) and binary ActionScript instructions. ActionScript is the ECMA standards-based scripting language used by Flash that features APIs designed to allow the creation and manipulation of client-side user interface elements, and for working with data.

Flash Player is designed to allow all SWF file content to be viewable and available consistently across a broad range of platforms, browsers, and devices. Flash Player is also designed to provide a robust environment to ensure security and privacy for the author, user, host institutions, and any of their respective data. Flash Player executes on the client's computer to view Flash content, typically from a host web server.

## About security

This document focuses on the security protections and access rules that apply for code and data running in Flash Player 10.0.12. All authors and users should be aware that Adobe cannot make security claims for, or significantly modify or enhance, the security capabilities and attributes of the infrastructure in which its products execute. Adobe products also utilize various external security components, such as browser- or OS-provided cryptography, and must rely on the strength of such components as chosen or made available on a given platform.

Security flaws might exist in the underlying environment (including the operating system and web browsers) that can potentially be exploited regardless of the applications (including Flash Player) running in that environment. The approach of Adobe is to implement robust security within its own products while "doing no harm" to the rest of the environment (in other words, to introduce no exposures to the rest of the environment, nor allow any avenues for additional exploitation of any existing platform security weaknesses). This provides a consistently high level of security for what Flash applications can do (as managed within Flash Player), regardless of the platform. Because Adobe products are also designed to be backwards-compatible when possible, some environments may be more vulnerable to weaknesses in the browser or operating system, or have weaker cryptography capabilities. Ultimately, users are responsible for their choices of platforms and maintenance of appropriate operational environments, while Adobe products target support for all reasonable combinations.

The various default protections and security-related features of Flash Player provide all stakeholders with assurances about the security and privacy of their code and data when processed by Flash Player and its related components. Flash Player also attempts to make the most appropriate security decisions without requiring explicit involvement by the author or user where possible, and minimizes where the author might need to make significant decisions on behalf of all users (as contrasted to other environments, such as Java and .NET). Where any of the Flash Player default access controls may appear overly restrictive for a given type of application or intended data sharing, configuration and administration options exist to allow explicit permissions for broader sharing.

The Flash Player security model protects against three broad classes of potential security breaches:

- Unauthorized access to data. This data could be on local disks, networked disks, or web servers that are communicated with over the network or stored in memory by an application or process. (Examples might include password lists, address books, protected documents, and application code.)
- Unauthorized access to end-user information. This includes personal and financial data, among other information that might be on the end user's computer. This also includes information about the end-user's security settings for Flash Player.

- Unauthorized access to host system resources. This includes gaining control of applications, devices, or resources attached to the system for the purposes of disabling, or denying or redirecting access, to those resources. (Examples might include buffer overruns and denial of service attacks.)

Overall, Flash Player provides very controlled and selective access to other resources. The functionality available to Flash application developers is a small, constrained subset of the functionality that could potentially allow exposures. For example, Flash Player does not allow content to inspect directories of local file systems, erase or modify arbitrary local files, shut down the user's computer, or make changes to the operating system. Because the system functionality that Flash Player (or its applications) can access is carefully limited, the risk of creating content that could gain unauthorized access to the host system or resources attached to it is minimized.

## Other sources of information

This document is not the only source of information on Flash Player security. The following table lists documents and web sites that provide more information:

**Table 1: Other sources of security information**

Document	Audience	Contains
The Flash Player Security Topic Center ( <a href="http://www.adobe.com/go/devnet_security">http://www.adobe.com/go/devnet_security</a> )	All	Articles and links to various sources of information, including information on changes in each Flash Player version.
The article "Understanding the security changes in Flash Player 10" ( <a href="http://www.adobe.com/devnet/flashplayer/articles/fplayer10_security_changes.html">www.adobe.com/devnet/flashplayer/articles/fplayer10_security_changes.html</a> )	All	An overview of the security enhancements implemented in Flash Player 10.
The article "Working with policy file changes in Flash Player 9 and Flash Player 10" ( <a href="http://www.adobe.com/devnet/flashplayer/articles/fplayer9-10_security.html">www.adobe.com/devnet/flashplayer/articles/fplayer9-10_security.html</a> )	Website administrators	Detailed information on more secure implementation of policy files in Flash Player 9 Update 3 (9,0,115,0), Flash Player 9 April 2008 Security Update (9,0,124,0), and Flash Player 10.0.12.
The article "Cross-domain policy file usage recommendations for Flash Player" ( <a href="http://www.adobe.com/devnet/flashplayer/articles/cross_domain_policy.html">www.adobe.com/devnet/flashplayer/articles/cross_domain_policy.html</a> )	Website administrators	Discusses some of the common security issues to consider when deciding how to use a cross-domain policy file on your server.
The article "Creating more secure SWF web applications" ( <a href="http://www.adobe.com/devnet/flashplayer/articles/secure_swf_apps.html">www.adobe.com/devnet/flashplayer/articles/secure_swf_apps.html</a> )	Developers	Explains many of the security considerations associated with common tasks and provides samples of techniques that can be used to help secure code against those threats.

<b>Document</b>	<b>Audience</b>	<b>Contains</b>
The "Security" chapter in Programming ActionScript 3.0 ( <a href="http://www.adobe.com/go/flashcs4_prog_as3_security_en">www.adobe.com/go/flashcs4_prog_as3_security_en</a> )	Developers	Detailed security-related information for developers creating Flash Player 10 content.
The ActionScript 3.0 Language Reference ( <a href="http://www.adobe.com/go/learn_flashcs4_langref_en">www.adobe.com/go/learn_flashcs4_langref_en</a> )	Developers	Detailed information on all ActionScript 3 APIs.
The Flash Player Administration Guide ( <a href="http://www.adobe.com/go/flash_player_admin">http://www.adobe.com/go/flash_player_admin</a> )	IT professionals and system administrators	Information on administering Flash Player in enterprise environments.
Flash Player Help ( <a href="http://www.adobe.com/support/documentation/en/flashplayer/help/index.html">www.adobe.com/support/documentation/en/flashplayer/help/index.html</a> )	End users	Information on understanding and controlling security in client environment.

# The Flash Player security environment

The Flash Player client runtime security model has been designed around *resources*, which are objects such as SWF files, local data, and Internet URLs. *Stakeholders* are the parties who own or use those resources. This document has been organized to reinforce that model.

## Stakeholders

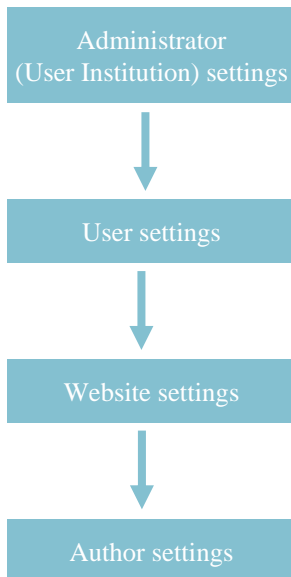
In any computer system, there are multiple stakeholders (individuals or classes of individuals) with interests related to correct operation and the protection of their data and resources. This is particularly important in an environment in which a user may obtain code from multiple sources (such as Flash Player from Adobe and a SWF file from another source), plus data from an outside website, in order to run an application on their local computer.

The following stakeholders may have security or privacy interests in this environment:

- Administrative user and the user institution
- User
- Website administrator
- Author

Within the Flash Player security model, each stakeholder can exercise controls (security settings) over their own resources, and each resource has four stakeholders. Flash Player strictly enforces a hierarchy of authority for these controls, as the following figure shows:

Figure 1: Hierarchy of security controls



This means, for instance, that if an administrator restricts access to a resource, no other stakeholders can override that restriction. In Flash Player, it is common for multiple stakeholders to have the ability to control access to a resource, and for some stakeholders to formally delegate the right of control to a lower level in the hierarchy. For example, Administrators regularly allow users to make security decisions about their own environment.

This hierarchy doesn't mean that all decisions about permissions can be enforced or modified by all stakeholders. For example, using ActionScript to communicate directly with another SWF file requires the read permission. Only the author can grant read permission for cross-scripting by calling the `Security.allowDomain()` method.

The following sections describe the stakeholders in more detail.

## **Administrative user (of a particular client computer) and the user institution**

A client computer has administrative settings that can only be modified by *administrative users*, as determined by the *user institution*, which is the entity on whose behalf the computer is used. The administrative user may simply be the user (in an in-home usage), but in work environments this can be restricted to administrators in an IT department. Such an institution typically owns and administers the computer (to varying extents). In some cases, each user may have multiple user institutions, as when an independent contractor contracts with multiple user institutions while using one computer. An institution may have data and configuration information on that computer (or on internal networks available to that computer) that it wants to protect from corruption or theft by programs that the user may select, install, or execute. Both users and user institutions are likely to have data on the client computers that they do not want shared on the external network.

## **User (of a particular computer and programs)**

The *user* is the single end-user *consumer* running Adobe and third-party products and applications on a client computer. Modern operating systems partition the computer to provide a degree of protection between multiple users of a single personal computer, and to a minor extent between programs running simultaneously on behalf of a single user on the same computer. It is possible that more than one user might use the same computer, or that one user may have multiple applications running and not want data shared between them. Privacy protection is considered one aspect of the security goals.

## **Website administrator**

Websites hosting Flash applications rely on Flash Player behavior to deliver their content and application features. Website administrators also implement security measures such as authentication, access controls, and network firewalls to ensure the integrity of their website. It is also important to distinguish between internal websites (behind one or more firewalls common to the client computer) and external websites (the rest of the Internet world). Users often access external websites as part of doing company business, and Flash programs from such external sites must not compromise security, such as unintended exposure of data from the user's local (intranet) data back to the external website, nor from the external website to the user.

## **Author (developer of a Flash application)**

An *author* is the program developer and publisher of a Flash application. The author might want to protect code (and data) from unauthorized modification or use. Protecting a Flash application means providing an environment in which the program can continue to work correctly (and be affected by only those things it chooses to be affected by), keep secret all of its logic and state except what it chooses to reveal, and impose proper security policies on its components.

# Overview of permission controls

The Flash security architecture provides each stakeholder with a set of permission controls for controlling access to their assets. The following table provides an overview. For more information, see “Permission Controls” on page 24.

**Table 2: Stakeholder permission controls**

Stakeholder	Control	Description
Administrator	The mms.cfg file	The mms.cfg file, accessible only to an administrative user, controls security-related Flash Player settings for the client computer, such as access to any camera or audio input devices attached to the computer and control of local file reading, file upload, and file download capabilities.
	Global Trust files	When installing a Flash application to a client computer, an installer (run by an administrator) can establish specified local files and directories as being trusted.
User	Settings Manager Settings UI Dialog Boxes	Flash Player includes a Settings Manager that lets the user set the security-related options (for that specific user of the computer), such as access to any camera or audio input devices attached to the computer and local Flash Player disk usage and third-party storage of persistent shared objects  When <i>legacy content</i> (content built for an earlier version of Flash Player) attempts to access resources that are protected in a newer version of Flash Player, Flash Player presents warning messages to the user.
	User Trust files	When installing a Flash application to a client computer, the installer (run by a specific user of the computer) can establish specific local files and directories as being trusted (for that user).

Website administrator	URL policy files	These files control Flash applications' access to resources on the domain.
	Socket policy files	These files authorize ActionScript socket-level connections.
	URL meta-policies and socket meta-policies	A meta-policy is a "policy on policies"—a designation by an administrator as to what policy files are permitted to exist on a server.
Author	Cross-script APIs and access to data originating from domains other than that of the SWF file	Flash Player provides a number of API calls related to security.

## Sources for potential risk

The Adobe Flash platform protects stakeholders from other sources of potential risk, described in this section. These risks can result from both malicious and accidental actions by other stakeholders or third parties.

### Innocent bugs

When a piece of software is part of a much larger system, it is a good practice to conceptualize software as being surrounded by malevolent entities, even when it is not. Design and implementation bugs can lead to security holes that can be exploited by malevolent entities, or simply can lead to unexpected (unwanted) behavior of the program. The most common security vulnerabilities are the result of innocent bugs, and Flash Player is designed to prevent introduction of a wide variety of these bugs, such as buffer overflows and cross-site scripting.

### Other stakeholders

Other users and user institutions with access to the same system are also entities to be protected against. They might want to obtain access to data that is not intended for sharing in those circumstances. Flash Player provides a clearly articulated model for sharing information: by default, Flash applications may not inspect or modify data without explicit permission from a stakeholder of that resource.

### Internet providers

While Internet providers might not intend to be potential sources of risk, they provide services that are vulnerable to certain classes of attacks. Internet providers include backbone operators, with significant responsibility for the correct functioning of DNS and packet routing.

# Flash Player security architecture

The Flash Player client runtime provides a comprehensive security architecture that defines the permissions granted to SWF files. Generally, these permissions are of the type `read` or `send`. Permissions are granted to a SWF file by the administrative user, end user, or website based on the origin of that SWF file. For example, although authors may grant permissions to specific SWF files, the `send` permission is generally allowed between sandboxes because the recipient is expected to handle the information in a secure manner. In some places where a security exposure might result from sending, the Flash Player runtime may restrict this functionality if explicit permission is not provided.

## Basic sandbox security model

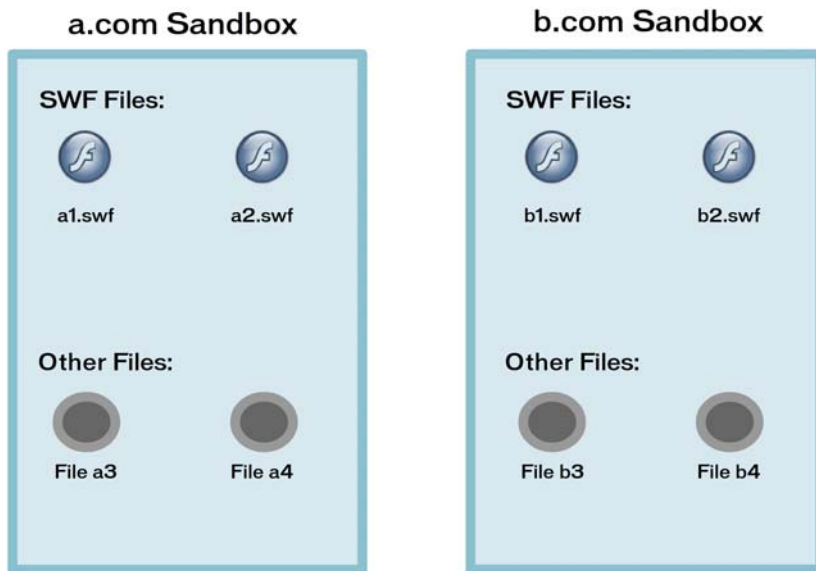
A significant component of security in Flash Player is based on *sandboxes*, which are logical security groupings that Flash Player uses to contain resources. Flash Player uses these security sandboxes to define the range of data and operations that each Flash application may access—that is, which resources can be accessed. Everything within each sandbox is securely controlled by stakeholders of that sandbox. This includes file requests, local data storage (shared objects), and any other resources used by a particular domain and its content. Each sandbox is isolated from the operating system, file system, network, other applications, and even other Flash Player sandbox instances.

Flash Player classifies SWF files downloaded from the network (such as from external websites) in separate sandboxes that correspond to their website origin domains. By default, these files are authorized to access additional resources that come from the specific (exact domain name match) site.

Flash Player places SWF files from local origins (local file systems or UNC—universal naming convention—network paths) into one of three specific sandboxes for local SWF files only. Any two SWF files that run in the same sandbox may interact freely with each other (for example, two SWF files downloaded from the same network domain). SWF files may also interact with SWF files from other sandboxes (and with servers), but only in accordance with specific security rules and configuration settings, which are described in “Permission controls” on page 24.

An author of a SWF file can use the `ActionScript Security.sandboxType` property to determine the type of sandbox to which Flash Player has assigned the SWF file (for more information, see “`Security.sandboxType`” on page 42).

Figure 2: Flash defines sandboxes based on source domains



## Domain of origin

Flash Player gets domain information from the host application (such as a browser). That information is only as reliable as the underlying protocol. For example, HTTP uses DNS for domain information, which is subject to spoofing; HTTPS uses SSL (secure socket layer) certificates, which provide cryptographically verified domain information. Flash Player applies a set of rules to the URL information it receives from the host application to determine if the content is from a local source.

Network SWF files can communicate freely with each other (and with other documents) only when they come from the same domain. For example, *www.adobe.com* and *store.adobe.com* are considered different domains.

By default, content loaded with a protocol other than HTTPS cannot access content that was loaded with HTTPS, even if from the same domain. The reverse direction is allowed; HTTPS content may access content loaded with other protocols from the same domain.

## Default permissions

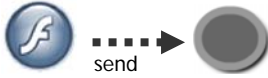
In the following series of diagrams, the arrows illustrate types of access:

- A solid arrow drawn between a SWF file and a non-SWF file resource depicts a read operation where the SWF file at the tail of the arrow is making the request on the resource indicated by the head of the arrow. The read access of a data resource outside of Flash Player is often referred to as *data loading*, because data is being brought into Flash Player for use by a SWF file. A solid arrow may also be drawn between two SWF files to depict a read permission. This may be described as a *cross-scripting*, because the SWF file at the tail of the arrow may invoke functions in the SWF file indicated by the head of the arrow. (Cross-scripting is not possible between AVM1 SWF files, which use ActionScript 1.0 or 2.0, and AVM2 SWF files, which use ActionScript 3.0.)
- A dashed arrow indicates data sent from the SWF file at the tail of the arrow and handled by the object indicated by the head of the arrow.

Figure 3: A read request from a SWF file to another resource



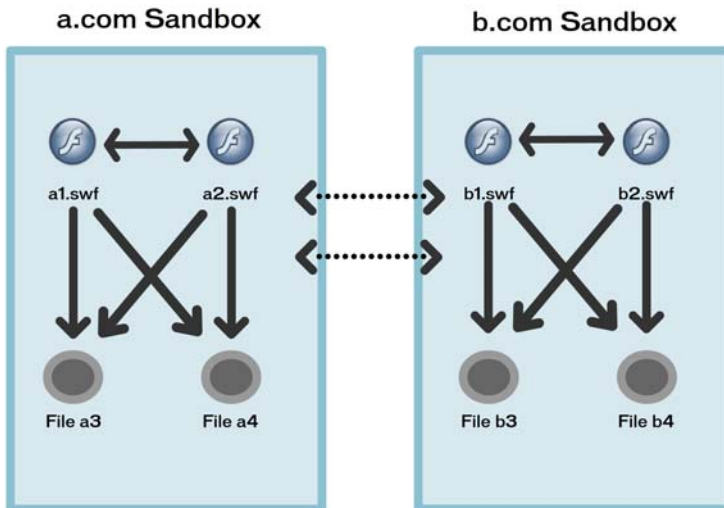
Figure 4: A message sent from a SWF file to another resource



Any two SWF files in the same sandbox may interact freely with each other. From the *a.com* sandbox, a SWF file may read from the server at *a.com* (by using the ActionScript `URLLoader.load()` method, for example), and may send anywhere on the network. However, it cannot by default read from *b.com*.

Diagrams in later sections explain how SWF files may interact with SWF files from other sandboxes, and with servers.

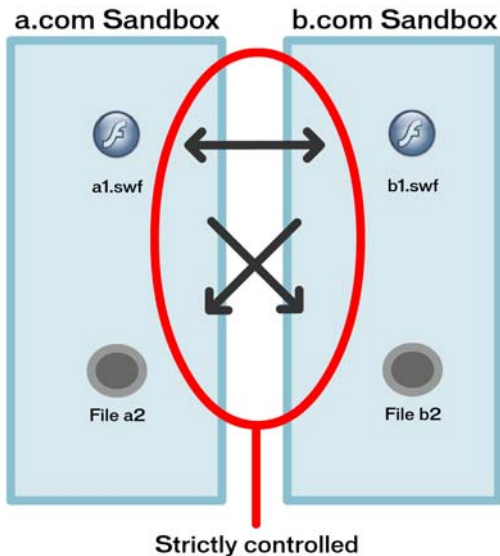
Figure 5: Default sandbox permissions



## Accessing data in another sandbox

In order for a SWF file to read data in another sandbox, it must be granted explicit permission by stakeholders of that other sandbox. There is a well-defined chain of precedence such that each stakeholder can protect the assets within their domain or system from external access (see “The Flash Player security environment” on page 9).

Figure 6: Read across sandbox boundaries is strictly controlled



A SWF file from *a.com* may read from the server at *b.com* (using the ActionScript `URLLoader.load()` method, for example) if *b.com* has a policy file that permits access from *a.com* (or from all domains). A SWF file from *a.com* may cross-script a SWF file from *b.com* (calling an ActionScript method in the *b.com* SWF file, for example) if the *b.com* SWF file calls the ActionScript `Security.allowDomain()` method to permit access from *a.com*.

Limitations on reading across sandboxes also apply to `DisplayObject` instances:

- `MouseEvent.relatedObject`: During a `mouseover`, `mouseout`, `rollover`, or `rollout` event, this property refers to the other object, if any, that was involved in the change of objects underneath the mouse pointer—the object that either was previously or is currently under the pointer.
- `FocusEvent.relatedObject`: During a `mouseFocusChange`, `keyFocusChange`, `focusIn`, or `focusOut` event, this property refers to the other object, if any, that was involved in the change of focus—the object that is either relinquishing or taking focus.
- `ContextMenuEvent.mouseTarget`: During a `menuSelect` or `menuItemSelect` event, this property refers to the object, if any, over which the context-click (right-click or Command-click) occurred.

If an object that would be referred to by any of these properties resides in a different security sandbox (for example, because it is part of a different SWF that was served from a different domain), and the two sandboxes do not both trust each other (by means of the `Security.allowDomain()` method), then the value of this property is changed to null.

If a developer wants to determine whether a null value resulted from this restriction, they can check the values of these properties, respectively:

- `MouseEvent.isRelatedObjectInaccessible`
- `FocusEvent.isRelatedObjectInaccessible`

- `ContextMenuEvent.isMouseEventInaccessible`

## Permissions for specific domains

### Network files

All resources in a network sandbox follow the basic sandbox model. Each domain is given its own sandbox. (For more information, see “Basic sandbox security model” on page 13.)

### Local files

Local files also follow the basic sandbox model, but they have different default permissions. *Local file* describes any file referenced using the “file:///” protocol or a UNC path, which does not include an IP address or a qualifying domain. For example, “\\test\test.txt” and “file:///test.txt” are considered local files, while “\\test.com\test.txt” and “\\192.150.18.61\test.txt” are not considered local files.

There are three different sandboxes for local files:

- local-with-filesystem
- local-with-networking
- local-trusted

For more information on these sandboxes, see “Basic sandbox security model” on page 13.

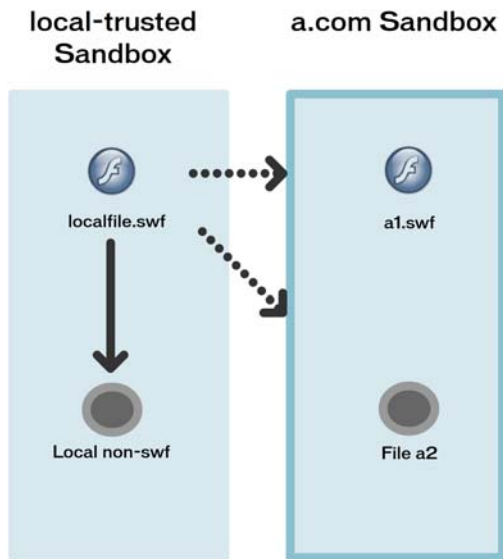
By default, local SWF files are placed in the local-with-file-system sandbox and may read from files on local file systems. They may not communicate with the network except in instances where the network resource is considered a local file.

Local SWF files may be placed in the local-with-networking sandbox by virtue of a developer control (see “Options when publishing” on page 44). Local SWF files may be placed in the local-trusted sandbox by virtue of various administrative or user controls (see “User Flash Player Trust directory” on page 33 and “Global Flash Player Trust directory” on page 26).

By default, network SWF files may not cross-script local SWF files. However, authors can grant permissions for network SWF files to cross-script local SWF files in the local-with-networking sandbox or local-trusted sandbox by using the `ActionScript Security.allowDomain()` method. A local SWF file in the local-with-file-system sandbox is not allowed to grant such permissions, because this would make it possible for the local SWF file and a network SWF file to cooperate in reading data from a local file system and sending the data to a web server.

The following diagram shows the unrestricted permissions granted to local-trusted SWF files:

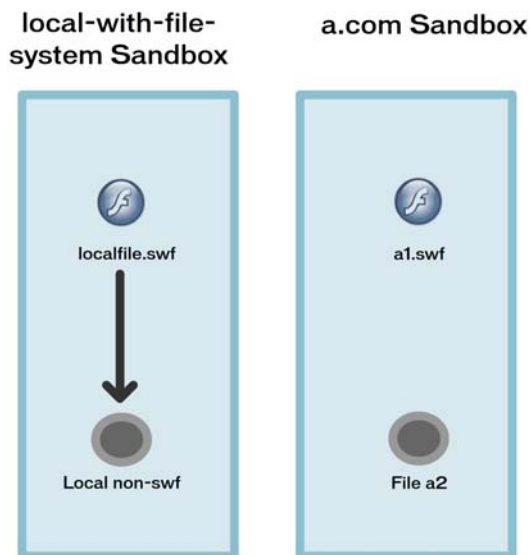
Figure 7: Default permissions for a SWF file in the local-trusted sandbox



Local-trusted SWF files may read from local files; read or send messages with any server; and script any other SWF file.

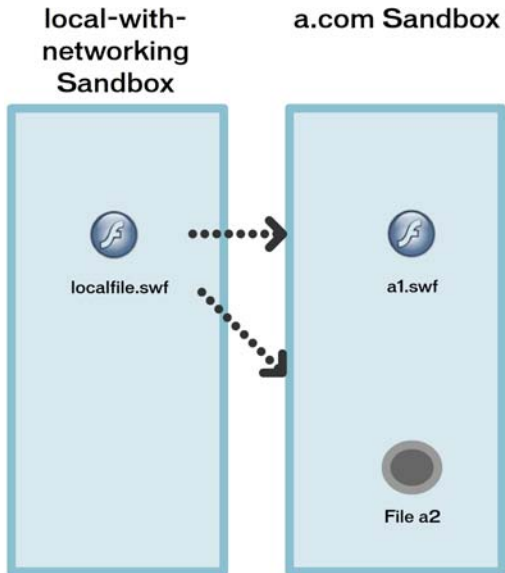
The following diagram shows the default permissions available to local SWF files in the local-with-file-system sandbox. These SWF files are not granted access to any network resource, as are files in the local-trusted sandbox, so they cannot be granted access directly to any network resource.

Figure 8: Default permissions for local-with-file-system SWF files; these cannot be modified



The following diagram shows the default permissions available to local SWF files in the local-with-networking sandbox. These SWF files are not granted permissions available to the local-trusted sandbox, but they do have default access to send to the network. Also, they can be granted read access to network sandboxes.

Figure 9: Default permissions for local-with-networking SWF files



In the absence of permissions, local-with-networking SWF files are only allowed to communicate with other local-with-networking SWF files and to send to network servers.

A local-with-networking SWF file is allowed to read (using the ActionScript `URLLoader.load()` method, for example) from the server at *a.com* if *a.com* has a policy file that permits access from all domains.

A local-with-networking SWF file is allowed to cross-script a SWF file from *a.com* (calling an ActionScript method in the *a.com* SWF file, for example) if the *a.com* SWF file calls the ActionScript `Security.allowDomain()` method to permit access from all domains. Similarly, a local-with-networking SWF file is allowed to cross-script a local-trusted SWF file if the local-trusted SWF file calls the ActionScript `Security.allowDomain()` method for all domains.

Developers can grant permissions to local-with-networking SWF files by granting permissions to all domains. The all-domains requirement reminds developers and server administrators that allowing access by local-with-networking SWF files is equivalent to allowing access by anyone, because Flash Player cannot determine the origin of a local-with-networking SWF file.

Developers can never grant permission to allow local-with-networking SWF files to read from local files. Using permissions, local-with-networking SWF files can be made completely interoperable with network SWF files and servers. Permissions also make it possible for local-trusted SWF files and network resources to communicate freely.

However, even with the maximal set of permissions, data cannot flow from local file systems to the network without going through a local-trusted SWF file.

# Server interaction

There are a number of instances in which Flash Player interacts with a remote server. Flash Player provides several means of reducing any security risks associated with server interactions, which are discussed in this section.

## Port blocking

Browsers have restrictions on HTTP access to certain ports, as does Flash Player. Specifically, browsers prevent HTTP requests to special ports that may run servers that can mistake HTTP traffic for the kind of traffic expected on that server. For example, some SMTP servers (port 25, for outgoing mail) apparently may mistake an HTTP POST for a valid SMTP request.

Port blocking also applies to Shared Library importing and the use of the `<img>` tag in text fields.

The following list shows the ActionScript APIs to which port blocking applies:

```
FileReference.download(), FileReference.upload(), Loader.load(),  
Loader.loadBytes(), navigateToURL(), NetConnection.call(),  
NetConnection.connect(), NetStream.play(),  
Security.loadPolicyFile(), sendToURL(), Sound.load(), URLLoader.load(),  
URLStream.load()
```

The following lists show which ports are blocked:

HTTP: 20 (ftp data), 21 (ftp control)

HTTP and FTP: 1 (tcpmux), 7 (echo), 9 (discard), 11 (sysstat), 13 (daytime), 15 (netstat), 17 (qotd), 19 (chargen), 22 (ssh), 23 (telnet), 25 (smtp), 37 (time), 42 (name), 43 (nickname), 53 (domain), 77 (priv-rjs), 79 (finger), 87 (ttyp), 95 (supdup), 101 (hostriame), 102 (iso-tsap), 103 (gppitnp), 104 (acr-nema), 109 (pop2), 110 (pop3), 111 (sunrpc), 113 (auth), 115 (sftp), 117 (uucp-path), 119 (nntp), 123 (ntp), 135 (loc-srv / epmap), 139 (netbios), 143 (imap2), 179 (bgp), 389 (ldap), 465 (smtp+ssl), 512 (print / exec), 513 (login), 514 (shell), 515 (printer), 526 (tempo), 530 (courier), 531 (chat), 532 (netnews), 540 (uucp), 556 (remotefs), 563 (nntp+ssl), 587 (smtp), 601 (syslog), 636 (ldap+ssl), 993 (ldap+ssl), 995 (pop3+ssl), 2049 (nfs), 4045 (lockd), 6000 (x11)

## Content-disposition headers

If Flash Player sees a "Content-Disposition: attachment" response header while downloading a SWF file, it will ignore the SWF file rather than play it. This restriction applies only to SWF files and not to other types of content, such as images, sounds, text, or XML files, policy files, etc.

The purpose of a "Content-Disposition: attachment" response header is to indicate to client software (browsers, Flash Player, e-mail clients, etc.) that the file being returned should not be rendered inline as active content. For example, if you're reading e-mail messages on a web-based service and you click a link that represents a file attached to a message, the server may respond with this header to indicate that the file should be saved and not opened.

If a server provides SWF files that are uploaded by untrusted users, the "Content-Disposition: attachment" header can help prevent those files from being executed in the server's domain.

If you control the HTTP server on which the SWF file resides, determine whether you trust the SWF file to execute in the server's domain. If so, remove the "Content-Disposition: attachment" header by changing your HTTP server's configuration.

If you do not control the HTTP server on which the SWF file resides, you have two options:

- Contact the server administrator, inquiring whether the "Content-Disposition: attachment" header can be removed.

- Download the SWF file from another SWF file that uses ActionScript 3.0. In the loading SWF file, use the `URLLoader` or `URLStream` class to download the SWF file. After the SWF file has finished downloading, call `Loader.loadBytes()` to transform the SWF data into a running SWF file. Note that this technique won't work if the downloaded SWF file refers to relative URLs, or relies on security sandbox privileges, that are not available from the loading SWF file's context.

## POST commands and file uploads

When a SWF file uses the `FileReference.browse()` and `FileReference.upload()` methods to upload a file to a server, Flash Player enforces two security rules:

- `FileReference.browse()` must be called from within a user-event handler (mouse or keyboard event).
- If the uploaded file goes to a server in a different domain than the SWF file calling `FileReference.upload()`, a URL policy file is required on the target server, trusting the domain of the SWF file. For information on URL policy files, see "Website controls (policy files)" on page 33.

Flash Player enforces these same rules any time a networking API is called to perform a POST that appears to the server to contain an upload. The format for these uploads is RFC1867, and it consists of a Content-Type of "multipart/form-data", with a section in the POST body that includes a "filename" attribute in a "Content-Disposition" header.

## User interaction

There are a number of APIs that either require user interaction (that is, they can only be invoked in response to a user action, such as a mouse or key handler) or limit user interaction.

### User interaction required

The following APIs can be initiated only through ActionScript that originates from user interaction, such as clicking the mouse or pressing a key on the keyboard.

- `FileReference.browse()`
- `FileReference.download()`
- `Clipboard.generalClipboard.setData()`
- `Clipboard.generalClipboard.setDataHandler()`
- `System.setClipboard()`

Also, data can be read from the system clipboard using the `Clipboard.generalClipboard.getData()` API, but only when it is called from within an event handler that is processing a `flash.events.Event.PASTE` event.

### User interaction limited

When content is displayed in full-screen mode, only a limited number of keys are available. These include the Tab key, the Spacebar, and the arrow keys.

# Interpreters and byte code

Flash Player includes a virtual machine to run instructions that are contained in the byte code of a SWF file. As you may know, Flash Player 9 introduced an entirely new ActionScript Virtual Machine, AVM2. The AVM2 byte codes are geared for recompilation to machine code (known as JIT or just-in-time compilation). Because Flash Player 10 includes both AVM2 and the older AVM1, it can run applications written in ActionScript 3.0 and, for backward compatibility, ActionScript 1.0 and ActionScript 2.0. The following sections discuss factors in common for all versions.

## Background

A common type of virtual machine construct is a program that interprets another program written in a particular language. This is in contrast to a somewhat different concept of a virtual machine that subdivides a real machine so as to provide the illusion of several concurrently running similar machines. These two technologies have many attributes in common, but this discussion concerns the first case, a virtual machine that runs within a traditional operating system and interprets programs written in a specific interpretive language (such as Java, JavaScript, ActionScript, Python, Perl, SmallTalk, or TCL). These languages typically have interpreters that run within several operating systems to support common applications across a broad spectrum of computers.

Java was perhaps the first to emphasize limitations that the interpreter placed on the programs that it interpreted. As both Java and web browsers grew in popularity, it became strategic to limit the actions that a program could take in the context of a browser fetching a Java program within a web page. For example, a Java program cannot delete (or even read) any local files on the user's computer. The Java program is at least partially constrained and said to run within a Java sandbox. Similarly, JavaScript, which is only distantly related to Java, also travels from a website to the user's computer to be interpreted by software typically included in the browser. This interpretation similarly limits the actions that a JavaScript program can take.

Usually, Java arrives at the browser as byte codes in a .class file, and JavaScript arrives at the browser embedded in HTML files. ActionScript is closely related to JavaScript as a language, but delivers its code to the client's computer in the form of a SWF file (with a .swf extension), which can also carry data, such as audiovisual content. The Flash Player client runtime then synchronizes the execution of the ActionScript code with the audiovisual content. The ActionScript code can also augment and override the simple audiovisual content.

## Flash Player and byte codes

The Flash authoring tools transform Flash applications completely to a byte code representation on the developer's computer (in the debugging cycle or when publishing). These byte codes are then transmitted from the website to the client computers in a SWF file (or projector file). The Flash Player AVM therefore is only an interpreter of byte codes (rather than of ActionScript), and byte codes are significantly smaller and faster to interpret. With ActionScript 3.0, authors are allowed (and encouraged) to declare the types of values that variables and parameters will use, meaning that the byte codes will have types that can be known before the program begins to run. This helps reduce potential development bugs.

ActionScript is therefore only delivered to the client's computer in the form of byte codes. The byte codes are for a stack-based virtual machine, so the meaning or validity of a particular byte code depends on the state of the stack as the code is encountered during interpretation. In a stack-oriented example, an add operation replaces the top two values on the value stack with their sum, after which the stack has one less value. There is also a control stack upon which is pushed the byte code cursor when a subroutine byte code is encountered. The subroutine finds its arguments on the value stack and places any return results there, and then returns by consuming the top element of the control stack.

Byte code verification is done in the client's computer (by the Verifier) and consists of determining the types that each byte code will consume and produce, by forming a map of the types of the values on the value stack for each byte code. Therefore, byte code programs that use invalid byte codes (given their operand types) can be detected and rejected early. This eliminates some runtime checking that would otherwise be necessary for system integrity.

The Verifier handles descriptions of classes and types, mapping of bindings, and so on. The Verifier both checks that the code is complete and valid, and provides some optimizations of the code (such as elimination of null checks at runtime). The Verifier checks all code loaded into the virtual machine; for example, the Verifier can throw error exceptions that user code can handle, but anything re-introduced is then re-verified.

At this point, ActionScript 3.0 produces native machine instructions to perform the actions of the program, having assured itself of the types of the values it will obtain as the program runs. This translation to native instructions depends mainly on knowledge that byte code verification must generate as it proceeds.

Although this process is fairly complex, it is also well defined and it is clear what the interpreter would do under the conditions that the Verifier has proven will occur. Therefore, it is relatively direct for AVM2 to map these to the appropriate machine instructions.

## Code isolation

A Flash application can potentially express actions that do any of the following:

- Read files
- Make connections over the network interface
- Contact other SWF files

Each of these operations is, in fact, ultimately performed by routines included in and controlled by the native (Adobe) Flash Player code (not by any third-party or application code), and then only after checking and enforcing all applicable access policies as established by the security model and the current runtime security controls.

Additionally, the byte codes are isolated, non-native code that cannot execute on the user's local processor. This constraint further ensures that Flash application code (SWF files) cannot affect other programs or data on the same computer.

The only machine instructions executed as a result of running an ActionScript program with Flash Player are those instructions that are part of Flash Player itself (as signed and distributed by Adobe) and those instructions produced by Flash Player by its translation of the byte codes of the SWF file (and these only after verifying compatibility of such instructions with the data types produced by preceding instructions, and applying Flash Player security policies).

## Disk, memory, and processor protections

Flash Player includes security protections for disk data and memory usage on the client computer.

### Disk storage protections

The only type of persistent storage is through *shared objects*, which are embodied as files in directories whose names are related to that of the specific owning SWF files. An ActionScript program cannot write, modify, or delete any files on the client computer other than shared objects, and it can only access shared objects under the established settings per domain. There are no primitives (no mechanisms) available to Flash applications that can read, create, modify, or delete directories or files.

Shared objects are therefore normally associated with a given SWF file's particular domain and sandbox. They actually use a finer-grained model than the usual concept of a sandbox, so that they are (by default) associated with an individual SWF file within the sandbox. An author can create shared objects, however, to cover the scope of an entire sandbox by providing a (non-default) "localPath" when creating them. The file path to the shared object data contains pseudo-random data so that the storage is not in a predictable location. Flash Player can also store shared object data loaded from HTTP and HTTPS paths in separate locations. So a movie loaded by HTTPS can specify that its shared objects cannot be accessed by a movie loaded via HTTP, even if the URL is the same otherwise.

Flash Player helps limit potential denial-of-service attacks involving disk space. Disk space is conserved through limits automatically set by Flash Player (the default is 100K of disk space for each domain). Capabilities exist for the author to include the ability for the Flash application to prompt the user for more, or Flash Player automatically prompts the user if an attempt is made to store data that exceeds the limit. In either case, the disk space limit is enforced by Flash Player until the user gives explicit permission for an increased allotment for that domain.

A Flash Player user interface lets the user set persistent disk storage allocation settings (per domain) that limit the use (amount) of permanent disk storage by the Flash applications within that domain. The user can also indicate a global setting for how much disk storage any new Flash applications (from domains not yet individually specified) can use. (For more information, see "Storage settings" on page 31.)

## Memory usage protections and processor quotas

Flash Player contains memory and processor safeguards that help prevent Flash applications from taking control of excess system resources for an indefinite period of time. For example, Flash Player can detect an application that is "stuck" in a script for more than 15 seconds (possibly because it is running an infinite loop) and select it for termination by prompting the user. The resources used by the application are immediately released when the application closes.

Flash Player uses a garbage collector engine common to other Adobe products. The processing of new allocation requests always first ensures that memory has been cleared so that the new usage always obtains only clean memory and cannot view any prior data.

## Permission controls

The range of stakeholders and differences in their perspectives over security and privacy issues requires a layered approach to the specification (and checking) of access controls. Flash Player offers a range of configuration mechanisms to address corporate security policy issues and user privacy concerns without significantly reducing the key benefits of Flash Player.

The chosen options (and defaults) in the configuration files, along with the overall Flash Player security model and other administrator, developer, user, and default options, may make Flash Player more secure than an individual stakeholder desires with respect to local files and local shared files, but should not be less secure. This may also mean that some new restrictions are enforced on some legacy applications when they are executed in Flash Player 10.

Some security control features in Flash Player target user choices, and some target the modern corporate and enterprise environments, such as when the IT department would like to install Flash Player across the enterprise but has concerns about IT security and privacy. To help address these types of requirements, Flash Player provides various installation-time configuration choices (some new). For example, some organizations do not want Flash Player to have access to the computer's audio and video hardware; other environments do not want Flash Player to have any read or write access to the local file system.

For a table with descriptions of permission controls, see "Overview of permission controls" on page 11.

# Administrative user controls

The system administrator of the internal network (where users may execute Flash applications) can designate rules about Flash Player options and Flash application access with the `mms.cfg` file and Global Flash Player Trust files.

## The `mms.cfg` file

The primary purpose for the `mms.cfg` file is to support the corporate and enterprise environments where the IT department would like to install Flash Player across the enterprise, while enforcing some common global security and privacy settings (supported with installation-time configuration choices).

On operating systems that support the concept of user security levels, the file is flagged as requiring system administrator (or root) permissions to modify or delete it. On Mac OS X systems using `mms.cfg`, the security configuration file is located at `/Library/Application Support/Macromedia/mms.cfg`. On Microsoft Windows XP and Vista, the file is located in the Flash Player folder within the system directory (for example, `C:\windows\system32\macromed\flash\mms.cfg` on a default Windows XP installation).

When Flash Player starts, it reads its security settings from this file, and uses them to limit or change functionality. There are several types of controls included in the `mms.cfg` file, such as data loading controls, privacy controls, update controls, and socket controls. Details on the `mms.cfg` file settings are available at [http://www.adobe.com/go/flash\\_player\\_admin](http://www.adobe.com/go/flash_player_admin).

Some of the `mms.cfg` options affect the settings in the tabs in the Flash Player Settings UI (see “Settings UI” on page 29), and may override settings specified by the end user:

- **Display**— This tab lets the user enable or disable graphics hardware acceleration. There is no `mms.cfg` setting related to this tab.
- **Privacy**—If the `AVHardwareDisable` `mms.cfg` setting is set to `true`, all user actions related to this tab are ignored. The tab does, however, appear functional.
- **Local Storage**—If the `LocalStorageLimit` `mms.cfg` setting is set, this tab shows the limit specified in that option. However, the user can use this tab as if the limit does not exist. If the user selects settings higher than the limit set in the configuration file, the user’s settings are ignored. If the user sets more restrictive settings, they are honored (and displayed the next time the Settings dialog box is invoked). The local file storage limit is best obtained from the Settings dialog box, because this security setting is just a maximum value, and the user may have set a lower limit.
- **Microphone**—If the `AVHardwareDisable` `mms.cfg` setting is set to `true`, the recording level meter is disabled. All other controls appear to work, but their values are ignored.
- **Camera**—If the `AVHardwareDisable` `mms.cfg` setting is set to `true`, clicking the camera tab does *not* bring up a thumbnail of what the camera is seeing.

Flash authors can query the `Capabilities.avHardwareDisable` and `Capabilities.localFileReadDisable` properties to change the behavior of their Flash applications based on features disabled by security; however, they cannot modify those values. Authors should be familiar with the range of information (such as pixel aspect ratio, screen size, color support, and so on) available with the `Capabilities` class (all properties of this object are read-only). For additional information on the `Capabilities` class, see the *ActionScript 3.0 Language Reference* at [www.adobe.com/go/learn\\_flashcs4\\_langref\\_en](http://www.adobe.com/go/learn_flashcs4_langref_en).

## Global Flash Player Trust directory

Local files are placed in the local-trusted sandbox only at the direction of a user, an administrative user, or an installer program. Administrative users, and installer programs that are run with administrative rights, have the option of designating trust by creating configuration files in a directory called the Global Flash Player Trust directory. Flash Player reads all configuration files in this directory; the configuration files may have any names, but the recommended convention is to use the extension ".cfg", and choose a filename that describes the files being trusted, so as to avoid name collisions. Each line of each configuration file lists a local path that is to be trusted. These paths may name individual files or entire directories. If a directory is specified, the directory and all of its descendant directories are trusted. Configuration files in the Global Flash Player Trust directory affect all users of the computer.

The Global Flash Player Trust directory is alongside the `mms.cfg` file (the system configuration file discussed in the previous section), in the following location, which requires administrator access:

- **Windows XP and Vista:** `system\Macromed\Flash\FlashPlayerTrust`  
(for example, `C:\windows\system32\Macromed\Flash\FlashPlayerTrust` on a default Windows XP installation)
- **Mac:** `app support/Macromedia/FlashPlayerTrust`  
(for example, `/Library/Application Support/Macromedia/FlashPlayerTrust`)
- **Linux:** `/etc/adobe`

There are also individual User Flash Player Trust directories, used by installers to register an application for specific users of a computer (see "User Flash Player Trust directory" on page 33). The Global and User Flash Player Trust directories have different security precedence in relation to each other and to settings in the `mms.cfg` file (see "Hierarchy of local file security controls" on page 45).

Depending on whether Flash Player will be embedded in a non-browser application, one of two strategies may be appropriate: register SWF files and HTML files to be trusted, or register applications to be trusted. Only applications that embed the browser plug-ins can be trusted—the stand-alone players and standard browsers do not check to see if they have been trusted.

## User controls

Flash Player provides users with four different mechanisms for setting permissions. The Settings UI provides a quick, interactive mechanism for configuring the settings for the domain of a SWF file being displayed. The Settings Manager presents a more detailed interface than the settings UI and provides the ability to make global changes that affect permissions for many or all domains. Additionally, at the moment a new permission is requested by a SWF file, requiring runtime decisions concerning security or privacy, Flash Player presents dialog boxes that allow users to adjust some Flash Player settings. Also, users can set permissions for specific applications in the Flash Player Trust directory.

In the case of the Settings Manager, although it appears that these settings are configured within the context of a web page, Flash Player actually retrieves the settings locally and only displays them in the apparent context of the web page being viewed. Adobe does not collect any information about Flash Player user settings or preferences, and at no time are the user's settings transmitted off of the user's computer. The Settings Manager appears to be an application hosted on the Adobe web page ([www.adobe.com](http://www.adobe.com)), but it is in reality a SWF file served from the Adobe domain and running on the user machine. Although the Settings user interface (UI) and the small context-sensitive runtime Settings dialog boxes appear superimposed over other Flash Player content, Flash Player runs these in separate sandboxes.

## Settings Manager

The Settings Manager allows the individual user to specify various security, privacy, and resource usage settings for Flash applications executing on their client computer. For example, the user can control application access to select facilities (such as their camera and microphone), or control the amount of disk space allotted to a SWF file's domain. The settings it manages are persistent and controlled by the user.

The user can indicate their personal choices for their Flash Player settings in a number of areas, either globally (for Flash Player itself and all Flash applications) or specifically (applying to specific domains only). To designate choices, the user can select from the six tab categories along the top of the Settings Manager dialog box:

- Global Privacy Settings
- Global Storage Settings
- Global Security Settings
- Global Notifications Settings
- Website Privacy Settings
- Website Storage Settings

You can view the Settings Manager at:

[http://www.adobe.com/support/documentation/en/flashplayer/help/settings\\_manager.html](http://www.adobe.com/support/documentation/en/flashplayer/help/settings_manager.html)

The following figures show samples of the Settings Manager.

Figure 10: Global Privacy Settings tab

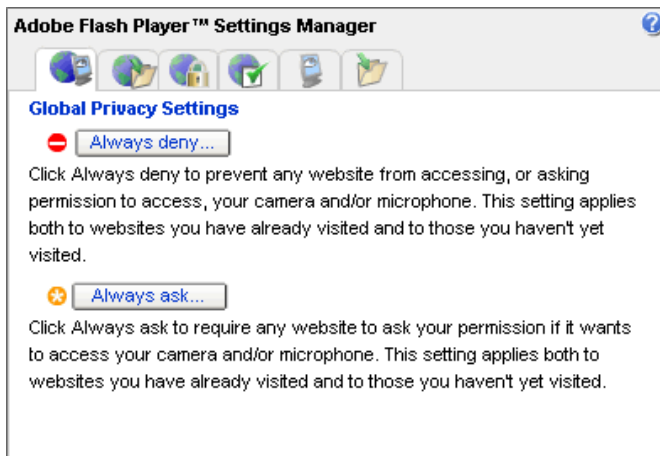


Figure 11: Global Storage Settings tab

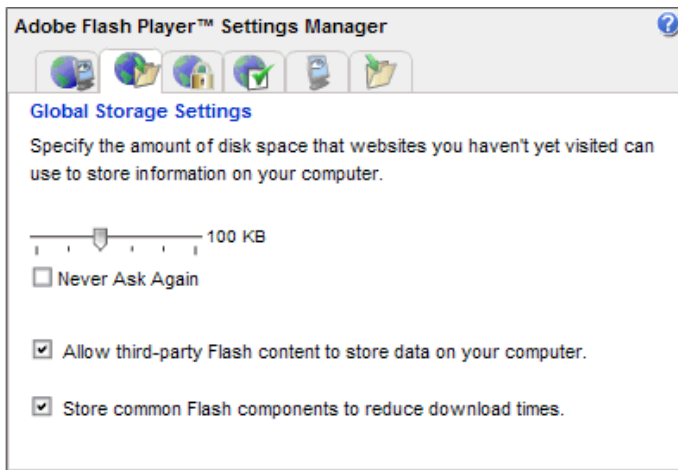


Figure 12: Global Security Settings tab



Figure 13: Global Notification Settings tab

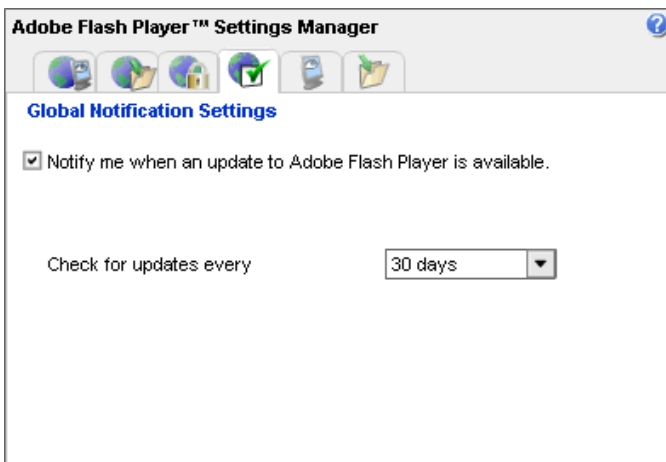


Figure 14: Website Privacy Settings tab

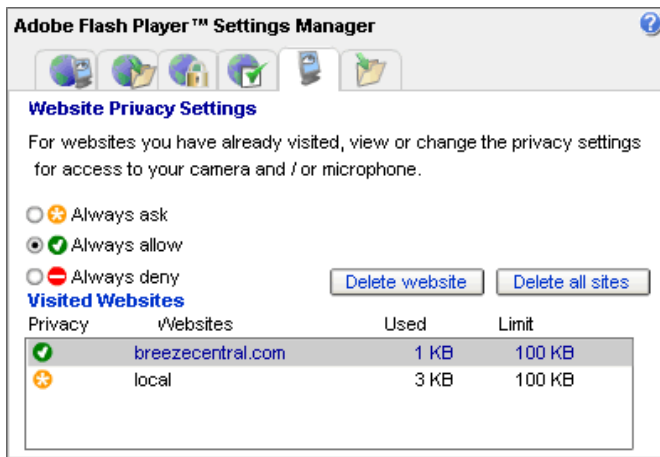


Figure 15: Website Storage Settings tab



All of the settings information managed by the Settings Manager is retained in persistent Flash Player data, and cannot be accessed via ActionScript. These objects are accessed only to modify the settings on the local machine in direct response to a user request through the Settings Manager. The content is not reviewed by Adobe, nor is it transmitted across the network.

## Settings UI

These settings are most commonly accessed by right-clicking (or, on the Macintosh, Command-clicking) an executing SWF file and selecting the Settings option. The Settings user interface (UI) provides users with the ability to modify settings and security controls for the domain of the main SWF file running in the player while the Settings UI is displayed, and it also provides an access point to the Settings Manager by using the Advanced button on the Privacy tab.

In the following circumstances, the Settings context menu is disabled, and the Settings UI can't be displayed. If user approval of an action is required, Flash Player, by default, treats the operation as if Deny were selected.

- If the application is too small to display the Settings UI or a runtime dialog box (the minimum size is 215 x 138 pixels)
- If the window mode (set with the HTML "wmode" attribute) is "direct" or "gpu"
- On Linux, if the window mode is "transparent" or "opaque"

There are five tabs in the Settings UI, which are shown in the following sections.

## Display settings

An interface lets the user specify whether to enable hardware acceleration on their computer.

Figure 16: Display settings dialog box



## Privacy settings

An interface lets the user set persistent privacy settings per domain that control access by Flash applications to the system's camera and microphone. The default is to ask the user, and Flash Player denies access to the camera and microphone if the user does not explicitly grant access. In the example shown below, the user has chosen to permanently allow access from [website]. (The user can change this setting by using the Global Privacy Settings tab in the Settings Manager, if desired).

Figure 17: Privacy settings dialog box



## Camera settings

An interface lets the user set camera information. This panel does not have direct security or privacy implication. Use the Privacy settings to control whether the camera is accessible to Flash applications.

Figure 18: Camera information dialog box



## Microphone settings

An interface lets the user set microphone configuration. This panel does not have direct security or privacy implications. Use the Privacy settings to control whether the microphone is accessible to Flash applications.

Figure 19: Microphone settings dialog box



## Storage settings

Users can also set storage limits for all Flash applications from the current domain using the following dialog box. Setting the storage to 0 kilobytes causes Flash Player to prompt the user if a Flash application requests a shared object. If the user selects 0 KB and selects the *Never Ask Again* option, Flash Player disables storage by the current domain.

Figure 20: Individual Flash application storage settings dialog box



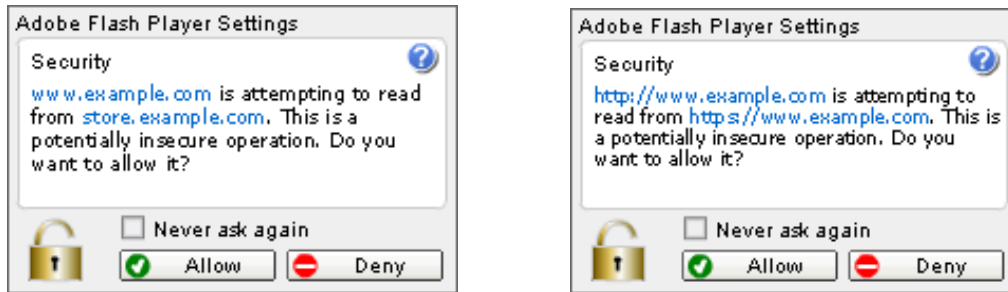
## Runtime dialog boxes

Flash Player tries to minimize any exposure of security decisions to end users. However, some runtime behavior may require user intervention or approval, such as for privacy-related issues (cameras and microphones), or when older applications attempt access that is no longer permitted by default.

## Domain match and HTTP/HTTPS warnings

One or both of the following screens may appear for users when they run an older (Flash Player 6 or earlier) application that makes a data loading request that is not permitted in current versions of Flash Player, but which would have been permitted in Flash Player 6. The dialog box on the left is an example for a request that was formerly acceptable due to the similarity of the domain names. The dialog box on the right is an example of a request from an insecure (HTTP) site to a secure (HTTPS) site.

Figure 21: Domain match and HTTP/HTTPS warnings

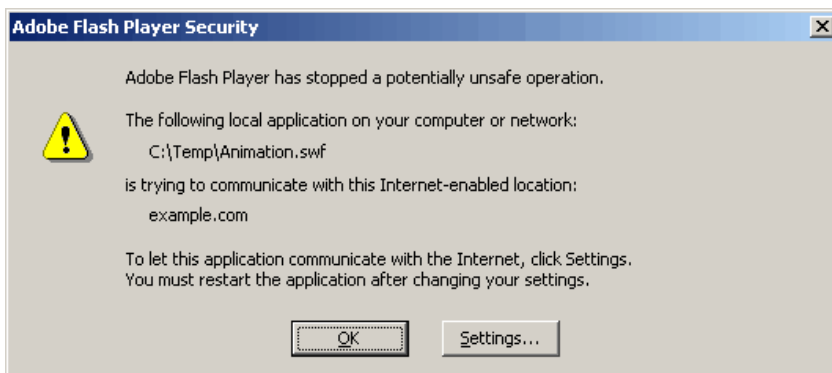


If the user clicks Allow, the request succeeds. If the user clicks Deny, the request fails. The dialog boxes do not appear again if the user selects the Never Ask Again check box.

## Network Access Warning

The following dialog box appears if an untrusted local SWF file (in the local-with-file-system sandbox) tries to access the network. The dialog box appears only for operations that would have succeeded in Flash Player 7 or earlier, and are attempted by SWF files that were produced for versions of Flash Player prior to Flash Player 8. The dialog box appears no more than one time for each time the player is executed. Also, it does not appear if either the administrator or user options have been set to request silent failures. Authors cannot disable this by requesting silent failures or successes; if authors don't avoid such prohibited actions, the dialog box appears based on user controls (via the Settings Manager) and administrative controls (via the mms.cfg file).

Figure 22: Network access warning



## User Flash Player Trust directory

The User Flash Player Trust directory is in the following location (which are specific to the current user, not to other users who log on to the computer):

- Windows Vista: `AppData\Roaming\Macromedia\Flash Player\#Security\FlashPlayerTrust` (for example: `C:\Users\JohnD\AppData\Roaming\Macromedia\Flash Player\#Security\FlashPlayerTrust`)
- Windows XP: `app data\Macromedia\Flash Player\#Security\FlashPlayerTrust` (for example, `C:\Documents and Settings\JohnD\Application Data\Macromedia\Flash Player\#Security\FlashPlayerTrust`)
- Mac: `app data/Macromedia/Flash Player/#Security/FlashPlayerTrust` (for example, `/Users/JohnD/Library/Preferences/Macromedia/Flash Player/#Security/FlashPlayerTrust`)
- Linux: `$HOME + "/.macromedia/Flash_Player/#Security/FlashPlayerTrust` (for example, `/home/JohnD/.macromedia/Flash_Player/#Security/FlashPlayerTrust`)

If a user without administrative rights installs an application in their own portion of the system, the User Flash Player Trust directory lets the installer register the application as trusted for that user. Similar capability is provided interactively to the user with the Settings Manager.

There is also a Global Flash Player Trust directory, used by the administrative user or installers to register an application for *all* users of a computer (see “Global Flash Player Trust directory” on page 26). These two sets of trust directories have different security precedence (see “Hierarchy of local file security controls” on page 45).

## Website controls (policy files)

When a SWF file downloads data from a server, it does so with certain credentials from the user, which may include cookies, passwords, private network access, etc. This is why, by default, a SWF file can download data only from servers in its own domain. If the administrator of a server wishes to permit SWF files from other domains to access data from that server (using any user credentials that the server may have provided), the administrator can create policy files specifying such permissions. This is always safe for data that is freely available on the public Internet, but may be risky for data that requires user authentication.

There are multiple ways in which a SWF file can read data from the web directly into ActionScript variables, such as by using the ActionScript `URLLoader.load()` method or by using the ActionScript socket object. The default for network sandboxes is to restrict read permissions to data sources from the origin domain (exact match) of the SWF file. The system administrator of a domain (website) that hosts resources used by Flash applications can designate what resources can be downloaded from their site using two types of cross-domain policy files:

- Most networking in ActionScript is URL-based, using HTTP, HTTPS, and FTP; this type of networking is authorized by *URL policy files*.
- ActionScript also supports the `Socket` and `XMLSocket` classes, which enable networking directly at the lower TCP socket level; socket-level connections are authorized by *socket policy files*.

All policy files have two essential attributes: permissions (the list of domains authorized to connect) and scope (the set of resources to which access is allowed). Socket policy files work identically to URL policy files as far as permissions are concerned, but differ with respect to scope. The scope of a URL policy file is determined by its location, but the scope of a socket policy file is determined by the file's contents—specifically, by `to-ports` attributes that specify the TCP ports to which socket connections may be made.

The following two sections describe URL policy files; for a discussion of socket policy files, see “Socket policy files” on page 37. For more information on policy file usage and syntax, see the section entitled “Website controls (policy files)” in the Security chapter in *Programming ActionScript 3.0* ([www.adobe.com/go/flashcs4\\_prog\\_as3\\_security\\_en](http://www.adobe.com/go/flashcs4_prog_as3_security_en)) For a discussion of security issues that should be reviewed prior to deployment of a policy file, see “Cross-domain policy file usage recommendations for Flash Player” at [www.adobe.com/devnet/flashplayer/articles/cross\\_domain\\_policy.html](http://www.adobe.com/devnet/flashplayer/articles/cross_domain_policy.html).

## URL policy files

For a SWF file to retrieve data from a domain other than its own, the sandbox from which it is to be fetched (the provider domain) must include a policy file that permits that action. These are the *URL policy files*, which allow a website administrator to specify (as consulted by and enforced by Flash Player) that the documents that domain serves be freely available to all domains, or available to specific other domains (such as by specifying an exact URL or domain, or specifying a set of related URLs or subdomains using wildcard notation).

### About URL policy files

URL policy files grant cross-domain permissions for reading data. They permit operations that are not permitted by default. It is important to understand what a policy file enables, rather than simply regarding the default rules as a potential barrier and creating a policy file without considering the consequences.

Note: If you plan to implement policy files, you must also declare a meta-policy that specifies the locations on your server that are permitted to serve policy files. See the following section, “Meta-policies,” for more information.

The URL policy file mechanism is a simple XML file that does the following:

- Modifies the read and send permissions for data between sandboxes and across the network. It does not apply to cross-scripting of SWF files.
- Applies only to the protocol and port of the server, rather than opening up an entire domain.

The URL policy file is located, by default, in the root directory of the target server, with the name *crossdomain.xml* (for example, at *www.example.com/crossdomain.xml*). Flash application developers can specify another location by calling the ActionScript `loadPolicyFile()` method. The URL policy file located in the root directory is referred to as the *master policy file*.

The scope of the permissions defined in a URL policy file includes all resources within the directory and within nested subdirectories. Policy files are only able to grant access to a resource; they cannot restrict access. The search order of the policy files does not need to be well-defined, because there is no precedence among them. If a resource is within the scope of more than one policy file, any one of the policy files may grant access.

Note: There are two distinct (basically unrelated) controls that apply to cross-domain actions that might be confused: URL policy files (covered here) are managed by domain administrators to allow network access to data; the ActionScript `allowDomain()` method is specified by authors to allow client-side cross-domain scripting (see “Developer controls” on page 39).

When a domain is specified in a URL policy file, the site declares that it is willing to allow the operators of any servers in that domain to obtain any document on the server where the policy file resides. Often this is the effect that you want. For example, if a site serves only public documents from a particular server, the site should have no qualms about who can obtain documents on that server, because anyone can simply visit the server and download files that they want. In this situation, it is safe to open the server to all domains (use of a single asterisk "\*" as a pure wildcard is supported):

```
<cross-domain-policy>
  <allow-access-from domain="*" />
</cross-domain-policy>
```

Alternatively, if the site serves private documents or anything that requires some form of authentication (such as a password), or if the server is behind a firewall where only certain users can access it, it is risky to put a public policy file on that server. Doing so would enable Flash applications to download documents from the server whenever they run on the computers of users that the server trusts. These applications could potentially reveal private data from the server to people whom the user or website administrator does not trust.

If it is necessary to create a policy file in such a situation, it is best that the file permit access for domains that you specifically know need access. For example, if you run a server at example1.com that provides XML data, and you serve Flash applications from www.example2.com, which needs to load the XML data, you could put a policy file on example1.com that enables access specifically from www.example2.com.

```
<!-- http://www.example1.com/crossdomain.xml -->
<cross-domain-policy>
  <allow-access-from domain="www.example2.com" />
</cross-domain-policy>
```

If you run a server that serves especially sensitive documents, and you know there are no SWF files on other servers that need to access those documents, it is safest to create a *deny-all* policy file on your server:

```
<cross-domain-policy>
</cross-domain-policy>
```

A Boolean *secure* attribute is supported in the `<allow-access-from>` directives of policy files on HTTPS servers. When `secure="true"` is specified, SWF files from the domain or domains listed in that `<allow-access-from>` directive are permitted to retrieve data only if those SWF files are themselves retrieved over HTTPS. Here is an example of an HTTPS policy file using this feature:

```
<!-- https://www.example2.com/crossdomain.xml -->
<cross-domain-policy>
  <allow-access-from domain="example1.com" secure="true">
</cross-domain-policy>
```

## Meta-policies

It is important for a server administrator to be able to limit the creation of URL policy files (thus avoiding potential data vulnerabilities), and to easily find all policy files on the server at any given time (thus auditing the current set of permissions). Some servers may allow content contributions from personnel who lack administrative rights, and other servers may even serve content defined by end users. It is often not appropriate to allow these kinds of contributors to create policy files.

By choosing an appropriate meta-policy, an administrator can completely prohibit the creation of policy files, completely allow the creation of policy files in any location, or take an intermediate approach, permitting only certain policy files by whatever criteria are desired.

A meta-policy is implemented by declaring an option placed inside the master policy file or in an HTTP response header.

- In the master policy file, the meta-policy is an attribute of the `<site-control>` tag called `permitted-cross-domain-policies`, as shown in the following example:

```

<cross-domain-policy>
  <site-control permitted-cross-domain-policies="by-content-type"/>
  <allow-access-from domain="www.example2.com" />
</cross-domain-policy>

```

- To declare a meta-policy in an HTTP response header, use the header X-Permitted-Cross-Domain-Policies, as shown below:

```

HTTP/1.1 200 OK
X-Permitted-Cross-Domain-Policies: none

```

You should choose only one of the two mechanisms to declare a meta-policy. However, if you accidentally declare a meta-policy in both places, the meta-policy declared in the HTTP response header will take precedence over the meta-policy declared in the master policy file.

The following table briefly describes the available meta-policy options.

**Table 3: URL meta-policy options**

Meta-policy option	Notes
All	All policy files are allowed. This meta-policy is appropriate only for servers that are on the public Internet and do not serve any content that requires login credentials.
by-content-type	All policy files whose Content-Type is exactly text/x-cross-domain-policy are allowed. No other policy files are allowed. This meta-policy is available only for HTTP and HTTPS servers.
by-ftp-filename	Only policy files named crossdomain.xml ( <i>i.e.</i> , those files whose URLs end with /crossdomain.xml) are allowed. This meta-policy is only available for FTP servers.
master-only	Only the master policy file (named crossdomain.xml and located in the root directory) is allowed. This is the default meta-policy.
None	No policy files are allowed. If this meta-policy is declared in the master policy file, then the master policy file is permitted to exist solely for the purpose of declaring this meta-policy, and any <allow-access-from> directives that it contains will be ignored.
none-this-response	This is a special meta-policy that can only be specified in an HTTP response header. It helps prevent that particular HTTP response from being interpreted as a policy file. Unlike all other meta-policies, none-this-response affects only a single HTTP response, rather than declaring an overall meta-policy for the entire server. Also unlike all other meta-policies, none-this-response can appear in combination with any other meta-policy.

The default meta-policy for URL policy files, "master-only," favors security by ensuring that a site must explicitly permit meta-policies at locations other than the root directory.

For more information on creating and declaring meta-policies, see [www.adobe.com/devnet/flashplayer/articles/fplayer9\\_security\\_03.html](http://www.adobe.com/devnet/flashplayer/articles/fplayer9_security_03.html).

## Policy file timeouts

When a SWF file checks for a master policy file, Flash Player waits three seconds for a server response. If a response isn't received, Flash Player assumes that no master policy file exists; any meta-policies within the master policy file are ignored. However, there is no default timeout value for calls to `loadPolicyFile()`; Flash Player assumes that the file being called exists, and waits as long as necessary to load it. Therefore, if you are implementing policy files for specific purposes, make sure the developers use `loadPolicyFile()` to call the master policy file explicitly.

## Header sending permissions

Flash Player offers the ability to send HTTP requests with custom HTTP request headers specified in ActionScript. Certain fundamental HTTP request headers are disallowed. In addition, when a SWF file wishes to send custom HTTP headers anywhere other than its own host of origin, there must be a policy file on the HTTP server to which the request is being sent. This policy file must enumerate the SWF file's host of origin (or a larger set of hosts) as being allowed to send custom request headers to that host.

The policy files that grant header-sending permission are the same as the policy files that grant data-loading permission, but header-sending permissions are expressed using a different XML tag, meaning that the two types of permissions are controlled independently of each other. Header-sending permissions are declared in the tag `<allow-http-request-headers-from>`, while data-loading permissions are declared in the tag `<allow-access-from>`. The `<allow-http-request-headers-from>` tag has the required attribute "domain" and the optional attribute "secure", which function exactly like the corresponding attributes of `<allow-access-from>`. In addition, `<allow-http-request-headers-from>` has the required attribute "headers", which specifies the particular headers that may be sent from the host(s) named in the "domain" attribute of that tag.

Here is an example of a policy file that permits certain custom request headers:

```
<cross-domain-policy>

  <!-- Establish a meta-policy -->
  <site-control permitted-cross-domain-policies="master-only"/>

  <!-- Allow *.myDomain.com to send any headers desired -->
  <allow-http-request-headers-from domain="*.myDomain.com" headers="*" />

  <!-- Allow any SWF to send the "X-Howdy" header -->
  <allow-http-request-headers-from domain="*" headers="X-Howdy" />

  <!-- Data loading permissions are separate, and don't affect headers -->
  <allow-access-from domain="another.com" />

</cross-domain-policy>
```

## Socket policy files

ActionScript supports the `Socket` and `XMLSocket` classes, which enables networking directly at the lower TCP socket level. These socket-level connections are authorized by socket policy files. Socket policy files serve the same purpose as URL policy files: they authorize connections from client content. Unlike a URL policy file, which authorizes data loading from its own HTTP, HTTPS, or FTP server, a socket policy file authorizes socket connections to its own host.

A socket policy file is required if you want to enable socket access. URL policy files do not permit socket access (although they did so in previous versions of Flash Player).

The usual location for a socket policy file is a dedicated *socket master policy file* server on port 843, but two other options are available: a dedicated policy file server on a port other than 843 (requiring a call to `loadPolicyFile()` in SWF files that connect), or a dual-mode server that both serves policy files and handles the main connections made by ActionScript. If the socket policy file is hosted from the master policy file location, be sure to include meta-policies to specify where socket policy files are allowed to be located (see the next section, “Socket meta-policies,” for more information).

You can serve a socket policy file from a port other than 843, but note that a socket policy file from port 1024 or above may authorize connections only to ports 1024 and above. A socket policy file served from a port below 1024 may authorize connections to any port on the host.

While the scope of a URL policy file is determined by its location, the scope of a socket policy file is determined by the file's contents—specifically, by `to-ports` attributes that specify the TCP ports to which socket connections may be made.

The format of a socket policy file is the same as that of a URL policy file, with additional attributes, as shown below.

```
<cross-domain-policy>
  <allow-access-from domain="mysite.com" to-ports="999,8080-8082"/>
</cross-domain-policy>
```

Like URL policy files served from HTTPS servers, socket policy files also support a Boolean `secure` attribute in the `<allow-access-from>` directive. When `secure="true"` is specified, SWF files from the domain or domains listed in that `<allow-access-from>` directive are permitted to retrieve data only if those SWF files are retrieved over HTTPS. This attribute should be used only in local socket policy files (that is, those served from the same machine as the Flash Player is running on). Here is an example of a local socket policy file using this feature:

```
<cross-domain-policy>
  <allow-access-from domain="my.com" secure="true" to-ports="3050"/>
</cross-domain-policy>
```

The only permission granted by such a policy file would be for a SWF file loaded over HTTPS from `my.com` to connect to port 3050. A SWF file loaded over HTTP from `my.com` would not be granted any permissions by this particular policy file.

For more information on the use of the `secure` attribute, see [www.adobe.com/devnet/flashplayer/articles/fplayer9\\_security\\_04.html](http://www.adobe.com/devnet/flashplayer/articles/fplayer9_security_04.html).

## Socket meta-policies

Socket meta-policies are similar to URL meta-policies, with the following differences:

- Socket meta-policies affect socket policy files, not URL policy files.
- A socket meta-policy affects an entire host, not just a single URL server.
- Socket meta-policies can be declared only in a socket master policy file. The syntax is the same as for declaring a meta-policy in an URL master policy file, using the `<site-control>` tag. Socket meta-policies cannot be declared in HTTP response headers, as HTTP is not involved.
- A different set of meta-policies is available for use as socket meta-policies than is available for use as URL meta-policies.
- The default socket meta-policy is “all,” while the default URL meta-policy is “master-only.”

The following table briefly describes the available socket meta-policy options.

Table 4: Socket meta-policy options

Meta-policy option	Notes
all	Socket policy files are allowed from any port on this host. This is the default meta-policy.
master-only	Only the socket master policy file, served from port 843, is allowed on this host.
none	No socket policy files are allowed on this host. If this meta-policy is declared in the master policy file, then the master policy file is permitted to exist solely for the purpose of declaring this meta-policy, and any <code>&lt;allow-access-from&gt;</code> directives that it contains will be ignored.

For more information on socket policy files, socket master policy files, and socket meta-policies, see [www.adobe.com/devnet/flashplayer/articles/fplayer9\\_security\\_04.html](http://www.adobe.com/devnet/flashplayer/articles/fplayer9_security_04.html).

### Policy file timeouts

As with URL policy files, when a SWF file checks for a master policy file, Flash Player waits three seconds for a server response. If a response isn't received, Flash Player assumes that no master policy file exists; any meta-policies within the master policy file are ignored. However, there is no default timeout value for calls to `loadPolicyFile()`; Flash Player assumes that the file being called exists, and waits as long as necessary to load it. Therefore, if you are implementing policy files for specific purposes, make sure the developers use `loadPolicyFile()` to call the master policy file explicitly.

### Socket timeouts

There is an additional socket timeout, which regulates how long Flash Player will wait before signaling connection failure to ActionScript. All failures, including lack of network connection, lack of response from the server, active refusals from the server, and lack of socket policy file permission, will be reported in this same amount of time. The default for this timeout is 20 seconds; developers can change it using the `Socket.timeout` and `XMLSocket.timeout` properties. Longer times favor greater tolerance of marginal network conditions, at the expense of longer delays in detecting problems. Shorter times favor quicker detection of problems, at the expense of reduced tolerance for temporary problems.

### Policy file logging

The policy file log shows messages for every event relating to policy files, including attempts to retrieve them, successes and failures in processing them, and the outcome of the requests that depend on them. Steps for implementing policy logging are located at [www.adobe.com/devnet/flashplayer/articles/fplayer9\\_security\\_05.html#\\_Using\\_Logging](http://www.adobe.com/devnet/flashplayer/articles/fplayer9_security_05.html#_Using_Logging). A complete reference to the messages you will find in the policy file log can be found at [www.adobe.com/devnet/flashplayer/articles/fplayer9\\_security\\_07.html](http://www.adobe.com/devnet/flashplayer/articles/fplayer9_security_07.html).

## Developer controls

Developers have many ways that they can designate or control some of the security aspects that apply to a Flash application that they are publishing. Security options available to developers are discussed in the following sections.

## Permission mechanisms

There are a number of mechanisms available for the author to designate aspects of the desired security environment for the resulting Flash application. *Permission mechanisms* are APIs that provide for altering the calling application's security environment. The following table and the sections that follow it describe these APIs.

**Table 5: Permission mechanism APIs**

API Names	Description
<code>Security.allowDomain()</code> <code>Security.allowInsecureDomain()</code>	Permits SWF files from a specified sandbox to read (or cross-script) the calling SWF file. The <code>allowInsecureDomain()</code> API also permits access to HTTPS SWF files by non-HTTPS SWF files.
<code>Security.loadPolicyFile()</code>	Informs Flash Player of the location of a URL policy file in a non-default location, or the location of a socket policy file.
<code>Security.exactSettings</code>	Determines whether exact or old-style superdomain rules are used to determine the scope of shared objects and privacy settings. The value of this property is scoped to the calling SWF file only.
<code>LocalConnection.allowDomain()</code> <code>LocalConnection.allowInsecureDomain()</code>	Specifies one or more domains that can send LocalConnection calls to a LocalConnection instance. The <code>LocalConnection.allowDomain()</code> method functions like <code>Security.allowDomain()</code> : receiving code calls <code>allowDomain()</code> . Like <code>Security.allowDomain()</code> , <code>LocalConnection.allowDomain()</code> supports a single asterisk "*" as a global wildcard.

**Table 6: HTML parameter permission mechanism APIs**

API Names	Description
<code>allowNetworking</code>	<p>This HTML parameter governs a number of ActionScript APIs. It has the following possible values:</p> <ul style="list-style-type: none"> <li>▪ "all"—the default. No networking restrictions; player behaves normally.</li> <li>▪ "internal"—SWF files may not call browser navigation or browser interaction APIs, but may call any other networking APIs.</li> <li>▪ "none"—SWF files may not call any networking APIs, nor any SWF-to-SWF communication APIs.</li> </ul> <p>Note: Using the value "internal" or "none" is not recommended for SWF files from the same domain as their enclosing HTML pages.</p>

API Names	Description
<code>allowScriptAccess</code>	<p>This HTML parameter governs whether ActionScript can call JavaScript (or other script) in the HTML page (container). It has the following possible values:</p> <ul style="list-style-type: none"> <li>▪ "never"— the SWF file cannot communicate with any HTML page. Note that using this value is deprecated and not recommended, and shouldn't be necessary if you don't serve untrusted SWF files from your own domain. If you do need to serve untrusted SWF files, Adobe recommends that you create a distinct subdomain and place all untrusted content there.</li> <li>▪ "sameDomain"— the default. The SWF file can communicate with the HTML page in which it is embedded only when the SWF file is from the same domain as the HTML page.</li> <li>▪ "always"—the SWF file can communicate with the HTML page in which it is embedded even when the SWF file is from a different domain than the HTML page</li> </ul>

**Table 7: Native method permission mechanism APIs for host applications**

API Names	Description
<code>DisableLocalSecurity</code> <code>EnforceLocalSecurity</code>	These are methods exposed to host applications to control security settings. They can be invoked only by native code in the host application.

## Security.allowDomain()

By default, Flash Player requires that SWF files and non-SWF files must come from the same domain to be able to script one another. In addition, applications that are served over non-secure protocols, such as HTTP, cannot script those that are served over HTTPS. The same restrictions apply to HTML pages scripting Flash applications.

Note: When two applications are from different domains, Flash Player ensures that they have different copies of the ActionScript global object. The global object is usually implicitly referenced. For example, all objects in the Flash Player standard library, such as MovieClip or Array, are part of the global object.

Applications served from different domains that want to be able to communicate with each other with scripting must be granted cross-domain scripting permission. This is done using the ActionScript method `Security.allowDomain()`.

For example, suppose an application at `http://www.mysite.com/controller.swf` needs to load an application from `http://utility.flashutils.com/helper.swf` and to call methods defined in `helper.swf`. Flash Player permits this operation only if the following ActionScript statement is in the `helper.swf` file:

```
Security.allowDomain("www.mysite.com");
```

This statement permits any application from the `www.mysite.com` domain to script `helper.swf`.

A SWF file may also call `Security.allowDomain()` with the wildcard parameter `""` to allow any domain. This is necessary to allow a local-with-networking SWF file to cross-script a network SWF file.

When a SWF file served over the secure HTTPS protocol calls the `Security.allowDomain()` method, this permits access by other SWF files that are also served over HTTPS, but it does not permit access by other SWF files that are served over insecure protocols, such as HTTP. In order for an HTTP SWF file to cross-script an HTTPS SWF file, the HTTPS SWF file must call the `Security.allowInsecureDomain()` method. For example, a SWF file at `http://www.mysite.com/controller.swf` that needs to load and call methods in a SWF file from `https://secure.mysite.com/creditcard.swf` is only permitted to do so if the following ActionScript exists in the `creditcard.swf` file:

```
Security.allowInsecureDomain("www.mysite.com");
```

Adobe does not recommend this practice, because allowing non-HTTPS documents to access HTTPS documents compromises the security offered by HTTPS. It is best to serve all Flash applications that require scripting access to HTTPS applications over HTTPS.

The Flash application that uses the `Security.allowDomain()` method may become vulnerable to cross-sandbox hijacking, because this allows full read permissions and cross-scripting to the application by members of the specified sandbox. (Contrast this with the `LocalConnection.allowDomain()` method, which works differently; see “The LocalConnection class” on page 43).

Note that calling the `Security.allowDomain()` method is an all-or-nothing switch; you cannot use the method to open up just a single part of a SWF file to another domain. You should call the `Security.allowDomain()` method only if you entirely trust the administrator of the specified domain not to abuse its rights to read and modify data in your SWF file, and to call ActionScript code in your SWF file. If you have a need to open up just a controlled API to another domain, the `LocalConnection.allowDomain()` method is a better solution, because you can provide a `LocalConnection` object that provides only limited functionality to its clients.

## Security.loadPolicyFile()

A policy file allows administrators to grant an application read access to specific server locations (URL policy files) or to specific port numbers (socket policy files). The default URL policy file is named `crossdomain.xml` and is located in the root directory of the target server. The default socket policy file is served from port 843. Use of the default location technique is typically best, as it opens the policy file for the entire server and is compatible with all versions of Flash Player 7 and higher.

An author can use the `loadPolicyFile()` method to load a policy file from another location. However, if there is no master URL policy file, then `loadPolicyFile()` cannot access other locations; this is because the default meta-policy for URL policy files specifies that only the master policy file can grant server access permissions.

For more information on policy files and on meta-policy requirements for their use, see “Website controls (policy files)” on page 33.

## Security.exactSettings

This is a Boolean value that specifies whether to use superdomain (`false`) or exact domain (`true`) matching rules when accessing local settings (such as camera or microphone access permissions) or locally persistent data (shared objects). The default value is `true` for files published for Flash Player 7 or later, and `false` for files published with earlier versions of Flash Player.

## Security.sandboxType

This read-only property indicates the type of security sandbox in which the calling SWF file is operating (for information on sandboxes, see “Basic sandbox security model” on page 13). The `Security.sandboxType` property has one of the following values:

- `remote`: This SWF file is from an Internet URL, and will operate under domain-based sandbox rules.

- `localWithFile`: This SWF file is a local file, but it has not been trusted by the user and was not published with a networking designation. This SWF file may read from local data sources but may not communicate with the Internet.
- `localWithNetwork`: This SWF file is a local file and has not been trusted by the user, but it was published with a networking designation. This SWF may communicate with the Internet but may not read from local data sources.
- `localTrusted`: This SWF file is a local file and has been trusted by the user, using either the Settings Manager or a FlashPlayerTrust configuration file. This SWF file may both read from local data sources and communicate with the Internet.

## The LocalConnection class

The ActionScript LocalConnection class lets you develop SWF files that can send instructions to each other. A LocalConnection object allows two Flash applications to communicate with each other, even when they are not located in the same instance of Flash Player (for example, when two SWF files are in separate browser windows). LocalConnection objects are available in Flash Player 6 and later.

Every LocalConnection object can use the `LocalConnection.allowDomain()` method to specify a set of domains that may send messages to this instance of the LocalConnection class. An application that calls `LocalConnection.allowDomain()` agrees to consider messages from other domains that it can examine further and act on or ignore as it chooses. It can therefore effectively select the effects (if any) that other domains can have on it. It can also choose what information to reveal back to the other domains. This supports mutually suspicious applications and allows communication without making either side vulnerable to the other.

### Security restrictions for LocalConnection objects

An application calls the `LocalConnection.send()` method to initiate a remote procedure call over a LocalConnection. When an application calls the `LocalConnection.send()` method and there is a receiver for the specified channel, Flash Player checks whether the domain of the sender is one of the domains allowed to use the LocalConnection, as specified by the receiver. If so, the call proceeds.

By default, Flash Player requires that a LocalConnection sender must come from the same domain as the receiver. In addition, applications that are served over non-secure protocols, such as HTTP, may not make LocalConnection calls to applications that are served over HTTPS. (Conversely, HTTPS applications may make LocalConnection calls to HTTP.)

When an application calls the `LocalConnection.domain` property, the return value is the application's exact domain. Thus, an application can use the `LocalConnection.domain` property to determine its own domain.

### LocalConnection channel names

One aspect of LocalConnections continues to use superdomains, even for applications made for Flash Player 7 and later: the domain in a channel name. A *channel name* is simply the name that a listener connects with and that a sender uses to identify a listener to send to. When a Flash application calls the `LocalConnection.connect()` method with a channel name that begins with an underscore (`_`), Flash Player uses the channel name exactly as provided, without regard to domain. However, for channel names that do not begin with an underscore, there is an implicit domain name added to the beginning of the channel name. For example, when a listener from `www.mysite.com` calls the following:

```
receiving_lc.connect("myChannel");
```

the channel name becomes `mysite.com:myChannel`. If a sender from `www.mysite.com` or `store.mysite.com` calls the following API:

```
sending_lc.send("myChannel", "methodName");
```

Flash Player sends the call to the channel *mysite.com:myChannel*, which corresponds to the listener's `connect()` call in the example.

The only time an application must add a domain name to a channel name is when it sends to a listener outside its own superdomain. In that case, the sender must explicitly specify the domain and channel name. If a sender from *www.anothersite.com* were to send to the listener in the previous example, the sender would use the following syntax:

```
sending_lc.send("mysite.com:myChannel", "methodName");
```

This situation uses the listener's superdomain, not the listener's exact domain.

## Granting cross-domain LocalConnection permissions

Applications served from different domains that need to be able to make LocalConnection calls to each other must be granted cross-domain LocalConnection permissions. To do this, the author must invoke the `LocalConnection.allowDomain()` method.

There is a second LocalConnection security method: `LocalConnection.allowInsecureDomain()`. This functions analogously to the `Security.allowInsecureDomain()` method: it is required in order for non-HTTPS SWF files to access HTTPS SWF files.

Adobe does not recommend using `LocalConnection.allowInsecureDomain()`, because allowing non-HTTPS documents to access HTTPS documents compromises the security offered by HTTPS. It is best that all Flash SWF files that make LocalConnection calls to HTTPS SWF files are served over HTTPS.

## Options when publishing

When publishing SWF files, authors have two choices for local file security: “access local files only” places the loaded SWF file in the local-with-file-system sandbox, and “access network only” places the loaded SWF file in the local-with-networking sandbox. The default is to place SWF files into the local-with-file-system sandbox. These settings do not affect SWF files loaded from the network.

The local content updater is another way to place a SWF file in the local-with-networking sandbox after compilation. This option is valuable for publishers who would like to modify the sandbox as a post-processing step. The local content updater is distributed as both binaries and source, and is available at <http://www.adobe.com/support/flashplayer/downloads.html>.

## ActiveX control and browser plug-in APIs

Applications hosting the Adobe Flash Player ActiveX control or Flash Player plug-in can use the `EnforceLocalSecurity` and `DisableLocalSecurity` API calls to control security settings. If `DisableLocalSecurity` is invoked, the application does not benefit from the local-with-networking and local-with-file-system sandboxes introduced in Flash Player 8. If `EnforceLocalSecurity` is invoked, the application is able to use all three local sandboxes, as described in “Hierarchy of local file security controls” on page 45.

The default behavior for an ActiveX control hosted in a client application is `DisableLocalSecurity`. The default behavior for the browser plug-in is `EnforceLocalSecurity`.

# Hierarchy of local file security controls

The preceding sections provide information local file sandboxes, as well as the local file security controls that are provided to administrative users, users, and authors. This section describes how these controls interact to determine which sandbox is used for a specific SWF file. These sandboxes (and the controls related to the sandboxes) have no effect on web-based content.

## Loading into the local-trusted sandbox

Any local SWF file can be placed into the local-trusted sandbox by either the administrative user or the end user; websites and developers do not have the authority to place files into the local-trusted sandbox. The Global Flash Player Trust directory allows administrative users to list SWF files that should be placed into the local-trusted sandbox. Users have this ability by default, but the administrative user restricts this ability with the `AllowLocalUserTrust` control in the `mms.cfg` file. By default, users are allowed to specify which SWF files should be placed into the local-trusted sandbox by using either the User Flash Player Trust directory or the Settings Manager.

## Loading into the local-with-networking sandbox

There is one control that may cause a SWF file to be placed in the local-with-networking sandbox rather than the default local-with-file-system sandbox; developers can mark a SWF file for the local-with-networking sandbox prior to providing it to the user. This can be done using the authoring tool, or with a post processing of the SWF file. In either workflow, this marker in the SWF file indicates that if Flash Player loads the SWF file from the local file system, the SWF file should be loaded into the local-with-networking sandbox.

Adobe provides a post-compiler tool (and source code) to configure a SWF file to use the local-with-networking sandbox at: <http://www.adobe.com/support/flashplayer/downloads.html>

## The default setting: local-with-file-system

There is no control that places files into the local-with-file-system sandbox. By default, Flash Player loads local SWF files into the local-with-file-system sandbox.

## Flash Player integration with native applications

Flash Player's standard local sandboxes may not be appropriate for the security model of all native applications hosting Flash Player, so a control was added to enable the Flash Player client runtime integration with native applications. Application developers integrating the plug-in version of the player should call an API to `DisableLocalSecurity` or `EnforceLocalSecurity`. If Flash Player integrators choose `EnforceLocalSecurity`, local content is placed into one of the three local sandboxes, according to the hierarchy of controls described previously. If Flash Player integrators choose to invoke `DisableLocalFileSecurity`, all files loaded from the local file system are placed into the local-trusted sandbox.

Adobe enables local file security in the stand-alone player and when it integrates Flash Player with any of the major browsers, including Internet Explorer, Mozilla, Netscape, Safari, and Firefox. On the other hand, the authoring player and projectors disable local file security.

Flash Player integrators should note that the default behavior for Flash Player varies between the ActiveX control and the browser plug-in. By default, the browser plug-in has local file security enabled, while the ActiveX control has local file security disabled.

## Deployment of the Flash Player runtime

There are a number of possible runtime deployment methods for installing the Flash Player client runtime on a client computer, making it available to run Flash applications for that client. These deployment options are not new to Flash Player 10. However, with some of the new security characteristics of Flash Player 10, there are some potential security-related runtime effects with the new environment. To provide as much upward compatibility as possible, Flash Player recognizes older release applications; however, in some instances, applications by default use a more restrictive security model than they would have previously encountered. Flash Player also provides some mechanisms for authors and users to specify security rules for an application for which you might want different handling. (Previous sections of this document provide more information on the sandbox security model and on security controls.)

Regardless of their packaging, delivery method, or target environment, Flash Player plug-ins and ActiveX control components are produced using a common Flash Player base and are digitally signed by Adobe. (For instance, features of Flash Player 10 are common to all Flash Player 10 plug-ins.)

## Browser plug-ins and ActiveX controls

A variety of plug-ins and similar options are available to users for the installation of Flash Player, depending on the combination of the operating system and browser(s) being used on the target client computer. These include Windows plug-ins (such as those for Netscape, Mozilla, and Opera), Macintosh plug-ins (such as those for Safari, Opera, and Netscape), and ActiveX control implementations (such as those for Windows Internet Explorer and AOL).

Some plug-ins are installed by default (initially deployed and configured) with various operating system and browser packages, or completely packaged and installed with selected third-party applications, for example, through various Adobe partnership agreements. Typically, the administrative user is the one who installs, uninstalls, or upgrades plug-ins. Sometimes the administrative user is simply the user of the client computer, but inside many security-sensitive organizations such administrative permissions are frequently restricted to specific authorized systems administrators.

Plug-ins run in the same process and address space as the browser itself, and may even share address space with other browsers, which is up to the browser's implementations.

There is also an API (`EnforceLocalSecurity`) provided for use by third-party applications to enable local file security (see “ActiveX control and browser plug-in APIs” on page 44). However, since some third-party applications can integrate ActiveX controls, developers should exercise special care to ensure that the proper sandbox domain model is used to avoid security issues (real or perceived).

## Authoring player

The Flash authoring tools, produced and signed by Adobe, include a version of Flash Player specifically for integrated previewing of content. While creating a Flash application, an author can run content using the authoring player within the authoring environment. This allows the author to test and review the application without (or prior to) deploying it to the target network environment.

However, there could be some differences in how the Flash Player security model affects the operation of the application between its execution in the author’s domain and its execution in its eventual deployment environment.

More importantly, while the same sandbox security model rules apply in either case (see “Basic sandbox security model” on page 13), the resulting interpretation of those rules in the author’s *location* might vary significantly from the ultimate deployment configuration. For example, if the author is an internal employee of a corporation (developing the application totally within that corporation’s internal network environment), but creating an application to be deployed to external customers on the Internet, the perspective of the sandbox boundaries seen on the developer’s computer can differ from those experienced by the outside users. If the application draws data from elsewhere within the internal network (considered *local* for the author residing within that network), that same data source would be outside the *local* domain when executing the SWF file on the external client’s computer outside the company. For example, internal help systems files might be available to an external SWF file only if they were delivered with the SWF file.

For these reasons, before deploying their applications, authors should always test them in the target network environment, not only in the authoring environment.

## Stand-alone player and Flash projector

The stand-alone player is distributed with the Flash authoring tools as an EXE file that can load and run SWF files. A Flash projector is an executable file (EXE file on Windows, or an APP file on Mac OS) that incorporates the stand-alone Flash Player. Since a Flash projector file is a SWF file packaged with the specific version of Flash Player, it does not necessarily run in the most current version of the environment available on the platform when it is executed; rather, it runs using the specific packaged version of Flash Player present at the time the Flash projector was produced and published by the author. Therefore, the Flash projector file could encounter differences in access authorizations or other behavior from what another SWF file would encounter on the same computer, such as one loaded as a current SWF file from the web and executed in a more current Flash Player environment.

Additionally, the version of Flash Player embedded within a projector file can be signed by the distributor, but it is *not* signed by Adobe. From the user’s perspective, there may be real or perceived security differences in the assurance ascribed to who signed the application or in the trust that they place in that source. Similarly, users might be concerned about downloading EXE files to run on their computer. For more information, see “Executable projector files” on page 48.

## Other distributions

Flash Player can also be distributed in other forms or with other products or applications, such as Adobe Flash Lite™ or third-party applications. Documentation specific for those products details the version of the client runtime used and any other architecture or security issues uniquely related to each of these other products.

## Deployment of Flash applications

Just as there are multiple ways to distribute and install Flash Player, there are multiple methods to distribute individual Flash applications.

## SWF files

Client computers can obtain individual SWF files from a number of sources, such as from external websites or from a local (internal) file system. SWF files are individually assigned to security sandboxes based on their origin when they are loaded into Flash Player. For more information, see “Flash Player security architecture” on page 13.

## Executable projector files

A projector file is an executable (EXE or APP) application, which not all users (or their systems security personnel and company security policies) accept from outside sources for downloading and execution on their local computers. From a security perspective, especially considering the perception and acceptance of security-sensitive users, authors may not want to deploy projector files.

In addition, as discussed in “Stand-alone player and Flash projector” on page 47, embedded in a projector file is a specific version of Flash Player that may be significantly older than the latest version available when the file is used, thereby forgoing all benefits (including security enhancements) of the newer Flash Player version. The user has no easy way to know which Flash Player runtime version was used by the application distributor, and therefore what security environment to expect.

Those considerations aside, projectors are a common mechanism for deploying content to environments where Flash Player may not be installed or where having absolute control over the version of Flash Player is preferred by an application producer.

# Other security-related information

## Network protocols

Flash Player supports a wide range of common network protocols. The following sections provide brief overviews of the most commonly used protocols and, where particularly relevant, describe some security-related effects of different approaches or protocols.

### AMF

Flash Player handles serializing and deserializing ActionScript objects to and from a proprietary terse binary data format called ActionScript Message Format (AMF). AMF serialized objects are the payload of HTTP requests and responses sent between the Flash Player client and the application server.

The client-side ActionScript libraries provide the ActionScript objects that a Flash developer uses to connect to and invoke methods on components in the application server. The libraries also provide objects that are helpful for debugging the connection.

### SMB

SMB (Server Message Block) is a message format used by DOS and Windows to share files, directories, and devices. Flash Player can load animations and SWF files from remote SMB shares. Flash has restrictions on what Flash SWF files loaded from SMB shares are allowed to do.

### RTMP

Flash Player uses the Real-Time Messaging Protocol (RTMP) for client-server communication. This is a TCP/IP protocol designed for high-performance transmission of audio, video, and data messages. RTMP sends unencrypted data, including authentication information (such as a name and a password).

Although RTMP in and of itself does not offer security features, Flash communications applications can perform secure transactions and secure authentication through an SSL-enabled web server.

Flash Player cannot access bitmap data or sound spectrum data for media loaded from RTMP sources, although it can display and play bitmaps and sounds loaded from these servers.

Flash Player also provides support for versions of RTMP that are tunneled through HTTP and HTTPS. RTMPT refers to RTMP transmitted within an HTTP wrapper, and RTMPS is RTMP transmitted within an HTTPS wrapper.

### HTTP

HyperText Transfer Protocol (HTTP) defines how messages are formatted and transmitted, and what actions web servers and browsers should take in response to various commands. HTTP is called a stateless protocol, because each command is executed independently, without any knowledge of the commands that came before it. HTTP is an insecure protocol subject to a variety of security weaknesses, so it is not appropriate for applications that transmit or provide access to sensitive data.

## HTTPS

HTTPS (HTTP over Secure Sockets Layer) is designed to transmit individual messages securely. SSL creates a secure connection between a client and a server, over which any amount of data can be sent securely. SSL works by using a private key to encrypt data that is transferred over the SSL connection. Flash Player uses the operating system or browser to determine whether its data was obtained over a secure HTTPS connection, and records that fact (for instance, by using separate sandboxes). Data loaded from HTTPS sites is subsequently treated differently than data from HTTP or other, less-secure, sources. Flash applications can call the ActionScript `Security.allowInsecureDomain()` method to allow scripting from non-HTTP sites to HTTPS sites, although an author should use this method only after reading the considerations listed in the *ActionScript 3.0 Language Reference*.

## TCP sockets

Transmission Control Protocol (TCP) is used as the underlying protocol for most of the previously described transmission methods. It does not provide any inherent capabilities for securing the data that it transmits.

Flash Player can use the ActionScript `XMLSocket` and `Socket` objects to create persistent sockets, which do not use the browser to communicate with the server. Because of this, Flash Player cannot take advantage of the built-in encryption capabilities of the browser. However, it is also possible to use encryption algorithms written in ActionScript to further protect the data that is being communicated.

Because these objects establish and maintain an open connection to the server, they have restrictions for security reasons. A socket policy file is required to permit connections to the host socket. The socket policy file includes all domains that are allowed to connect to the socket, including itself. For more information, see “Socket policy files” on page 37.

Additionally, computer administrators can use directives in the `mms.cfg` file on the user’s local computer to globally enable socket connections (the default), specify a whitelist of servers to which socket connections may be made, or disable socket connections altogether. For more information on the `mms.cfg` file, see “The `mms.cfg` file” on page 25.

# SSL (Secure Sockets Layer) utilization

## Basic SSL–browser plug-ins

PKI (Public Key Infrastructure) is built into all web browsers that use SSL, and Flash Player uses the browser to do all the work in the interpretation of client-side PKI and in using the browser's certificate store. An SSL connection is secured by using the PKI certificate of the web server to share a symmetric key with the web browser that is used to encrypt data exchanged between them. When SSL is being used to communicate with a web server, the security functions of the web browser may allow the end user to view and check the validity of the associated web server's certificate.

This is currently the most common application of SSL. Since it works with no further user interaction, most people are unaware of the other PKI certificate and security features. Some web browsers also allow you to store and use personal PKI certificates for authentication. The key pair and certificate are used with web servers and sites that require authentication through client-side SSL connections. In a client-side SSL connection, the web browser authenticates using a private key to decrypt a message encrypted by the public key. Depending on the features of the browser, the certificate to be used may have to be specified, if there are many certificates available. Some browsers select a certificate that works based on which other certificates were used to sign it.

Because Flash Player does not itself implement SSL, all behavior related to certificate verification is determined by the browser. This approach simplifies administration of the client, but it may also result in some variation in behavior between different browsers and operating systems. For example, the symmetric key size and the specific algorithm used for an SSL connection are negotiated by the browser. Similarly, Flash Player does not handle client behavior for certificates that are expired, revoked, self-signed, or do not match the URL of a requested resource.

## Projector files

Because Adobe Flash projector files are executables that run outside a browser, they cannot take advantage of the SSL capabilities of a browser. If sensitive information needs to be transferred between the projector file and a server, authors must do one of the following:

- Encrypt the data within the application with an ActionScript algorithm, such as an Advanced Encryption Standard (AES) routine.
- Require that the application's users have a secure network connection to the server, such as a virtual private network (VPN) connection.