macromedia®

WHITE PAPER

Macromedia Flash Player 8 Security

by Adrian Ludwig

September 2005

Contents

Introduction

The Flash Platform architecture

As the following figure shows, the Macromedia Flash Platform provides an end-to-end architecture for delivering Rich Internet Applications (RIAs), content, and communications across multiple platforms and devices.

Figure 1: The Flash Platform architecture supports the delivery of RIAs, content, and communications.
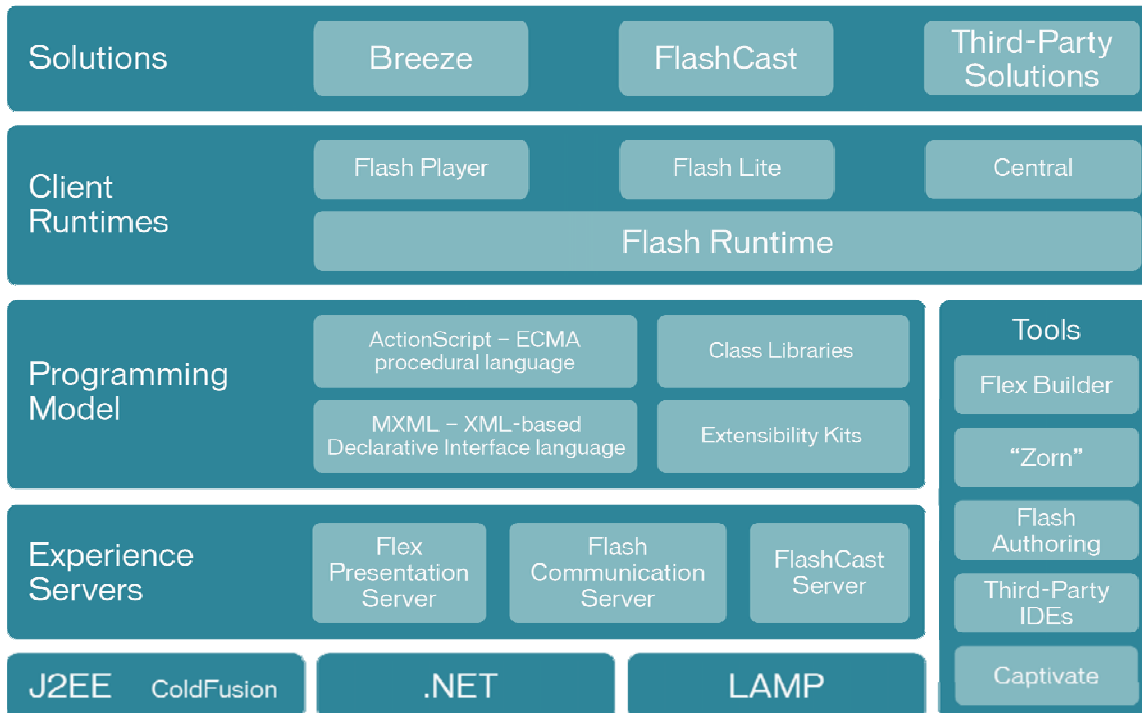
The following are the elements of the Flash Platform architecture:

- Servers and data services that deliver Flash based applications, content, and communications

- A standards-based programming model grounded in industry best practices

- Development tools provided by Macromedia as well as third-party independent software vendors

- A client runtime that delivers a consistent user experience across the widest range of platforms and devices

- Communication and collaboration solutions that address the increasingly complex communications issues of today's organizations

For more information on the Flash Platform, see www.macromedia.com/platform/.

About this document

This document is intended for the following audience:

- Enterprise architects who are evaluating or need to better understand the security model of the Flash Platform

- IT managers interested in the security of Flash applications in their network environment

- Website owners that deploy Flash applications from their sites

- Developers (including programmers and other authors) designing and publishing Flash applications

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

This document focuses on the security-relevant features of the Flash Player client runtime, including those previously 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. It is important, however, that there are no distinctions in the resulting runtime security model between applications created using different development tools, such as Macromedia Flex 1.0, Flash MX 2004, or Flash MX Professional 2004.

Flash Player client runtime

Macromedia Flash Player client runtime (also known simply as *Flash Player*) runs Flash applications (also referred to as SWF files). Flash 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. The SWF files themselves are typically hosted on a server and then downloaded to, and displayed on, the client computer when requested. SWF files consist of multimedia content (vectors, bitmaps, sound, 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.

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. 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 2: Hierarchy of security controls

Administrator
(User Institution)
settings

↓

User settings

↓

Website settings

↓

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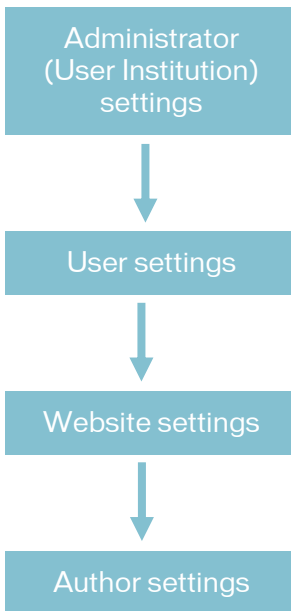
This means, for instance, that if an administrator restricts access to a resource, no other stakeholders can override that restriction. In Flash Player, it is common for multiple stakeholders to have the ability to control access to a resource, and for some stakeholders to formally delegate the right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

The following sections describe these stakeholders.

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 Macromedia 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 owner

- Author

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 a recreational usage at home), 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 Macromedia 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 owner

Websites hosting Flash applications rely on Flash Player behavior to deliver their content and application features. Website (and their data) owners 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 (of a Flash application)

An *author* is the program developer and publisher of a Flash application. The author might want to protect his 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 but that which 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 17.

Table 1: 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, including the following: ▪ Access to any camera or audio input devices attached to the computer ▪ Global control of local file reading, as well as file upload and download capabilities ▪ Local Flash Player disk usage and third-party storage of persistent shared objects ▪ Flash Player auto-update settings
	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 the following: ▪ Access to any camera or audio input devices attached to the computer ▪ Local Flash Player disk usage and third-party storage of persistent shared objects ▪ Flash Player auto-update settings When *legacy content* (content built for an earlier version of Flash Player) attempts to access resources that are protected in the new version of 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	Cross-domain policy files	Website administrators can use these files to control Flash applications' access to resources on the domain.
Author	Cross-script APIs	Flash Player provides a number of API calls related to security. For more information, see "Developer controls" on page 30.

Sources for potential risk

It is also useful to categorize those sources of potential risk that the Macromedia Flash platform protects stakeholders against. These include sources that 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.

Flash applications and other programs (not Flash) sharing the same computer

Computer operating systems do not consistently or securely protect one application, such as Flash Player, from another (except to some limited degree when multiple user IDs are used). Other software running on the platform might include malevolent software, such as viruses and worms. A Flash application should protect against using such malevolent pieces of software in its execution.

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 claims

This paper focuses on security related to Flash Player. Flash Player executes on the client's computer to view Flash content, typically from a host web server. It protects all stakeholders 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 allocate its own memory, install software, or make changes to the operating system without permission. 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.

Flash Player security architecture

The Flash Player client runtime provides a comprehensive security architecture that allows content to access functionality only if that content has been explicitly granted permission to use that functionality. 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. Authors may also grant permissions to specific SWF files.

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 what permissions are enabled. 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 assigns SWF files to sandboxes based on their origin. Flash Player bases sandbox boundaries on Internet domains or, for local SWF files, on the class of local SWF file. As described earlier, Flash always runs SWF files downloaded from the network in separate sandboxes that correspond to their origin domains. 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. Security enhancements in Flash Player 8 differentiate between more sandboxes than were available in earlier versions.

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 17.

An author of a SWF file can use the ActionScript `System.security.sandboxType` property to determine the type of sandbox to which Flash Player has assigned the SWF file (for more information, see "System.security.sandboxType" on page 34).

Flash Player places files, shared objects, and other resources in sandboxes corresponding to their origin domain.

Figure 3: Flash defines sandboxes based on source domains
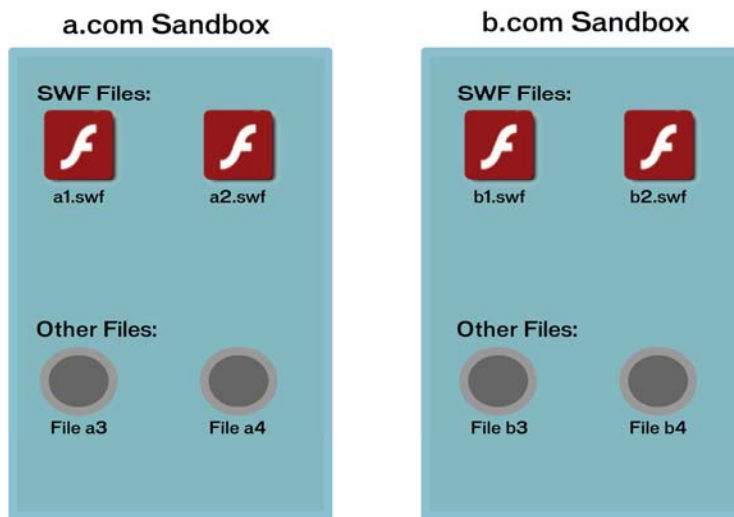
Domain of origin

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.

Prior to Flash Player 7, the cross-domain security decisions were made on the basis of a *superdomain* comparison, to attempt to accommodate SWF files from similar domains (for example, *www.macromedia.com* and *store.macromedia.com*) to communicate freely with each other (and with other documents). However, security weaknesses could stem from a policy this broad.

The Exact Domain Match feature of Flash Player 7 strengthened the domain-matching behavior, requiring that network SWF files can only communicate freely with each other (and with other documents) when they come from the same domain. Besides being much more secure, this also more closely matches the behavior of models used by DHTML and Java on the client.

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 *dataloading*, 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.

- 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 4: A read request from a SWF file to another resource

Figure 5: A message sent from a SWF file to another resource
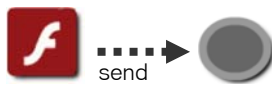
From the *a.com* sandbox, a SWF file may read (using the ActionScript `XML.load()` method, for example) from the server at *a.com* (but cannot by default read from *b.com*), and may send (using the ActionScript `XML.send()` method, for example) anywhere on the network. (For more information, see the *Flash Security-Related APIs* document.)

Any two SWF files in the same sandbox may interact freely with each other. Diagrams in later sections explain how SWF files may interact with SWF files from other sandboxes, and with servers.

Figure 6: Default sandbox permissions
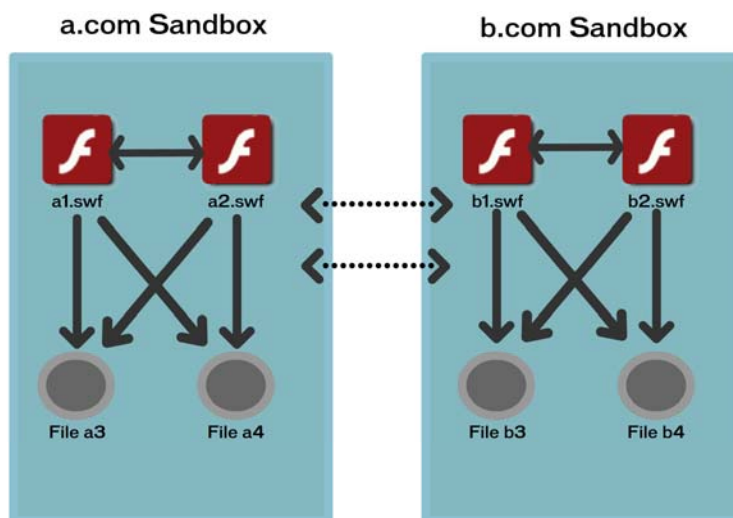
Accessing data in another sandbox

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. As detailed in "The Flash Player security environment" on page 3, the following have authority to grant access to files, and the precedence of control is in the order of this list:

- User institution administrators
- Users
- Website administrators
- Flash application authors

In other words, user institution administrator decisions take priority over user decisions, which take priority over website administrator decisions, which take priority over Flash application author decisions. This should not be taken to imply that all decisions about permissions can be enforced or modified by all stakeholders, because that is not true. 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 `System.security.allowDomain()`. Similarly, 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, unless explicit permission is provided.

Figure 7: Read across sandbox boundaries is strictly controlled
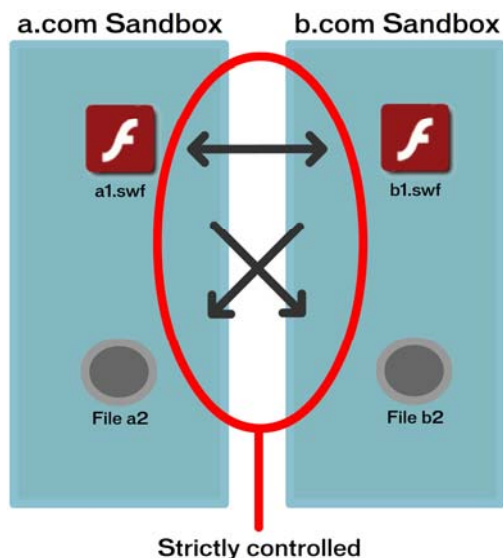
Administrative users, users, website administrators, and developers can grant permissions that are not available by default to SWF files (see "Permission controls" on page 17).

A SWF file from *a.com* may read from the server at *b.com* (using the ActionScript `XML.load()` method, for example) if *b.com* has a cross-domain 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 `System.security.allowDomain()` method to permit access from *a.com*.

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 8.)

Local files

Local files also follow the basic sandbox model, with this exception: they have different default permissions. (For more information, see "Basic sandbox security model" on page 8.)

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

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 `System.security.allowDomain()` API. 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 8: Default permissions for a SWF file in the local-trusted sandbox
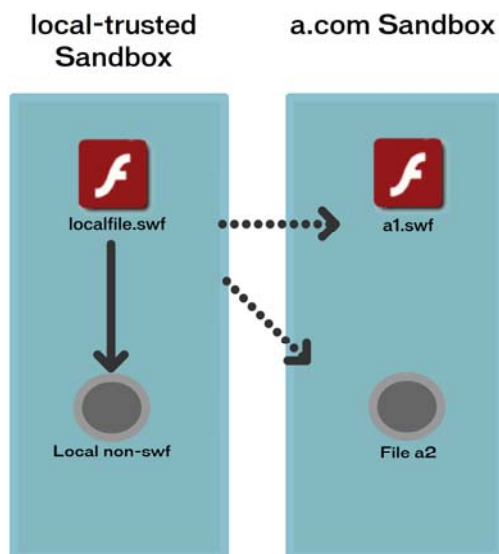
Local-trusted SWF files may read from local files; read or send messages with any server; and script any other SWF file.

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

Figure 9: Default permissions for local-with-file-system SWF files; these cannot be modified
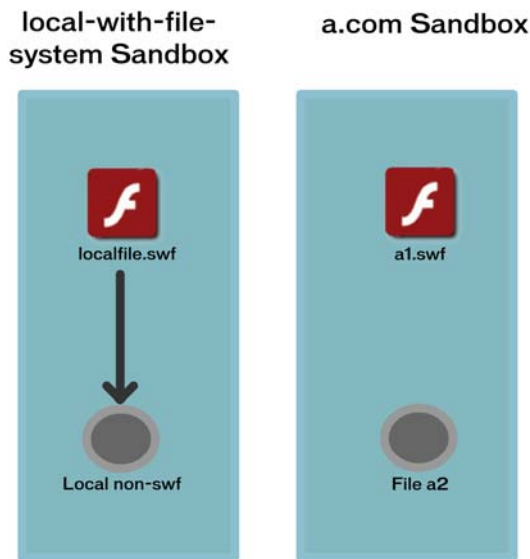
The following diagram shows the default permissions available to local SWF files in the local-with-networking sandbox. These SWF files are not granted permissions available to the local-trusted sandbox, but they *do* have default access to send to the network. Also, they can be granted read access to network sandboxes.

Figure 10: Default permissions for local-with-networking SWF files
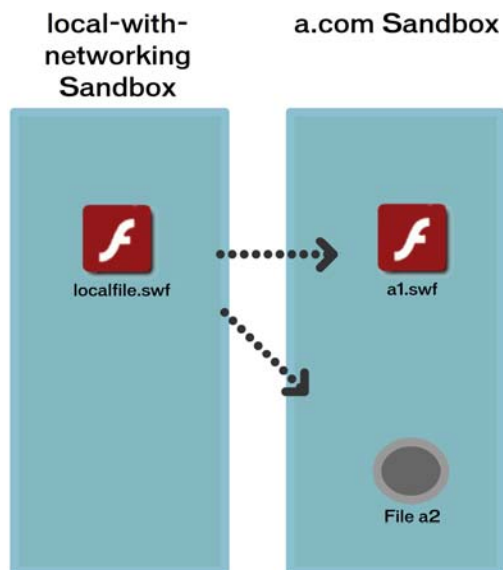
In the absence of permissions, local-with-networking SWF files are only allowed to communicate with other local-with-networking SWF files and to send to network servers (using the ActionScript `XML.send()` method, for example).

A local-with-networking SWF file is allowed to read (using the ActionScript `XML.load()` API, 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 `System.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 `System.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.

Interpreters and byte code

Flash Player includes a virtual machine to run instructions that are contained in the byte code of a SWF file.

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.

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 2.0, 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.

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 (Macromedia) 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. This is true for all versions of Flash Player, although there are some new differences in the granularity and implementation of the access control rules in Flash Player 8 (discussed in other sections of this document).

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 Macromedia) 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 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 (nondefault) "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 helps limit potential denial-of-service attacks involving disk space (and system memory) through its monitoring of the usage of these key system resources. 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 26.)

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 in 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 8 uses a garbage collector engine common to other Macromedia 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.

Flash Player enforces quotas on memory usage and on processor cycles.

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 8.

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 8 provides various installation-time configuration choices (some new). For example, some corporations 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 5.

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 Macromedia Security configuration file (mms.cfg) and Global Flash Player Trust files.

Macromedia Security Configuration file

The primary purpose for the Macromedia Security Configuration file (mms.cfg) 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, the file is located in the Macromedia Flash Player folder within the system directory (for example, C:\winnt\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 functionality as described here (along with the list of settings that are supported for each entry):

Data Loading Controls

`LocalFileReadDisable=[0, 1]`
(0=false, 1=true)

Setting this option to `1` disables all local content execution.

If this is set to `1`, Flash Player cannot read any files referenced by a path (including the first SWF file that Flash Player opens) on the user's hard disk. Specifically, the ActionScript `XML.load()`, `loadVariables()`, `loadMovie()`, and `getURL()` methods, and anything else that maps to a target on the local drive, is blocked. Reading runtime-shared libraries is also blocked if this flag is set. Reading local shared objects is allowed, although only if the administrative user has not revoked the permission to write them.

`FileDownloadDisable=[0, 1]`
(0 = false, 1 = true)

If this is set to `1`, the ActionScript `FileReference.download()` method is disabled; the user is *not* prompted to allow a download, and no downloads using the FileReference API are allowed. If this is set to `0`, Flash Player allows the ActionScript `FileReference.download()` method to ask the user where a file can be downloaded to, and then Flash Player downloads the file after the user approves the file save location.

`FileUploadDisable=[0, 1]`
(0 = false, 1 = true)

If this is set to `1`, all `FileReference.upload()`, `FileReference.browse()`, and `FileReferenceList.browse()` activity is disabled; the user is not prompted to upload files, and no uploads using the FileReference API are allowed. If this is set to `0`, Flash Player allows files to be uploaded using the FileReference API. The user is prompted to select a file to upload and to approve the selection.

`LocalStorageLimit=[0, 1, 2, 3, 4, 5]`
(0 = false, 1 = 0K, 2 = 10K, 3 = 100K, 4 = 1 MB, 5 = 10 MB, 6 = unlimited)

If this has a value other than `0`, that value indicates a hard limit on the amount of local storage that Flash Player uses. The Settings dialog box is not able to specify more storage than this limit, but it is able to specify less. If it does specify less, that setting is honored. If the value is `0`, the Settings dialog box is honored.

Privacy Controls

AVHardwareDisable=[0, 1]
(O=false, 1=true)

If this is set to 1, the camera and microphone are disabled, and they cannot be enabled using the Settings dialog box (the Settings dialog box does not appear if a SWF file requests access to these assets). This disables the camera and the microphone, and also disables the camera and microphone pages of the Settings dialog box. (The user can use the controls there, but it won't affect anything. The user can also play with the Privacy page of the Settings dialog box, but again it has no effect.)

WindowlessDisable=[1,0]
(O = false, 1 = true)

If this is set to 0 (the default), Flash Player can play windowless content. If this is set to 1, Flash Player suppresses windowless content from playing.

ThirdPartyStorage=[1,0]
(O = false, 1 = true)

Third-party refers to SWF files that are executing within a browser and have an originating domain that does not match the URL displayed in the browser window. If this is set to 1, third-party SWF files can write locally persistent shared objects. If this is set to 0, third-party SWF files cannot write locally persistent shared objects.

Update Controls

AutoUpdateDisable=[1,0]
(O = false, 1 = true)

If this is set to 0 (the default), Flash Player allows auto-update based on user settings. If this is set to 1, Flash Player disables auto-update.

AutoUpdateInterval=[number]

If this is a negative value (the default), Flash Player uses the auto-update interval value specified in the Settings Manager. If this is set to 0, Flash Player checks for an update every time it starts. If this is a positive value, the value specifies the minimum number of days between update checks.

AutoUpdateVersionUrl=[string]

If no value is specified (the default), Flash Player uses the Macromedia server to check for auto-updates. If a string is specified, it represents the URL that Flash Player uses to retrieve player update data.

AutoUpdateSettingsUrl=[string]

If no value is specified (the default), Flash Player uses the Macromedia server as the destination for the Settings button in the Auto-Update dialog box. If a string is specified, Flash Player uses the specified URL as the destination for the Settings button in the Auto-Update dialog box.

AutoUpdateInstallerUrl=[string]

If no value is specified (the default), Flash Player uses the Macromedia server as the download location for player update. If a string is specified, Flash Player uses the specified URL as the download location for player update.

AutoUpdateStartDelay=[non-negative number]

If no value is specified (the default), Flash Player uses five minutes as the delay before showing the System tray icon. If a number is specified, it is the number of milliseconds to pause before showing the system tray icon.

AutoUpdateFrequency=[-1, 0, 1...N]
(-1 = never, 0 = undefined, 1...N = number of days)

If this is set to `-1`, auto-update is disabled. If this is set to `0`, the Settings dialog box is honored. If this has a value from `1` to `N`, that is the number of days between auto-update attempts.

AutoUpdateLock=[0, 1]
(0 = false, 1 = true)

If this is set to `1`, Flash Player does not allow the user to modify the auto-update frequency. If this is set to `0`, the Settings dialog box is honored.

DisableProductDownload=[0, 1]
(0 = false, 1 = true)

If this is set to `0` (the default), the Flash Player can use the `system.product()` API to install native code applications that are digitally signed by Macromedia. If this is set to `1`, the `system.product()` API is disabled.

Legacy Controls

LegacyDomainMatching=[1,0]
(no value, 0 = false, 1 = true)

If no value is specified (the default), the user can determine whether to allow a SWF file produced for an older version of Flash Player to execute an operation that has been restricted in a new version of Flash Player – this decision may be made by the user in a global manner using the Settings Manager, or on a case-by-case basis using an interactive dialog box. If `LegacyDomainMatching` is set to `1`, Flash Player behaves as though the user answers "allow" whenever they make this decision. If `LegacyDomainMatching` is set to `0`, Flash Player enforces the deny decision for all users.

LocalFileLegacyAction=[0, 1]
(0 = false, 1 = true)

If this is set to `0` (the default), Flash Player uses a hierarchy of controls to decide which local sandbox should be used to execute a specific SWF file that was originally produced for an older version of Flash Player. If this is set to `1`, all SWF files produced for older versions of Flash Player are placed into the local-trusted sandbox. The hierarchy of local file security controls is described in detail in the "Hierarchy of local file security controls" section on page 37.

Local File Security Controls

AllowUserLocalTrust=[0, 1]
(0 = false, 1 = true)

If this is set to `1` (the default), Flash Player allows the user to specify whether local files can be placed into the local-trusted sandbox. If this is set to `0`, the user cannot place files into the local-trusted sandbox. The Settings Manager trust panel and user trust files are ignored.

The mms.cfg file and the Settings dialog box

The Macromedia security configuration file (mms.cfg) affects the options in the Flash Player Settings dialog box, and some mms.cfg file settings may override individual settings. The Settings dialog box has four tabs:

- **Privacy**—If `AVHardwareDisable` is `true`, all user actions related to this tab are ignored. The tab does, however, appear functional.

- **Local Storage**—If `LocalStorageLimit` 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 `AVHardwareDisable` is `true`, the recording level meter is disabled. All other controls appear to work, but their values are ignored.

- **Camera**—If `AVHardwareDisable` is `true`, clicking the camera tab does *not* bring up a thumbnail of what the camera is seeing.

Flash authors can query `System.capabilities.avHardwareDisable` and `System.capabilities.localFileReadDisable` (both new in Flash Player 8) 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 System.capabilities object (all properties of this object are read-only). For additional information on the System.capabilities object, see *ActionScript 3.0 Language Reference*.

Global Flash Player Trust directory

A new Flash Player 8 feature provides a way for administrative users to register certain local files so that they are always loaded into the local-trusted sandbox. Often an installer for a native application or an application that includes many SWF files will do this. Depending on whether Flash Player will be embedded in a nonbrowser 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.

The installer creates files in a directory called FlashPlayerTrust. These files list paths of trusted files. This directory, known as 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:** *system*\Macromed\Flash\FlashPlayerTrust
  (for example, C:\winnt\system32\Macromed\Flash\FlashPlayerTrust)

- **Mac:** *app support*/Macromedia/FlashPlayerTrust
  (for example, /Library/Application Support/Macromedia/FlashPlayerTrust)

These settings affect all users of the computer. If an installer is installing an application for all users, the installer can register its SWF files as trusted for all users.

There are also individual User Flash Player Trust directories, used by installers to register an application for specific users of a computer (see "Flash Player Trust directories and files" on page 28). 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 37).

User controls

Flash Player provides users with three different mechanisms for setting permissions. The Settings UI provides a quick, interactive mechanism for configuring the settings for a specific domain. 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.

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. Macromedia does not collect any information about Flash Player user settings or preferences, and at no time are the user's settings at risk of being snooped or hijacked. The Settings Manager appears to be an application hosted on the Macromedia web page (www.macromedia.com), but it is in reality a SWF file served from the Macromedia 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 (which includes new options in Flash Player 8)
- Flash Player Update Settings
- Privacy Settings for Individual Websites
- Storage Settings for Individual Websites

The following figures show sample dialog boxes:

Figure 11: Global Privacy Settings dialog box
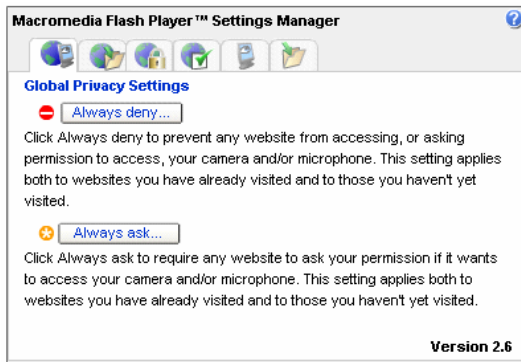
Figure 12: Global Storage Settings dialog box
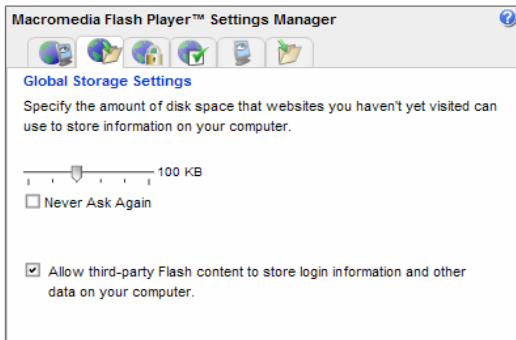
Figure 13: Global Security Settings dialog box
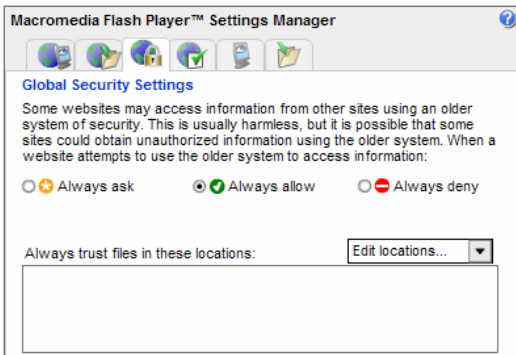
Figure 14: Flash Player Global Notification Settings dialog box
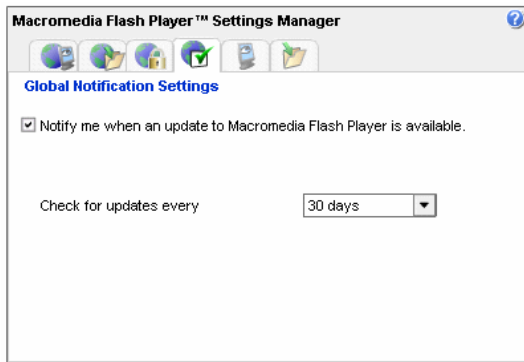
Figure 15: Individual Website Privacy Settings dialog box
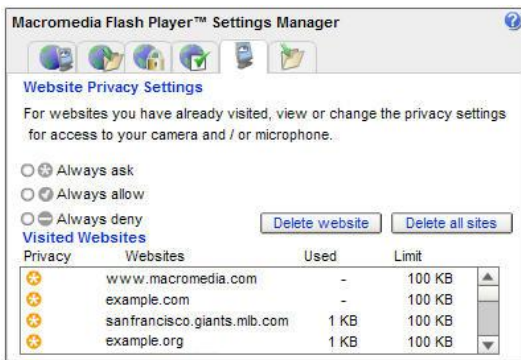
Figure 16: Individual Website Storage Settings dialog box

Figure 17: Global Security Settings dialog box (Flash Player 8)

All of the settings information managed by the Settings Manager is retained in shared objects of the *Macromedia.com* domain. 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 Macromedia, nor is it transmitted across the network.

Settings UI and runtime dialog boxes

These settings are most commonly accessed by right-clicking an executing SWF file and selecting the Settings option. The Settings 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 global Settings Manager by using the Advanced button on the Privacy tab.

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 (particularly earlier than Flash Player 7) applications attempt access that is no longer permitted by default.

If the application is too small to display the Settings UI or a runtime dialog box (for example, an application smaller than 215 x 138 pixels), the dialog box does not appear and Flash Player, by default, treats the operation as if Deny were selected.

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.

Figure 18: Privacy settings
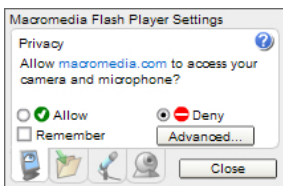
Camera settings

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 19: Camera information

Microphone settings

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

Figure 20: Microphone settings

Storage settings

Users can also set storage limits for all Flash applications from the current domain using the following UI. Setting the storage to 0 kilobytes causes Flash Player to prompt the user if a Flash application requests a shared object.

Figure 21: Individual Flash application storage settings

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 forbidden in current versions of Flash Player, but which would have been permitted in Flash Player 6. The box on the left is an example for a request that was 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 22: Fallback access warnings
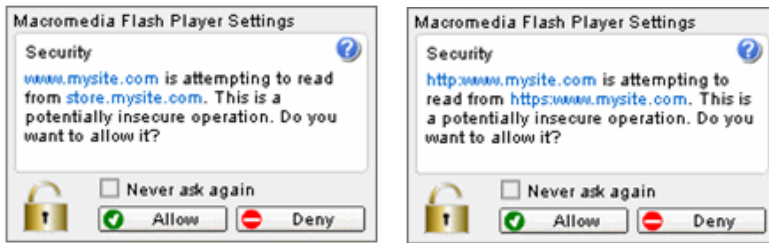
If the user clicks Allow, the request succeeds. If the user clicks Deny, the request fails. The dialog boxes do not appear if at some time previously the user selected the Never Ask Again check box.

Network Access Warning

The following dialog box is new in Flash Player 8 and is directed at users if they attempt to run an untrusted local SWF file (in the local-with-file-system sandbox) that tries to access the network. The dialog box only appears for operations that would have succeeded in Flash Player 7 or earlier, and are undertaken 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; authors must either avoid such prohibited actions or the dialog box appears based on user and administrative controls.

Figure 23: Network access warning
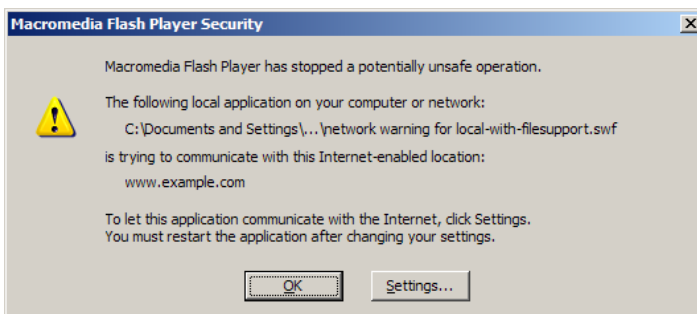
Effects of configuration files

The choices users have at the time they attempt to execute a Flash application are restricted by options set by the author at publication time (see "Developer controls" on page 30) and particularly by other system-wide control settings that may have been set by their system administrators (see "Administrative user controls" on page 18).

Flash Player Trust directories and files

A new Flash Player 8 feature provides a way for users and installer applications to register specified local files as trusted. The User Flash Player Trust directory is alongside the Flash shared objects storage area, in the following locations (which are specific to the current user):

- Windows: *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)

These settings affect only the current user (not other users that log in on the computer). 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 21).

These two sets of trust directories have different security precedence (see "Hierarchy of local file security controls" on page 37).

Website controls

The system administrator of a domain (website) that hosts Flash applications or resources used by Flash applications can designate what resources can be downloaded from their site using a cross-domain policy file.

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 `XML.load()` or `loadVariables()` API, using Flash Remoting with NetConnections, and using XML socket connections. The default for network sandboxes is to restrict read permissions to data sources from the origin domain (exact match) of the SWF file.

For a Flash application to fetch data over the web, the sandbox from which it is to be fetched (the provider domain) must include a policy file that permits that action. These are the cross-domain 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). For more information on policy file formatting, see the following Flash Player TechNote:

http://www.macromedia.com/cfusion/knowledgebase/index.cfm?id=tn_14213

The Cross Domain Policy File mechanism is a simple XML file (*crossdomain.xml*) that does the following:

- Modifies the read permission for data between sandboxes and across the network. It does not apply to cross-scripting of SWF files.

- Is specified with a text file (served as a policy file from the site to be accessed as an ordinary web page), and is consulted by Flash Player, which then enforces the rules, rather than requiring (or allowing) a SWF file to declare or interpret the permissions.

- Applies only to the protocol and port of the server, rather than opening up an entire domain, with one exception: HTTP servers can provide the policy files that govern XML socket connections.

The Cross Domain Policy File is located in the root directory of the target server by default (for example, at *www.applicablesite.com/crossdomain.xml*), or Flash application developers can specify another location by calling the ActionScript `System.security.loadPolicyFile()` API.

When a server is serving a policy file, SWF files that want to take advantage of the policy file can simply issue a cross-domain request, and Flash Player fetches the server's policy file to determine whether access is allowed. This applies to all players on all platforms and all SWF file versions.

Note: There are two distinct (basically unrelated) controls that apply to cross-domain actions that might be confused because they have similar names: cross-domain policy files (covered here) are managed by domain administrators to allow network access to data; the ActionScript `System.security.allowDomain()` API is specified by authors to allow client-side cross-domain scripting (discussed in "Developer controls" on page 30).

Policy file usage

Policy files grant cross-domain permissions for reading. 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.

When a domain is specified in a 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):

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

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 *bigbang.staticland.net* that provides XML data, and you serve Flash applications from *www.bigbang-corp.com*, which needs to load the XML data, you could put a policy file on *bigbang.staticland.net* that enables access specifically from *www.bigbang-corp.com*.

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

```
<cross-domain-policy>

</cross-domain-policy>
```

The scope of the permissions defined in a 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 policy files does not need to be well-defined, because there is no precedence among policy files. If a resource is within the scope of more than one policy file, any one of the policy files may grant access.

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.

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 2: Permission mechanism APIs

API Names	Description

`System.security.allowDomain()`
 	
`System.security.allowInsecureDomain()`
	Permit SWF files from a specified sandbox to read (or cross-script) the calling SWF file. The `allowInsecureDomain()` API also permits access to HTTPS SWF files by non-HTTPS SWF files.
	
	Note: For Flash Player 7 and earlier, these methods permitted access to all SWF files in the sandbox of the calling SWF file. Starting with Flash Player 8, these methods permit access only to the calling SWF file itself.

`System.security.loadPolicyFile()`	Informs Flash Player of the location of a policy file in a nondefault location, or the location of an XMLSocket policy file.
`System.exactSettings`	Determines whether exact or old-style *superdomain* rules are used to determine the scope of shared objects and privacy settings. *Note:* For Flash Player 7 and earlier, the value of this property was scoped to an entire sandbox–changing it from one SWF file in a sandbox also changed it for all other SWF files in the same sandbox. In Flash Player 8, the value of this property is scoped to the calling SWF file only.

API Names	Description

`LocalConnection.allowDomain()`
 	
`LocalConnection.allowInsecureDomain()`
	If present, called by Flash Player whenever a `LocalConnection` method call is about to occur. The caller's domain is passed as an argument. The method returns `true` or `false` to allow or disallow the call. If unimplemented, calls are only allowed from callers in the same domain.

`DisableLocalSecurity` `EnforceLocalSecurity`	This is not an ActionScript API. It is a method exposed to host applications to control security settings. It can only be invoked by native code in the host application.
`allowScriptAccess`	This is not an ActionScript API. It is an HTML parameter. Its value governs whether ActionScript can call JavaScript (or other script) in the HTML page (container). It has the following values: `"never"`–not allowed `"sameDomain"`–allowed if the calling SWF file is from the same sandbox as the container. Note: this is the default value for Flash Player 8; `"always"` was the default value for Flash Player 7 and earlier SWF files. `"always"`–allowed

System.security.allowDomain()

The Flash application that uses the `System.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, as described in "System.security.sandboxType" on page 34).

Security restrictions for scripting between SWF files

Scripting between SWF files occurs when one SWF file loads another SWF file using `MovieClipLoader.LoadClip()`, `loadMovie()`, or `loadMovieNum()`, and then one of the SWF files uses ActionScript to examine or modify variables in the other, or calls functions or methods in the other. Scripting between SWF files requires read permission, so by default it is only permitted with SWF files that are in the same sandbox. Loading of a SWF file, on the other hand, requires only the send permission. So, SWF files are generally allowed to load SWF files from other sandboxes, but security restrictions may prevent those files from communicating with each other.

Scripting permissions also affect when an HTML page that uses JavaScript (or another scripting language) may script a Flash application. Flash Player only permits this operation when the HTML page is in the same sandbox as the Flash application it attempts to script, or if appropriate sandbox permissions have been explicitly provided.

Applying the rules

By default, Flash Player 7 and later requires that SWF files and non-SWF file script must come from the same domain to be able to script one another. In addition, applications that are served over nonsecure protocols, such as HTTP, cannot script those that are served over HTTPS. The same restrictions apply to HTML pages scripting Flash applications.

These rules apply whenever one or both of the SWF files are made for Flash Player 7 or later. If both applications are made for Flash Player 6 or earlier, Flash Player 8 uses the old rules. The old rules permit applications from the same superdomain to script each other; they also permit HTTP applications to script HTTPS applications. A superdomain is defined by removing all subdomains in a DNS name, except for the trailing two domain names (for example, www.test.com and ftp.test.com are both members of the same superdomain).

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. The global object also holds global variables created by assigning properties to `global`. Separating global objects between applications from different domains has occurred in Flash Player 6 and later. However, Flash Player 7 introduced a new restriction: applications made for Flash Player 6 or earlier can never share a global object with those made for Flash Player 7 or later, even when they are from the same domain and protocol. This may have subtle repercussions for sites that mix SWF files made for Flash Player 6 and those for later versions.

Granting scripting between SWF files

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 `System.security.allowDomain`, first introduced in Flash Player 6, but with a slightly different behavior in Flash Player 7 and later.

For example, Flash Player allows an application at *http://www.mysite.com/controller.swf* that needs to load another application from *http://utility.flashutils.com/helper.swf* and call methods defined in helper.swf*,* as long as the following ActionScript is in the helper.swf file:

```
System.security.allowDomain( "www.mysite.com" );
```

In Flash Player 7, this ActionScript permits any application from the *www.mysite.com* domain to script any application from the *utility.flashutils.com* domain.

Flash Player 8 introduces the ability to call `System.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 an application made for Flash Player 6 calls `System.security.allowDomain()` and a second application made for Flash Player 6 or earlier attempts to cross-script the first application, `System.security.allowDomain()` works with superdomains. For example, with the previous ActionScript example, any application from *www.mysite.com*, *store.mysite.com*, and so forth can cross-script any application from *utility.flashutils.com*, *www.flashutils.com*, and so forth. When either the application calling `System.security.allowDomain()` or the one performing cross-file scripting is made for Flash Player 7 or later, `System.security.allowDomain()` interprets domains exactly. This means that the previous ActionScript example would only permit applications from *www.mysite.com* to access those from *utility.flashutils.com*.

When an application made for Flash Player 6 calls `System.security.allowDomain()`, it permits non-HTTPS Flash applications of any version from the permitted domain to access HTTPS applications in the domain of the granting application. For example, the previous ActionScript example would permit any application from *www.mysite.com* to script HTTPS applications in the *utility.flashutils.com* domain.

In contrast, when an application made for Flash Player 7 or later calls `System.security.allowDomain()`, cross-scripting of HTTPS applications by non-HTTPS applications is not permitted. To grant permission for applications made in Flash 7 or later, authors invoke the new `System.security.allowInsecureDomain()` method. Therefore, an application at *http://www.mysite.com/controller.swf* that needs to load another from *https://secure.mysite.com/creditcard.swf* and call methods in *creditcard.swf* is only permitted by Flash Player 7 if the following ActionScript exists in the *creditcard.swf* file:

```
System.security.allowInsecureDomain( "www.mysite.com" );
```

Macromedia 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.

System.security.loadPolicyFile()

The policy file allows administrators with write access to a portion of a website to grant an application read access to that portion (see "Policy file usage" on page 29). By default, this file is located in the root directory of the target server.

Use of the default location technique is typically best, as it opens the policy file for the entire server; it is compatible with all versions of Flash Player 7 and higher, and it does not require applications to declare anything about policy files. However, if there are reasons why the policy file cannot be placed in a root location on the server, or the policy file needs to be served from an XMLSocket server, the alternative would be to use the `loadPolicyFile()` method. This API was introduced in Flash Player 7 (7.0.19.0) to allow the website to specify a nondefault location for the policy file. This mechanism is used by the Flash application to indicate to Flash Player where to look for a policy file that (if it exists and if it indicates permission) would allow that application to read data from that part of that site. An author must call this API prior to any operation that may require the policy file.

System.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.

An important change with Flash Player 8 is that previously the value of this property was scoped to an entire sandbox—changing it from one SWF file in a sandbox also changed it for all other SWF files in the same sandbox. In Flash Player 8, the value of this property is scoped to the calling SWF file only.

System.security.sandboxType

This read-only property indicates the type of security sandbox in which the calling SWF file is operating. `System.security.sandboxType` 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.

LocalConnection.allowDomain()

The ActionScript LocalConnection class lets you develop SWF files that can send instructions to each other. LocalConnection objects allow communication between two instances of Flash Player, as might occur for a Flash application that includes multiple browser windows.

Every LocalConnection object has a `LocalConnection.allowDomain()` method that accepts a set of domains that may send messages to this instance of the LocalConnection class. An author 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 LocalConnections

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.

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.

An application may also call the `LocalConnection.domain()` method to determine its own domain.

Applying the rules for legacy applications

By default, Flash Player 7 or later requires that a LocalConnection sender must come from the same domain as the receiver. In addition, applications that are served over nonsecure 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.)

These rules apply only when any of the applications are made for Flash Player 7 or later. If both are made for Flash Player 6, Flash Player uses the old rules. The old rules permit an application to make a LocalConnection call to an application from the same superdomain, and permit HTTP SWF files to make LocalConnection calls to HTTPS sources.

When an application made for Flash Player 6 calls the `LocalConnection.domain()` method, the return value is the application's superdomain. When an application made for Flash Player 7 or later calls the `LocalConnection.domain()` method, the return value is the application's exact domain.

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 when 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 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.

This is done by implementing the `allowDomain` event handler on the LocalConnection listener. The `LocalConnection.allowDomain()` method exists in Flash Player 6 and later, although its behavior was slightly modified in Flash Player 7. Previously, it used superdomains. Now the domain expected by that code is the exact domain of the caller. To preserve backward compatibility in Flash Player 7 and later, if the LocalConnection caller and the LocalConnection listener are both Flash Player 6, the argument for `LocalConnection.allowDomain` remains the caller's superdomain.

When a `LocalConnection` listener appears in an application made for Flash Player 6, Flash Player 7 or later uses its `allowDomain` handler for all `LocalConnection` calls, even when the listener is in an HTTPS SWF file and the caller is not. However, when a listener appears in an HTTPS application made for Flash Player 7 or later, and a caller makes a `LocalConnection` call in a non-HTTPS SWF file, the listener must implement a new handler called `LocalConnection.allowInsecureDomain`, or else Flash Player does not permit the call.

Macromedia does not recommend implementing `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.

Local file system options for authors

Flash Player 8 provides a mechanism to simplify workflow for those who frequently work with local SWF files, such as authors and developers using Macromedia Studio or Macromedia Flex Builder. If `LocalSecurityPrompt=Author` is present in a FlashAuthor.cfg file, Flash Player alters its local security behavior as to display the security dialog box when a local SWF file developed for Flash Player 8 accesses a resource in another sandbox.

This type of access silently fails if the FlashAuthor.cfg file does not exist (as is the case for most users of Flash Player), or if `LocalSecurityPrompt=Author` is in the FlashAuthor.cfg file. This gives individuals building a Flash application a chance to see and correct all potential failures, rather than experiencing silent failures that could be harder to detect or resolve.

Options when publishing

When publishing SWF files, authors have two choices for local file security that are new with Flash Player 8: "access local files only" or "access network only". These equate to the resultant application, when loaded as local SWF files, being put in the local-with-file-system or the local-with-networking sandboxes, respectively. These settings do not affect SWF files loaded from the network. The default is to place SWF files into the local-with-file-system sandbox. The Flash authoring tool includes a new control in the Publish Settings dialog box:

Figure 24: Flash author publish settings for Flash Player 8 security
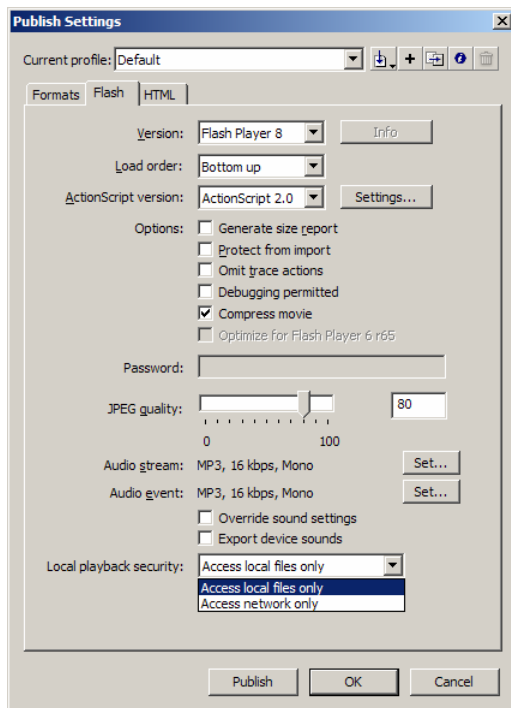
The local content updater is another way to have a SWF file be in the local-with-networking after compilation. This is valuable for publishers who would like to modify the sandbox as a post-processing step. The local content updater is distributed as both binaries and source, and is available at http://www.macromedia.com/support/flashplayer/downloads.html.

ActiveX control and browser plug-in APIs

Applications hosting the Macromedia 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 the following section, "Hierarchy of local file security controls."

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 about the new local file sandboxes introduced in Flash Player 8, 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 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 UserTrust control in the mms.cfg file. By default, users are allowed to specify which SWF files should be placed into the local-trusted sandbox with either the User Flash Player Trust directory or by using the Settings Manager.

An additional set of controls is provided specifically for backwards compatibility with legacy SWF files. Administrative users or regular users may indicate that SWF files of versions prior to Flash Player 8 that are not explicitly trusted based on the controls listed previously should (or should not) be placed into the local-trusted sandbox. By default, any SWF file prior to Flash Player 8 is placed into the local-with-file-system sandbox. If that legacy SWF file attempts to access the network, Flash displays a dialog box to the user, indicating that the file may require the user to specify that the file should run in the local-trusted sandbox to operate correctly.

The administrative user may allow or disallow legacy support via the `LocalFileLegacyAction` option in the mms.cfg file. In either case, this dialog box does not appear and the administrator's preference is exercised. If the administrative user does not specify a preference in the mms.cfg file, the Settings Manager provides the user with a mechanism to indicate that all SWF files prior to Flash Player 8 should be placed into the local-trusted sandbox. If neither the administrative user nor the end user exercises these controls, the default behavior applies and the dialog box may appear.

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.

Macromedia provides a post-compiler tool (and source code) to configure a SWF file to use the local-with-networking sandbox at: www.macromedia.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 8 loads local SWF files into the local-with-file-system sandbox. This is a change from Flash Player 7 and earlier, which placed all local SWF files into the local-trusted sandbox.

Flash Player integration with native applications

The new 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 Flash Player 8 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.

Macromedia 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.

The default behavior of the browser plug-in has been changed in Flash Player 8, so it may be possible for an application to function incorrectly after the administrative user upgrades from an earlier version of Flash Player. Therefore, administrative user and user controls provide functionality equivalent to the `DisableLocalSecurity` API for applications that use the browser plug-in. The Global Flash Player Trust directory allows administrative users to specify whether a particular native application that integrates the browser plug-in should use the local-trusted sandbox for all local content. If the administrative user wants, he or she can provide (or not provide) this control to the user with the UserTrust control in the mms.cfg file. By default, users are allowed to specify the use of the local-trusted sandbox for all local content, and they can use either the User Flash Player Trust directory or the Settings Manager to indicate that an application should use only the local-trusted sandbox.

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 8. However, with some of the new security characteristics of Flash Player 8 (such as the enhanced sandbox model), there are some potential security-related runtime effects with the new environment. (These are highlighted throughout this document.)

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. (Later 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 Macromedia. (For instance, features of Flash Player 8 are common to all Flash Player 8 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, CompuServe, and Opera), Macintosh plug-ins (such as those for Safari, AOL, CompuServe, Opera, Netscape, and Internet Explorer for Macintosh), and ActiveX control implementations (such as those for Windows Internet Explorer and AOL). For the complete list of currently supported options, see the following listing of system requirements:

> http://www.macromedia.com/software/flash/productinfo/systemreqs/

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 Macromedia partnership agreements). Typically, the administrative user is the one who installs (or 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 37). 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 Macromedia, include a version of Flash Player specifically for integrated previewing of content. While creating a Flash application, the author can run their 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 8), 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 his 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 only be available to an external SWF file if they were delivered with the SWF file.

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, but 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 (such as one loaded as a current SWF file from the web and executed in a more current Flash Player environment) would encounter on the same computer.

Additionally, the version of Flash Player embedded within a projector file can be signed by the distributor, but it is *not* signed by Macromedia. 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 " on page 44.)

Other distributions

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

Platform and runtime environment

This document focuses on the security protections and access rules that apply for code and data running in Flash Player. All authors and users should be aware that Macromedia cannot make security claims for, or significantly modify or enhance, the security capabilities and attributes of the infrastructure in which its products execute. Macromedia 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 Macromedia 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 Macromedia 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 Macromedia products target support for all reasonable combinations.

Flash Player adds significant security features over earlier versions of Flash Player, and over what is typically included in other runtime environments. 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.

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. Flash Player 8 includes some new application deployment configurations, and therefore some new potential security-related concerns.

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. The following sections note the rules, enforced by Flash Player, about what any SWF file within a given sandbox can access. (For more information, see "Basic sandbox security model" on page 8.)

Network SWF files

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. (For more information, see "Basic sandbox security model" on page 8.) Network SWF files can be allowed to access additional data from other domains by explicit website and author permissions. (For more information about cross-domain policy files, see "Website controls" on page 28.)

Local SWF files

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 "\\192.168.0.1\test.txt" are not considered local files.

Local SWF files from local origins, such as local file systems or UNC network paths, are placed into one of three sandboxes in Flash Player 8 (this is a change from earlier versions). By default, local SWF files are placed in the local-with-filesupport sandbox. Local SWF files that the author has decided should have network access are placed in the local-with-networking sandbox.

Local SWF files that are registered as trusted (by users or by installer programs) are placed in the *local-trusted* sandbox. Users also have the ability to reassign (move) a local SWF file to or from the local-trusted sandbox based on their security considerations.

There are three groups that can make security choices: the author (using developer controls), the administrative user (using administrator controls), and the local user (with user controls). For information about the available options and mechanisms for each of these three areas, see "Permission controls" on page 17. Communication between the local-with-network and local-with-file-system sandbox is strictly forbidden. Permission to allow such communication cannot be granted by any stakeholder.

Local-with-file-system

For security purposes, Flash Player 8 places all local SWF files, including all legacy local SWF files, in the local-with-file-system sandbox, by default (unless some other setting is made). For some SWF files built for earlier versions, operations that were allowed prior to Flash Player 8 may be restricted, but this provides the most secure default for the users' protection.

From this sandbox, SWF files may read local files (by using the `XML.load()` method, for example), but they may not communicate with the network in any way. This assures the user that local data cannot be leaked out to the network or otherwise inappropriately shared.

Local-with-networking

When local SWF files are assigned to the local-with-networking sandbox, they forfeit their local file access. In return, the SWF files are allowed to access the network, with no restrictions on where they may send data. However, a local-with-networking SWF file still is not allowed to read any network-derived data unless permissions are present for that action. Therefore, a local-with-networking SWF file has no local access, yet it has the ability to transmit data over the network and can read network data from those sites that explicitly allow reading to the local-with-networking sandbox through their site-specific access permissions.

Local-trusted

SWF files assigned to the local-trusted sandbox can interact with any other SWF files, and load data from anywhere (remote or local).

Earlier versions of Flash Player (those prior to Flash Player 8) treated *all* local SWF files as members of the local-trusted sandbox that is available in Flash Player 8. This allowed authors to work freely with content under development, and allowed flexibility for SWF file deployments on CD-ROM or other local media. The basis for this rule was the assumption that if a user consented to download or install a SWF file, they were implicitly indicating their trust for that SWF file. This type of local file security is also consistent with other scripting models, such as those used by JavaScript files.

With the proliferation of SWF files and their sources, and the variety of ways to distribute them, the Flash platform required additional sandboxes. Therefore, in Flash Player 8, the three separate sandboxes described here allow for the differentiation of access controls enforced by Flash Player 8 for three distinct types of local SWF files.

Executable projector files

From a security perspective, especially considering the perception and acceptance of security-sensitive users, authors may not want to deploy 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.

In addition, 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 (other than possible simple assertions by the distributor) 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 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 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, 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 `System.security.allowInsecureDomain()` method to allow data sharing.

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 persistent sockets (through the ActionScript XMLSocket object), 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 the XMLSocket object establishes and maintains an open connection to the server, the XMLSocket object has restrictions for security reasons. By default, the `XMLSocket.connect()` method can connect only to TCP port numbers greater than or equal to 1024. One consequence of this restriction is that the server daemons that communicate with the XMLSocket object should also be assigned to port numbers greater than or equal to 1024. Port numbers less than 1024 are often used by system services, such as FTP, Telnet, and HTTP, thus the XMLSocket object is by default unable to access these services. The port number restriction limits the possibility that these resources will be inappropriately accessed and abused. By default, the `XMLSocket.connect()` method can connect only to computers in the same domain where the SWF file resides.

HTTP servers provide the cross-domain policy files that govern XMLSocket connections. A policy file can enable cross-domain access or access to ports with numbers less than 1024.

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 check the validity of, and view, 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.