Flash Security

Mihai Corlan

Platform Evangelist, Adobe

OWASP, Copenhagen

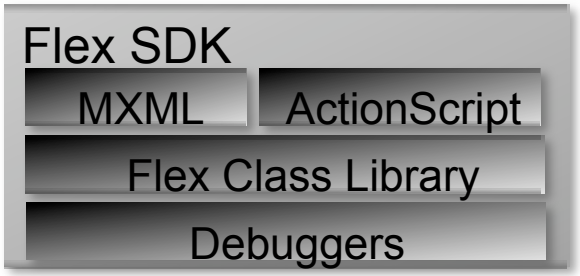
About me

- Platform Evangelist, member of the Adobe's European team
- Used to be a Flex Builder engineer (Java Desktop SWT, Eclipse Platform)
- Used to be a web developer (PHP, ColdFusion, DHTML, JS, MySQL, Sybase…)
- ~9 years experience with web related technologies
- I write articles, I do presentations, and I code

Today's Agenda

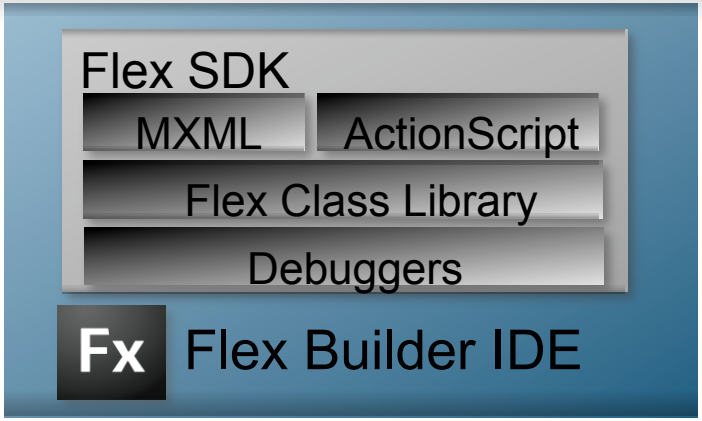
- Short overview of Flash Platform Runtimes and Frameworks
- Flash Player security
- Adobe AIR security
- Q & A

What is Flex?



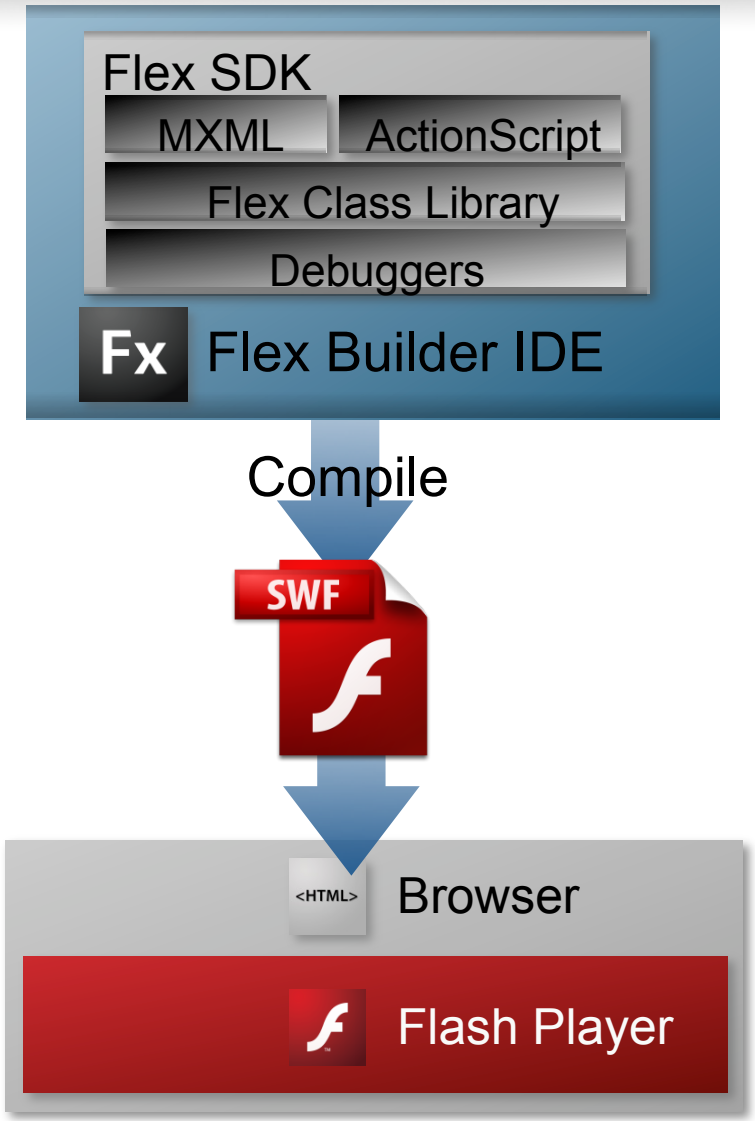
- Flex SDK
  - 2 languages
    - MXML
    - ActionScript 3
  - Compilers
  - Rich Component Library
  - Debuggers

What is Flex?



- Flex SDK
  - 2 languages
    - MXML
    - ActionScript 3
  - Compilers
  - Rich Component Library
  - Debuggers
- Flex Builder IDE
  - Eclipse Plug-in or turn-key install
  - Accelerates Design and Development
  - Design view and code view

What is Flex?

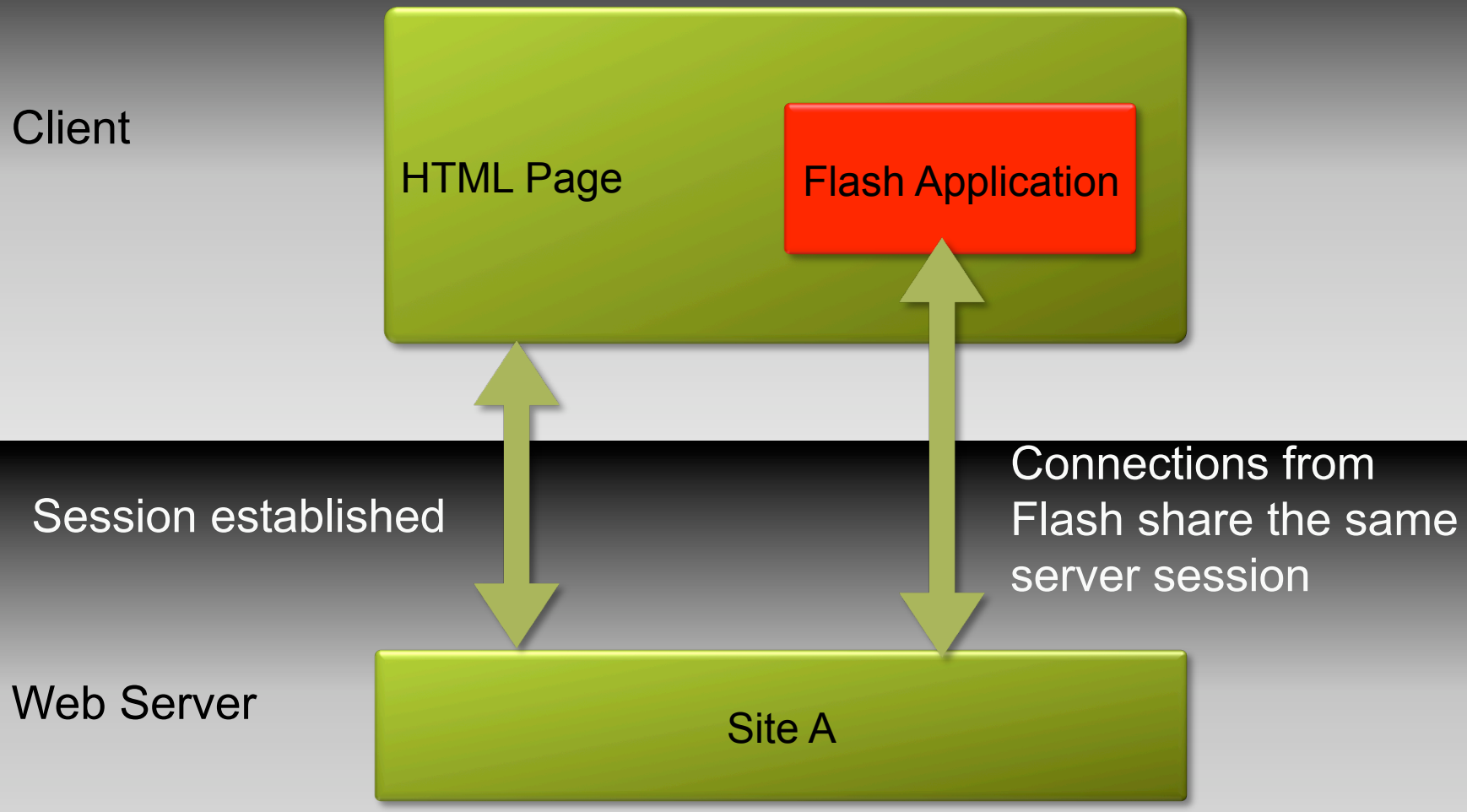


- Flex SDK
  - 2 languages
    - MXML
    - ActionScript 3
  - Compilers
  - Rich Component Library
  - Debuggers
- Flex Builder IDE
  - Eclipse Plug-in or turn-key install
  - Accelerates Design and Development
  - Design view and code view

Flash Player Security

- Flash Player and browser sessions
- Crossdomain.xml
- Flash Player and JavaScript / HTML
- User-initiated Action Requirements
- Loading vs. Importing
- Local Connection
- HP SWFScan application

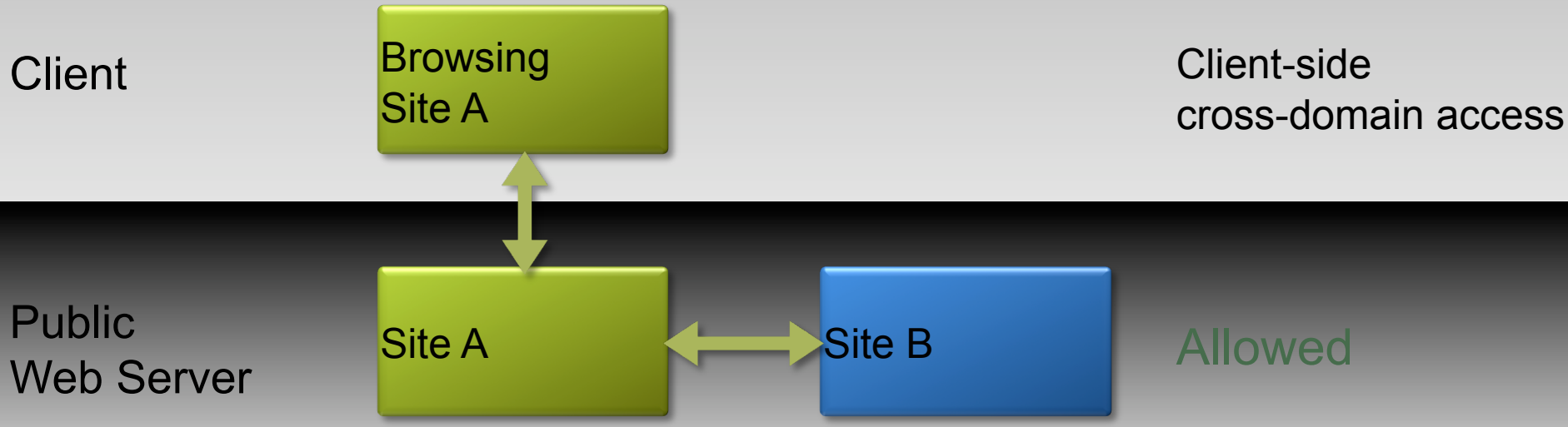
Flash Player and browser sessions



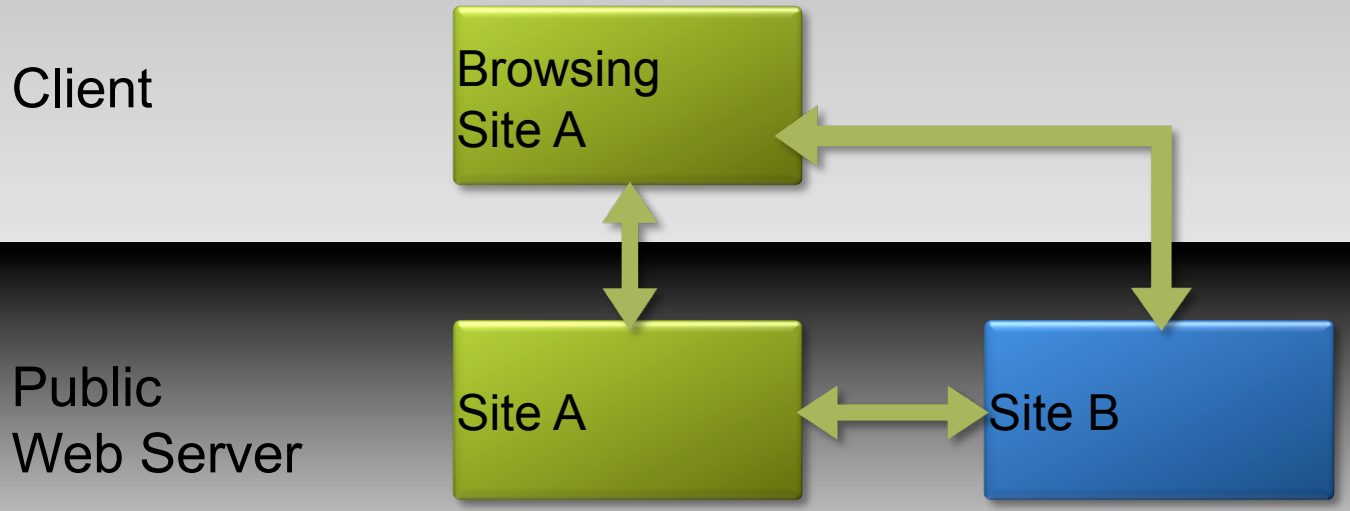
Flash Player and HTTP / HTTPS

- When you send sensitive data from Flash applications you should use HTTPS , and not HTTP.

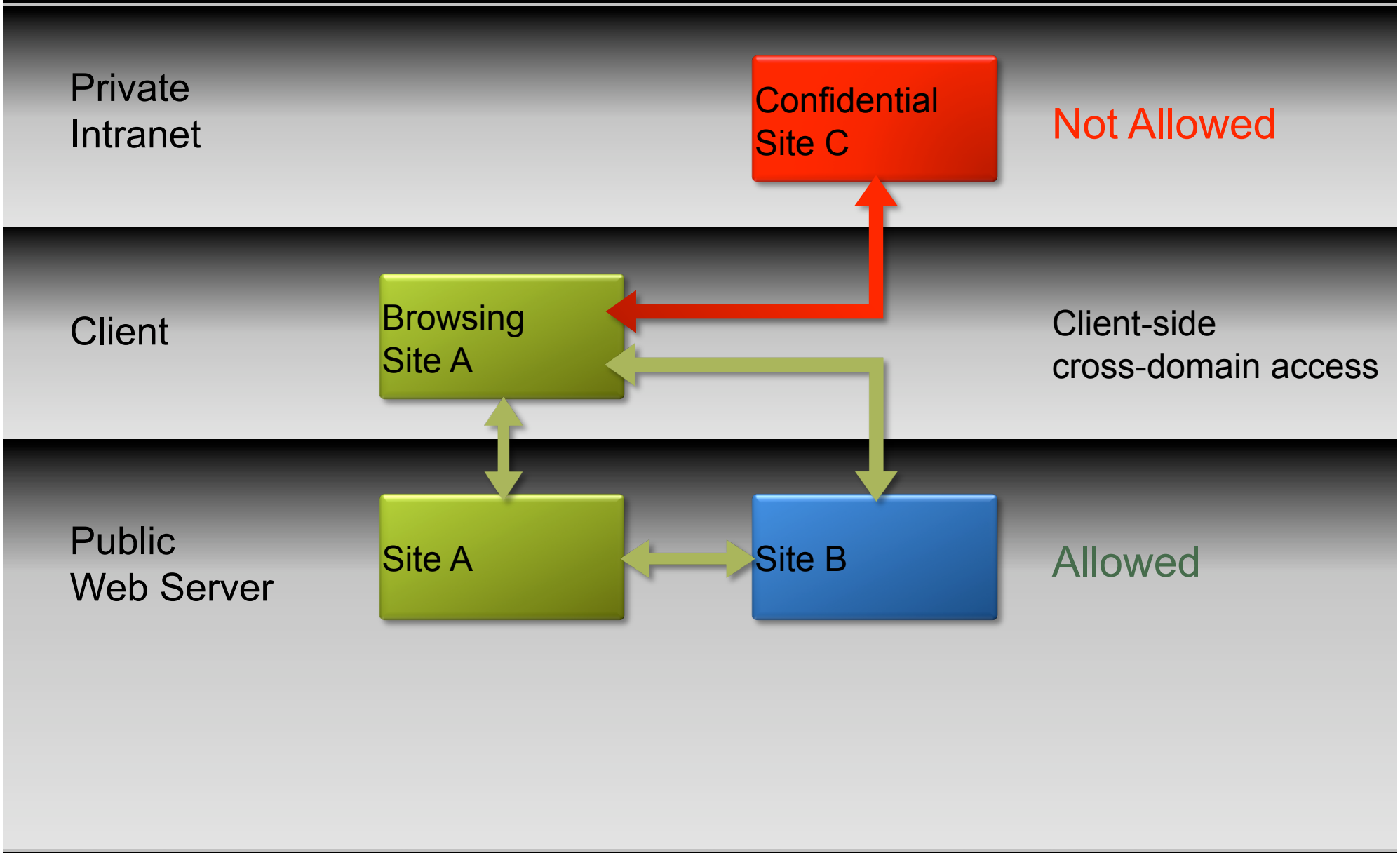
Crossdomain.xml



Crossdomain.xml



Crossdomain.xml



Crossdomain.xml

Private
Intranet

Confidential
Site C

Not Allowed

Client

Browsing
Site A

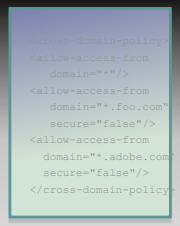
Client-side
cross-domain access

Public
Web Server

Site A

Site B

Allowed



crossdomain.xml

Crossdomain.xml

- Crossdomain.xml policy file present on 36% of Alexa 100 sites*

Yahoo	Forbes	eBay	ABC
MySpace	Walmart	NY Times	CBS
MSN	Travelocity	Amazon	CNET
YouTube	CNN	AOL	MLB

- Client implementation in Flash Player today.
- W3C is working on similar mechanism called Access Control
- For more information: http://adobe.com/go/crossdomain

Crossdomain.xml

```
<?xml version="1.0"?>
<!-- http://www.adobe.com/crossdomain.xml -->
<cross-domain-policy>
        <allow-access-from domain="www.a.com" />
        <allow-access-from domain="www2.a.com" />
</cross-domain-policy>
```

Cross domain offer protection for cross domain loading data, not for sending data from Flash to server.

Flash Player and JavaScript / HTML

- *allowScriptAccess* parameter for Object tag

If set to *sameDomain*, only the SWFs loaded from the same domain as the HTML page can access the JS/HTML DOM of the page

Use wise allowScriptAccess set to *all* for SWFs loaded from other domains!

- *allowNetworking* parameter for Object tag

1. *All* – Application can communicate with the browser and networking
2. *Internal* – communication to browser is cut, only networking communication is available
3. *None* – no external communication is allowed (browser/networking)

User-initiated Action Requirements

Full Screen Mode

To counter Spoofing threats, the application can enter in Full Screen Mode only if the user initiate this by triggering a click/keypress event

allowFullScreen can be used to restrict the Full Screen Mode for untrusted Flash apps (loaded from different domains)

While in Full Screen Mode, you cannot type in in form fields

User-initiated Action Requirements

- writing to the user's Clipboard
- creating pop-up windows
- launching a FileReference dialog box (download/upload files)
- certain POST operations (when a file is uploaded)

Loading vs. Importing

Loading

- The loaded SWF remains in a separate domain of security (no access to the loader Page/SWF)
- Loader.load(URLRequest("http://somesite.com/my.swf"))
- Iframe

Importing

- Brings the imported SWF in the same domain of security as the loader
- Loader. loadBytes(URLRequest("http://somesite.com/my.swf"))
- Loader.load() and LoaderContext.securityDomain = SecurityDomain.currentDomain

Loading: cross-scripting vs. cross-domain scripting

Cross-scripting

SWFs loaded from the same domain can modify by default each others variables, objects, properties etc

- http://mysite.com/swfA.swf
- http://mysite.com/swfB.swf
- swfA can script swfB, and vice-versa

Loading: cross-scripting vs. cross-domain scripting

Cross-domain scripting

SWFs loaded from different domains **CAN NOT** modify by default each others variables, objects, properties etc

- http://mysite.com/swfA.swf
- http:/someothersite.com/swfB.swf
- swfA can not script swfB, and vice-versa

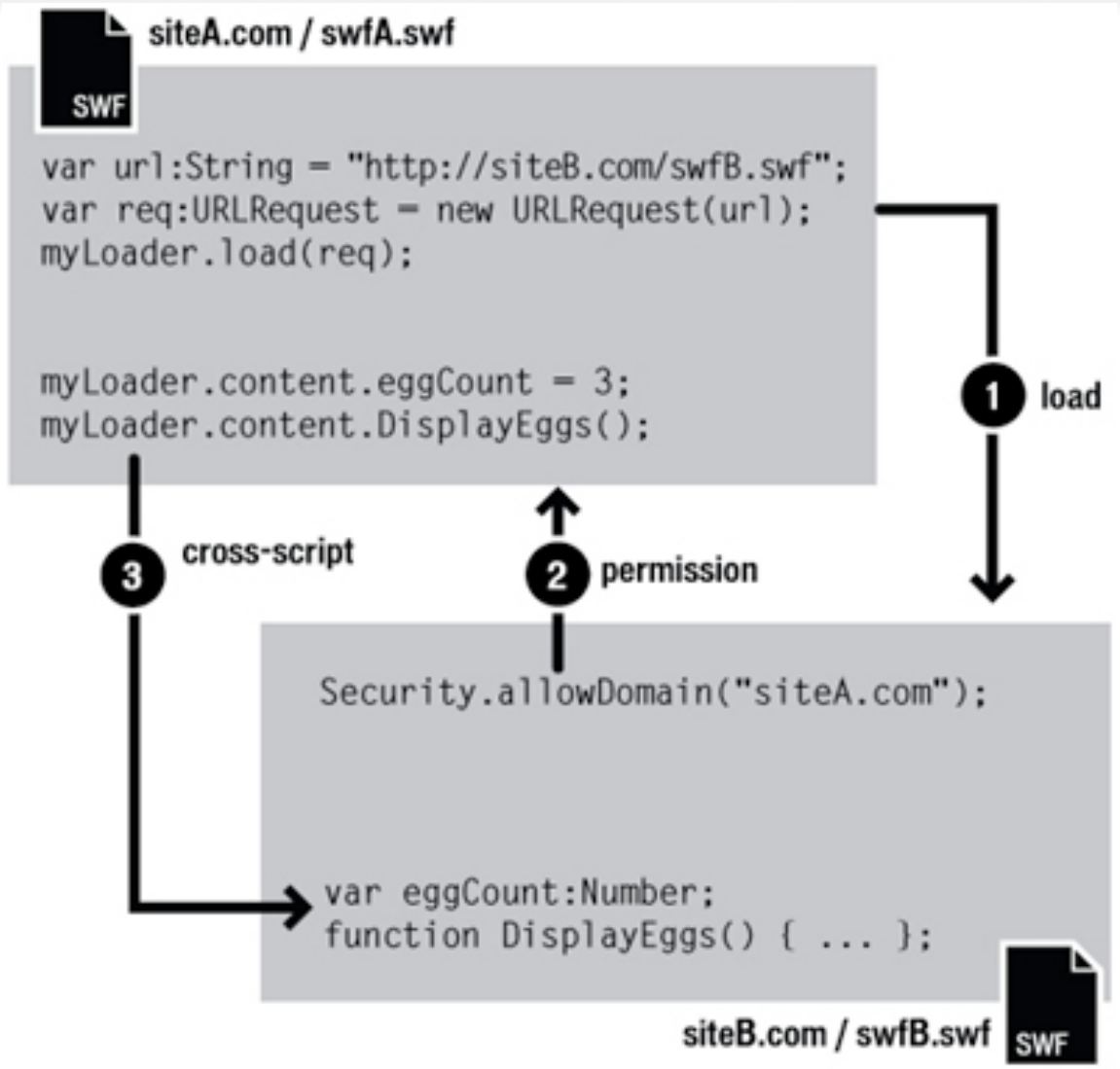
Loading: cross-scripting vs. cross-domain scripting

Cross-domain scripting

- http://mysite.com/swfA.swf
- http:/someothersite.com/swfB.swf
- swfA can script swfB, if in swfB Security.allowDomain("swfA domain")
- swfB can script swfA, if in swfA Security.allowDomain("swfB domain")

When HTML-to-SWF scripting crosses domains, the SWF file being accessed must call Security.allowDomain(), just as when the accessing party is a SWF file, or the operation will fail.

Loading: cross-scripting vs. cross-domain scripting



Loading: cross-scripting vs. cross-domain scripting

Cross-domain scripting: HTTPS and HTTP

- https://mysite.com/swfA.swf
- http:/somesite.com/swfB.swf
- swfB can script swfA, if in swfA Security.allowInsecureDomain()

Use this with caution! It is better to not allow unsecure content to access secure content!

Local Connection

- Use Local Connection to communicate between two ore more SWF that run on the same machine (could be loaded in the same web page, in different browsers/browser tabs)
- 40KB limit per message
- Opt in

```
//receiving_lc is in http://mysite.com/receiving.swf
receiving_lc.allowDomain("store.example.com");
receiving_lc.connect('connectionName');
function myMethod():String { return "Hello World"};

// sendingLC is in http://store.example.com/sending.swf
// Even though the receiving SWF uses full domain, the sending SWF
// only uses the superdomain
sendingLC.send("example.com:connectionName", "myMethod");
```

HP SWFScan

- HP created SWFScan app to test Flash App for vulnerabilities
- Decompiles the SWF file and performs static analysis on the code
- Detects the problems, higlight them in the source code, so you can fix
- It looks for ~ 50 vulnerabilities (hard-coded passwords, developer debugging information, XSS vulnerabilities)
- It's free

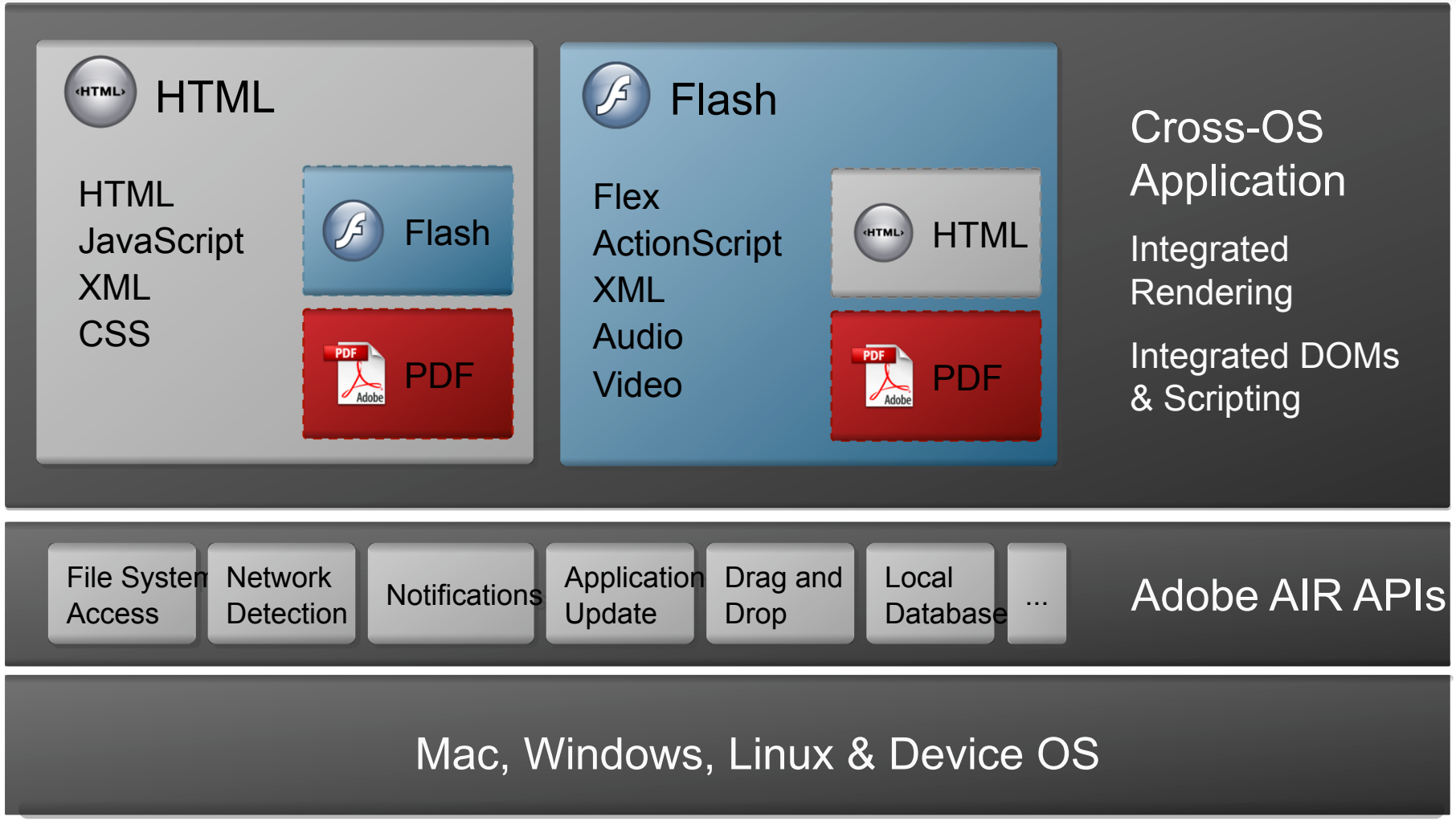
Other things to keep in mind

- Compile your app for the latest version of Flash Player whenever is possible
- Omit Trace Actions compiler flag: strip debugging code from the compiled file
- Validate the data you receive from the user input or from other apps
- Always set a mask for Loader objects (protection against spoofing) when loading untrusted SWFs

What is AIR?

Adobe® AIR™ lets developers use their existing web development skills in HTML, AJAX, Flash and Flex to build and deploy rich Internet applications to the desktop on Windows, Mac or Linux.

Adobe AIR Application Stack



Outline

- Application Signing
- Update Framework
- Storing data locally
- AIR Security Sandboxes
- Validates Data

0

What is AIR

- Conceptually similar to EXEs files
- Digitally signed + user approval for installation
- Created with HTML/JS or/and ActionScript
- Cross platform: Windows, Mac and Linux
- Full access to the desktop
- Runs with user permissions

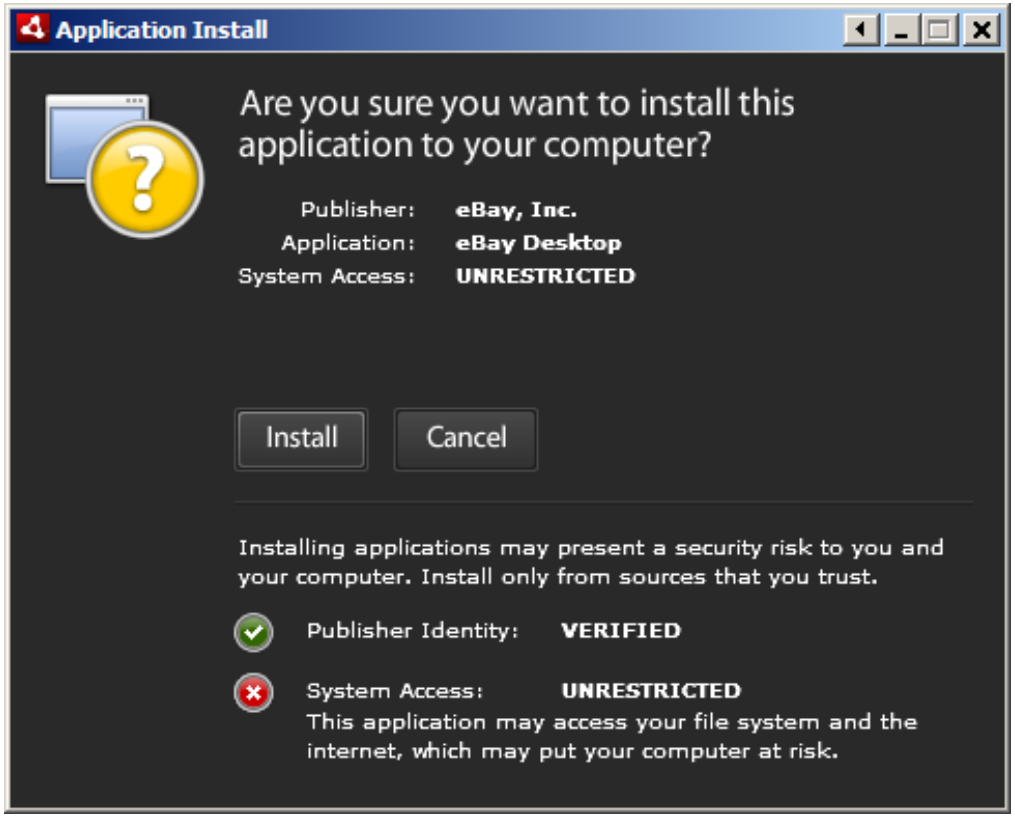
Application signing

- How?
  - Using a digitally signed certificate from an authorized certificate authority (CA)
- What type of certificate?
  - AIR requires certificates designed specifically for use in code signing
- What it does?
  - Validates the source of the application (the application publisher is indeed the one from the certificate)
  - Ensures that the application wasn't modified in its way from the server to the user machine (server content replaced, middle man attack)
- Where to buy?
  - Any big Certificate Authority can be used

Application signing

- Adobe AIR does not provide any facilities for directly managing trusted certificates and, therefore, no facilities for managing trusted identity
- These facilities are already provided by your operating system, along with tools that you can use to add and remove trusted certificates.
- Adobe AIR considers a certificate to be trusted if:
  - either the same certificate is stored in the system certificate store as a trusted certificate
  - or if it is possible to establish a certificate chain from the signing certificate to some trusted certificate in the system store

Application signed with a trusted certificate



Application signed with self generated certificate

Workflows for signing an application

There are two different workflows supported by the Flex Builder IDE and ADT for signing :

1. You can sign the application from Flex Builder while exporting for release
2. Or your programmer can export an intermediate file format (AIRI):
    1. This file cannot be installed.
    2. Using this file, the person who has the certificate, can sign the application using ADT command line tool

Migrate your app from one certificate to another

You can change the certificate used for signing, from the existent one, to a new one as long as the old certificate is still valid.

1. Sign the application with the new certificate
2. Use ADT tool with –migrate command to sign the .air file with the old certificate

Thus you can:

- Migrate from self signed certificate to one issued by a CA
- Changing from one commercial certificate, to another

It works both for new installations, and for updates.

More on signing an application

- You can not sign an AIR application with an expired or revoked certificate
- You can sign an AIR application with a renewed certificate
- You can install an AIR application that was signed with a valid certificate, even though in the mean time the certificate expired

Application Identity

- AIR uses certificates to establish the application identity
- Application identity is used to securely indentify application when:
  - Update the application
  - When the browser API is used for install, detect, launch applications from web browser
  - Communicate through LocalConnection with other AIR or web apps
  - Indentify the application when it access the Encrypted Local Store (ELS)

Updating AIR applications

The update framework provided by the AIR 1.5 offers a secure way to update an installed application:

1. By using the version number from the application descriptor file it isn't possible to update an installed version with an older version
2. A signed application with a certificate, cannot be updated by another version unless it is signed with the same certificate like the original installation (though it can be a renewed one). Exception apps packed with migrate command.
3. The publisher ID is the same even if the updated version is signed with a renewed Certificate

Updating AIR apps with Update Framework

Using the Update Framework you can implement easily this workflow:

- Each time the application is launched, first checks for a new update
- If an update is found, you can choose to ask for user permission or you can automatically update the app
- Once the update was downloaded, it is installed, and the application is automatically restarted

With this workflow, the user, as long as it is online, doesn't have a choice, the application will be updated no matter what.

It is pretty flexible, you can highly customize.

Updating AIR apps with Update Framework

It comes in two flavors:

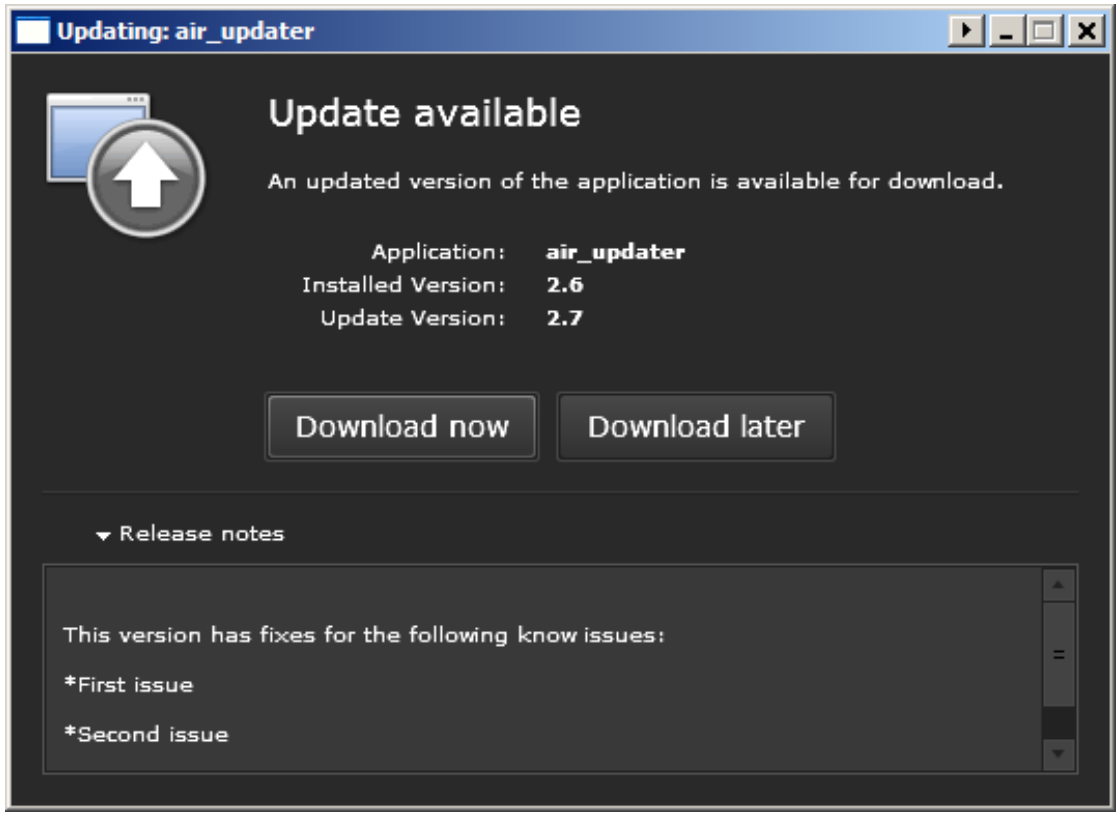
- With UI
- Without UI – you must implement your own UI

It can be used for AIR apps created with:

- Flex (both flavors)
- AJAX (both flavors)
- Flash (only the non UI)

Updating AIR apps with Update Framework

You can display to the user a dialog with information about the reason for updating:



Storing data locally

AIR offers different ways to store data on the client machine:

1. Save files locally in the file system
2. Serialize ActionScript objects to files in the local system file
3. Save ActionScript objects to the Encrypted Local Store
4. Save data using the local SQLite database

Storing data locally

The first two don't offer any security; any other application can read the info.

You should use *app-storage:/* directory. Thus you can keep the data per user account and application. If the same application is used on multiple accounts on the same machine, the data will not be overwrite.

Do not create or modify files in *app:/* directory

Encrypted Local Store

Encrypted local store, offers security:

- AIR uses DPAPI on Windows® and KeyChain on Mac® OS® to associate the encrypted local store to each application and user. The encrypted local store uses AES128-bit encryption
- It is good for small chunks of data; 10MB space per Application – performance limit
- The content is available only to the code run from the Application Sandbox
- By default the data is bound to the Publisher ID; you can bound it also to the application bits; downside – if the application is updated, you cannot read the data anymore
- One ELS per application and user account

EncryptedLocalStore.setItem(key:String, data:ByteArray, stronglybound:Boolean)

Encrypted Local Databases

- SQLite stores the whole database in a single file, and this file will be encrypted
- You use a key (16 bytes) for encrypt/decrypt the database file
  - You can use SHA2-256 for generating the key, via as3corelib library
- A database created as a non encrypted one, cannot be encrypted later

```
sqlConnection =  new SQLConnection();

sqlConnection .open(reference:Object = null, openMode:String = "create",
    autoCompact:Boolean = false, pageSize:int = 1024, encryptionKey:ByteArray = null):
```

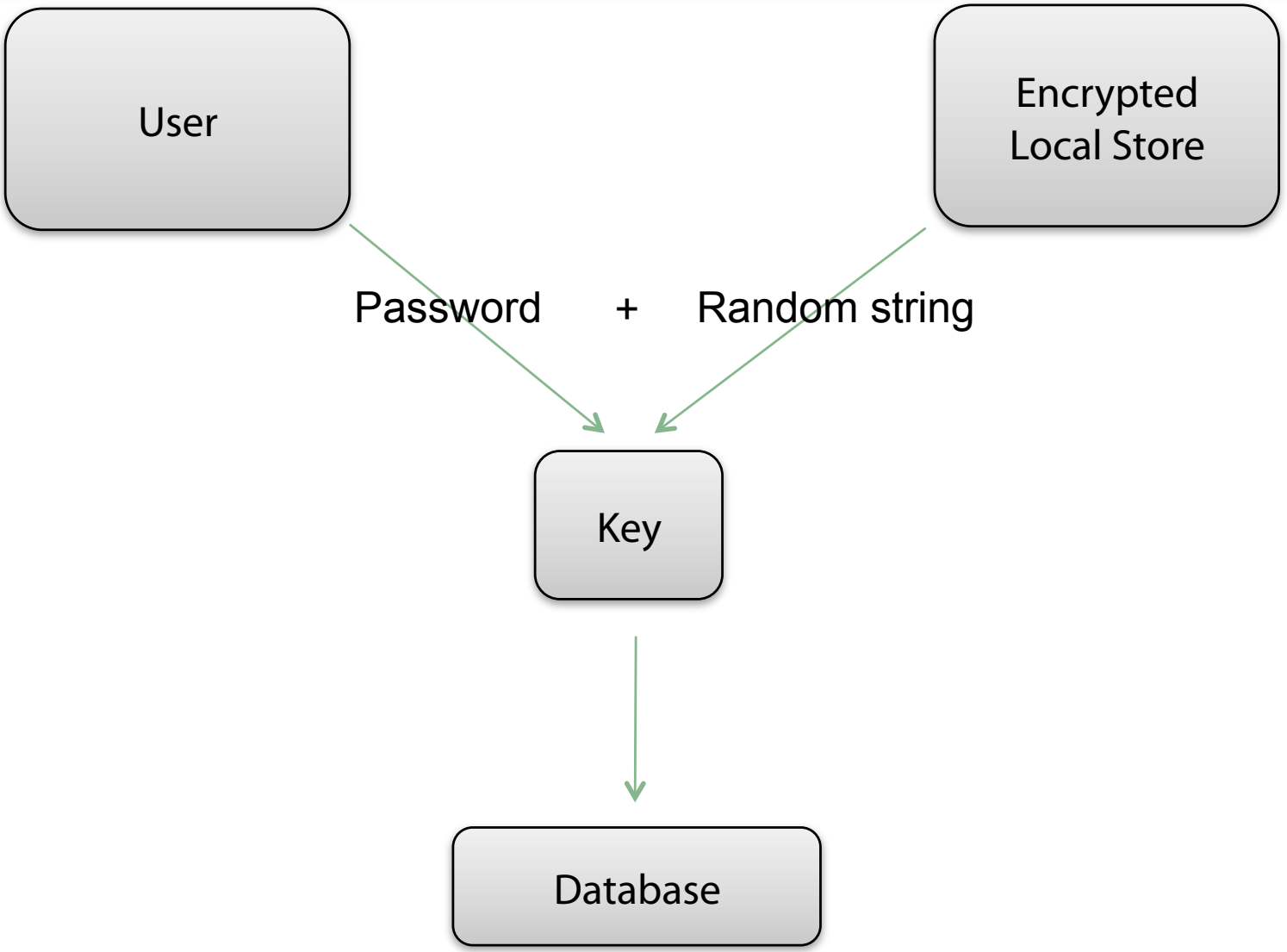
Encrypted Local Databases

Things to keep in mind:

- Encrypting the database does not prevent against SQL injection attacks
- The security of the database depends on how you keep the key safe. AIR applications can be decompiled, so do not hard code the key within the application!
- For the key you can use the user password, or you can generate a random number and combine the password with this number to get the key
- You can use ELS for keeping the key or parts of the key
- You can use SHA256.hash() from as3Corelib library for hash passwords

Look at how **BlackBookSafe** handles this.

Encrypted Local Databases



Sandboxes

There are two sandboxes from the point of view of security and how the code is executed:

1. Application sandbox – permits access to the privileged AIR APIs (read/write OS files…).
   - Any file that is coming from within the AIR installer file it is put in this sandbox
2. Non-application sandbox – this content doesn't have access to AIR APIs
   - Any content loaded from local or remote that is not coming from the application installation folder

More on SandBoxes

You can execute the code coming from the application installation folder in a non-application sandbox for extra security

Use *sandboxRoot* and *documentRoot* for this

Communicate between Sandboxes

You can use SandboxBridge API to communicate between the two sandboxes.

- Any data passed through, it is passed by value, and not by reference; there is no reference leaked
- You shouldn't expose generic functions to the Non-aplication sandox: **deleteConfigFile()** instead of **deleteFile(fileToDelete:String)**
- There is no security through obscurity. Any methods exposed through SandBoxBridge can be discovered on the other side
- Opt-in on both sides

Imported vs. Sandboxed

You can load executable content (SWF, HTML/JS) in two ways:

1. Sandboxed: stays in its own domain;
    1. HTML: <frame>, <iframe>, <img> tags
    2. SWF: Loader.load(), SWFLoader
2. Imported: runs in loader's sandbox with loader's privilege
    1. HTML/JS: eval(), innerHTML, <script>
    2. Loader.loadBytes(), HTMLLoader.loadString()

When you import external content to Application Sandbox, you give access to the AIR APIs for this content!

AIR Prevents Accidental Importing

You can load executable content (SWF, HTML/JS) in two ways:

- In HTML, eval() and friends are restricted.
  - Before onLoad, they operate normally
  - After onLoad, they will not generate code.
  - Restricted eval() supports pure JSON, but some systems require generated code.
- Loader.loadBytes() and HTMLLoader.loadString() require opt-in
  - LoaderContext.allowLoadBytesCodeExecution (AIR Only Requirement)
  - HTMLLoader.allowLoadStringInAppSandbox (NEW in AIR 1.5!!!)

Verify Imports

Especially when you import in the Application Sandbox, you should verify that the imported code is what you assume to be.

The safest way to do that, is by using code signing.

Bad news: there is no built-in support into the runtime and SDK for easy verifying

Good news: Alchemy could be used to port a library from C/C++ to ActionScript, and write your own framework to verify the content.

Server name and Server + SSL are not good enough. You can have DNS attacks, server content attacks, man in the middle.

Data Validation

What you do in web (never trust the client), you should do in AIR: always validate the data.

Use Flex Validators, Flash Validators (Google Code Project) to validate data

SQL injection: when writing data in SQLite, use prepare statements

Intead of:

```
employees.text = "SELECT  FROM employees WHERE employeeID = " + remotedata.ID;
employees.execute();
```

Do:

```
employees.text = "SELECT FROM employees WHERE employeeID = :empID";
employees.parameters[":empID"] = remotedata.ID;
employees.execute();
```

Thank you!

Today's presentation: http://corlan.org/downloads/FlashSecurity.pdf

Flash Player Security center:
http://www.adobe.com/devnet/flashplayer/security.html

HP SWFScan: http://www.adobe.com/devnet/flashplayer/articles/swfscan.html

AIR Security: http://tv.adobe.com/#vi+f15384v1025

Mihai Corlan

Blog: http://corlan.org

E-mail: mihai.corlan@adobe.com