

Maintaining security with Adobe AIR

Mihai Corlan

Adobe Evangelist

December 3rd, 2008



AIR is secure as Fort Knox!



There is no point of discussing any more, let's have a beer instead!

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

Today's presentation is about

- Techniques to make the application secure against attacks
- Techniques to make the user data stored by the application secure
- It is not about DRM (protecting the content against piracy)

- 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

Why security is so important?

When a web app is compromised, usually the server data are exposed.

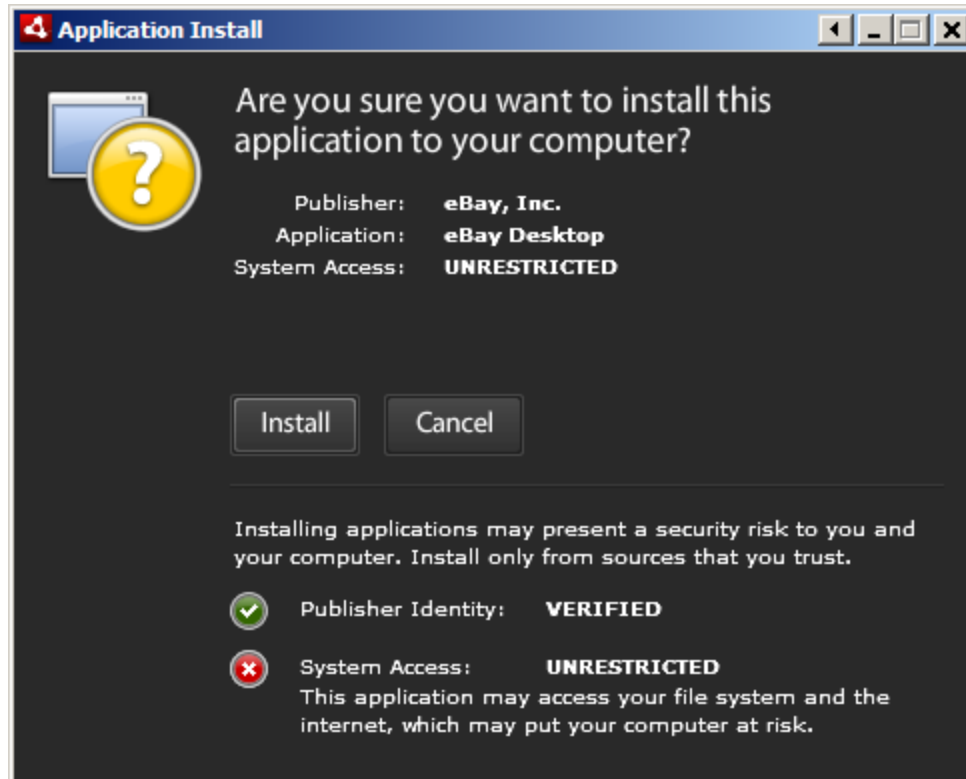
When a desktop app is compromised, the attacker can gain the control over the entire machine.

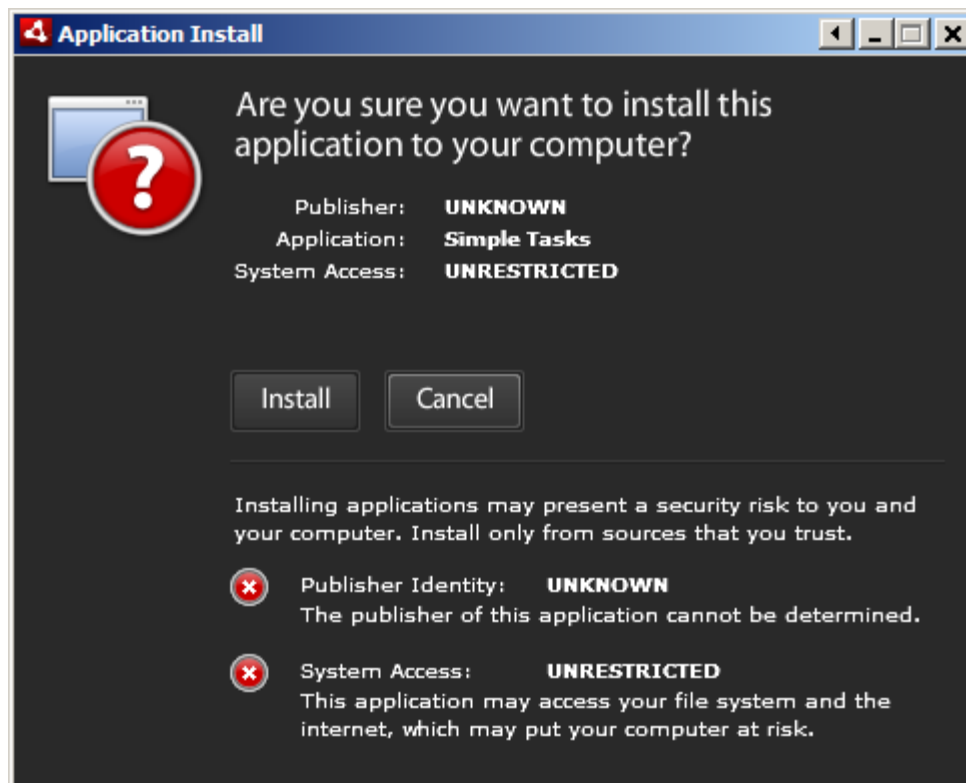
Adobe AIR offers a set of tools that can assist you while implementing secure applications.

- **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

- 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





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

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.

- 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

- 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)

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

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.

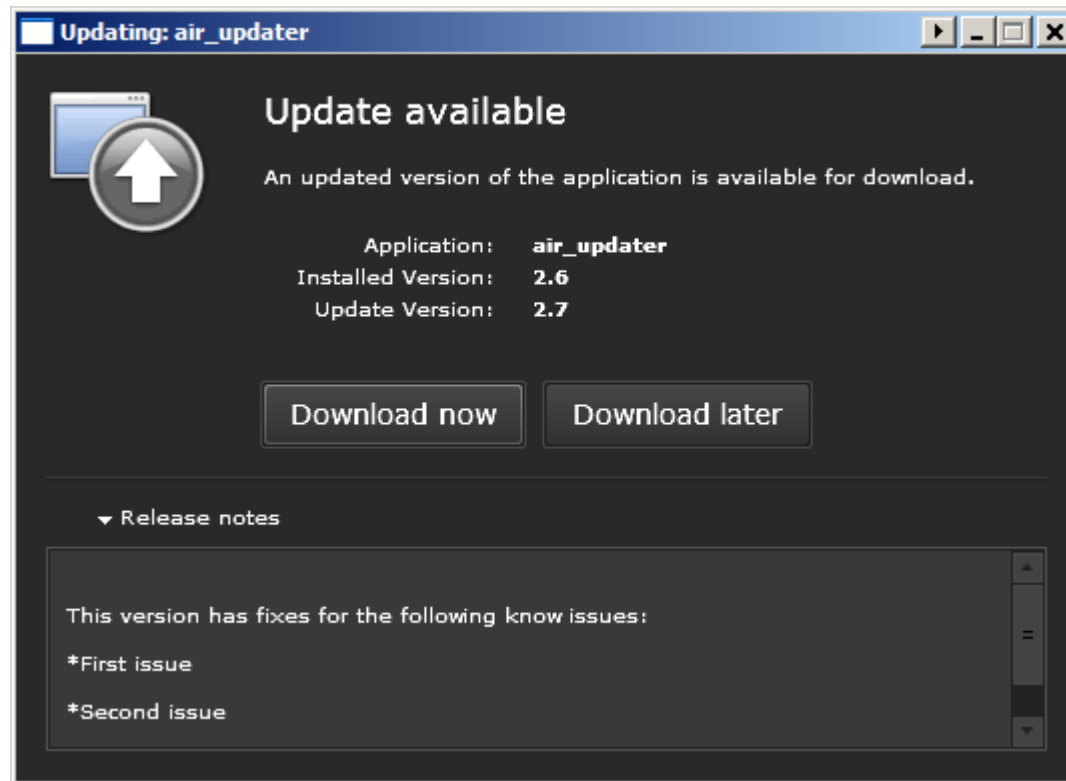
It comes in two flavors:

- 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)

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



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

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, 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)`

- 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

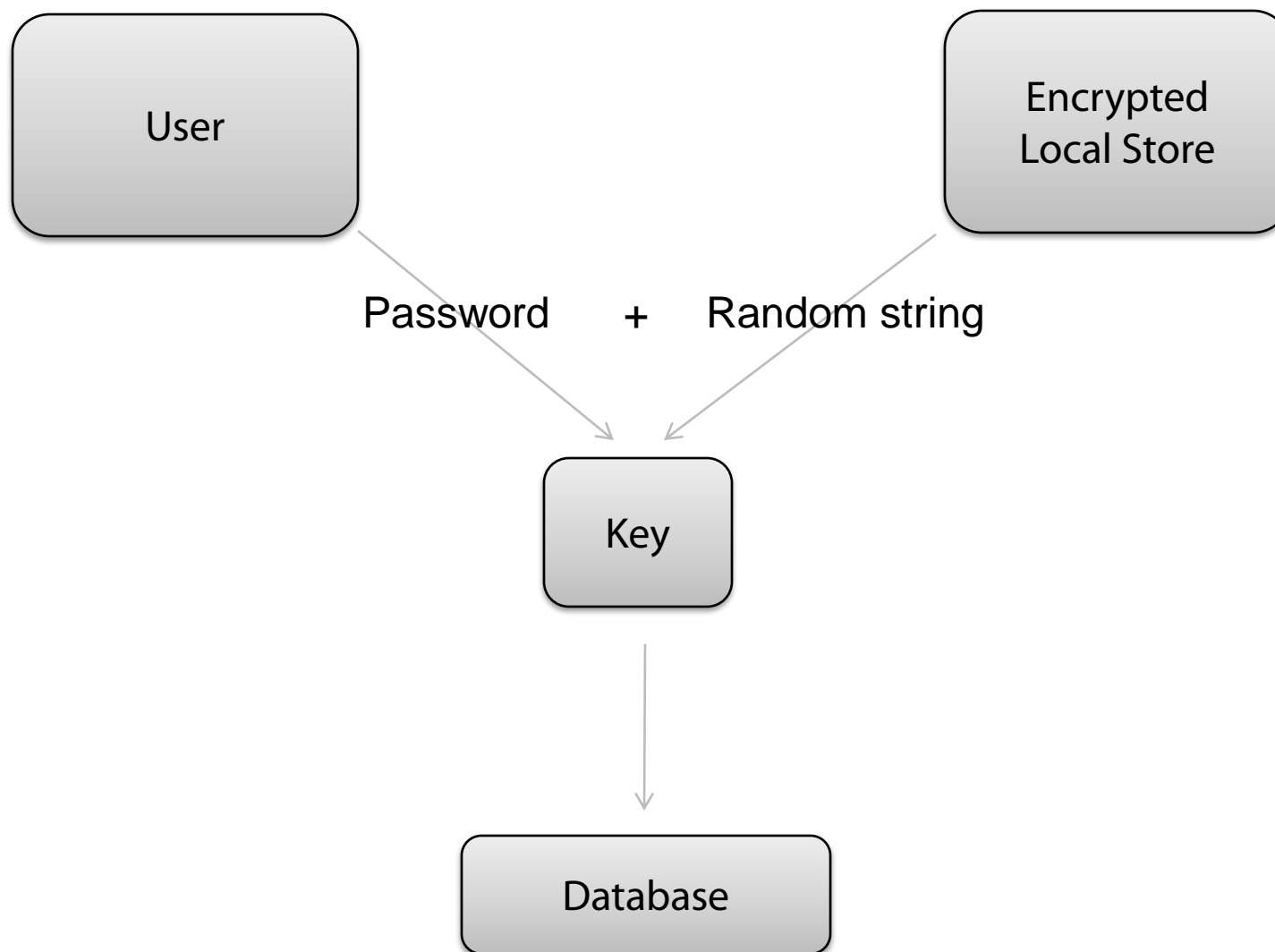
```
sqlConnection = new SqlConnection();
```

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

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.



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

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

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-application 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

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

1. Sandboxed: stays in its own domain;
 1. HTML: <frame>, <iframe>, 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!

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!!!)

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.

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();
```

<http://ddj.com/web-development/210004209>

http://www.adobe.com/devnet/air/ajax/articles/blackbooksafe_anatomy.html

http://www.adobe.com/devnet/air/articles/air_update_framework.html

http://www.adobe.com/devnet/air/articles/introduction_to_air_security.html

http://www.adobe.com/devnet/flashplayer/articles/secure_swf_apps_print.html

<http://onair.adobe.com/blogs/videos/2008/06/23/ethan-malasky-developing-secure-air-applications/>

http://weblogs.macromedia.com/emalasky/archives/2008/04/remote_plugins.html

http://www.adobe.com/devnet/air/flex/quickstart/xml_signatures.html

Thank you!

mihai.corlan@adobe.com

<http://corlan.org>