Technical Note TN2038

Movie Data Security

CONTENTS

This technote describes techniques you can use to protect data in QuickTime movies.

[Dec 13 2001]

Overview

Developers will often find it necessary to safeguard data in movie files. Such safeguards could involve password protecting the data or various other schemes. QuickTime provides a number of techniques for protecting movie data, including password protection, and the ability to prevent users from altering movie data or saving it to another file. Listed below is a discussion of these and other techniques for protecting movie data.

Back to top

The No Save User Data Item

You can add a No Save user data item (type 'nsav') to your movies to prevent them from being edited or re-saved within the QuickTime environment. The No Save user data item was introduced with QuickTime 4. Once this user data item has been added to a movie, any attempts at using Movie Toolbox editing functions (`CutMovieSelection`, etc.) will fail. Furthermore, Movie Toolbox functions that attempt to save the movie (`UpdateMovieResource`, etc.) will fail. Here's a brief overview of this user data item:

Type	Data Size	Data Description
'nsav'	4 bytes	Indicates whether editing or saving of a movie file is possible. A data value of 1 means no saves or edits are allowed. A value of 0 means saves and edits are allowed.

Adding this item to a movie should prevent all but hard-core QuickTime hackers from editing or re-saving movies. However, the No Save user data item is a first level of defense, and a statement of intent to users. Hackers could remove this user data item from a movie and thus allow the movie to be edited or re-saved. If you need more security than this, you'll want to explore some of the other options discussed below.

Listing 1. A short code snippet showing how to add the No Save user data item to a movie:

```
enum
{
     kUserDataNSAV  =     FOUR_CHAR_CODE('nsav')
};

enum
{
    kNoSavesOrEditsAllowed    =    1    /* no saves or edits allowed */
};

OSErr PutFile (ConstStr255Param thePrompt, ConstStr255Param theFileName,
FSSpecPtr theFSSpecPtr, Boolean *theIsSelected,
```

```
Boolean *theIsReplacing)
{
    StandardFileReply    myReply;
    OSErr            myErr = noErr;

    if ((theFSSpecPtr == NULL) || (theIsSelected == NULL) ||
         (theIsReplacing == NULL))
         return(paramErr);

    // assume we are not replacing an existing file
    *theIsReplacing = false;

    StandardPutFile(thePrompt, theFileName, &myReply);
    *theFSSpecPtr = myReply.sfFile;
    *theIsSelected = myReply.sfGood;
    if (myReply.sfGood)
        *theIsReplacing = myReply.sfReplacing;

    return(myErr);
}

OSErr Add_NSAV_Item(    Movie     theMovie,
                    short    theRefNum,
                    short    theResID
                     )
{
    OSErr        err = noErr;
    UserData        theUserData;
    long            nsavData;
    Handle        ourNSAVdataH = NULL;

    /* get the movie's user data list */
    theUserData = GetMovieUserData(theMovie);
    if (theUserData == NULL)
        return (GetMoviesError());

    /* specify the actual data for this user item */
    nsavData = kNoSavesOrEditsAllowed;
    PtrToHand(&nsavData, &ourNSAVdataH, 4);
    if (ourNSAVdataH == NULL)
        return (MemError());

    /* add the No Save item to the user data list */
    AddUserData (theUserData, ourNSAVdataH, kUserDataNSAV);
    err = GetMoviesError();
    if (err == noErr)
    {
        FSSpec        myFile;
        Boolean        myIsSelected = false;
        Boolean        myIsReplacing = false;
        Str255         myPrompt = "\pSave new movie:";
        Str255         myFileName = "\pNewMovie.mov";

        /* prompt the user for a new movie save file */
        PutFile(myPrompt, myFileName, &myFile, &myIsSelected, &myIsReplacing);
        if (myIsSelected)
        {
        /* create a new movie file containing our No Save user
            data item - we'll also compress the movie resource
            to make it more difficult for hackers to remove this
            user data item */
            FlattenMovieData(theMovie,

     flattenAddMovieToDataFork | flattenForceMovieResourceBeforeMovieData
     | flattenCompressMovieResource,
     &myFile, FOUR_CHAR_CODE('TVOD'),
     smSystemScript, createMovieFileDeleteCurFile);
            err = GetMoviesError();
        }
    }

    DisposeHandle(ourNSAVdataH);
```

```
    return err;
}
```

Listing 1. Adding a No Save user data item.

Back to top

Compressed Movie Resources

All QuickTime movies have a movie header describing the type and location of their media data. Media data can, of course, be compressed using a variety of video and sound compression algorithms. Beginning with QuickTime 3, it also became possible to compress the movie header - more commonly known as the movie resource.

Notice the code in Listing 1 makes use of the `flattenCompressMovieResource` flag when calling the `FlattenMovieData` function to save the movie file. Specifying the `flattenCompressMovieResource` flag will compress the movie resource of the new movie (this flag can be used with the `FlattenMovie` function as well). Compressing the movie resource will make it a bit more difficult for hackers to remove this user data item from the movie.

Most application developers won't need to know the details of how movie resources are compressed. For information on how to compress the movie resource without using the FlattenMovie and FlattenMovieData functions, see Letters From The Ice Floe #12 "Compressed Movie Resources"). Using the techniques described in this Ice Floe note it is possible to compress a movie resource in-place, without writing another copy of the file (as long as the compressed movie resource is really smaller than the uncompressed one).

Note it is also possible to encrypt the movie resource for your movie with a custom encryption scheme. For example a custom data compressor component could be used to provide the custom encryption. Again, refer to Ice Floe Note #12 for the details. Your custom component could also make use of the QuickTime access keys as described below.

After applying a custom encryption scheme to your movie atom it is also possible to use the QuickTime `NewMovieFromUserProc` function and a custom decrypting data procedure to open and view the movie.

Back to top

QuickTime Access Keys

QuickTime access keys (also referred to as "media keys") make it possible to protect data. QuickTime access keys were introduced with QuickTime 3. The QuickTime access keys allow an application that supplies data to register a password for the data with QuickTime and allows a user to enter the password to gain access to the data. For example, a codec can protect data it compresses with a password, so that it is available only to someone with the password. Similarly, the creator of a movie can require a password to view the movie. In order to gain access to protected data, the user enters the access key in the QuickTime Settings control panel.

There are actually two kinds of access keys:

- system access keys
- application access keys

Data protected by a system access key can be unlocked by any QuickTime caller on the computer. Data protected by an application access key can be unlocked only by QuickTime clients for that application.

System access keys are useful for data that needs to be used by more than one application. When a system access key is registered for data, applications do not have to perform any additional registration to unlock the data. In contrast, application access keys are normally registered by the application in which the data is available, and each application registers the application access keys it uses.

Certain compression formats, such as the Intel Indeo and Sorenson Vision 3 formats for video compression, allow data to be encrypted when it is compressed. For software to gain access to the encrypted data, the access key for the data must be registered with QuickTime. For example, a CD-ROM title could register one or more access keys for encrypted video data it presents. Once the access keys are registered, the keys are available to the decompressor component that will be used to decompress the data. The CD-ROM title can then use the encrypted data in the same way it uses unencrypted data.

Sorenson Vision 2 video compression also makes use of the QuickTime access keys to protect data, but it does not actually encrypt the data.

Back to top

Custom Data Structures

If you don't want "casual" users to be able to open movies stored in your application you could hide them in a custom data structure. That way, they would not be easily identifiable (by simple browsing) and thus not easily opened by a user. However, keep in mind if you alter the movie atom for a movie after the movie has been saved this might prevent the movie from correctly locating the movie sample data. Check the QuickTime File Format pdf document) for more information regarding the various movie atoms.

Back to top

Custom Codec or Data Handler

Finally, one could write a custom codec or data handler. Access to proprietary data in a movie file encrypted with a custom codec would of course require the codec for decryption. Check the QuickTime API documentation for more information about writing image compressor components.

Similarly, a custom decrypting data handler could be used to perform decryption of movie data, though the manner in which data handlers receive data may preclude some encryption techniques (cipher block chaining might be a challenge, for example). Check the QuickTime data handler documentation) for additional information.

Also, keep in mind writing a custom data handler or a custom codec capable of keeping up with cutting-edge media is a major undertaking.

Back to top

Summary

There are a number of techniques you can use to protect movie data. However, a critical issue one needs to resolve when formulating a protection scheme is this - what level of security is necessary to protect the data? Any protection can be broken by someone determined enough - but each of the above techniques would present a certain degree of challenge.

Also, one final consideration worth noting is how much the chosen technique will inconvenience and put-off legitimate viewers of your content. Users may become frustrated with elaborate security techniques, and could end up not using your product as a result.

Back to top

References

Movie Toolbox Access Keys

QuickTime File Format

Letters From The Ice Floe #12 "Compressed Movie Resources"

Image Compressor Components

QuickTime Data Handler Components

Back to top

Downloadables

Acrobat version of this Note (K) Download

Back to top
