APPLICATION SECURITY IN AN INTERACTIVE MEDIA ENVIRONMENT

STATEMENT OF RELATED APPLICATION

[0001] This application claims the benefit of provisional application number 60/695,944, filed July 1, 2005, which is incorporated by reference herein.

BACKGROUND

[0002] Some multimedia playback systems provide limited interactive graphics during audio/video playback. The greater capabilities of interactive playback systems present greater opportunities for malfeasance. It is critical to maintain the security of the playback system against viruses, spyware and other malicious software. Malicious software could cause the interactive playback system to malfunction or gather and transmit private user information. In addition, an interactive playback system may be connected to a network. The software or user information could propagate from the playback system to other computing systems attached to the network. Consequently, it is critical that the interactive playback system include adequate security provisions.

SUMMARY

[0003] A security system is provided which controls the privileges of unsigned applications in the field of interactive multimedia. Interactive multimedia is an environment in which applications typically manage multimedia objects including graphics, audio and video responsively to user input events on a synchronized real-time, frame-accurate basis. Applications here are termed “iHD” applications as they relate to high-definition DVD (digital versatile disk) media. However, the disclosed security system is applicable to other interactive multimedia environments more generally.

[0004] The system in particular applies to application security, not content security, and establishes a framework for application security, including a signature system, and further provides file formats that support security. Interactive multimedia applications run on an interactive playback system (that is implemented as a standalone hardware device, or alternatively as a software application running, for example, on a personal computer) may be either signed or unsigned.

[0005] Signed applications are allowed practically unlimited applications. Unsigned applications are greatly restricted in what the same can access. Moreover, if both signed and unsigned applications are running, both are given only the security level and access privileges of the unsigned application. Providing for unsigned applications allows for home-authored discs customizable with rich interactivity features, but restricts access to

networks, e.g., the Internet, and sensitive information stored within the playback system, to authorized parties.

[0006] Signed applications may be provided with special file formats, allowing determination of the signature status without requiring parsing of the entire file.

BRIEF DESCRIPTION OF THE DRAWINGS

[0007] Fig. 1 is a flowchart illustrating a method of assigning privileges to applications where signature statuses of applications are detected from a disk.

[0008] Fig. 2 is a flowchart illustrating a method of assigning privileges to applications where the signature status of an applications is detected upon loading into a playback system.

[0009] Fig. 3 is a flowchart illustrating creation of an author identifier – keyed directory.

[0010] Fig. 4 is a schematic depiction of an application file.

DETAILED DESCRIPTION

[00010] Interactive multimedia applications are those in which the application is responsive to user events. An example is a menu implemented within an application that is accessed by the user, in which the user submits an input that causes the application to change state. In such a case, the interactivity is with the menu graphics which are rendered while video plays beneath them, e.g., on the z=0 layer, on a real time, frame-synchronous basis. The interactivity may lead, for example, to changes in how the video stream is displayed.

[00011] For example, an underlying video may be a high-definition movie. The graphic overlay may be part of a commentary by the director of the movie, showing, e.g., a schematic of various camera locations overlayed on top of the scene itself. The user may, employing the remote control, switch to a view envisioned by any of those camera locations.

[00012] As noted above, the greater capabilities of interactive playback systems present greater opportunities for malfeasance. Malicious software could cause the playback system to malfunction or gather and transmit private user information.

[00013] In the current system, interactive applications for use in the playback system may be either signed or unsigned. Signed applications are those which inherit a root certificate from a trusted root authority (e.g., a movie studio) and are considered safe.

[00014] Signed applications are given high-level access privileges. This almost-unrestricted privilege allows access to, e.g., networking, file I/O, security and diagnostic

APIs, and may access persistent storage to store and retrieve data that is to persist across invocations of the application.

[00015] Unsigned applications, on the other hand, are given low-level access privileges. They are denied access to the type of functionality afforded by high access. They may be limited to the utilization of the markup language, as well as, e.g., certain objects from the following exemplary APIs in ECMAScript: XML (without the I/O functionality); globalization; drawing functions associated with graphics elements; and user input operations.

[00016] This level of functionality prohibits access to any networking, security, or file I/O. Any attempt to call a function outside the above namespaces or load resources from persistent local storage may result in an exception which will terminate the application.

[00017] In one embodiment, a set of applications is present on a media disk, e.g., a HD-DVD, and the same are employed to run an interactive graphics and video application. Referring to Fig. 1, the media disk is received by the playback system (step 12). The playback system, which may be a general purpose computer system or a more specialized media center system, determines the signature status of the applications on the media (step 14). If the signature status of all applications is determined (step 16) to be signed, then all of the applications are given the high access privileges (step 18). If the signature status of any one application is determined to be unsigned, then all of the applications are given low access privileges (step 22). That is, if an unsigned application is running, all concurrently running applications, whether signed or unsigned, may be restricted to the unsigned application permission level. This prevents an unsigned application from leveraging the privileges of a concurrent signed application.

[00018] In another embodiment, a similar method may apply directly to applications loaded into the playback system. Referring to Fig. 2, an application may be loaded into the playback system (step 24). The signature status of the application is then detected (step 26). If the signature status is determined (step 28) to be signed, then the application may be run at a high privilege access level (step 32). However, if the signature status is determined (step 28) to be unsigned, then the application is run at a low privilege access level (step 34). In this case, the application is run directly from the media (step 36), e.g. a disk. This provides enhanced security, as all unsigned applications are then prevented from running or loading resources from local persistent storage of the playback system. If additional applications are loaded (step 38), then they may be tested or not for their signature status: in general, they will be afforded a low access level (step 34). If an

application is signed and thus given high access privileges, and then a later application is loaded and is unsigned, then the high access application is lowered to the low-access level.

[00019] Referring to Fig. 3, for signed applications, the playback system may include employment of a set of author identifiers that are detected (step 44) upon the media introduction (step 42) into the playback system. That is, each media or application may be associated with an author identifier, which uniquely identifies content authors, and which is particularly important to the security of applications in persistent storage, i.e., those which can access persistent storage to store and retrieve data that are desired to be persistent across invocations.

[00020] The author identifier is then associated with creation (step 46) of a directory associated with that author identifier. The application from that media may access only the directory corresponding to its author identifier in persistent storage. The file system, as the application views it, is rooted in that directory. While the application can manage subdirectories, it cannot go above its root directory and see other author's data.

[00021] The author identifier may be associated with either a disk, including all the applications on that disk, or a single application, either on that disk, spread over several disks, or otherwise loaded into the playback system, e.g., via an internet download. Furthermore, a given media may be associated with a single author identifier, but a given author identifier may be found on multiple media. Another embodiment may be to use the identifier referred to by the key that signed the application. Assuming that different applications may be signed by different persons on a single media, this embodiment would lead to even greater segregation of storage. Using the chain of certificates instead of the last signature would do so even more.

APPLICATION STRUCTURE

[00022] The structure of a signed application is now described. Referring to Fig. 4, a signed application 50 may include a manifest file 52 and at least one resource file 54. The manifest file 52 is signed with the author's signature and certificate and authenticates all the resources it references.

[00023] The application may have its manifest file 52 and all resource files 54-58 bundled into an uncompressed archive 48. The file format for archive 48 need not even support encryption. The archive file 48 is in essence a container and generally does not need to be signed independently. The manifest file 52 may reference each of the resource files 54-58 of the interactive application. The archive 48 architecture may be specified such that the archive 48 may be streamed efficiently, e.g., the signed manifest file 52 may

be the first file within the archive 48, allowing the verification of the signature without reading the entire archive. Subsequent versions of the archive format may be backwards compatible with previous ones.

[00024] Authentication of the data in the archive 48 may be provided by the use of, e.g., XML-Signature as defined by RFC 3275.

MANIFEST FILE FORMAT

[00025] In one example, the format of the signed manifest file 52 may make use of a subset of the W3C Recommendation for XML-Signature Syntax and Processing defined by RFC 3275. In this way, the following subset of elements may be included and supported:

ds:Signature

ds:SignatureValue

ds:SignatureType

ds:Reference

ds:Reference/ds:DigestValue

[00026] Other elements may be determined by the system. Digest values for each resource item included in the manifest may be listed as ds:Reference elements.

CERTIFICATES AND SIGNATURES

[00027] As an example, the required certificate type may be, e.g., X.509. The signature method as defined by ds:SignatureMethod may be RSA-SHA1. The canonicalization method may be specified to be Exclusive XML Canonicalization 1.0. The digest method may be the same as the signature method, RSA-SHA1. The key information may be inferred by the system from the identity of the media or local storage area from which the application is being run.

CERTIFICATE REVOCATION LISTS

[00028] To provide a mechanism for revocation and replacement of compromised applications, each interactive video and graphics application author can include a Content Revocation List ("CRL") which lists the bundle file digest values of revoked applications. This CRL may be included in a separate file. This file contains a list of bundle signature digests which have been revoked, and the signature of the content creator who authored the disc. Assuming that the original author's signature of each revoked application matches that of the signature in the CRL file, the application digests listed will be stored in the content provider's restricted area of local storage and will be no longer allowed to run.

If a CRL is included in an application, it may be given a recognizable name such as Revocation.xml.

[00029] The application author may desire to replace the revoked application with a new version. This can be accomplished in several different ways. Titles running on Internet-connected players may be caused to check their home servers for updated playlists or interactive video and graphics bundles which specify newly downloaded applications. Alternatively, the media revoking an application could also supply the replacement itself.

[00030] The following table describes one possible format of the archive file, along with comments describing the fields. It should be noted that numerous other formats may be used. In this table, abbreviations are used to represent types: Ui*n* represents a unsigned integer of *n* bits. For example, Ui8 is an 8-bit unsigned integer, and Ui32 is a 32-bit one. An array of type is indicated by using square brackets, and the length of the array is indicated in between those brackets. If the length depends on a previous field, that field's name, or a shorter name indicated in the field, can be used to refer to the value of that field. Hexadecimal values are indicated using the 0xdd notation. All variable-length strings, and hence resource names, may be encoded using UTF-8 using a Pascal string notation (8-bit length followed by bytes).

	Field		Type	Comment
Archive Header	Magic		Ui8[5]	The values of the 5 bytes must be : 0x69, 0x48, 0x44, 0x61, 0x72.
	Version		Ui8	Version of the format The value must be 0x01.
Resource Catalog	Resource Entry #1	Resource Entry Length	Ui16	Length of this resource entry.
		Resource Offset	Ui32	Byte offset of the resource in the resource data block.
		Resource Length	Ui32	Length of the resource in bytes.
		Resource Checksum	Ui32	CRC-32 checksum of the resource bytes.
		Resource Type		
		Resource Name Length	Ui8	RNL
		Resource Name	Ui8[RNL]	The filesystem name of the resource.

		Field	Type	Comment
		Resource Entry Length	Ui16	Length of this resource entry.
	Resource Entry #n	Resource Offset	Ui32	Byte offset of the resource in the resource data block.
		Resource Length	Ui32	Length of the resource in bytes.
		Resource Checksum	Ui32	CRC-32 checksum of the resource bytes.
		Resource Type		
		Resource Name Length	Ui8	RNL
		Resource Name	Ui8[RNL]	The filesystem name of the resource.
		All the resource data, in continuous blocks.		
Resource Data Block				

[00031] Certain rules may apply to the application resources noted above. For example, resource names must be file system names or logical URIs. The directory in which an archive file will be extracted may be considered to be the root of the file system during that extraction. In that way, all names will be made relative to that directory, so that absolute paths will behave the same as relative ones. If a name results in a location outside of that directory, the name and the entry may be considered invalid.

[0033] The following sections give more detailed information about various fields and sections of the above exemplary archive file.

Archive Header

Magic Field

[0034] This is a “magic number” used to uniquely identify archives. It may consist of the string “iHDar”, i.e., iHD archive, coded as a sequence of 5-character values in UTF-8, ASCII, etc., i.e. 0x69, 0x48, 0x44, 0x61, 0x72.

Version Field

[0035] The version field allows an archive reader to read different versions of the archive format. By looking at the version field, one can know what to expect in the different sections of the file, and thus read information that was not present in some

versions of the file format. The value of this field may be, e.g., 0x01. Future versions may have values from 0x02 to 0xff.

Resource Catalog and Resource Entries

[0036] The resource catalog includes a number of resource entries. Each entry follows the same format.

Resource Entry Length

[0037] This is the length, in bytes, of the resource entry itself. This value is used by readers who are reading a format whose version they do not understand. Assuming that an archive written using version 2 of the format is seen by a reader made for version 1, the reader can read the fields it knows about and then skip to the next resource entry since it knows the length of the current entry.

Resource Offset

[0038] This indicates the byte offset of the resource in the resource data block. The offset of the first resource is 0x0000.

Resource Length

[0039] This is the length of the resource, in bytes.

[0040] If resources A and B are contiguous in the archive file, then the resource offset for B is equal to the sum of A's resource offset and resource length.

Resource Checksum

[0041] This represents a CRC-32 checksum of the resource, as defined by ISO 3309. Note that this checksum should only be used for simple verification of the integrity of a resource transported over an unreliable medium. Because the CRC-32 checksum is neither keyed nor collision-proof, it should not be used for authentication purposes. If one needs to authenticate resources, the signature mechanism described above may be employed.

Resource Type

[0042] This is the MIME type of the resource.

Resource Name Length

[0043] This is the length of the resource name in bytes. The resource name immediately follows this field.

Resource Name

[0044] This is the resource name itself.

Resource Data Block

[0045] The resource data block contains all the bytes for the resources, in the order they appear in the resource catalog. There is generally no explicit separation between two resources, as their offsets and lengths are well-known quantities.

[0046] The system may be described in the general context of computer-executable instructions, such as program modules, being executed by a computer. Generally, program modules include routines, programs, objects, components, data structures, etc. that perform particular tasks or implement particular abstract data types. The system and method may also be practiced in distributed computing environments where tasks are performed by remote processing devices that are linked through a communications network. In a distributed computing environment, program modules may be located in both local and remote computer storage media including memory storage devices.

[0047] The instructions which execute the method and system may be stored on a variety of computer readable media. Computer readable media can be any available media that can be accessed by a computer and includes both volatile and nonvolatile media, removable and non-removable media. By way of example, and not limitation, computer readable media may comprise computer storage media and communication media. Computer storage media includes both volatile and nonvolatile, removable and non-removable media implemented in any method or technology for storage of information such as computer readable instructions, data structures, program modules or other data. Computer storage media includes, but is not limited to, RAM, ROM, EEPROM, flash memory or other memory technology, CD-ROM, digital versatile disks (DVD) or other optical disk storage, magnetic cassettes, magnetic tape, magnetic disk storage or other magnetic storage devices, or any other medium which can be used to store the desired information and which can accessed by a computer. Communication media typically embodies computer readable instructions, data structures, program modules or other data in a modulated data signal such as a carrier wave or other transport mechanism and includes any information delivery media. The term “modulated data signal” means a signal that has one or more of its characteristics set or changed in such a manner as to encode information in the signal. By way of example, and not limitation, communication media includes wired media such as a wired network or direct-wired connection, and wireless media such as acoustic, RF, infrared and other wireless media. Combinations of the any of the above should also be included within the scope of computer readable media.

[0048] Although described in connection with an exemplary computing system environment, including a computer, the system is operational with numerous other general purpose or special purpose computing system environments or configurations. The computing system environment is not intended to suggest any limitation as to the scope of use or functionality. Moreover, the computing system environment should not be interpreted as having any dependency or requirement relating to any one or combination of components illustrated in the exemplary operating environment. Examples of well known computing systems, environments, and/or configurations that may be suitable that may be used include, but are not limited to, personal computers, server computers, hand-held or laptop devices, multiprocessor systems, microprocessor-based systems, set top boxes, programmable consumer electronics, mobile telephones, network PCs, minicomputers, mainframe computers, distributed computing environments that include any of the above systems or devices, and the like.

[0049] The systems and methods described herein may be implemented in software or hardware or both using techniques some of which are well known in the art.

[0050] The order of execution or performance of the methods illustrated and described herein is not essential, unless otherwise specified. That is, elements of the methods may be performed in any order, unless otherwise specified, and that the methods may include more or less elements than those disclosed herein.

[0051] When introducing elements of the present invention or the embodiment(s) thereof, the articles "a," "an," "the," and "said" are intended to mean that there are one or more of the elements. The terms "comprising," "including," and "having" are intended to be inclusive and mean that there may be additional elements other than the listed elements.

[0052] As various changes could be made in the above constructions, products, and methods without departing from the scope of the invention, it is intended that all matter contained in the above description shall be interpreted as illustrative and not in a limiting sense.

[0053] Although the subject matter has been described in language specific to structural features and/or methodological acts, it is to be understood that the subject matter defined in the appended claims is not necessarily limited to the specific features or acts described above. Rather, the specific features and acts described above are disclosed as example forms of implementing the claims.

CLAIMS

1. A method for ensuring security of an application in an interactive multimedia environment, comprising:
   a. receiving an application implementing interactive video and frame-synchronous graphics;
   b. detecting the signature status of the application;
   c. if the signature status is signed, then giving permission access to a source of local storage and a network resource;
   d. if the signature status is unsigned, then denying permission access to a source of local storage and a network resource.
2. The method of claim 1, further comprising if the signature status is signed, then giving permission access to file I/O and security and diagnostic APIs.
3. The method of claim 1, wherein the detecting comprises reading a manifest file associated with the application, and determining if the manifest is signed with an author's signature and certificate.
4. The method of claim 3, wherein the application contains an archive which houses the manifest file and at least one resource file, and wherein the manifest file is the first file in the archive.
5. The method of claim 4, wherein the archive has a format such that the archive may be streamed efficiently.
6. The method of claim 1, further comprising if the signature status is signed, then the application includes a signed root certificate, a content revocation list, or an author identifier.
7. The method of claim 1, wherein the source of local storage is a directory keyed to the author identifier.
8. The method of claim 1, further comprising if the signature status is unsigned, then denying permission access to file I/O and security and diagnostic APIs.
9. The method of claim 1, further comprising if the signature status is unsigned, then giving permission access to utilization of a markup language and utilization of the objects consisting of: XML without I/O functionality, globalization, drawing functions, and user input operations.
10. The method of claim 1, further comprising if the signature status is unsigned, and the application is received from an optical disk, then running the application only from the optical disk.

11. The method of claim 1, further comprising:

a. receiving another application;

b. detecting the signature status of the another application;

c. if the signature status of the application or the another application is unsigned, then denying permission access for both applications to both a source of local storage and a network resource; or

d. if the signature status of the application and the another application are both signed, then granting permission access for both applications to both a source of local storage and a network resource.

12. A method for ensuring security of a media disk, comprising:

a. receiving a media disk in a playback system, the media disk containing one or more applications whereby the applications implement interactive video-frame-synchronous graphics in an interactive multimedia environment;

b. detecting the signature status of each application on the media disk;

c. if the signature status of all the applications is signed, then giving high permission access to each application to a source of local storage and a network resource;

d. if the signature status of any of the applications is unsigned, then denying to any of the applications high permission access to a source of local storage and a network resource.

13. A multimedia playback system for applications whereby the applications implement interactive video-frame-synchronous graphics in an interactive multimedia environment, comprising:

a network resource;

a source of local storage;

a device to receive an application incorporating interactive graphics and video;

a processor to detect the signature status of the application;

wherein if the application is signed, the application is given permission access to the source of local storage and the network resource; and

wherein if the application is unsigned, the application is denied permission access to the source of local storage and the network resource.

14. The system of claim 13, wherein the processor detects the signature status of the application by reading a manifest file associated with the application, and determining if the manifest is signed with the author's signature and certificate.

15. The system of claim 14, wherein the application contains an archive which houses the manifest file and at least one resource file, and wherein the manifest file is the first file in the archive.

16. The system of claim 13, wherein if the signature status is signed, then the application includes a signed root certificate, a content revocation list, or an author identifier.

17. The system of claim 16, wherein the source of local storage is a directory keyed to the author identifier.

18. The system of claim 13, wherein if the signature status is signed, then the application is given permission access to file I/O and security and diagnostic APIs.

19. The system of claim 13, wherein if the signature status is unsigned, then the application is:

a. denied permission access to file I/O and security and diagnostic APIs; or

b. given permission access to utilization of a markup language and utilization of the objects consisting of: XML without I/O functionality, globalization, drawing functions, and user input operations

20. The system of claim 13, wherein the processor is configured to detect the signature status of another application, and wherein

a. if the signature status of the application or the another application is unsigned, then permission access is denied for both applications to both a source of local storage and a network resource; or

b. if the signature status of the application and the another application are both signed, then permission access is granted for both applications to both a source of local storage and a network resource.

21. The system of claim 13, further comprising if the signature status of the application is unsigned, and the application is received from an optical disk, then the application is only run from the optical disk.

22. A method of authoring secure applications , comprising:

developing an application implementing interactive video-frame-synchronous graphics in an interactive multimedia environment;

converting the application into an archive file format having a manifest file and at least one resource file;

disposing within the manifest file a certificate containing a signature; and

disposing the manifest file at the beginning of the archive file.

23. The method of claim 22, further comprising burning or saving the application onto a disk.