RC 21944 (98481) 20 November 2000 Computer Science

IBM Research Report

WebGuard: A System for Web Content Protection

Magda Mourad[+], Jonathan Munson[+], Tamer Nadeem[++], Giovanni Pacifici[+], Marco Pistoia[+], Alaa Youssef[+]

IBM Research Division
Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

[++]University of Maryland at College Park

Research Division
Almaden - Austin - Beijing - Haifa - T. J. Watson - Tokyo - Zurich

LIMITED DISTRIBUTION NOTICE: This report has been submitted for publication outside of IBM and will probably be copyrighted if accepted for publication. It has been issued as a Research Report for early dissemination of its contents. In view of the transfer of copyright to the outside publisher, its distribution outside of IBM prior to publication should be limited to peer communications and specific requests. After outside publication, requests should be filled only by reprints or legally obtained copies of the article (e.g. payment of royalties). Copies may be requested from IBM T. J. Watson Research Center, P. O. Box 218, Yorktown Heights, NY 10598 USA (email: reports@us.ibm.com). Some reports are available on the internet at http://domino.watson.ibm.com/library/CyberDig.nsf/home

WebGuard: A System for Web Content Protection

Magda Mourad[†], Jonathan Munson[†], Tamer Nadeem[‡], Giovanni Pacific?, Marco Pistoia’, Alaa Youssef[†]

[†]IBM T.J. Watson Research Center

[‡]University of Maryland at College Park

{magdam, jpmunson, giovanni, pistoia, ayoussef}@us.ibm.com

Abstract

In this paper we present *WebGuard*, a content protection system for Web documents. WebGuard allows content owners to enforce control over the distribution and access to high-value digital content. We define high-value digital content as either commercially valuable content (e.g. course material, artwork, etc) or personal and confidential property (e.g. family photos, confidential material, etc). The novelty of our approach is that WebGuard enables existing Web browsers and browser plug-ins to handle protected content, in a way that is completely transparent to the browser and its plug-ins. Our solution centers around three components: an application certification process and a content-protection enabled http protocol handler, and an application-independent user-interface control module. We show that using these three components a complete end-to-end content protection system can be achieved. To demonstrate our approach we have built a prototype of WebGuard using the Internet Explorer Browser and the Microsoft protocol handler technology.

1. Introduction

With the advent of digital distribution and the many mechanisms available for it on the Internet, it is now possible for a single person to make a perfect copy of a digital content and distribute it to millions of others. While in most cases content owners welcome this widespread distribution, in some cases the content owners may wish to enforce some control over the distribution and access to high-value digital content. High-value digital content can be commercially valuable content (e.g. course material, artwork, etc) or personal and confidential property (e.g. family photos, confidential material, etc).

Digital Rights Management (DRM) technology allows content owners—such as publishers, artists and instructors— to distribute high-value digital content with the confidence that the terms and conditions they set for the use of their content will be respected. For example, the content owners may wish to specify terms and conditions that prevents the digital content from being copied or disables the content after it has been accessed a certain number of times.

These benefits of DRM technology have been available for some time in the specific industries of music and e-books, however, for general multimedia Web content—HTML, GIF and JPEG images, animations, audio, etc.—DRM technology has been less successful.

The reason for this, to a large measure, lies in the current state of the art of the DRM technology and the specific requirements it imposes on end-user applications. Typically a Digital Rights Management (DRM) system works by encrypting the content and providing a program capable of playing (or displaying) the content to the user. This player (often referred to as a trusted player) ensures that the user does not make unauthorized use of the digital content. To play, or view, the content, users must use this trusted player; thus the content producers must ensure it is available to them. While for producers of digital music this situation may not present a problem since there are several DRM-enabled music players available, it is particularly problematic for producers of Web content. They enjoy a great range of third-party players to choose from to develop their content for. To restrict them to develop protected content only for DRM-enabled players is to restrict their choices to practically nil. In addition, browsers tend to be highly personalized, and users would prefer not to use a different browser to access protected content, especially if the user has to use a different web browser to access content protected by different DRM systems. Furthermore, DRM developers should not be burdened by the need of providing and maintaining a fully featured web browser, making the DRM solution cost effective only for large scale distributions and commercially viable content.

In this paper we present *WebGuard*, a content protection system that allows users to access protected content using existing web browsers. Our solution centers around three major components: an application certification process, a DRM-enabled http protocol handler, and an application-independent user-interface control module. We show that using these two components a complete end-to-end content protection system can be achieved. We have built a prototype of WebGuard using the Internet Explorer Browser and the Microsoft protocol handler technology.

This paper is organized as follows. In Section 2, we first review DRM technology and discuss the requirements for end-user applications. In Section 3, we describe the novel approach we are proposing for designing player independent DRM systems. We then present WebGuard, describing its components and mechanisms. We conclude with a discussion and future plans for the WebGuard system.

2. Current State-of-the-art in Digital Rights Management

In the recent years we have witnessed the emergence of several content protection systems. The current most popular systems include InterTrust's *MetaTrust* system [5, 9], Microsoft's Windows Media Rights Manager [8], the *ContentGuard* system [3], IBM's *Cryptolope*™ system for digital libraries [6], Adobe's *WebBuy* component in their Acrobat Reader™ product [10], and IBM's Electronic Media Management System (EMMS) [4].

Figure 1 shows the prototypical components of a generic DRM system. There are five basic subsystems that are generally used by content protection systems: *content packaging*, *e-store*, *content hosting*, *clearinghouse* and *user player* (or *viewer*).

The *content packaging* component is used by content owners to encrypt and package the content. Together with the encrypted content, the content packaging subsystem produces a content description object. The content description object is used to uniquely identify the digital content, to specify the set of usage rights allowed by the content owner as well as provide marketing information (such as price, promotional material etc).

The *e-Store* is responsible for promoting the content and for granting usage rights to individual users (customers). Usually the e-store supports e-commerce transactions. Rights are granted to users in the form of unique digital certificates that uniquely identify the content as well as the set of usage rights being granted.

The *Content hosting* component is responsible for hosting the encrypted content packages and releasing them only to authorized users, who have acquired the rights to download the content. Logically separating this component from the rest of the system allows for flexibility and independence of the distribution channel used.

The *Clearinghouse* is responsible for delivery the keys for decrypting the content to authorized users. It is also the locus of authorization and usage tracking. In general, the Clearinghouse is the only component that is trusted by all parties (content owners, distributors, and consumers).

The *Player/Viewer.* runs on the user side and is the application with which the consumer accesses the content. It may be a music player, a document viewer, or, in our case, a Web browser. In addition to its primary content-handling functions, it also encapsulates, or invokes, functions of a DRM client. The DRM client is responsible for maintaining the content in encrypted form using keys hidden from the user at all times; decrypting the content on demand; interacting with the

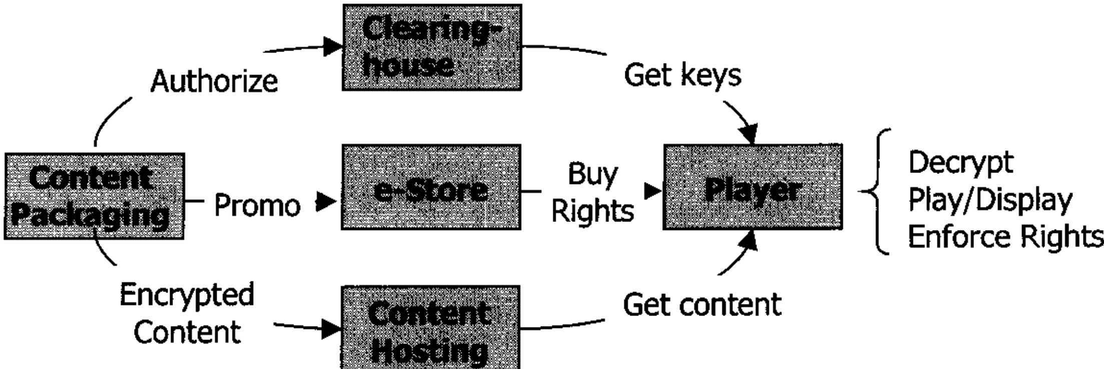

Figure 1: Components of a typical DRM system.

Clearinghouse to obtain keys, report usage, and other purposes; and for enforcing the usage conditions associated with the content.

The development of player/viewer is the most challenging from a technical point of view. The need to hide keys from users on their own systems, which can be inspected exhaustively; the need to protect the execution of the decryption code against attacks; and the need to protect the decrypted content from being siphoned off by malicious code: all of these require clever and careful programming. The challenge is compounded when the goal is, as it was in our case, to provide all this protection equally to off-the-shelf Web browsers and browser plug-ins. To better explain the particular challenge we faced, we first review the approaches taken by current DRM systems.

2.1. Current Approaches to End-user DRM Software

Current DRM systems take one of two approaches to the end-user side of DRM software. In the first approach, the DRM functions are integrated with the content-rendering functions of the player/viewer. Systems that take this, the "integrated player" approach, include IBM's *Cryptolope*™ system for digital libraries [6], Adobe's *WebBuy* component in their Acrobat Reader™ product [10], and IBM's Electronic Media Management System (EMMS) [4]. In these systems the DRM functions of key handling, decryption, and rights management are packaged in the same executable with the content rendering code. Any tamper-resistance measures applied are effective for the entire executable, and thus form a single protected execution environment. The strength of such systems is that they offer the strongest protection for the content, since it never leaves the control of the protected environment. In addition, the integration with the DRM system allows DRM operations, such as acquisition of additional rights, to be invoked within the user interface of the application and thus provide a smoother experience for the consumer. The weakness of the integrated player approach is that their DRM functions are not easily extendible to new forms of content. This makes the approach appropriate for content producers whose content fits into the fixed range of content that these applications handle, such as HTML (pure), digital music, and PDF documents, or who are large enough, such as music studios or book publishers, to have a player specially developed for their content.

The second approach is to offer DRM functions in a toolkit, then bind content-rendering applications to the toolkit using a secure mechanism, to ensure that only legitimate applications can access the content. Systems that use the toolkit approach include InterTrust's *MetaTrust* system [5, 9], Microsoft's Windows Media Rights Manager [8], and the *ContentGuard* system

[3]. (Details of the secure mechanisms used to authenticate applications to the toolkits are proprietary to each vendor.) The strength of this approach is that it offers extendibility to various forms of content since any application can utilize the toolkits. And, as with the integrated player approach, it still allows for integration of DRM operations in the user interface with the operations of the player. The weakness of this approach is that if content producers want to choose their DRM solution, they must also be the developers of the player application that renders their content, or have a strong relationship with its developers. Thus, as with the integrated player approach but to a lesser extent, it is appropriate for content owners who have the means to have players specifically developed for their content.

The shared drawback of both approaches is that they require the player application to be built against a specific DRM solution, either integrated with it or using it in the form of a toolkit. This presents a significant problem for the content producers who neither own the applications that render their content, nor whose content can be rendered by an existing DRM-enabled player. Most producers of Web-based content fall into this situation. To meet their requirements, we developed WebGuard, which represents a new approach in DRM-enabled software, that we call "transparent DRM." The specific class of applications that WebGuard supports are those that execute as plug-ins within the Microsoft Internet Explorer browser.

3. New Approach: Transparent DRM Extensions

By "transparent DRM," we mean that DRM functions are provided to an application without requiring it to be specially "DRM-enabled." Our approach has three main elements: a trusted content handler, certificate-based code verification, and UI control through event blocking.

Certificate-based code verification. In the integrated player approach, the developers of the DRM software have also developed the content-rendering software, and so this code is implicitly trusted. In the toolkit approach, the development process offers opportunities to verify trustworthiness, and the player software may be specially configured to implement the toolkits secure binding mechanisms. Our approach uses a three-step process of establishing the trust of an application at call-time: (1) the application goes through an off-line certification process that generates a trust certificate containing the digital signature of the application, (2) this signature is checked against the signature of the executable image of the application at launch time, and (3) this validation is remembered and checked by the Trusted Content Handler (below) upon each request the application makes for content. We describe specifics of this process in Section 4.

Trusted content handler. Since DRM functions must be transparent to the application in this approach, all rights-management functions and content decryption must be invisible to the application in its requests for content. To achieve this, we use Internet Explorer's protocol handler extension mechanism. URLs for WebGuard-protected content use the HTTP method names `rmfile` (for local content) and `rmhttp` (for remote content). When the browser receives URLs with these method names, the WebGuard Trusted Content Handler is invoked. The TCH invokes the necessary DRM functions, decrypts the content, and passes it back to the browser. We describe the structure of the TCH and the rights specification mechanism in more detail in Section 5.

User interface control through event blocking. An application handling protected content must prevent the user from invoking unauthorized operations on the content. To provide this function for an application in a transparent manner proved to be one of the more challenging tasks in the project. In the end, we developed a mechanism that will allow the user-interface operations of any Windows application to be controlled. We rely on the event-handler structure of Windows programs and the ability of other program modules to register themselves as listeners of the events and to receive them before the application does. In this way, if the received event represents an operation that should be blocked, according to the rights in effect for the subject content, the application can be prevented from receiving the event. We describe this system in Section 6.

In the following sections, we describe in some detail each of the above three principal elements of our approach. We then present the rights-specification mechanisms we developed for WebGuard, and then describe a lightweight end-to-end content distribution and clearinghouse system that we have developed as part of WebGuard.

4. Certificate-based Code Verification

Recall that in order for the Trusted Content Handler to pass content in the clear to applications requesting it, it must be confident that the applications can be trusted not to mishandle the content (by, for example, discreetly writing it to a file). Since it is infeasible to automatically inspect the application's executable code to determine this, the TCH relies on WebGuard's trust verification system, which has two parts: an off-line certification process, and run-time verification process.

4.1. Certification process

Figure 3 shows the essential elements of the certification system. The certification system consists of a Certificate Generator and a Certificate Repository. To obtain trust certification for

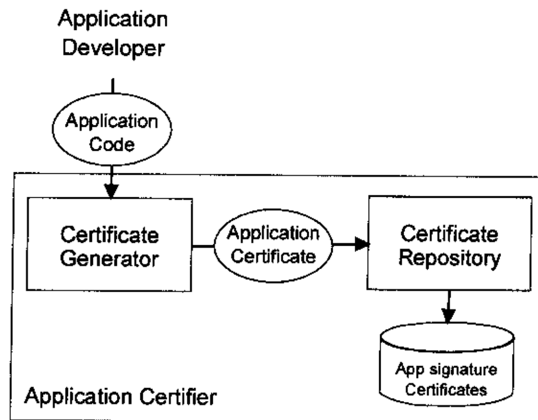

Figure 3: Certification subsystem.

their applications, application developers submit their applications to the Certificate Generator, in the same form as they will be distributed to end-users. If the operators of the Certificate Generator decide that the application may be trusted with protected content, the CG produces a Trust Certificate and stores it in the Certificate Repository. The trust decisions may be based simply on the source of the application (e.g., if from Macromedia, it is deemed trusted), or in some cases may require manual inspection of the source code.

The form of the Trust Certificate (TC) is as follows.

Program identifier. This is a string that identifies the program with the previous code digest. This may be a hierarchical name such as "Microsoft/Internet Explorer/5.01".

Property name. This is a string that identifies precisely what is being certified by this certificate. For example: "IBM Rights Manager Trusted".

Code digest(s). This is created with a conventional message digest function such as MD5 or SHA. There will be a digest for each application module that exists in a separate file.

Digital signature. This is the digital signature of the TC, using the private key of the Application Certifier.

Certifier identification. This is a conventional digital certificate containing the public key of the Application Certifier, signed by a public certificate authority.

The TC may contain other elements such as date, certificate version, and cryptography parameters.

Trust Certificates are used by WebGuard's run-time verification system to verify that the application being given protected content has in fact gone through the certification process. We have developed two variations of the run time verification process: one using a Verifying Launcher, and the other performing In-call Verification.

4.2. Code verification with a verifying launcher

The Verifying Launcher (VL) is responsible for verifying at launch time that the viewer application is certified as a trusted application for safely handling protected content entrusted to it. As mentioned above, each trusted viewer must undergo an offline certification process, which results in a trust certificate that includes the signed digest(s) of the application code modules. Before launching the viewer, VL verifies the integrity of the code. This is done by applying a message digest algorithm to the code module in question and comparing the result to the pre-signed digest. An exact match means that the code installed on the client host is identical to the one certified, and hence is safe to handle the content. VL then instructs the operating system to load the application from the verified code files. By virtue of its role as the application launcher, VL obtains OS-specific information, such as the process ID or the process creation date, that uniquely identifies the loaded application instance within the system. VL uses this information to compute a stamp that still uniquely identifies the application instance but is hard to guess or forge. The stamp is computed using a hashing function, which is known to TCH as well. The algorithm used by the hashing function must be deterministic (always generating the same result given the same input, for reasons described later. One such algorithm may be a common encryption algorithm using a predetermined key

Figure 4 shows the out-of-process verification subsystem. The procedure for an application to use the Trusted Content Handler is as follows:

By opening a file with an extension that is registered to the VL, and which also indicates the application to load, the user invokes the VL and requests it to load the application. (Alternatively, given the name of the application, the VL may locate its code through an application registry if one is available.)

Given the unique name of the application, the VL looks up the associated certificate in its Certificate Cache, or from the Certificate Repository if the needed certificate does not exist in its

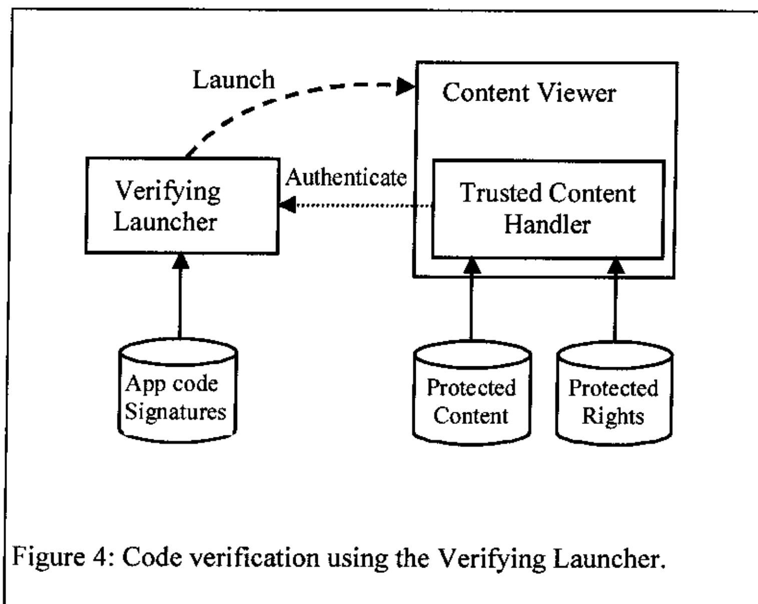

Figure 4: Code verification using the Verifying Launcher.

cache. If a certificate is not found in either the cache or the repository, the VL exits without loading the application.

The VL reads the file(s) of the application's executable code (the files to read are indicated in the certificate) and computes the code digest(s) using the same digest function as used by the certification system.

The VL compares each computed digest with the corresponding digest in the certificate. If any of the digests differ, the VL exits without loading the application.

If all computed digests match the digests in the certificate, the VL requests the host operating system to load the application. The VL then computes a *stamp* for the application, as described above. The VL then stores the stamp in internal memory.

When an application makes a call on the Trusted Content Handler (TCH) to access a protected resource, the TCH first verifies that the application was launched and verified by the VL. It does this by computing the stamp for the application using the same uniquely identifying information and scrambling information that the VL did, and then contacting the VL for comparison. If the TCH-computed stamp and the VL-computed stamp are the same, then the TCH was called by the same application instance that the VL verified and launched. The TCH may then cache its stamp so that no further communication with the VL is necessary, for this session with the application. The TCH and the VL communicate through a secure connection.

4.3. In-call verification

The Verifying Launcher mechanism has two drawbacks. One is the inconvenience of requiring that the application be loaded by the VL. Thus, if a user has a browser instance open and then wants to view protected content, a new instance will have to be loaded. The more significant drawback is that, at load time, the VL must anticipate all the executables that must be verified for a particular set of content. Presently we rely on the content producer to specify which applications will be handling the content, information which is included at content packaging time (content packaging is described in a later section). This process is, however, error-prone. Because of these drawbacks, we have recently developed an alternative mechanism, which we call the In-Call Verifier.

The In-Call Verifier (ICV), illustrated in Figure 5, uses the same off-line, signature-based certification as the Verifying Launcher, but performs the code integrity check at the time of the first request for content to the TCH. At this time the ICV makes a system service call to the host operating system to query it for the filenames of the modules that are currently loaded for the given process ID. (Currently we have implemented this mechanism for the Windows NT and Windows 2000 operating systems; we are investigating it for others.) Having the filenames of all modules loaded for the process, the ICV computes the file message digests and compares them with the digests in the corresponding trust certificates. The ICV uses a policy-based mechanism

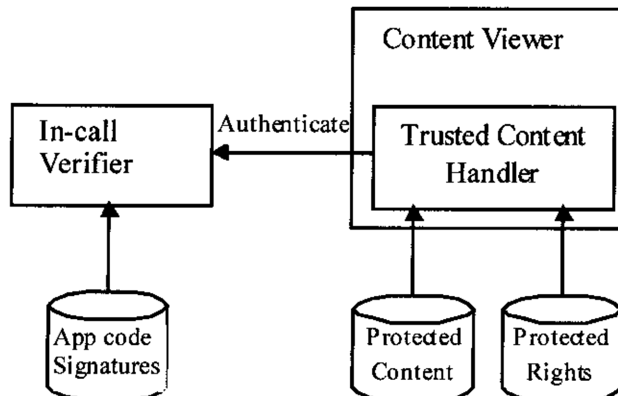

Figure 5: In-process verification.

(not described here) to determine which modules do or do not require a certificate. The mechanism also allows the ICV to reject a request for content if it detects the presence of malicious programs that may try to obtain the content directly from the system's resources such as the frame buffer or the audio driver.

5. Trusted Content Handler

The Trusted Content Handler (TCH) is a transparent extension to the content viewer (i.e., the Microsoft Internet Explorer Web browser) that is responsible for feeding the viewer with protected content. In this section we describe the structure of the TCH and the rights-specification mechanism we developed for it.

5.1. Components of trusted content handler

One of our objectives in the TCH was to develop a reusable DRM component that implemented the core DRM functions in an application-independent way. We could then use the component in other extendible platforms, such as a Java virtual machine and component-based productivity applications. Figure 6 illustrates the structure of the TCH. Note that all functions within the DRMC execute within a tamper-resistant environment designed to make its internal workings exceedingly difficult to inspect.

Figure 6 shows the structure of the TCH. The interface that the DRMC offers to the IE protocol

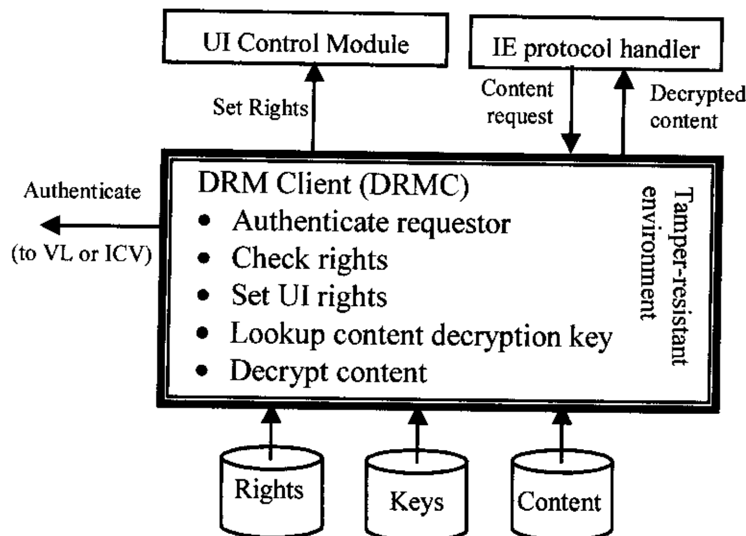

Figure 6: Structure of Trusted Content Handler.

handler is similar to a file system interface in following an "open-read-close" pattern. The content identifier received from the protocol handler is a URL with a hierarchical path name identifying the content. One issue we faced is, given a path to an item of content within a larger content package, how do we identify the package so that we may look up the rights for that item in the package (without associating a rights file with each item)? We settled on the following identifier scheme: the URL has the following form: <protocol name>://<package path>/<package name marker><package name>/<resource relative name>, where the <package name marker> is a constant used to aid the parser in identifying the package name. E.g., `rmhttp://....../PackageMark_P1/......` In this case the protocol is `rmhttp` and the package name is `P1`. Given the package name, we can locate the set of rights associated with the package.

5.2. WebGuard rights specification mechanism

The three elements of the transparent DRM approach—application certification, a trusted content handler, and application-independent UI control—are in fact largely independent of the rights specification mechanism used. Thus we could have adopted an existing rights specification language for our system. However, we found that rights specification for protected packages containing large numbers of individual content items—e.g., courseware, photography collections, literature anthologies—presents issues not addressed by existing rights management systems. First of all, there must be the ability to specify rights at a fine granularity. This is important in enabling content owners and distributors to offer consumers a variety of ways to purchase content in these large packages, and in giving consumers choice in how they purchase it. But rights specification must also be convenient—it should not be necessary to specify rights individually for each item of content. Finally, there should be an efficient run-time mechanism, both in speed and in storage requirements, for associating a particular item of content with its corresponding set of rights. Meeting these requirements was the design goal of the rights specification mechanism described here.

A complete rights specification for a protected package consists of (1) one or more rights files, (2) a key-table file, and (3) a rights-map file. These files are collected together in one directory, the directory having the same name as the package name.

5.2.1. Rights files

Rights files are text files with the "`.ini`" extension. A package may have multiple rights files but must have at least one. The format of the core of a rights file is a series of lines, each line granting or denying one right. Example:

```
Play: yes
Print: no
Save: no
Clip: no
```

More sophisticated rights are also supported, such as time durations, but we do not describe the complete rights specification language here.

5.2.2. Rights map

A rights map associates content items with rights files in a compact and efficient way. Conceptually, it is a set of <file specifier>: <rights file> pairs, as illustrated below.

File specifier	Rights file
file1	RightsSetFile1
.../Section1/*	Section1Rights
.../Section2/*	Section2Rights
...	...

Figure 7: Conceptual Rights Map

As implemented, a rights map is a text file with the name "`rightsmap.ini`". Following is a sample rights map.

```
[*]
RightsFile: default_rights.ini

[Andrew_Jackson.htm]
RightsFile: AJ_rights.ini

[Andrew_Jackson_files/*]
RightsFile: AJ_rights.ini

[Jacksons_Hermitage.htm]
RightsFile: JH_rights.htm

[Jacksons_Hermitage_files/*]
RightsFile: JH_rights.htm
```

Content-path specifiers (within '[' and ']' brackets) are relative to the parent directory of the course. Thus, for a file named "`D:\packages\RMHTTP_PACKAGE_ROOT_DB2Demo\index.html`", the content-path specifier in the content-attributes table is "`index.html`".

So that not every file in a package requires its own specification line, the rights map allows hierarchical specification. In the sample above, "[Andrew_Jackson_files/*]" specifies all the files in the directory "Andrew_Jackson_files".

It can be seen in the sample above that a given a file path name may match more than one path specification. For example, Andrew_Jackson.html matches both [*] and [Andrew_Jackson.html]. The rule is that specification with the longest matching prefix is chosen. This makes it possible to assign a default rights file and then override it for selected files.

Since prefix matching is strictly on the basis of whole path elements, the time complexity of any one rights-file lookup (i.e., given a file name, the time required to look up the associated rights file) is linear in the length, in path elements, of the file name. For example, "[index.html]" has one path element, and "[Andrew_Jackson_files/*]" has two. Since the depth of a tree of n nodes is proportional to $\log_2(n)$, the time complexity of rights-file lookup in a package of n content items is $O(\lg n)$.

5.2.3. Key tables

Given that any one package may contain hundreds of files, it may not be desirable, for security reasons, to use a single key for all files. Thus for the WebGuard system we developed a mechanism for scattering a set of keys among the various files in a package. While details of the algorithm are omitted for brevity, it provides for a uniform distribution of keys among the files, and an O(1) key-lookup time.

Keys are distributed separately from the content in a file called a key table. A key table is a text file with the name "keytable.ini". Following is a sample key table.

```
[Version]
0.1
[Parameters]
Algorithm: RC4
NumberOfKeys: 7
[Keys]
0123456789
ABCDEF0123
456789ABCD
EF01234567
89ABCDEF01
23456789AB
CDEF012345
```

Keys are specified in hexadecimal format, and must have an even number of digits. The keys in the sample each have 10 digits, and are thus 5 bytes, or 40 bits, long.

5.3. Content packaging

After the rights specification files—the rights files, a key table, and a rights map—have been prepared, a packaging tool encrypts the content files for distribution, using the keys specified in the key table. The rights specification files are also encrypted, and remain encrypted while on the user's system.

Because the names of the content files and rights specification files, and their organization in directories, is not protected against tampering, some users may attempt to circumvent the rights management system by renaming rights files and/or content files. Seeing a file named "`NoRights.ini`" and a file named "`AllRights.ini`", a user may be tempted to delete the `NoRights.ini` file, then make a copy of the `AllRights.ini` file and name it `NoRights.ini`. The obvious intent would be that any content assigned the rights in `NoRights.ini` would effectively have the rights in `AllRights.ini`.

To prevent this, the packager inserts into each rights specification file and content file, at encryption time, its name. (For rights specification files, the name is relative to the package's rights-specification directory; for content files, the name is relative to the content root directory.) When one of these files is accessed, the name used to access the file is checked against the name embedded in the file itself. If the names do not match, the file is not decrypted.

6. User interface control module

The User Interface Control module (UICM) blocks those UI operations that would represent actions not allowed by the rights a user possesses for the content. The "policy" of the UICM is set dynamically by the DRMC upon each request for content. When the user attempts a non-allowed operation by choosing it from a menu or using a "hot key" such as Ctrl-C, the UICM blocks the operation and informs the user in a dialog box. Operation of the UICM is transparent to the application. In this section we describe in overview the operation of the UICM.

6.1. Filtering UI messages using window subclassing

Since the standard method for handling the user activities and user requests in Windows™ applications is by sending messages to the application containing the command requested by the user. We found that by intercepting the messages sent to the browser, and filtering them appropriately, we can achieve the required controlled behavior. We use a technique known as "window subclassing."

Each application in Windows systems has a main procedure, *WinProc,* which is called by the windows system to handle the application's messages. The address of this procedure is stored in an application window's class information structure. By window subclassing we can replace the WinProc address in that structure by the address of our own procedure. However, rather than subclassing each individual window in an application, instead we subclass the window classes we are interested in. Then each time a new window of one of these classes is created, our subclassing information is created with it. To subclass a window class we call:

SetClassLong(hwnd, GCL_WNDPROC, (LONG)(NewWndProc))

where ***hwnd*** is the handle of the window we want to subclass, GWL_WNDPROC indicates that we need to change the WndProc information in that structure, and ***NewWndProc*** is the address of our procedure that we want to replace the original WndProc. We store original WinProc address in order to pass the valid messages to it to be processed in a normal way.

6.2. Determining application windows and events to block

Above we described the mechanism we use to block the window events of interest to us. But how do we know which windows to subclass and which events to block? Through a manual process of inspecting the execution of an application, using a tool such as Microsoft Spy++, we can determine which events are generated by certain user actions, and which windows they occur in. The basic procedure is to first use Spy++ to generate a graphical display of an application's windows. Doing this for Internet Explorer, for example, yielded the three window classes "MainHeader," "MainBody," and "Body." An instance of the latter window shows the current page. Having determined which windows are of interest, we then use Spy++'s Messages option to view the messages (events) sent to a specific window. For example, to intercept the Ctrl+C hot key, we would watch the messages sent to Body window and then press Ctrl+C and see what message and what parameters have been sent to that window. This information is then input to the UI Control module to set its policy for event blocking.

6.3. Dynamic UI control

The UI Control module has a single interface function:

int UIControlSetRights(HWND hwnd, WGRightsSet *rightsSet)

This function is called by the DRMC whenever a protected page is loaded in order to set the rights set for it. Also, the above function receives the handle ***(hwnd)*** of the current active

window. This is needed for supporting multiple windows or multiple frames per window, where each window or frame may have a different set of rights associated with it.

6.4. Intrusion detection

Using the same method of window subclassing that we have described here, an intruder could attempt to disable our security model by re-subclassing the windows which we had already subclassed. This way, the intruding program can intercept the window messages before our new WndProc receives them. Then the intruder could pass these messages to the original WndProc of the controlled application (e.g., IE). In order to prevent this, we should either prevent any additional subclassing for the windows we subclassed, or detect any such additional subclassing and terminate the application immediately. We implemented the latter method in our UI control module.

To detect intrusive subclassing, we perform a periodic test on the WndProc fields in the current windows of the controlled application and their corresponding classes. The values of these fields should be equal to the address of our new WndProc function. If the values differ, then there is an intruder who replaced our filtering module, and we terminate the application immediately.

7. Conclusions

We have presented WebGuard, a system that provides digital rights management in a transparent fashion to off-the-shelf Web browsers and browser plug-ins. We reviewed the current state-of-the-art in DRM systems, in particular discussing the prevailing approaches to providing end-user trusted players. We described how these approaches present difficulties for producers of Web-based content because they would require not only the browsers but each and every plug-in to be DRM-enabled. We then discussed our new approach to providing DRM transparently, which relies on certificate-based code verification, a trusted content handler, and event-based user interface control. We then described each element separately.

We are currently working on extending and improving WebGuard, in several ways. One area for development is the user-interface control module, which is at present specialized to Internet Explorer. We will be developing a configuration mechanism for the UICM so that it can be easily tailored to any application, given only the window classes and set of events it needs to monitor. This specification will be tied to the application certification process.

Another area for improvement is in devising protocols that enable access to protected web content, without requiring the user to go through an initial rights acquisition phase. The WebGuard system, as described, requires the user to go through a rights acquisition phase during

which rights to access protected content are explicitly acquired and downloaded to the user's machine. Instead, we would like to enable users to surf the web freely with rights being acquired automatically as they encounter protected content. The rights may be acquired in accordance to a user pre-set profile, possibly linked to a payment account.

Finally, we hope to be able to support other popular web browsers, such as the Netscape browser, in the near future.

Acknowledgements

The authors would like to thank Craig Bennett, Tim Crowley, and Chris von Koschembahr of IBM for their helpful and insightful comments on earlier versions of this work.

References

[1] J.T. Brassil, S. Low, N.F. Maxemchuk, Copyright protection for the electronic distribution of text documents, Proceedings of the IEEE, 87(7), pp. 1181-1196, July 1999.

[2] A. Choudhury, N. Maxemchuk, S. Paul, and H. Schulzrinne, "Copyright protection for electronic publishing over computer networks", *IEEE Network*, May/June 1995.

[3] ContentGuard, http://www.contentguard.com.

[4] IBM's Electronic Media Management System, http://www.ibm.com/software/emms.

[5] Intertrust's Metatrust system, http://www.intertrust.com.

[6] U. Kohl, J. Lotspiech, and S. Nusser, Security for the digital library—protecting documents rather than channels, in Proc. of the IEEE 9th International Workshop on Database and Expert Systems Applications, 1998.

[7] G. Lowton, Intellectual property protection advances with new technologies, IEEE Computer, January, 2000.

[8] Microsoft's Windows Media Rights Manager, http://www.microsoft.com/windows/windowsmedia/en/wm7/drm.asp.

[9] O. Sibert, J. Horning, and S. Orwick, A massively distributed trusted system, in Work-in-Progress session at the 16th ACM Symposium on Operating System Principles, Saint-Malo, France, October, 1997.

[10] WebBuy, http://www.adobe.com/products/acrobat/webbuy/main.html.