

國立交通大學
資訊科學與工程研究所
碩士論文

應用於 COTS 套裝軟體、以方針設定導引監控的
版權管理系統

OpenDReaMS: Policy Controlled Digital Rights Management

Wrapper for COTS



研究生：黃哲逸

指導教授：黃世昆 博士

中華民國九十五年六月

應用於 COTS 套裝軟體、以方針設定導引監控的
版權管理系統

OpenDReaMS: Policy Controlled Digital Rights Management

Wrapper for COTS

研究生：黃哲逸

Student: Tsue-Yi Huang

指導教授：黃世昆 博士

Advisor: Dr. Shih-Kun Huang

國立交通大學
資訊科學與工程研究所
碩士論文



A Thesis
Submitted to Institute of Computer Science and Engineering
College of Computer Science
National Chiao Tung University

In Partial Fulfillment of the Requirements
For the Degree of
Master
In
Computer Science

June 2006

Hsinchu Taiwan, Republic of China

中華民國九十五年六月

應用於 COTS 套裝軟體、以方針設定導引監控的 版權管理系統

研究生：黃哲逸

指導教授：黃世昆 博士

國立交通大學資訊科學與工程學系（研究所）碩士班

摘要

隨著網際網路的普及與快速成長，內容產品數位化已成為無可避免的趨勢。在此同時，如何保護數位化後資料的版權問題也越來越重要。本論文發展一開放源碼的系統，能針對一般常用的播放軟體(包括商用、非商用)提供數位版權管理(DRM)機制。透過我們的系統，能使原本沒有 DRM 機制的播放軟體也能達到具 DRM 控管的效果。此外，我們採用控制播放軟體的方式，能不受限於數位內容的格式，達到更廣泛的應用。因為商用軟體並沒有原始碼可供分析，我們運用外掛程式，進行動態偵測分析、並攔截播放軟體的控制及資訊流程，在用戶端上實作數位內容的保護機制。我們除了實作在用戶端上保護數位內容的系統外，也設計及實作一套簡單、完整的 DRM 控制流程(跨越伺服器及用戶端)，對許多常用的播放軟體進行測試與實驗，以驗證本系統的可行性。

OpenDReaMS: Policy Controlled Digital Rights Management wrapper for COTS

Student: Tsue-Yi Huang

Advisor: Shih-Kun Huang

Department (Institute) of Computer Science and Engineering
National Chaio Tung University

Abstract

With the fast development and increasing use of the Internet, it has been a major trend to distribute content in digital form. At the same time, how to protect the copyright of digital content is getting more important. In this thesis, we develop an open-source system that offers digital rights management (DRM) capability to the rendering software. It can be used for commercial off the shelf (COTS) software without source code. Through our system, we can introduce DRM to the non-DRM rendering software. Besides, we can adjust the control policy of rendering software and protect the digital content without considering the restriction of content format for wider applications. Since the source code of COTS is not available, we use the approach of software plug-in to detect, analyze and intercept the control and information flow of the rendering software, and implement the protection for digital content in client side. We also design a simple and complete DRM workflow which encompasses the management process of the server and client side. The system has been tested by wrapping popular COTS readers that are commonly used and prove the applicability and feasibility of our system.

誌謝

首先感謝我的父母，一直以來都沒有給我什麼壓力，也支持我做的每一個決定，同時還讓我擁有良好的成長環境來完成我想要完成的事，我真的很幸福。

感謝指導老師黃世昆教授，因為我是個跨領域的考生，對資工這個領域的專業知識、常識都懂的很少，但黃老師一直很有耐心的教導我，甚至在 meeting 的時候，一步一步指導並示範如何 debug，我永遠會記得這一幕，老師，我真的很感謝您。

再來一定要感謝實驗室的夥伴們，昌憲學長、建智、有德、孟勳，這二年真的麻煩你們了，當了我二年的電腦百科@@”，也因為有你們在，我才有辦法念完這個碩士，我們一起在實驗室徹夜打拼的情景，我會一直記在心中的^^b



Table of Contents

摘要.....	iii
Abstract.....	iv
誌謝.....	v
Table of Contents.....	vi
List of Tables.....	viii
List of Figure.....	ix
1. Introduction.....	1
1.1. Problem description.....	1
1.2. Background.....	1
1.2.1. Digital Rights Management.....	2
1.2.1.1. Content server.....	2
1.2.1.2. License server.....	3
1.2.1.3. Content Reader.....	3
1.2.1.4. Working flow.....	3
1.2.2. Rights Expression Language.....	4
1.3. Motivation.....	5
1.4. Objective.....	6
1.5. Synopsis.....	8
2. Related Work.....	9
2.1. DRM Architectures.....	9
2.2. System Call Intercepting Techniques.....	12
2.3. Rights Expression Languages.....	13
2.4. Existing DRM systems.....	14
3. Research Method.....	16
3.1. System Architecture.....	16
3.1.1. Content Server.....	17
3.1.2. License Server.....	17
3.1.3. License Proxy.....	17
3.1.4. Helper.....	17
3.1.5. User Wrapper.....	18
3.1.6. Encrypted, Decrypted and Scrambled Data.....	19
3.1.7. COTS Reader.....	19
3.2. Working Flow.....	19
4. Implementation.....	21
4.1. User Wrapper.....	21
4.1.1. Interception Steps.....	23

4.1.2.	Function Interception	24
4.1.2.1.	Render	24
4.1.2.2.	Print	27
4.1.2.3.	Copy, Cut, Paste	27
4.1.2.4.	Print Screen (in the keyboard)	27
4.1.2.5.	Save	28
4.1.3.	Security Interception	29
4.1.3.1.	Drag-Drop	29
4.1.3.2.	View Source Code (in web page)	29
4.2.	Helper	29
4.3.	License Server	32
4.4.	License Proxy	34
4.5.	Experience and Discussions	34
4.5.1.	Rendering	34
4.5.2.	Dialog box	36
4.5.3.	Clipboard	37
4.5.4.	Other Experiences and Discussions	38
5.	Results and Assessment	40
6.	Conclusion and Future Work	46
	References	47



List of Tables

Table 2-1 A comparison between WebGuard, SUMMER and our wrapper12	
Table 5-1 Comparison between our work and other DRM systems.....	44
Table 5-2 Experimental COTS Readers	45



List of Figure

Figure 1-1	Basic DRM architecture.....	2
Figure 1-2	Basic DRM working Flow	4
Figure 1-3	The difference between our work and the basic DRM architecture.....	7
Figure 3-1	System architecture	16
Figure 3-2	The working flow of our DRM architecture	20
Figure 4-1	The logging file of the traceapi	21
Figure 4-2	The basic algorithm of our implementation	22
Figure 4-3.	The control flow of rendering the protected content.....	25
Figure 4-4	The algorithm of implementation the ‘render’ functionality ..	26
Figure 4-5	The algorithm of implementing the ‘print’ functionality	27
Figure 4-6	The algorithm of implementing the ‘PrtScrn’ function key....	28
Figure 4-7	The algorithm of implementing the Helper.....	31
Figure 4-8	The situation that IE plays the multimedia files.....	32
Figure 4-9	Simple license file	33
Figure 4-10	The thread problem	36
Figure 4-11	The clipboard problem in our work	37
Figure 4-12	view source working flow in IE and firefox	39
Figure 5-1	Log-in web page in license server.....	40
Figure 5-2	The license page in license server	41
Figure 5-3	The page to open the COTS Reader.....	42
Figure 5-4	Denying rendering.....	43
Figure 5-5	Protection of viewing source.....	43

1. Introduction

1.1. Problem description

Digital rights management (DRM) is an important issue in today's e-business world. Digital rights management is the aggregative term referring to any of technologies used to enforce various policies, controlling access to software, music, movies, pictures, or other digital content and hardware including the revising, accessing, transferring to others...etc[1]. The content owner can take advantage of it to achieve his goal about business behavior, protecting contents, or other purpose, but there are some problems within it.

One of the major problems in Digital rights management system is the need for installing dedicated software for each content format. DRM-aware client systems are often expensive and vendor-dependent, hard to maintain the inter-operability between each content format. Due to variations of rights constraints and different requirements of rights enforcement, we cannot easily develop a generic DRM system for most of the Commercial off-the-shelf (COTS) readers. It's a time-consuming process to develop such a client side DRM wrapper.

Most of the existing DRM systems develop their own server, content player for their own content format or some specific content format, so they almost can't communicate with each other. In this kind of the circumstance, it will increase the cost and the difficulty to popularize the DRM system.

1.2. Background

First of all, we need to know what is the inter-operations in the Digital Rights Management and the important item, Rights Expression Language in order to help to understand our research work.

1.2.1. Digital Rights Management

DRM is a well-known and important system for digital contents. A DRM system generally contains three components DRM content server, License server for DRM, DRM-aware content reader [2]. Figure 1-1 describes the basic components of the DRM.

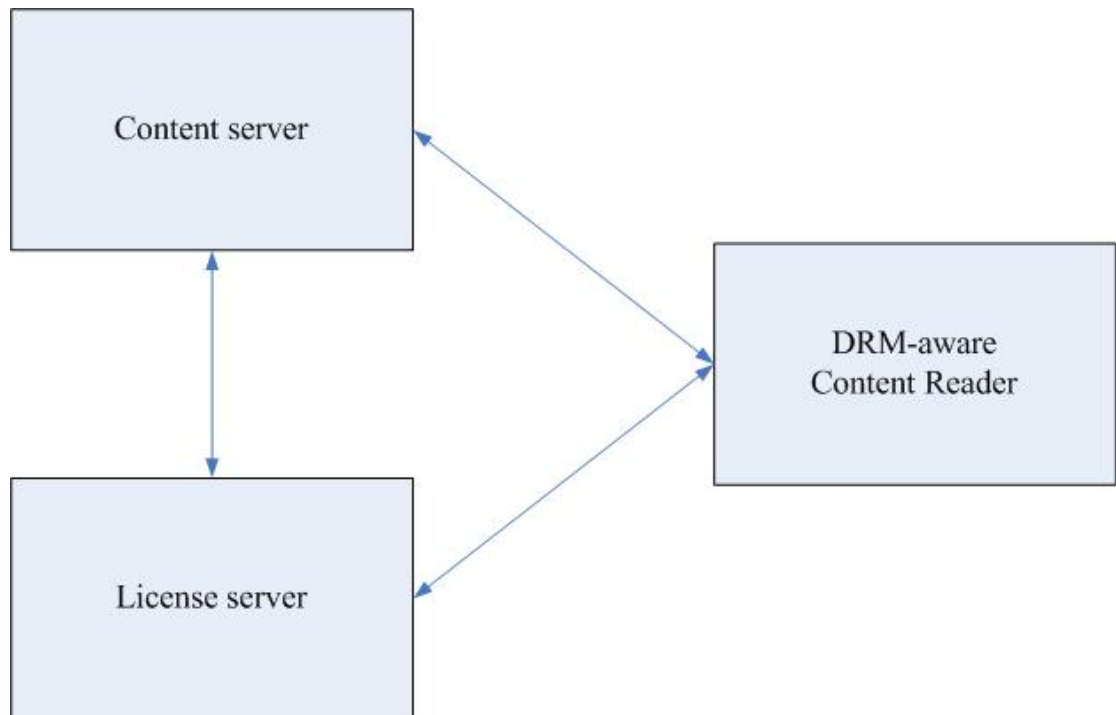


Figure 1-1 Basic DRM architecture

1.2.1.1. Content server

The Content Server is responsible for content processing about receiving content from content producer (owner), protecting content by cipher technologies, watermarking or other content protecting technologies and delivering the content to the client-side Content Reader.

The Content Server is also responsible for granting the client-side Content Reader, granting the License Server and delivering the necessary information to License Server for generating license.

1.2.1.2. License server

The License Server is responsible for license processing about license generating from the information receiving from the content server by some license generating algorithms such as XrML, ODRM, MPEG-21 REL...etc and processing the requests receiving from the Content Reader by checking the generated license. The License Server is often outsourcing to be the third-party certification.

1.2.1.3. Content Reader

The Content Reader is the content rendering application at the client side. This application can be the normal rendering application in the world such as Internet Explorer browser, Mozilla Firefox, Windows Media Player, Winnamp, Adobe Reader, ACDSsee, Djview...etc and the special rendering application designed for some special purpose such as decrypting the ciphered content, getting grants from the License Server, etc. It's all depending on the design of the DRM system.

1.2.1.4. Working flow

The Content Server gets the content from content owner (producer), authenticating the Content Reader, ciphering and delivering content to Content Reader and sends the necessary information for license generating to License Server. A user gets digital contents from the Content Server. Following that, the DRM-aware content reader receives the user's requests and gathers the necessary information for authorization, and then sends it to the license server. Finally, the license server authorizes the requests and gives the information for DRM-aware content reader executing the requests back to client. Figure 1-2 shows the basic working flow of the DRM system.

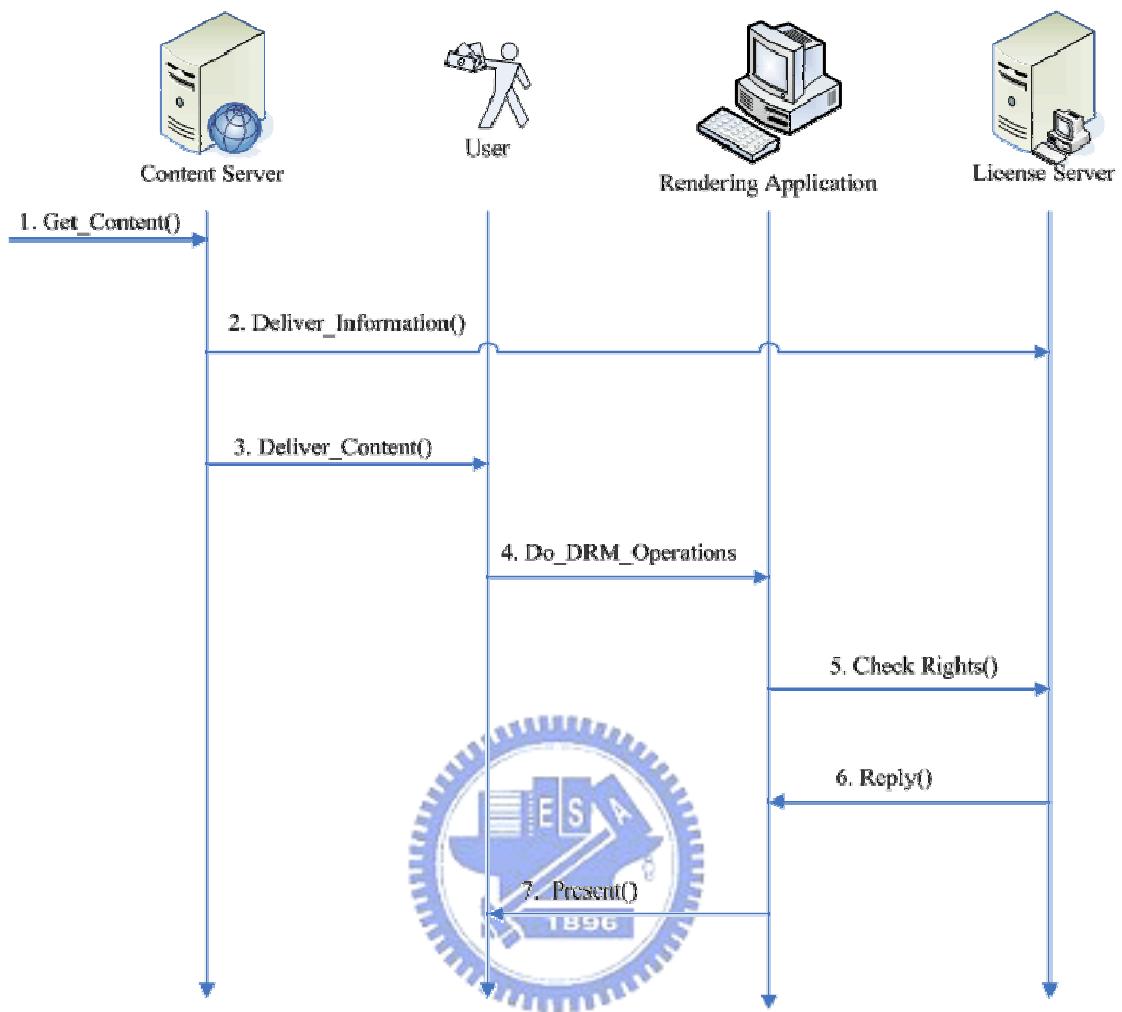


Figure 1-2 Basic DRM working Flow

1.2.2. Rights Expression Language

The commerce transaction of the multimedia content through the network today is based on the exchange of the rights. But, this situation is exactly one of the limitations of the Digital Rights Management (DRM) due to that the rights description must be machine-readable and can fit various business models. It is hard to express a complex and unambiguous rights permission to fit in with the requirements.

The main mission of the Rights Expression Language (REL) is the rights expression, which describes a permission to allow a user to use the content to achieve the protection purpose. The Rights Expression Language must be based on a syntax

that is recognized in order to reach the goal, machine-readable [3].

There are several Rights Expression Language have been proposed to describe the license. The famous two Rights Expression Languages are ODRL and MPEG-21 REL. Both of them are based on the XML format.

1.3. Motivation

The employees let out the business secrets. The staffs of the bank steal and sell the personal information of the consumer. There are many leaking out secret occurrences in quick succession to tell us that you can stop the hacker but the thieves within a household are difficult to guard against even if you have the conscientious and careful firewall. The digital rights management is becoming more and more important in today's network environments.

Following that the digital rights management has been applied to many kinds of the file formats, you can find that the digital rights management system is everywhere within the movies, music, documents...etc, and the number of the corporations that invest in the research of the digital content protection technology is also increasingly. At present, many famous companies, such as, Microsoft, Apple, Real Network, Adobe...etc, has been developing and selling their own digital rights management technologies. However, these technologies can't understand each other because of that all of them have their own standards, so there is a vast requirement which is the common standard of the digital rights management no matter the structure, working flow, Rights Expression Language...etc, or the translating frame of reference in the market and the sun corporation is developing an open-source project, called DReAM [4], to endeavor to achieve this goal through the open-source model.

We realize that the hardest and complicated part is DRM-aware content reader because of its functions such as rights constraints and rights enforcing, such that this

part often wastes a lot of time and money (the order of hundred thousands) during developing a DRM system. (For example, DJViewer)

Besides, most of the existing DRM systems are vertical-integration, that is, the content which they can process is a specific format or created by them. This kind of DRM system is designed for some special content format from server to client side software. If you want to view or use more than one DRM contents, then you must install or buy different kinds of DRM software for client side and these software systems are often expensive.

For this situation, we will design an open-source Policy Controlled DRM wrapper for COTS that can apply to most existing COTS Readers and can process all existing content format that the COTS Reader supports.

1.4. Objective

We develop a flexible DRM wrapper for COTS readers. One of the features of this flexible DRM wrapper is that it can be policy controlled by configuring for rights enforcement in any COTS readers instead of hard-coding policy in the program. By this way, we can configure client-side wrapper and helper easily by modifying the configuration files, similar to a DRM firewall for COTS readers. The other feature is that the policy controlled DRM wrapper can communicate with other DRM or non-DRM server and we also demo some security-protection functionality.

This wrapper will monitor the running COTS reader and perform the detection mechanism to analyze the rendering application's behavior. For the content protection and rights enforcement, this wrapper will uses the interception mechanism to achieve the content protection and indirect the target parameters and codes into the running memory ordered by the license from the license server in order for various rights enforcement.

Moreover, we design a flexible structure that can communicate with other existing non-DRM or DRM functionality servers in order for integration with existing DRM systems.

With this tools, we don't need to consider that whether the COTS-reader having DRM functionality or not when we render a digital content protected by a DRM system. For the content owner (producer), we don't need to consider that if the content server having DRM functionality or not when we using the server to store and protect our content. Figure 1-3 show the differences of our work based on the traditional DRM architecture.

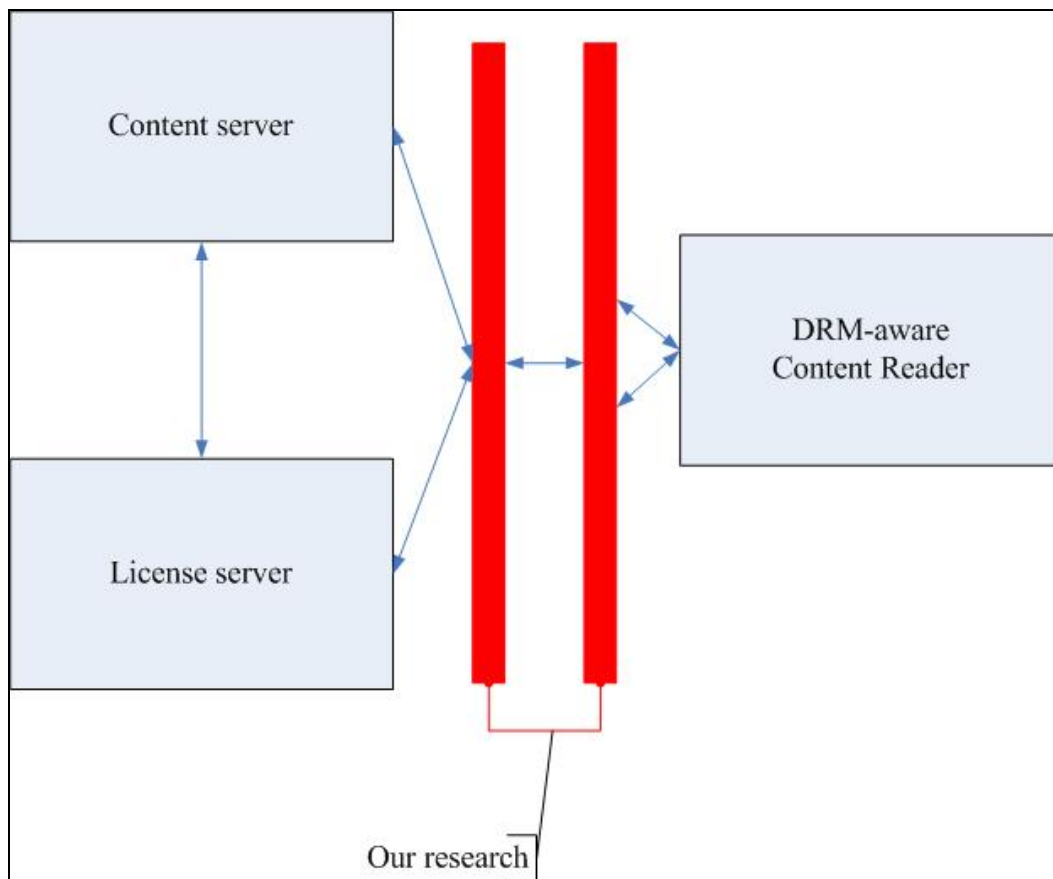


Figure 1-3 The difference between our work and the basic DRM architecture

1.5. Synopsis

In Section 2, we present the related work. The research method is explained in Section 3. The detailed ideas and implementation will be described in Section 4. The experimental results are in Section 5. We apply our system to an existing COTS Reader, IE, as an example to elaborate our research results. Finally, the conclusions are in Section 6.



2. Related Work

2.1. DRM Architectures

A basic DRM reference architecture is presented in [2]. There are three major components: the content server, the license server, and the client. These major components are consisting of many sub-components. It not only describes the detail working flow of the DRM system, but also compares many methods and techniques to explain the reason why this paper chooses its method. Furthermore, it introduces and discusses the two most prevalent core technologies involved in DRM implementation: encryption and water marking.

[5] presents an open and secure DRM solution which is named Open SDRM. It deploys the traditional DRM architecture and is based on open-source components. Its architecture is started from the OPIMA international specifications, the MPEG-4 IPMP Extensions and the emerging MPEG-21 IPMP architecture.

A careful idea about the flexible management control flow of certificates and authorities is proposed in [6]. There are two innovations here. 1) It combines the identity, attribute and rights to allow for maximum flexibility. 2) The digital licenses are generated on demand after the identity and the security attributes have been verified. It also explains the difference between the public key certificate, attribute certificate and digital license by mapping them into the relations among the passport, visa and residence permit in the work flow of the immigration. There two features in this security attribute based digital rights management system. 1) It use the public key certificate, attribute certificate, content identification and a secure of randomness to generate a secret, unique, personalized content key. 2) A hierarchy of authorities, for example, JI (identity) is a computer engineer (attribute) and he is only allowed to enter the system by a dedicate computer (attribute) and he can modify the source code (rights) and print (rights) it. It develops a prototype which is called SUMMER, a

secure distributed multimedia database management system, and the future work in the client-side component of this architecture is to build an independent application in order to interface to arbitrary Render Application via smaller plug-ins. It plans to solve the problem that the Renders and plug-ins run in an unsafe environment by using tamper resistant hardware with watermarking techniques. It uses a SPIN model to test their prototype against the ability that can prevent digital content from super re-distribution and find that the drawback of this work to catch the thief is the high price of caching all license keys. It may use an appropriate hashing technique to solve this problem, but the falls positives problems will be induced. A web content protection system, WebGuard, is proposed in [7]. It provides the digital rights management for off-the-shelf Web browsers and browser plug-ins by a serious of verification process to trust an application at call-time, a trusted content handler and a user interface control module.

The secure architecture that is allowing digital rights management in home networks which is consisting of consumer electronic devices is described in [8]. The main idea is that allows devices to establish dynamic groups, so called “Authorized Domains”, in order to allow the acquired rights content legally can move from device to seamlessly. This “Authorized Domain” is consisting of licensing organization, manufacturers, content providers, compliant devices, authorized domain manager device, and content manager devices. The security architecture is based on a novel compliance checking protocol which allows relying on public key certificates issued by a license organization. The great advantage of this architecture is that the public key operation is required seldom. Another advantage is that only the device that stores the device master key need tamper-resistant memory. One limitation of this architecture is that it still needs the public key authentication for device registration. Another limitation is that the size of the authentication credential set is proportional to

the maximum number of the devices in the domain. The maximum domain size is restricted by the given storage constraints with devices.

There is a different view point of the DRM system proposed by [9]. It divides the DRM system into three blocks of layers like that the OSI layered model and the TCP/IP protocol divide the process of the data communications into layers. It indicates that the Rights Expression and Interpretation is the key node when communicating upper with lower layers by mapping the DRM system into an hourglass structure like the IP in the TCP/IP protocol. The advantage of this layered approach is that it separates rights enforcements from services. Thus it allows development separately and independently in these areas. Furthermore, the effect that it will not disturb other layers when changing or adding functions into some layers is happened. The work in [10] is adding the communications between layers. It also proves that this layered approach is a helpful method to analyze the interoperability in a DRM system by mapping the Microsoft DRM 10 architecture into the layered DRM framework.

The sun's corporation is developing an open DRM system which is called DReaM [4]. This architecture is based on open-standards-based-solutions and supports both of the Conditional Access System (CAS) and the Digital Rights Management (DRM) models. The goals of this DReaM are that it wishes to work with any content type, multiple file formats and codec and can work cross the device types and operation systems. It also wishes to control the access to content regardless of the delivery media, whether it is a physical or a digital medium and can support widespread business models to provider the flexibility.

[11] describes a case study that incorporates an effective DRM system with previously deployed DRM system at the Greek Orthodox Archdiocese of America (GOA). Here also describes that how the Elisar's MediaRights technology protects

content and the container using encryption, watermarking, specific file format, time lock, tamper proofing, obfuscation and implementing itself in kernel level and how to incorporate with previously DRM systems at the GOA.

Our main ideas is that we wish our User Wrapper can work without considering the types of content and the COTS Readers via small DLL plug-ins. This combines the goals of the DReaM, the future work in client side of SUMMER and concept of the verifying the rendering applications.

	WebGuard (2001)	SUMMER(2002)	OpenDReaMS Wrapper
Rights implementation	Windows message	Windows message	Win32 API function
Conditions	No	No	Yes
policy	dynamic	Unknown	dynamic
Type	Plug-in	Plug-in	Plug-in
Apply scope	Web browser	Adobe Acrobat	COTS Reader

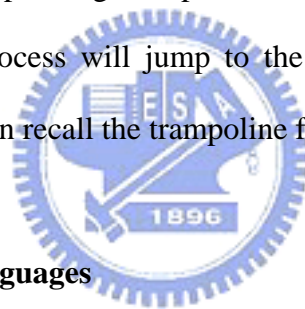
Table 2-1 A comparison between WebGuard, SUMMER and our wrapper

2.2. System Call Intercepting Techniques

[12] proposes a generic software wrapper system for hardening COTS software. It implements this wrapper in kernel level by a loadable kernel module and designing a Wrapper Definition Language (WDL) to listen for specified events to wrapper the specified system calls. The key element of the WDL is to augment the system call API with semantic information by using tag process. It can allow wrappers to refer to the system calls easily and isolate the wrapper writer from low-level details. This wrapper is efficient and protected because of no context-switch overhead and executing in kernel space. The limitation is that events occurs at the application level,

at the system call level, are difficult to sense in the low-level system calls stream. [13] inherits and extends this work. It improves the Software wrapper and makes use of it to do intrusion detection by using the Generic Software Wrapper Toolkit (GSWTK). If this kind of software wrapper be compromised, it will cause devastating damages because that it is implemented in kernel space.

An open-source binary interception tool, detours, that has been developed by Microsoft Research is a library for instrumenting Win32 functions on x86 machines [14]. It can monitor the target function and replace the first instruction of it with unconditional jump, which points to the detour functions that the user provides. Users can do their works in these detour functions. The instructions replaced from the target function are keep in a corresponding trampoline function. If the target function is called, the control of this process will jump to the user-provided detour function. Finally, the detour function can recall the trampoline function or return to the caller.



2.3. Rights Expression Languages

The MPEG-21 REL [15] developed by the standardization committee, the Moving Picture Experts Group, is designed for content owners to specify the usage grants for consumers, that is, the content owner can limit his content used by someone with some rights, restrictions or conditions using the functions supported by the MPEG-21 REL. It also allows consumers to set up secure personal parameters to protected individual privacy. The Open Digital Rights Language (ODRL) [16] is designed for the DRM to provide flexible and interoperable mechanisms to support clear and creative usage of digital resources. This is also based on XML grammar. It focuses on the definition of elements in the data dictionary and the semantics of using these elements. [3] introduces the REL, MPEG-21 REL and the ODRL and proposes an analysis of the similarities and the interoperability of both RELs. It is also

developing a tool, Distributed Multimedia Application Group (DMAG), to generate and check licenses describing by both RELs. Our License Server is based on this tool.

The rights expression language of the Rights Enforcing Access Protocol (REAP) [17] is generated from modifying the ODRL. It needs the entire ODRL language as input leading to two situations. It is aimed to demonstrate that how to publish the intellectual property in the Internet by digital libraries according to the copyrights laws. It is able to interpret all rights expressions in ODRL or to ignore parts of rights expressions in ODRL that it could not interpret correctly. The REAP rights language only can describe the usage rights like print, execute and play and the language doesn't have a security model. It is not as flexible as the ODRL language, but it is easier to understand.

An open-source PARMA is proposed in [18] for network and mobile applications based on ODRL. The PARMA REL is an extension of ODRL designed by them, so it is compatible with OMA REL which is used in mobile applications because that the OMA REL is also based on ODRL. Therefore, PARMA REL is compatible with current DRM system integrated with mobile phones. There is an issue that when we want to add a new rights specific call to specify the rights object, we need to modify the source code traditional, but this way of doing is inflexible and will increase the working load of the application developers. It solves this issue by the concept of Aspect-Oriented using an Aspect Oriented Software Development (AOSD) tool.

2.4. Existing DRM systems

Microsoft Windows Rights Management consists of three components, Windows Rights Management (RM) technology, Windows Rights Management Services (RMS), Windows RMS Software Development Kit (SDK), and Windows RM Client SDK

[19]. It provides a complete rights management system for the enterprise. [20] explains the Rights Management Add-on (RMA) for Internet Explorer, the .rmh (RMH) file format, and the Rights-Managed HTML (RMH) SDK and specifies how an organization can protect their sensitive information by using these technologies and the Microsoft Windows rights management technologies. The major differences between our DRM system and the Microsoft RMA/RMH is that the Microsoft RMA/RMH can only controls the specific file format but our DRM system will control the COTS Readers, that is, we can do the rights enforcement independent of the file formats.

TrustView developed by [21] is a powerful DRM system. It supports the content protection, security policy classification, using RSA 256-bit AES encryption, and can trace and control the content event if it obtained by some other people. Now it can support for Pro/E, PDF, Office and Web.

SecureAttachment [22] is an on-line service with the Adobe DRM for securing the Distribution Chain of Digital Documents in e-mail. It supports various rights protection policy consisted of rights and conditions. If a document, for example Office Word, is created, the owner can configure its DRM-policy when he wants to email it to other people. If the receiver has rights to view this attached document, then this file will be opened in Acrobat.

Windows Media DRM is a DRM service for securing delivery of audio and/or video content. It was using a combination of elliptic curve cryptography key exchange, DES block cipher, a custom block cipher, RC4 stream cipher and the SHA-1 hashing function in early version. It is designed to be renewable on-line based on the important assumption that it will be cracked [23].

3. Research Method

Our research uses the following architectures and approaches to accomplish this policy controlled DRM system.

3.1. System Architecture

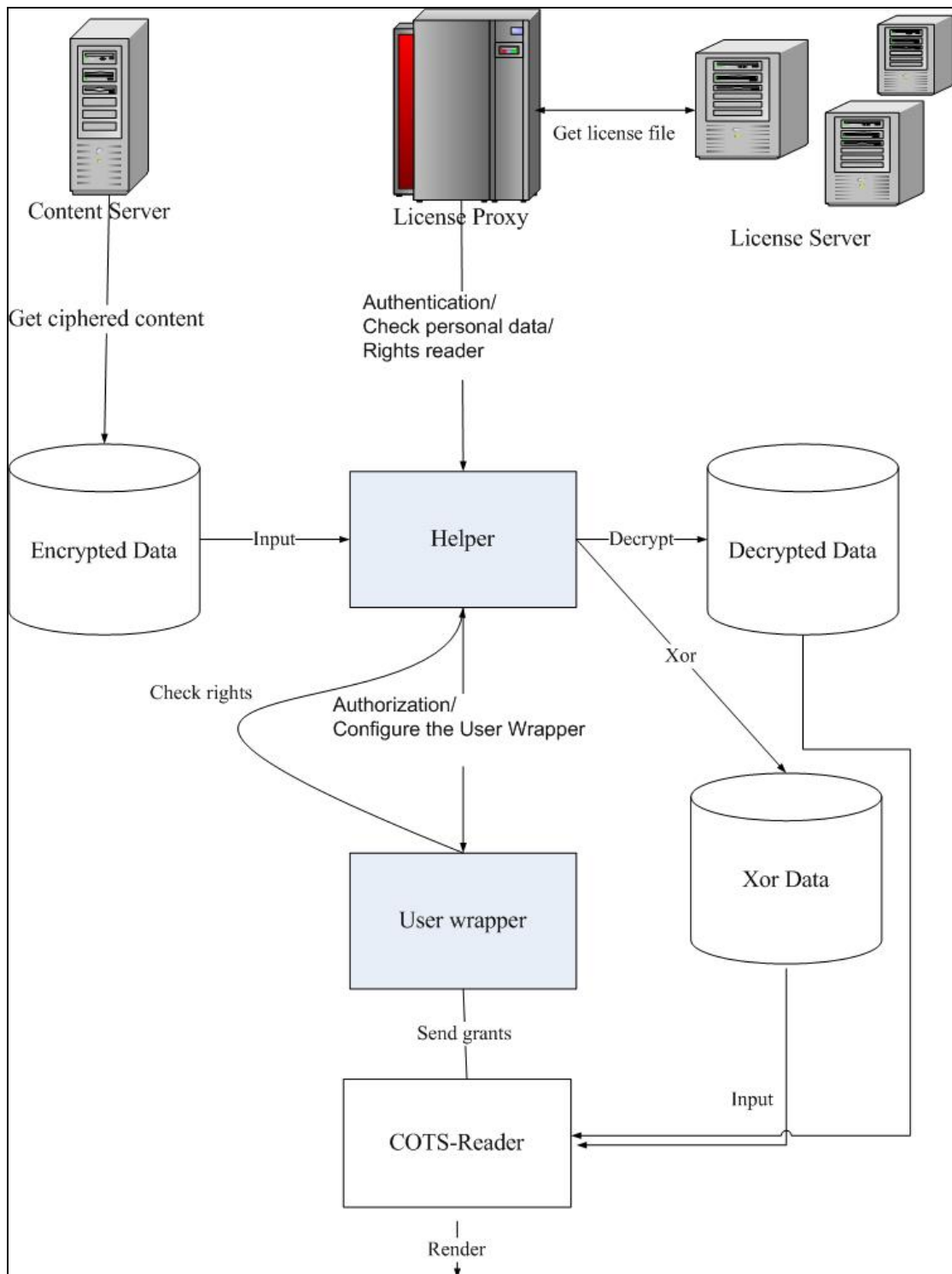


Figure 3-1 System architecture

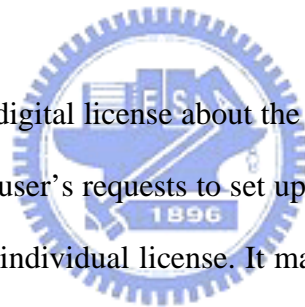
Figure 3-1 shows that what are the components within our architecture and how these components are deployed in our architecture. Following this architecture, we will discuss the functions of each component in our research.

3.1.1. Content Server

This server maintains the digital content including documents, movies, music...etc. It receives the raw content from the content owner (producer) and protects the content by cipher mechanisms. It also processes the requests from clients to download the digital content regardless of the client having license or not.

3.1.2. License Server

This server maintains the digital license about the personal data and relative rights and limitation. It receives the user's requests to set up and configure the personal data and the rights to generate the individual license. It may offer a friendly user interface to serve user, for example, a log-in web page and a web page for configuring.



3.1.3. License Proxy

This component is responsible for reading rights from License Server and processing the authenticating requests, such as, authenticating the rights holder, the COTS-Reader and the DRM actions it wants to execute from Helper. Furthermore, it performs in the role of communicating with other DRM or non-DRM license server by a license translator, it can translate XML-based license to our rights format as long as we know their standards.

3.1.4. Helper

This component is an important part of our research. The job of the Helper layer is to decrypt the encrypted digital contents received from Content Server and to function as a coordinator between content viewers and user wrapper by communicating with the License Proxy through sending the configure data about the user and un-authenticating actions information for authentication. Furthermore, the Helper will make a decision that if there is a requirement to do exclusive-or on the decrypted content or not by judging the content information which is sent from the User Wrapper. Then, it will instruct the User Wrapper to enforce the legal DRM actions.

3.1.5. User Wrapper

This component is the core part of our research. The User wrapper layer is responsible for content protection and rights enforcement. In order to archive these objectives, the first work is to do application layer interception. We do coding operations with certificate by creating built-in control policies in associated code segment for rights enforcement and program built-in methods, that is, we inject with source codes through compiler support or inject from COTS wrapper. For example, we can restrict any operations pertained to 'render', 'save', 'copy', 'paste', 'print'...etc by intercepting the Win32 API functions and then injecting generic codes for enforcing rights. So, we can apply the generic code segments with rights policies to enforce complicated rights, for example, we can add some restriction like render number of times or the date into the rights. The second work of this layer is to adapt for various content viewers. For this objective, we analyze many applications for viewing systematically, including the control flow monitor and the dataflow monitor to realize that which kind of the operations is the DRM operation and the DRM operation will call which Win32 API functions. Then, we can apply the information to operate in coordination with the configuration to

achieve various combinations to let User wrapper fit various COTS readers.

3.1.6. Encrypted, Decrypted and Scrambled Data

Here we will introduce the meaning and usage of these items at our research. Obviously, the encrypted and decrypted data is the production of the ciphering processing and all of them are in the client-side computer. There is an ordinary thing that using cipher mechanism in the DRM system and doing this will let the raw content lie in the client computer based on our implementation restriction (it will state later), but we think that is very dangerous, so we add a new state called Scrambled Data, that is, after decrypting, we will do exclusive-or operation on the raw content to let the raw content lie in the computer is more secret.

3.1.7. COTS Reader

This component represents the existing COTS rendering application. Unlike some DRM system, we don't need to develop a dedicated content viewer, because of that we use the plug-in scheme so that we don't need the source code's support.



3.2. Working Flow

Our working flow scenery is that a user gets the encrypted digital content from the Content Server and sets up or configures his license (rights) at the License Server. When the user does some DRM operations such as print and copy/paste, it will trigger the User Wrapper starting to work. The User Wrapper will detect and intercept any DRM operations immediately and get the information about each DRM operation for authenticating through communicating with the Helper. The procedure of the Helper processing the User Wrapper's requests is that the Helper will request the DRM operation's rights from the License Server. As long as the Helper gets the rights, it

will reserve the rights as a cache rights and the Helper can't change it. The Helper changes this cache rights only when the user changes his license and the Helper will re-request the rights from the License Server. When the Helper gets the reply message from License Server, it will decrypt the encrypted content and pass the rights checking message to the User Wrapper to enforce or deny the DRM operation's requests. Finally, the COTS reader can show the digital content to the user. Figure 3-2 shows the all working flow we describe above.

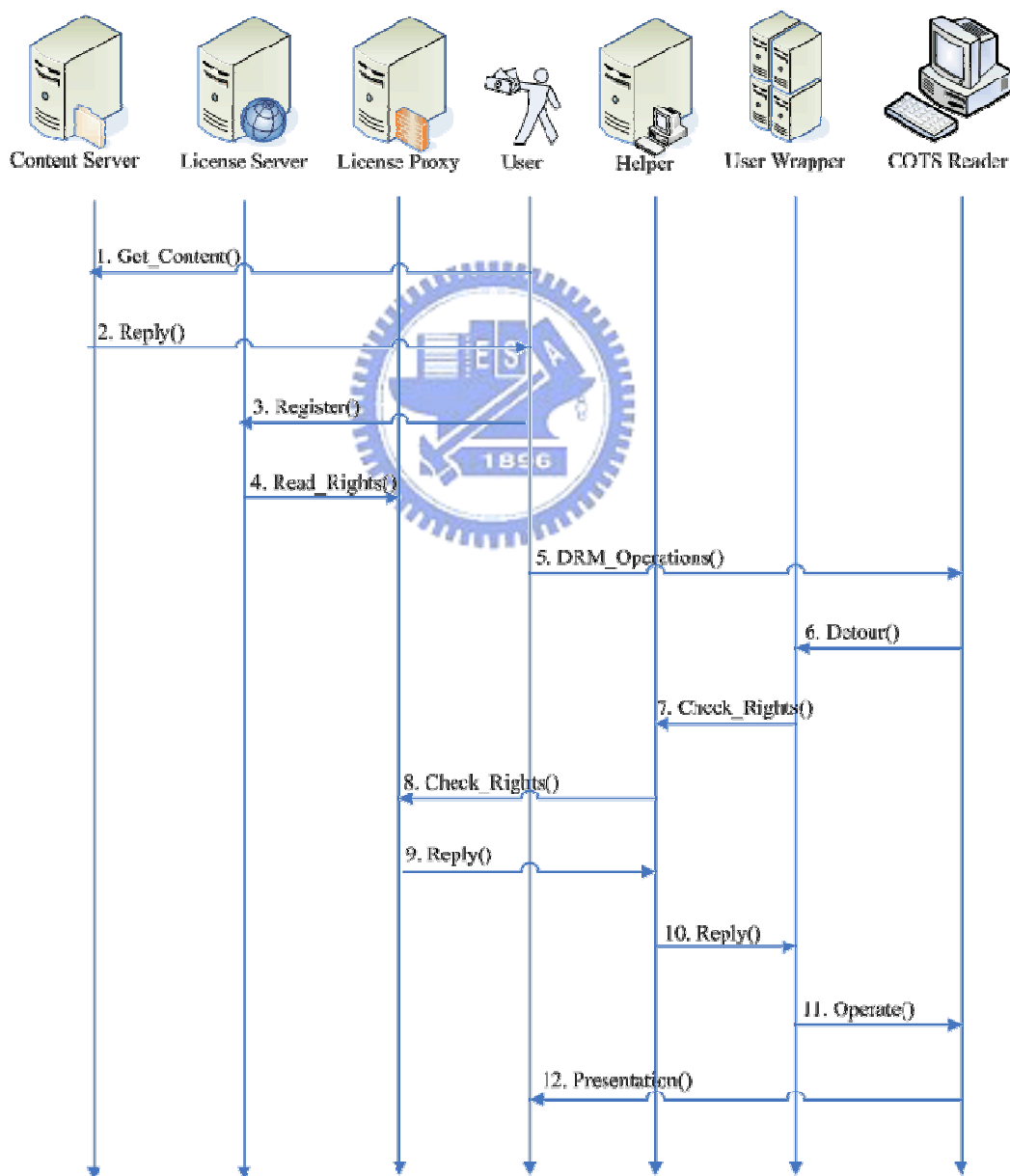


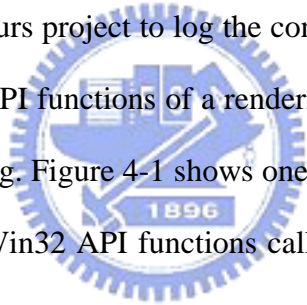
Figure 3-2 The working flow of our DRM architecture

4. Implementation

For our research, we implement four components, Helper, User Wrapper, License Proxy and License Server to complete our architecture and objectives.

4.1. User Wrapper

This User Wrapper is the most important part of our research. It controls the digital content and the COTS Reader directly. This component implementation is based on the detours [14]. Since that we want to apply this Wrapper to any existing COTS Readers that without source code support, we will implement it as the dynamic link library (DLL) file to attach to the existing COTS Readers by the function supported by the detours. First, we make use of the sample program, which is called traceapi, appended to the detours project to log the control flow and the data flow that are composed of the Win32 API functions of a rendering application. Following is an example of the control flow log. Figure 4-1 shows one section of the log file produced by intercepting and logging Win32 API functions called by the IE using the traceapi program.



```
traceapi: 001   SendMessageW(2106e8,443,5,12d46c)
traceapi: 001   GetPropW(2106e8,<C019>)
traceapi: 001   GetPropW(,) -> 16a638
traceapi: 001   IsWindow(2106e8)
traceapi: 001   IsWindow() -> 1
traceapi: 001   GetPropW(2106e8,<C019>)
traceapi: 001   GetPropW(,) -> 16a638
traceapi: 001   GetCurrentThreadId()
traceapi: 001   GetCurrentThreadId() -> 11f8
traceapi: 001   CallWindowProcW(771f433b,2106e8,443,5,12d46c)
traceapi: 001   GetWindowLongA(2106e8,0)
traceapi: 001   GetWindowLongA(,) -> 16a4c0
traceapi: 001   LocalReAlloc(16a860,a8,42)
traceapi: 001   LocalReAlloc(,,) -> 16a920
traceapi: 001   lstrlenW(?(&H))
traceapi: 001   lstrlenW() -> 6
traceapi: 001   LocalAlloc(40,e)
traceapi: 001   LocalAlloc(,) -> 16a860
traceapi: 001   InvalidateRect(2106e8,0,1)
traceapi: 001   InvalidateRect(,,) -> 1
traceapi: 001   CallWindowProcW(,,,,) -> 1
traceapi: 001   IsWindow(2106e8)
traceapi: 001   IsWindow() -> 1
traceapi: 001   SendMessageW(,,,) -> 1
```

Figure 4-1 The logging file of the traceapi

We wish to achieve one of our objectives that the User Wrapper can apply to any COTS Readers. We assume that every COTS Reader will call identical function call when executing an identical DRM operation, so we need to analyze the log file to find out the key function call sequence that every COTS Reader will call when executing a DRM operation and plays a critical role of this DRM operation.

We log the control flow of the same application twice when doing two different DRM operations in order to reduce the searching range to find out the critical function call sequence. Then, we need to understand the meaning of the remaining function call sequence by referring to the MSDN [24], a Microsoft online library, to help and speed us to perform the examination. We have implemented seven base functions about DRM operation, 'render content', 'print', 'copy/cut/paste', 'PrintScreen', 'save', 'drag-drop', 'view source'. We will introduce the implement detail and issues about the functions we implement.



```
If (detect the DRM operation)
{
    Get rights grants from Helper
    If (grants)
    {
        Do the DRM operations (the seven DRM operations)
    }
    Else
    {
        Show the deny message to user
    }
}
```

Figure 4-2 The basic algorithm of our implementation

Figure 4-2 shows the basic algorithm we used when implementing our research. All of our implementation functions are based on this simple algorithm.

4.1.1. Interception Steps

We will introduce how to intercept the software function calls using the detours and the hook hardware function calls.

- Intercept Hardware Event:

```
HHOOK hhkLowLevelKybd=0;
hhkLowLevelKybd = SetWindowsHookEx(WH_KEYBOARD_LL,LowLevelKeyboardProc, hinst,
0);
UnhookWindowsHookEx(hhkLowLevelKybd);

LRESULT CALLBACK LowLevelKeyboardProc(int nCode, WPARAM wParam, LPARAM lParam)
{
    BOOL fEatKeystroke = FALSE;
    //do something here
    return(fEatKeystroke ? 1 : CallNextHookEx(NULL, nCode, wParam, lParam));
}
```

We call the function call, `SetWindowsHookEx()`, to install an application-defined hook procedure into the hook chain. It can monitor the events of certain types. The above example is to install the low level keyboard hook to monitor the events produced by the keyboard and an example of the application-defined hook procedure to process the events.

- Intercept Software Function:

First, we must put the function call that we want to intercept into the trampoline. And rename it as `Real_function()`.


```
DETOUR_TRAMPOLINE(HRESULT __stdcall Real_DoDragDrop(IDataObject * a0,
IDropSource * a1, DWORD a2, DWORD * a3), DoDragDrop);
```

The above sample code is to put the function call, DoDragDrop(), into the trampoline and the return type of the trampoline is HRESULT.

Then, we declare a function to be enforced when the detours detect that the called function call is in the trampoline.

```
DetourFunctionWithTrampoline((PBYTE)Real_DoDragDrop, (PBYTE)Mine_DoDragDrop);
DetourRemove((PBYTE)Real_DoDragDrop, (PBYTE)Mine_DoDragDrop);
```

The above sample code is to declare the function, Mine_DoDragDrop, that will be executed when the detours detect the DoDragDrop() function.

Finally, we can program any thing we want to do in the Mine_function() and we can make use of the parameters in the Real_function().

```
HRESULT Mine_DoDragDropA(IDataObject * a0, IDropSource * a1, DWORD a2, DWORD *
a3)
{
    //do something here
    return DRAGDROP_S_CANCEL;
}
```

4.1.2. Function Interception

4.1.2.1. Render

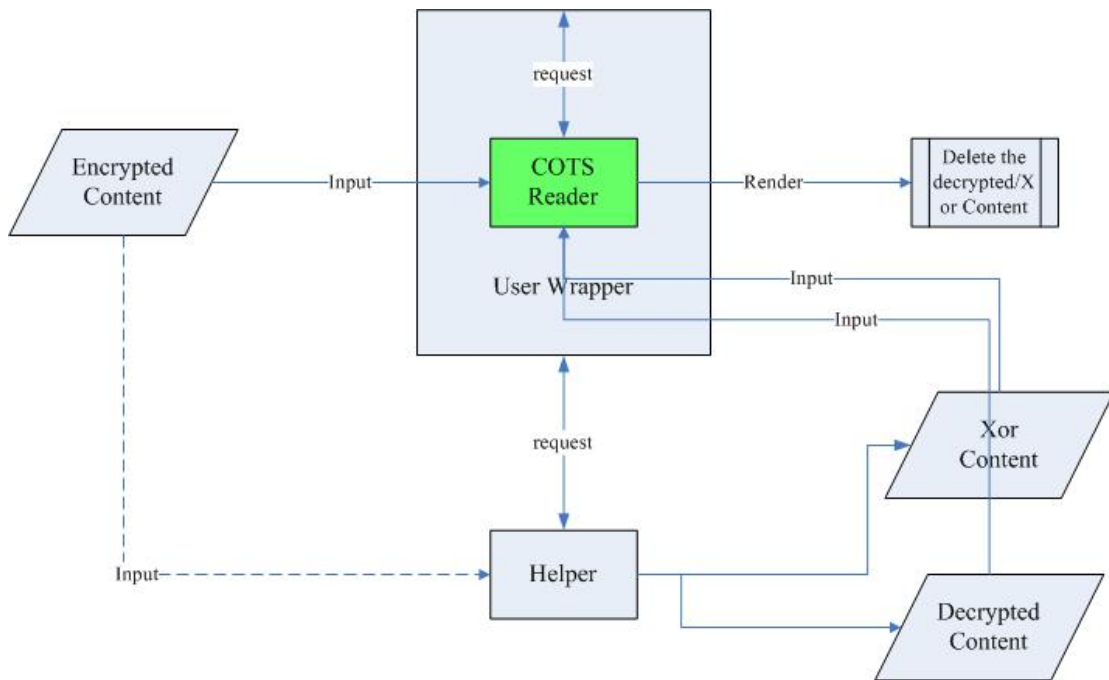


Figure 4-3. The control flow of rendering the protected content.

Figure 4-3 show that the ideas and the algorithm we use when implementing the ‘render’ functionality and figure 8 shows this concept. We intercept the system call, `createfile()`, because that windows will call `createfile()` when rendering contents.

Since we make use of the existing COTS Reader, we need to consider the normal use of the COTS Reader to render the digital content. When detecting the action that the COTS Reader will do is a DRM operation, we will send a message to Helper to get the grants. If the rights enforcement requests are allowed, then we will check the digital content that will be rendered is encrypted or not. If yes, then we send a message to the Helper to ask it to decrypt this content, else we just render it. If the reply message is `Xor_content`, it denotes that the Helper have done exclusive-or to the decrypted content in order to protect the digital content which is in the client-side computer. When we want to render it, we will need to do exclusive-or to this content again to recover its data, so we intercept the function call, `ReadFile()`, that read data from a file to get the exclusive-or data and process it. After completing rendering, the

wrapper will delete the decrypted or exclusive-or content automatically when the handle to the content is no more used by replacing the file flag parameter in CreateFile function with FILE_FLAG_DELETE_ON_CLOSE.

```
If (detect the Render operation)
{
    Get rights grants from Helper
    If (grants)
    {
        If (encrypted_content)
        {
            Send a request to Helper to ask to decrypted the content
            While (!receive_message) //wait for receiving reply message
            If (receive_message == Xor_content)
            {
                Intercept the Readfile() to Xor the Xor_content in the
                memory
            }
            Render the decrypted_content
        }
        Else
        {
            Render this content
        }
    }
    Else
    {
        Show the deny message to user
        Return 0
    }
}
```

Figure 4-4 The algorithm of implementation the 'render' functionality

4.1.2.2. Print

```
    If (detect the Print operation)
    {
        Get rights grants from Helper
        If (grants)
        {
            Return Real_StartDoc()
        }
        Else
        {
            Show the deny message to user and return 0
        }
    }
```

Figure 4-5 The algorithm of implementing the ‘print’ functionality

StartDoc(), this function call starts a print job. The print job will start behind calling this function call, so we intercept this function call in order to judge that whether let the print procedure work or not by getting rights from the Helper. Figure 4-5 shows the algorithm we used here.

4.1.2.3. Copy, Cut, Paste

Here we intercept the function call, openclipboard(), because that these DRM operations are using the clipboard to process the requests. They do the jobs about getting the data in the clipboard or storing it into the clipboard. We intercept this function in order to achieve the goal that disable or enable these DRM operations and the judge rules are the same as the 1.2.

4.1.2.4. Print Screen (in the keyboard)

```

    If (detect the PrintScreen operation)
    {
        Get rights grants from Helper
        If (grants)
        {
            Return 1
        }
        Else
        {
            OpenClipboard(NULL)
            EmptyClipboard()
            CloseClipboard()
            Show the deny message to user and return 0
            Return CallNextHookEx()
        }
    }
}

```

Figure 4-6 The algorithm of implementing the 'PrtScrn' function key

This part is also relative to the clipboard. If the DRM operation's request is denied, we call `OpenClipboard()` to open the clipboard preventing other applications from modifying the clipboard and then we clean the handle to the data in the clipboard. Finally, we close the clipboard by calling `CloseClipboard()` to let other application to access the clipboard. Figure 4-6 shows the basic ideas when implementing this functionality.

4.1.2.5. Save

We think that the function, save as a new file, is always need a dialog box to communicate with the user, so we intercept the dialog box to cut down the control flow of the 'save as' function. We intercept the function call, `DialogBoxIndirectParam()`, because of all of the dialog box is relative to this function.

Return zero indicates that this DRM operation's request is denied.

4.1.3. Security Interception

4.1.3.1. Drag-Drop

All the message passing job of the drag-drop actions is done by the function, DoDragDrop(), and analyze the return value of DoDragDrop() to judge that what actions are request, so we intercept this function call and return DRAGDROP_S_CANCEL, the cancel message in the DoDragDrop(), to refuse to do the operation to protect the digital content.

4.1.3.2. View Source Code (in web page)

The function of the 'view source' will get the absolute path in the input text of the network position to read the source file. But in our architecture, the Helper will put the decrypted content with exclusive-or effect into a secret place in the computer and tell the User Wrapper the place to get the content to process. At this time, the absolute path in the input text of the network position still point to the ciphered file such that the 'view source' function will read the encrypted file to achieve the protection purpose.

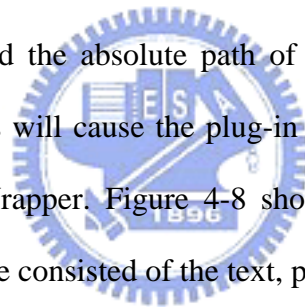
Finally, we compile this program to a dynamic link library file and use the sample program which is called 'withdll' that creates a process and inject a named DLL into the new process to inject this .DLL file into the target COTS Reader.

4.2. Helper

The jobs of this component are content decryption and rights checking. We use C# to implement here because of the convenient function call about decryption and network. The cipher mechanism we choose here is the Data Encryption Standard

(DES). The inter-process communication (IPC) mechanism we use is the TCP-IP standard. Figure 4-7 shows our ideas when implementing the Helper.

The Helper will classify the messages that receive from the User Wrapper into two cases. If the request is 'get_rights', then the Helper will check the rights_cache to see that if the right is in it or not. If the right is not in the cache, then the Helper will update the rights_cache by communicating with the License Proxy. If the request is the 'decrypt', then the Helper will get the decrypted key from License Proxy and the content by the absolute path receiving from the User Wrapper. If the content is prepared for the plug-in, for example, when IE wants to render the media content like .mp3 and .mid file, IE will call the Windows Media Player to play this content, then we don't do exclusive-or on the content because that if we do exclusive-or on the content and then IE will send the absolute path of the exclusive-or content to the plug-in for rendering and this will cause the plug-in render error. Finally, return the content place to the User Wrapper. Figure 4-8 shows that what IE will do when playing the compound web file consisted of the text, picture and media files.



```

Listen (ip, port)
  While (true)
  {
    While (!get message) //wait for message coming
    Switch (message)
    {
      Case get_rights:
        If (rights_cache contain this right)
        {
          Return request_accept
        }
        Else
        {
          Send the message to the License Proxy
          If (reply_message)
          {
            Add this right into the rights_cache
          }
          Else
          {
            Return request_fail
          }
        }
      }
      Case decrypt:
        Get the content_place from User Wrapper
        Get the decrypted key from License Proxy
        Decrypted the content
        If (!plug-in content)
        {
          Xor the content
        }
        Content_place = (Save it at another secret place)
        Return Content_place
    }
  }
}

```

Figure 4-7 The algorithm of implementing the Helper

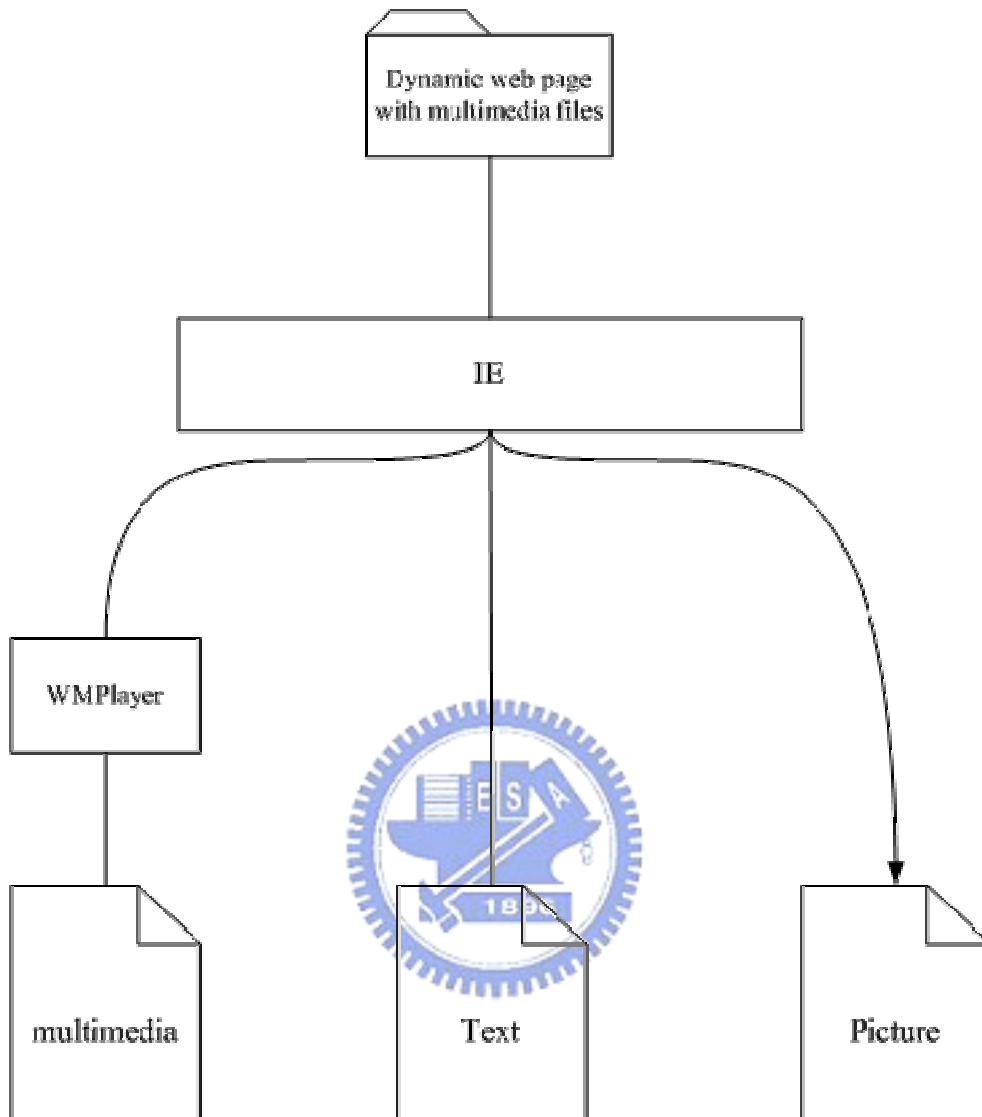


Figure 4-8 The situation that IE plays the multimedia files

4.3. License Server

This Server will provide many web pages for use log in, set up and configure his license. Then, it will transfer the user's setup into the XML-based license file that includes user id, password, COTS Reader (we have implemented), rights and restrictions. For example:

```

<?xml version="1.0" encoding="UTF-8"?>
<r:license xmlns:con="rights_conditions"
xmlns:uip="use_id_password" xmlns:mx="rights_enforcement"
xmlns:o="render_application" xmlns:r="rights">
  <r:Personal>
    <uip:id>JI</uip:id>
    <uip:pwd>1234</uip:pwd>
  <r:Rights>
    <mx:render>
      <con:numbers>10</con:numbers>
    </mx:render>
    <mx:ccp>
      <con:numbers>2</con:numbers>
    </mx:ccp>
    <mx:print>
      <con:numbers>4</con:numbers>
    </mx:print>
  </r:Rights>
  <r:RenderAP>
    <o:IE/>
    <o:Firefox/>
    <o:WMPlayer/>
    <o:Adobe/>
  </r:RenderAP>
</r:license>

```

Figure 4-9 Simple license file

Figure 4-9 shows the sample license file. This license file is denoted that the user id is 'JI' and his password is '1234', and the rights he can enforce are that he can render this content ten times, do 'copy, cut, paste' operations two times and print it four times at IE, Firefox, Windows Media Player and Adobe. Finally, it will trigger the License Proxy to read the configured license file by sending message to it when the license created or/and changed every time. We implement this server using

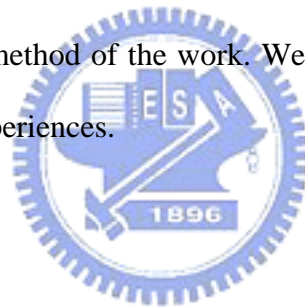
JavaScript and Java Servlet on Tomcat based on [3].

4.4. License Proxy

In this part, we implement a program as a license handler. There are two jobs of this program. One is that it will read the license file triggered by the License Server and put it into a 'rights_cache'. The Proxy will change the content in this cache only when the user changes his license and it will announce the Helper to update the rights synchronization. Another is that it is responsible for license transformation.

4.5. Experience and Discussions

When implement this DRM Wrapper, we encounter some issues that are not easy to overcome because of our method of the work. We address these issues below and bring up our solutions and experiences.



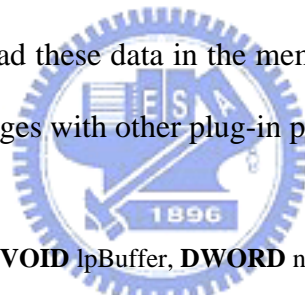
4.5.1. Rendering

Because of the method of the work that inject a DLL into the process, we need to know that the working flow of the process in order to make the execution of the process normal when we modify the parameters of some function call and return then to the process. Unfortunately, we can't understand the work flow of a process because that the COTS Reader is commercial without source code and won't release its detail implementation.

When implementing the 'Render' functionality, we decide to intercept the function, CreateFile(), and this function can creates or opens a file, directory, physical disk, volume, console buffer, tape drive, communications resources, mailslot, or named pipe, but we must fit in with the control flow of the process, so we must conform the object type that the function original opens.

With our example of IE rendering web pages, the object type we need is file, so there are many restrictions on the decision of the protection methods. When deciding the cipher algorithm, the streaming cipher is more secret than the block cipher because that there are no entity files existing in the computer, but we can not choose it because of the requirement we mentioned that 'the object type we need is file'. So we use block cipher and store an entity file into a secret place in the computer.

But we think this method is not secret enough, we add a simple method on the file to increase the level of secret. We do exclusive-or on this entity file after decrypting because we need to fit in with the working flow of IE, that is, the object type and size are must identical with them after doing the exclusive-or method. Here we encounter two issues that how to do exclusive-or on the data in the memory because that the IE doesn't read these data in the memory at a time and the rendering problems with the dynamic pages with other plug-in programs.



BOOL ReadFile(Handle hFile, LPVOID lpBuffer, DWORD nNumberOfBytesToRead, LPDWORD lpNumberOfBytesRead, LPOVERLAPPED lpOverlapped)

The ReadFile function will read the data in the memory that pointed by hFile witch gets from the return value of the CreateFile() into lpBuffer. IE calls this function to read the content into memory after a lot of setting steps after calling Createfile(). Our solution is to intercept this function and does exclusive-or on the lpBuffer, but the problem is how to know that what data are we want. We store and assign the return value of the CreateFile() to a global variable because that there are may many threads executing and we just have a shared source code and this situation will cause that a handle in a thread will be covered by the handle in another thread if we don't store it as a global type and check the hFile of the ReadFile function with it

to see that if they are equal or not to overcome this problem. Figure 4-10 shows this problem. If they are equal, then we do exclusive-or on the lpBuffer, else we don't.

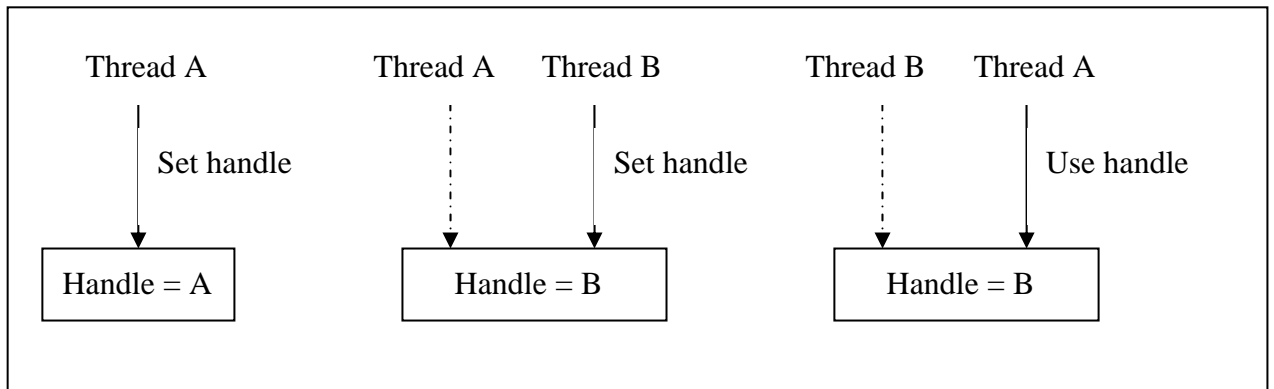


Figure 4-10 The thread problem

We have mentioned the situation in 2 that the rendering problem of IE rendering a dynamic page using JavaScript with multimedia contents. We don't propose a useful solution to solve it. We just make a decision by judging that if this file is need a plug-in application to render or not. Now we only implement the decision rule of the .mid and the .mp3 files in the Helper.

4.5.2. Dialog box

We have mentioned in 1.5 that we intercept the DialogBoxIndirectParam function. This way of making this functionality is not good because that all functionalities relative to dialog box such as 'save as' and 'open old file' will be locked. We have no idea to solve this situation except that we take some redeeming actions in IE. We observe that there is a different part between the 'save as' and 'open old file' in IE. The 'save as' function call DialogBoxIndirectParam() immediately, but the 'open old file' will call DialogBoxParam() before calling DialogBoxIndirectParam(). We can

make use of this divergence to distinguish them, but this is a special case. When working with other applications like Adobe or Windows Media Player, this method is still not working.

4.5.3. Clipboard

We encounter some issues when we implement the 'print screen' function. As we mentioned in 1.4, the User Wrapper opens the clipboard preventing other applications from modifying the data in the clipboard, empties and close the clipboard to release the use right of the clipboard. The algorithm we mentioned in 1.4 seems workable, but there will be some problems happened when work with other applications that will use the clipboard such as the Microsoft Office Word. Figure 4-11 is an example.

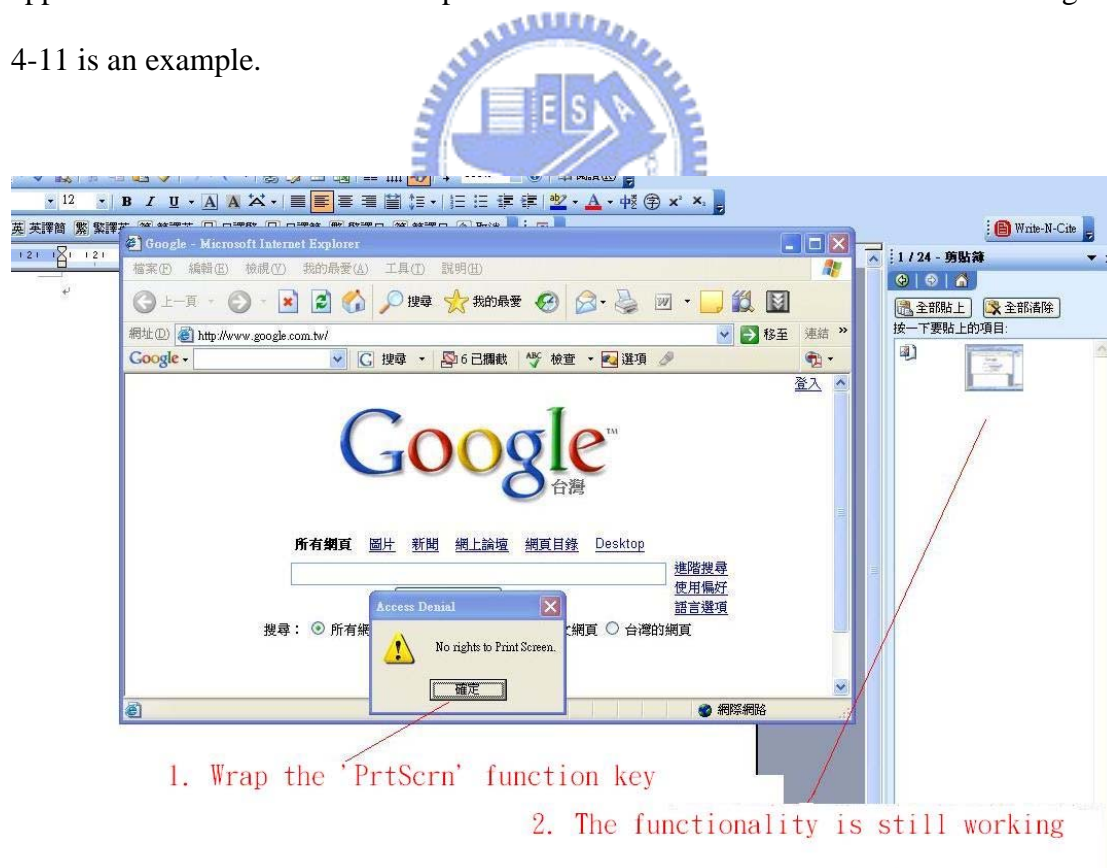


Figure 4-11 The clipboard problem in our work

When we test this functionality together with the Microsoft Office Word, the intercept functionality sometimes works successfully but sometimes is not. We believe that this is a timing problem, that is, the system puts the picture into the clipboard after executing our program. We even try to intercept the key down and key up separately in order to fit in with the time difference, but still not work. Finally, we implement successful by disabling the warning message box we prompt because that we have observed that IE will call the warning message box many times since IE will re-call the same function many times when calling fail and the delay time needed by the warning message box is about five seconds (depend on the speed of the testing computer). We think that this is a very possible reason causing the time difference.

4.5.4. Other Experiences and Discussions

There are some limitations in our work. We can't judge that the rendering content is protected or not in wrapped functions just in time because of the restrictions of applying the detours. Although we can access and change the parameters in the target functions, but we are still restricted by the information provided by the parameters. For example, we can't get the URL from the parameters in ReadFile(), but we only can get the handle to the createfile() and the URL we needed is one of the parameters in createfile(). We can't intercept the 'view source' functionality in FireFox. We suppose that the reason is that FireFox reads the decrypted content into memory and it will get the content in memory when doing the functionality, view source, but IE will send URL referring to the encrypted file to notepad for viewing source. Figure 4-12 shows this concept.

We also find out some issues when OpenDReaMS Wrapper works with the Acrobat 7.0 professional but it will work well with Acrobat 7.0 Reader. The 'drag-drop' function can not work well in Acrobat professional. It will crash it, but it

can work in IE. If the non-wrapped Acrobat is opened first, and then we will fail to wrap another one whether professional version or reader version, but IE does. We guess the reason is that the Acrobat will work as one daemon and IE works as a single instance, so there is one Acrobat daemon at a time and there can be many IE existing at the same time. This may need further work to understand it. Moreover, if we wrap IE, the any other IE opened by this wrapped IE are still under control of our User Wrapper.

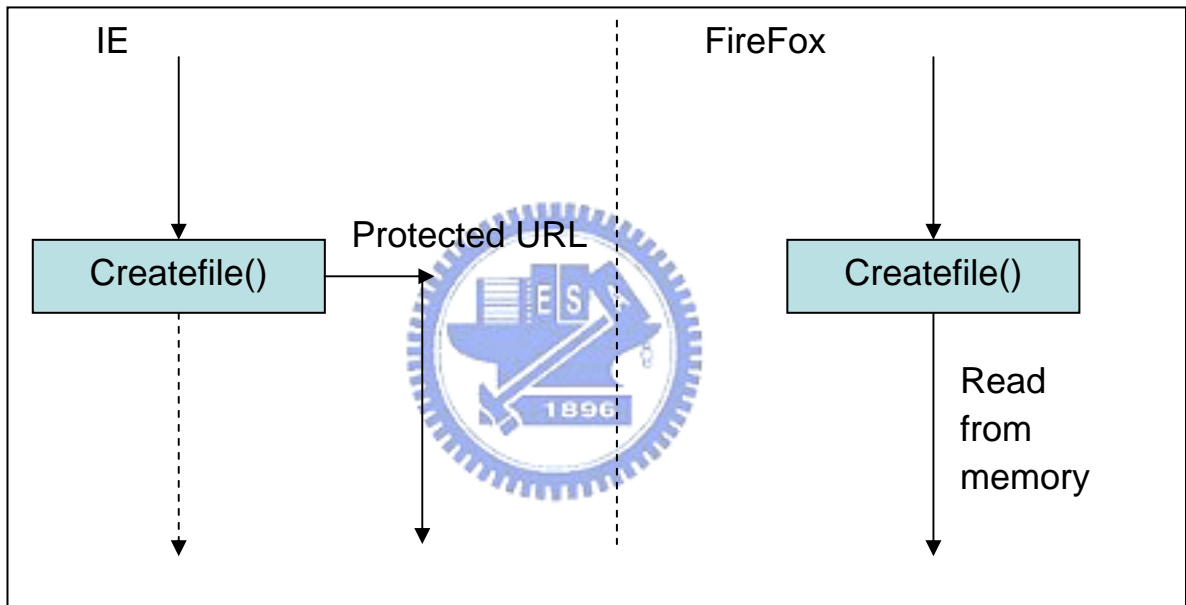


Figure 4-12 view source working flow in IE and firefox

5. Results and Assessment

We have implemented many functions about the DRM protection system. We can protect our content from render, print, copy, cut, paste with conditions such as the number of using times by our work. Moreover, we have considered some security problems. We can disable the view source functionality in IE preventing the rendering content from stealing from the source code. When rendering the content, the exposed content in the client side is also protected by simple exclusive-or method. After rendering, the User Wrapper will delete the content just in time for no content leaving in the end-user's computer even if the temporary files. Furthermore, the most different thing between our work and other existing DRM systems is that our work can apply to many kinds of COTS Readers no matter what kinds of content type and render applications such as IE explorer, Windows Media Player, Acrobat...etc. For proving our key idea, we also design and implement a complete DRM control flow.

Following, we will apply our system to the Web browser, IE, as an example to show our experiments.

Firstly, we need connect to the license server to configure our license (rights).

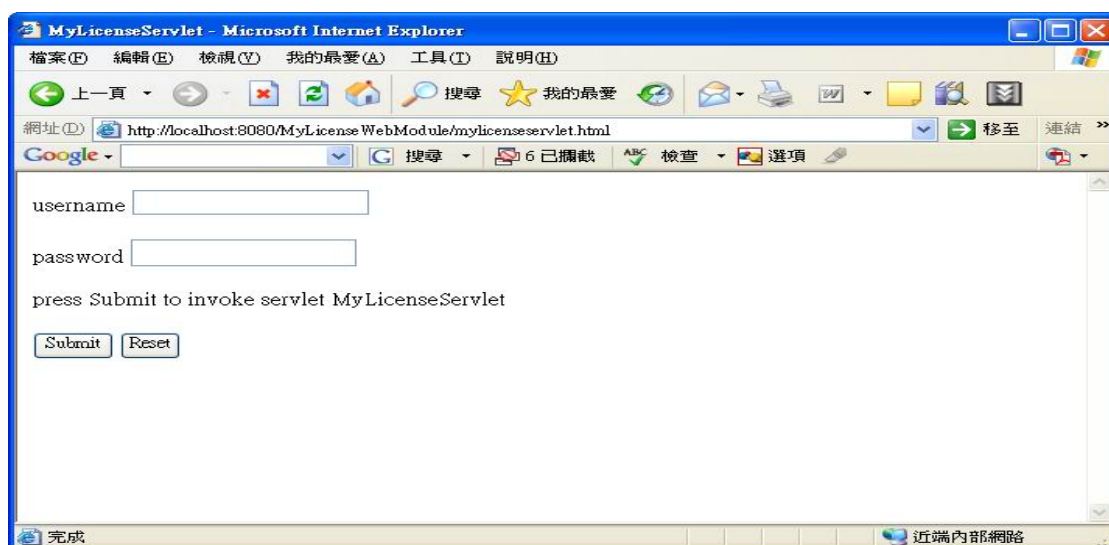


Figure 5-1 Log-in web page in license server

After logging, we can configure the individual license including the COTS Reader that we will use to render, rights and the cooperating conditions. For example, we can restrict the agreement numbers of executing each function. Figure 5-2 shows these attributes.

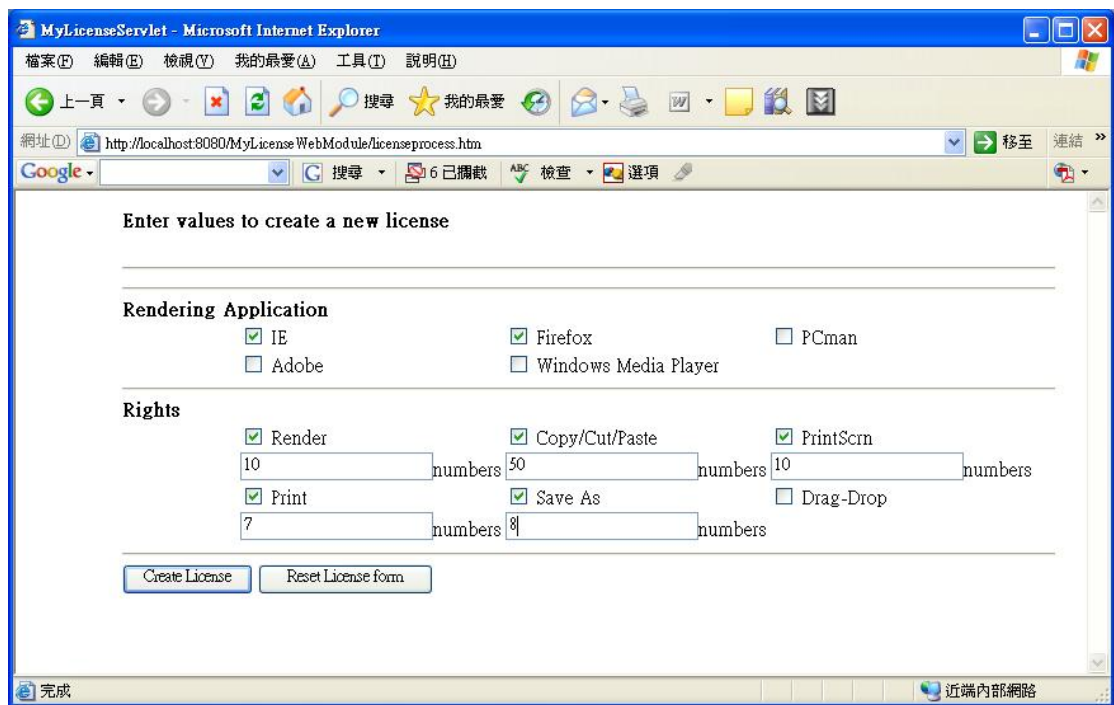


Figure 5-2 The license page in license server

Then, we can use the User Wrapper to protect the target content as well as the normal content depend on your policies, that is, we can control the whole activities of the application and the protection policy design is freedom. For convenient to demonstrate, we integrate the User Wrapper and the Helper into a simple user-interface application.

We start to use the protected content after input the user id and password for authenticating. Then, we can arbitrary choose the COTS Reader that you want to use.

Figure 5-3 shows the form to open COTS Readers arbitrary. There are some default icons in common use and other two ways to open the COTS Reader. We can type the absolute path in the input text and click the Browse button to choose the executive file of the COTS Reader.

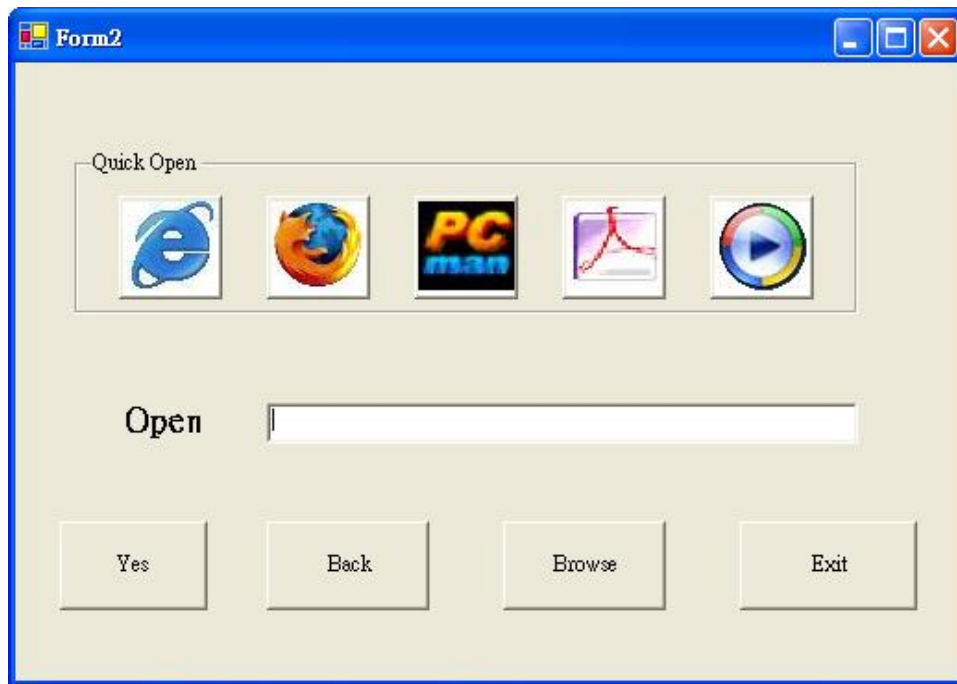


Figure 5-3 The page to open the COTS Reader.

When we open the protected digital content using the wrapped IE, the User Wrapper will check the rights just in time. Figure 5-4 shows that no rights to render the protected digital content and figure 5-5 shows that we protect the digital content from viewing source functionality.

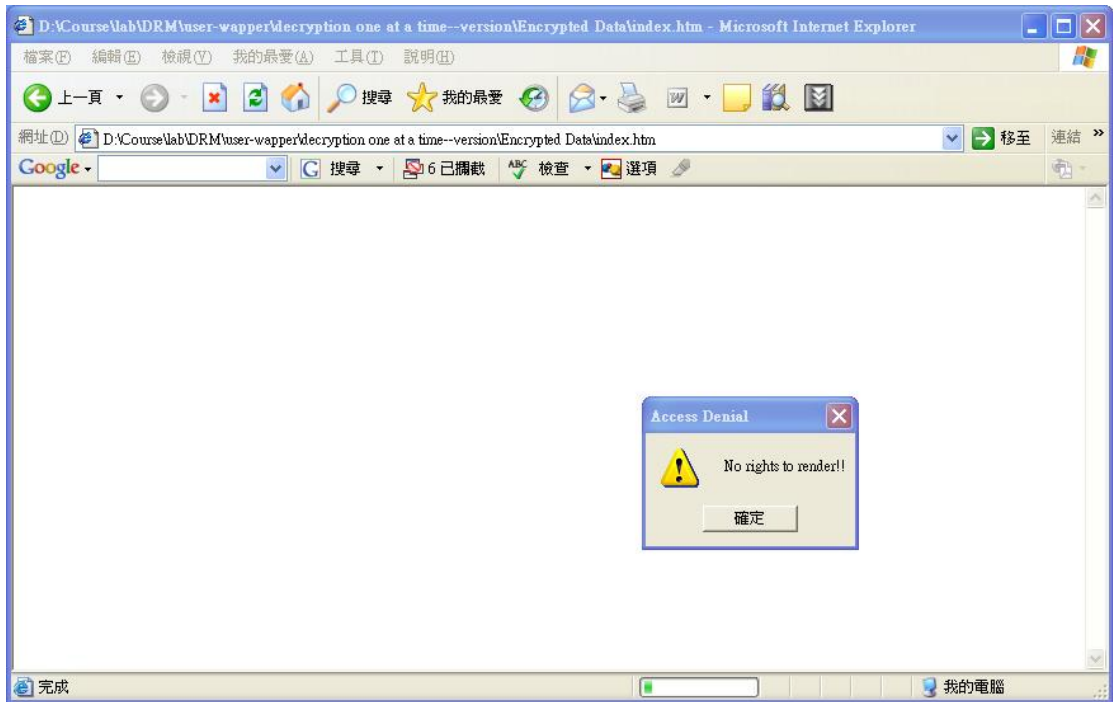


Figure 5-4 Denying rendering

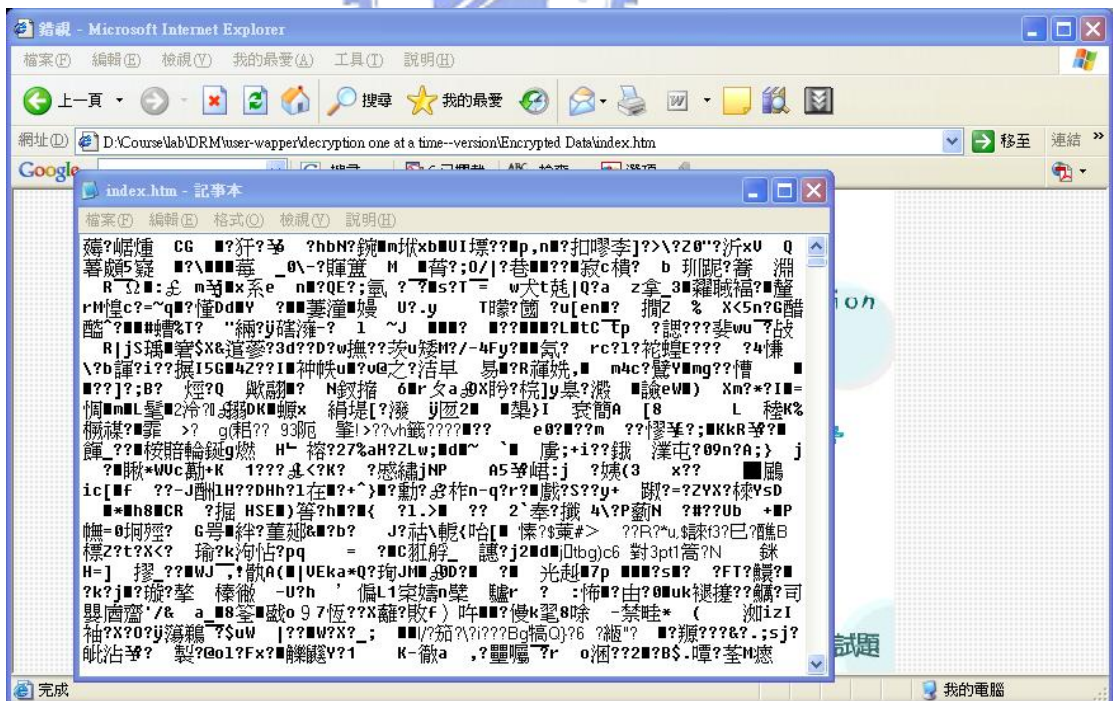


Figure 5-5 Protection of viewing source

DRM Characteristics	OpenDReaMS wrapper	OpenIPMP	Media-S (demo)	Adobe DRM
Price	Free	Free	Free	Unknown
Install requirement	No	Yes	Yes	Yes
Open source	Yes	Yes	Yes	No
Rights enforcement	Yes*	Play	Play	Yes**
Constraint enforcement	Number of usage times	Date time	Date time	Yes***
Mouse protection	Yes****	No	N/A	Unknown
Exposed content protection	Yes	N/A	No	Yes
Support content type	N/A	Mpeg4	.ogg	.pdf
Apply scope	COTS Reader	Dedicated Reader	Dedicated Reader	Dedicated Reader

*** copy, cut, paste, print, PrintScreen, save as, open old file, view source when opening encrypted content using IE**

**** print, copy/paste, save, file copy**

***** data time, expiry after x hours of reading, allow x sections copied/printed every y days**

****** drag/drop, functions related to * in “right click”**

Table 5-1 Comparison between our work and other DRM systems

This table shows that the main differences between OpenIPMP, Media-S, Adobe DRM and our work. Our system is free, open-source and doesn't need to do installation steps. We just need to execute the executive file. Obviously, the best advantage of our research is that we don't need to consider the content type that we

want to protect and we can protect it as long as we can wrap the COTS Reader, that is,
we make COTS Readers to be Dedicated Readers as long as we can wrap them.

	IE	Firefox	PCman	Adobe 7.0 Reader	DjVuViewer	WMPlayer	RealPlayer	Winamp
Render	✓	✓	✓	✓	✓	✓	✓	✓
Print	✓	✓	✓	✓	✓	N/A	N/A	N/A
Copy/ Cut/ Paste	✓	✓	✓	✓	N/A	N/A	N/A	N/A
Save as	✓	Δ	✓	✓	✓	✓	N/A	N/A
PrintScreen	✓	✓	✓	✓	✓	✓	✓	✓
Drag-Drop	✓	✓	✓	N/A	N/A	N/A	N/A	N/A
View source	✓	×	✓	N/A	N/A	N/A	N/A	N/A

Table 5-2 Experimental COTS Readers



6. Conclusion and Future Work

In this thesis, we propose a digital content protection scheme in the client side of the DRM system. We analyze and intercept the generic system call of the rendering applications at execution time to protect the digital content with various policies. We apply the protection mechanism to different COTS readers without considering the content format on the platform of proprietary Microsoft Windows. We also implement a complete DRM flow to prove that we can protect the digital content with various rights and restrictions by the policy configuration. Our contribution lies in that with the proposed mechanism, the end-user needs only for one installation of the DRM wrapper for all rendering software. This can eliminate the cost and promote the use of DRM system. With open-source distribution, it is helpful to advance the spread of the DRM system.

Beside the issues and problems discussed in the thesis, the shortcoming of current implementation is the need to manually analyze the generic system calls. We log the system call sequence and the caller-callee relation between them, but we can't exactly figure out which functions of the software will induce a certain series of system call sequence. It is a lengthy process to find out the relations and not an easy task to automatically analyze the semantic relation between functions and system calls. Moreover, it is hard to test if the parameters in the system call fit for the policy we need, even if the relations have been clarified. Some COTS Readers such as IE may have different architectures (for example, calling different system calls when doing the same functionalities) across the versions. If we can achieve the goal of automatic analysis, we can develop a new wrapper with associated configuration to fit the COTS Reader of the new version.

References

- [1] WEKIPEDIA, "http://en.wikipedia.org/wiki/Digital_rights_management,"
- [2] Bill Rosenblatt, Bill Trippe and Stephen Mooney, *DRM Building Blocks: Protecting and Tracking Content*. 2002,
- [3] J. Polo, J. Prados and J. Delgado, "Interoperability between ODRL and MPEG-21 REL." in *ODRL Workshop*, 2004, pp. 65-76 ee = {<http://odr.net/workshop2004/paper/odr-poo-paper.pdf>.
- [4] F. Gerard, J. Tom and S. Vishy, "Project DReaM-an architectural overview," September, 2005.
- [5] C. Serrão and D. c. = { . Neves, "Open SDRM -“ An open and secure digital rights management solution," *IADIS'03*, June. 2003.
- [6] C. N. Chong, R. van Buuren, P. H. Hartel and G. E. -. Kleinhuis, *Security Attributes Based Digital Rights Management*. , vol. 2515, 2002, pp. 339-352.
- [7] ". Mourad, J. Munson, T. Nadeem, G. Pacifici, M. Pistoia and A. Youssef", ""WebGuard: A system for web content protection"," in "*{WWW} Posters*", 2001,
- [8] B. C. Popescu, B. Crispo, A. S. Tanenbaum and F. L. A. J. Kamperman, "A DRM security architecture for home networks," in *DRM '04: Proceedings of the 4th ACM Workshop on Digital Rights Management*, 2004, pp. 1-10.
- [9] P. A. Jamkhedkar and G. L. Heileman, "DRM as a layered system," in *DRM '04: Proceedings of the 4th ACM Workshop on Digital Rights Management*, 2004, pp. 11-21.
- [10] G. L. Heileman and P. A. Jamkhedkar, "DRM interoperability analysis from the perspective of a layered framework," in *DRM '05: Proceedings of the 5th ACM Workshop on Digital Rights Management*, 2005, pp. 17-26.
- [11] T. Nicolakis, C. E. Pizano, B. Prumo and M. Webb, "Protecting digital archives at the greek orthodox archdiocese of america," in *DRM '03: Proceedings of the 3rd ACM Workshop on Digital Rights Management*, 2003, pp. 13-26.
- [12] ". Fraser, L. Badger and M. Feldman", ""Hardening COTS software with generic software wrappers"," in "*{IEEE} Symposium on Security and Privacy*", 1999, pp. 16.
- [13] ". Ko, T. Fraser, L. Badger and D. Kilpatrick", ""Detecting and countering system intrusions using software wrappers"," in "*{USENIX} Security Symposium*", 2000, pp. 17.
- [14] G. Hunt and D. Brubacher, "Detours: Binary Interception of Win32 Functions," *In Proceedings of the 3rd USENIX Windows NT Symposium*, July. 1999.
- [15] Rightscom Ltd, "The MPEG-21 rights expression language. A white paper," 2003.
- [16] I. R, *Open Digital Rights Language (ODRL) Version 1.1*. 2002,
- [17] O. Vestavik, "REAP: A system for rights management in digital libraries." in *ODRL Workshop*, 2004, pp. 79-85 ee = {<http://odr.net/workshop2004/paper/odr-estak-paper.pdf>.
- [18] Dominik Dahlem, Ivana Dusparic and Jim Dowling, "A Pervasive application rights Management Architecture (PARMA) based on ODRL," *Proceedings of the First ODRL International Workshop*, April 22-23. 2004.
- [19] C. Microsoft, "Technical Overview of Microsoft Windows Rights Management in the Enterprise,"

June. 2003.

[20] Microsoft Corporation, "Rights Management Add-on for Microsoft Internet Explorer,"
September. 2003.

[21] I. TrustView. [Http://www.trustview.com.tw/](http://www.trustview.com.tw/). Available: <http://www.trustview.com.tw/>

[22] D Soft, "<http://www.secureattachment.com/index.htm>,"

[23] Anonymous "http://en.wikipedia.org/wiki/Windows_Media_DRM,"

[24] Microsoft coporatoin, "MSDN,<http://msdn.microsoft.com/library/>,"

