An Internet File System for Random Access Protected Data

Ahmad Syahir* Yasushi SHINJO*,† Kozo ITANO*,† Akira SATO* Hisashi NAKAI*

*University of Tsukuba
†Japan Science and Technology Agency

1. Introduction

Internet is ubiquitous on these days and has been essential for collaborating people. Therefore, sharing data on the Internet is really common for easy access. In this paper, we deal with distributing protected data through the Internet. For example, in the sports coaching area, video data is shared among scattered organizations with annotations [1]. Typically, to distribute protected video data, Digital Rights Management (DRM) is often used. Representative DRM products include Windows Media DRM, Helix DRM, FairPlay DRM, and DReaM.

Existing DRM mechanisms for video have a problem in the random access capability. Random access is really important for video seeking. This problem is inherent for these existing DRM mechanisms because they are optimized for major users who watch video sequentially. In concrete, these existing DRM mechanisms use a large buffer and often download entire video data in advance. This feature is nice to mitigate jitters, high latency, and slow throughput. However, this feature is not suitable for applications that need random access.

In this paper, we propose a new data protection mechanism to access protected data through the Internet. In this mechanism, we reuse existing media players for the Windows operating system (OS) without any modifications to applications and the OS. Instead, we extend the OS in the Virtual File System (VFS) layer. To simplify the implementation of a VFS module, we use a framework for user-level file systems called Dokan [2]. Encrypted protected remote files are accessed thought HTTP in a VFS module. The VFS module decrypts protected data in the remote files, and gives to the OS. The OS passes the decrypted data to applications. The applications can access the remote files as if they are local files. Hence, the applications can open the remote files with a random access capability as if they are local files.

In this mechanism, we need to protect the data from malicious programs that run in the same operating system as the legitimate applications. We solve this problem by using secure channels between the legitimate applications and the VFS module.

2. Overview of Our Data Protection Mechanism

2.1 Distributing protected data

Our mechanism introduced in this paper consists of several modules as shown in Figure 1. The encryption software sends the filename and requests a token to the authentication server. The authentication server generates a random token for the file, replies the token to the software and stores the filename/token pair. We will elaborate details of token in Section 5. The encryption software encrypts the file based on the provided token. Encryption software can be executed on any computer as the encrypted file can be uploaded later to the web server. The application logins to the authentication server to receive the filename/token pair. Upon opening the file, the application passes the token to the VFS module. The VFS module accesses the data from the web server and decrypts it using the received token before sends it back to the application.

This paper focuses on the following programs: The VFS module, encryption software and applications. We assume that we can use some token distribution mechanism such as Smart System [1].

2.2 DirectShow API

DirectShow is a multimedia framework and API provided by Microsoft for software developers to perform various operations with media files [3]. Based on the Microsoft Windows Component Object Model (COM) framework, DirectShow provides a common interface for media across many of Microsoft's programming languages, and is an extensible filter-based framework that can render media files on demand by applications.

Most video-related applications on Windows, not only Microsoft's Windows Media Player but also most third-party applications use DirectShow to manage multimedia data. However, DirectShow has a problem with a random access capability. Applications that use DirectShow API can perform random access for local files but cannot when opening files over HTTP. In other words, the video seeking does not work when applications open files over HTTP.

3. VFS Module

The virtual file system (VFS) layer is an abstraction layer on top of more concrete file systems. The purpose of VFS is to allow applications to access different types of concrete file systems in a uniform way. VFS specifies an interface (or a "contract") between the kernel and a concrete file system. Therefore, it enables to add new file system types to the kernel by fulfilling the contract.

Windows has several ways to implement a new file system. The easiest way is through user land Shell namespace extensions [4]. However, this does not support the lowest-level file system access API including DirectShow API in Windows. Not all applications are able to access file systems that are implemented as namespace extensions. The other way is using Installable File System (IFS), a file system API in Microsoft Windows that enables the OS to recognize and load kernel module for file systems.

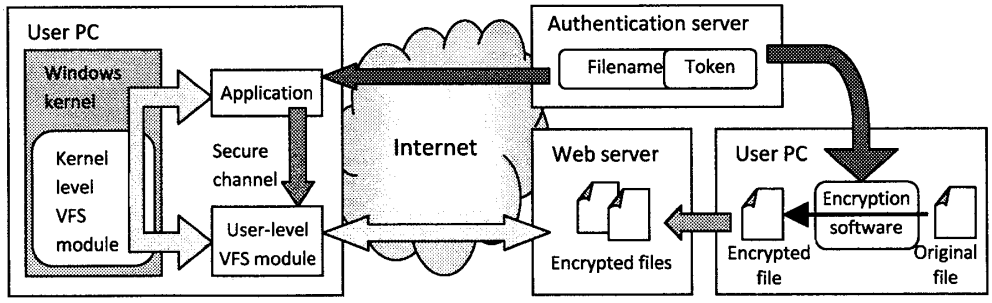

Figure 1 Dataflow and token (cryptography key) distribution in proposed data

The VFS module we use in this research project is same as IFS but with a difference. It is separated into two components: the kernel module and the user level module.

3.1 Dokan Library

In this research, we use Dokan library [2] for simplifying kernel level programming. Dokan library contains a user mode library and a kernel mode file system driver. File operation requests from applications like CreateFile and ReadFile are sent to the Windows I/O subsystem (runs in kernel mode) which subsequently forwards the requests to kernel-level module of Dokan. The kernel-level module of Dokan forwards the requests to the user-level module of Dokan. By using functions provided by the Dokan user mode library, user-level module is able to register callback functions which are invoked in order to response to the forwarded requests. The results of the callback routines are sent back to the kernel-level module and then to the application. The user-level module itself can be written in C, C# or Ruby.

3.2 User-level Module for Dokan

We implement the user-level module of Dokan in the C# language. Upon opening files, the CreateFile callback function is called and this module accesses the file header from the server to obtain the meta-information such as file sizes and file timestamps. These pieces of meta-information are store in memory for later use. When the ReadFile callback function called, the module opens a connection to the server. The module downloads data using the connection based on the given offset and buffer size. After that, the module decrypts the downloaded data using the passed key from the application and returns it to the OS via Dokan.

3.3 Secure Channels

It is vital to prevent malicious applications from opening the protected files. To realize this, a file needs to be encrypted and the legitimate application must pass a token to the VFS module, which is needed to decrypt the requested file. To pass the token, a secure channel between the VFS module and the application needs to be established. We are considering two ways to implement this: a named pipe and a filename extension.

3.3.1 Named Pipe

A safe way to make communication between the VFS module and the application is by using a named pipe. The VFS module sets up a named pipe that can be accessed by the application. Upon opening a file, the application passes the token along with self process ID (PID) to the VFS module through the pipe. The VFS module then compares the received PID with the PID of the application that opens the file. If the PIDs are same, the requested data is decrypted using the passed token.

3.3.2 Filename Extension

In this method, a token is passed with the filename upon opening a file. For example, if the application want to open 'sport.wmv', it must open 'sport.wmv;11-22-33-44-55-66-77-88-11-22-33-44-55-66-77-88' while '11-22-33-44-55-66-77-88-11-22-33-44-55-66-77-88' is the token. The VFS module parses and splits it using the semicolon as the separator. However, there is a limitation using this method. In Windows, a filename is limited up to 255 characters long for Windows XP and 260 characters for Windows Vista. This problem prevents the application from opening file since the token itself is can exceed 40 characters long.

3.4 Performance Issues

In file systems, performance always becomes a matter. To avoid high latency to the whole system, we perform several optimizations.

3.4.1 Pooled Connections

Upon accessing data by an application, data is usually accessed in small chunks from random positions in the file. This leads to rapid ReadFile requests in a very short time. To avoid delays to establish a new connection, we use pooled connections to the server. In other words, the VFS module reuses same connections to access several portions of data from the same file.

3.4.2 Cache and Prefetch

Since we implement a decryption capability, data needs to be downloaded based on block sizes instead of requested buffer sizes. The VFS module downloads data that is slightly bigger than the requested buffer size. Data chunks are cached first then decrypted before the module sends them to the application.

In addition, when playing audio and video files, either from beginning or after perform seeking across the file, data are usually accessed sequentially. Therefore, we improve the performance by using prefetching. The algorithm guesses next data, fetches it and stores it in memory. Each time user performs skipping backward or forward, a new prefetching session is started. To support prefetch properly, we perform the prefetch function in a dedicated thread.

4. Server-side Programs

For the web server, we use the most common web server application, Apache that supports HTTP 1.1. HTTP 1.1 is vital because it contains what we need to realize the random access idea. Since version 1.1, it supports range request (byte serving) and persistence connection (keep-alive) [5]. The range request is needed for random access and persistence connection is essential for pooled connections.

5. Encryption and Decryption

The encryption we use in our mechanism is Advanced Encryption Standard (AES) with Counter (CTR) mode. We use CTR mode because it is suitable with random access. In our mechanism, the token provided by the authentication server consists of two elements: a key and a nonce (or initialization vector). Encryption software splits the file into small blocks and fixes a counter for each block. Each block is encrypted based on the key and the nonce which is concatenated with the block counter. For decryption, the VFS module uses the filename/token pair passed from the application to decrypt the file block.

6. Conclusions

In the paper, we have proposed a new data protection mechanism and how it works. By using this mechanism, users are able to access protected data over HTTP with a random access capability while providing a tight security. In future, we evaluate the use of multiple pooled connections for each file to relief stress on the single connection. We are also considering implementing support to FTP servers.

References

[1] Chikara Miyagi, Koji Ito, and Jun Shimizu: *"Creating the SMART system - A Database for Sports Movement", The Engineering of Sport 6, Vol.3, pp.179-184, 2006.*

[2] Hiroki Asakawa: *Design and implementation of user-mode file system library, 2008* http://decas-dev.net/en/

[3] Microsoft Developer Network (MSDN) Documentation: *DirectShow, 2008*

[4] MSDN Documentation: *Registering Shell Extensions, 2008*

[5] RFC 2616: *Hypertext Transfer Protocol -- HTTP/1.1, 1999*