

Performance Implications of a Kernel Level DRM Controller

Work in Progress

Alapan Arnab, Duncan Bennett, Marlon Paulse and Andrew Hutchison
Data Networks Architecture Group
Department of Computer Science, University of Cape Town
Rondebosch, 7701
{aarnab, dbennett, mpaulse, hutch}@cs.uct.ac.za

Abstract—Digital Rights Management (DRM) aims to provide “persistent access control” of data [5]. A DRM controller is a software or hardware module that enforces the access control rules associated with a DRM enabled data. This paper discusses a project that aims to investigate the possibility of implementing a DRM controller in the kernel of the GNU-Linux operating system and the performance implications of such an implementation.

I. INTRODUCTION

Modern access control mechanisms in files systems are bound to the machine that they reside in. Although Microsoft Windows Active Directory services and Microsoft Windows Network domain controllers allow for access control in a networked environment, any user with administrator access to a machine can over ride these access controls. Furthermore, access controls are also dependent on the file system (for example FAT32 does not support access control mechanisms). And if a data file leaves the corporate network (legitimately or not) the enterprise loses all control over the data. Digital Rights Management (DRM) aims to counter this problem by providing “persistent access control” of data [5].

DRM systems use encryption to protect the confidentiality of data and a *use license* to specify a set of access control rules for the protected data. On the client side, a software or hardware module called the *DRM controller*, is charged with interpreting and enforcing these access control rules.

Currently all but two major DRM systems makes use of application level DRM controllers. Thus, all the rules in the use license are enforced at the application which lead to a number of problems, including:

- 1) every DRM system make use of proprietary DRM formats (Apple iTunes, Windows Media 9, Real Helix etc.) which are all incompatible with each other leading to no interoperability between the different systems.
- 2) application level DRM systems are less secure as many operating system operations cannot be restricted regardless of the use license [1].

DRM can also be implemented at the hardware level; and the DVD-CSS mechanism is the best example. However, hardware level DRM is still in a primitive stage and currently caters only for embedded devices catering for specific formats. Thus, currently we feel an operating system level DRM

controller will provide the best combination of flexibility and security. Microsoft’s Rights Management Services (RMS) does have DRM controller support in a modified Windows XP and 2003 kernels, but it still requires application level support to function [3]. However, some like Rosenblatt [4] have opined that kernel level DRM controllers will have too much overhead and thus impractical. In this paper we detail our proposals to implement a DRM controller as a GNU-Linux Kernel module.

II. PROJECT OVERVIEW

The main aims of the project are to:

- 1) Investigate the feasibility of implementing a kernel level DRM controller.
- 2) Investigate the performance implications of such an implementations.

The system is split into two parts:

- 1) A kernel module to enforce the access control rules.
- 2) A daemon to authenticate users and fetching, managing and interpreting use licenses. For this reason, the daemon itself has a few components:
 - a) A communication module to communicate with federated identity servers, license servers etc. The communication module will essentially be a web services client.
 - b) A license store to store, index and manage use licenses.
 - c) A revocation list to manage invalid use licenses.

III. PROCESS OVERVIEW

DRM systems should theoretically be able to handle any type of data, and ideally the complexities should be hidden from the user. With our approach, we aim to intercept requests from applications, evaluate the request in relation to the use license conditions and then allow or disallow the application to perform that request. The complete process is detailed in figure 1 and the steps are discussed below.

Step 1: The application attempts to access a DRM enabled file.

Step 2: The kernel DRM controller intercepts that request.

Step 3: The DRM controller looks up whether an use license exists for the data file in question. The license store

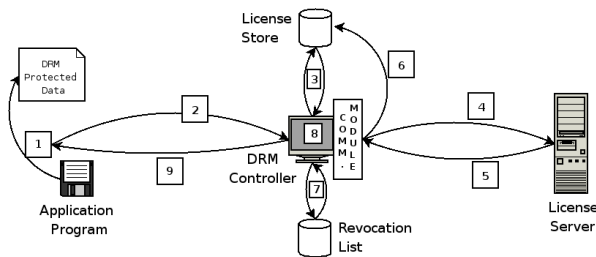


Fig. 1. Accessing DRM protected data

gives the DRM controller the license if it exists or a message indicating a license for the data file does not exist yet. If the license exists, the DRM controller moves to step 7. The license store could have an expired license, in which case the DRM controller continues with step 4.

- Step 4: The DRM controller contacts the license server for a use license. The user authenticates themselves to the license server, pays for the license if required and should also be allowed to negotiate with the license server for specific terms if the rights holder allows for such an option. This will be the first known implementation of negotiation enabled DRM system and based on language extensions discussed in [2].
- Step 5: If a license is granted, it is transferred to the DRM controller. If a license is not granted, an appropriate error message is sent to the DRM controller and the access request is denied.
- Step 6: The DRM controller stores the use license in the license store.
- Step 7: The DRM Controller checks the validity of use license against the revocation list. As explained previously, this action could be done online with the license server. If the license is invalid, the user could negotiate a new license with the license server (Step 4).
- Step 8: The DRM controller authenticates the user against the user specified in the use license. If authentication fails, the user should be prevented from accessing the file.
- Step 9: If authentication is successful, and the use license allows the action the application wishes to perform, the application's request is processed. For every subsequent action that the application wishes to perform, steps 2 and 9 are performed as the kernel will cache a copy of the use license in use. Should the license expire during use, steps 3 – 8 need to be redone. Authentication of the user is not necessary for every action, but can be scaled according to the use license – for example the license may stipulate that the user needs to authenticate themselves after every hour of use.

Use licenses are specified in Rights Expression Languages (RELs). In this project we aim to use the ODRL REL, together

with bi-directional negotiation support as discussed in [2]. However, the main aim is to investigate the feasibility of a kernel level DRM implementation and thus only a small number of permissions (the access control terms like render and open) and a small number of related constraints (boundaries of the control terms, like count or time) will be implemented. However the aim is to create a flexible implementation that can handle any REL and any number of permissions and their constraints.

IV. PERFORMANCE EVALUATION

There are two areas where we can investigate the performance implications of our DRM implementations, and we plan to explore both.

Firstly, will there be too much overhead as claimed by Rosenblatt in [4] or will the effect be negligible? To answer this question we plan to benchmark an unmodified kernel and a modified kernel with a few popular Linux benchmarking tools. These tools will just have a DRM use license but remain unencrypted.

Secondly, will there be any significant delays for users when accessing DRM protected data? To answer this question, we plan to perform user testing where users will be asked to perform a number of standard tasks on a standard kernel and a modified DRM enabled kernel. The user will then be asked on whether they noticed any latency between the different systems and if so, how distracting was the latency.

V. CONCLUSION

In this paper we have presented an overview of a kernel level DRM system implementation. To our knowledge, this is the first ever attempt at such an implementation. We have also discussed our aims in evaluating the performance of our implementations. We will consider this project a success if kernel level DRM controllers can be implemented with minimal wall time performance implications.

ACKNOWLEDGEMENTS

This work is partially supported through grants from the UCT Council and the National Research Foundation (NRF) of South Africa. Any opinions, findings, and conclusions or recommendations expressed in this paper/report are those of the author(s) and do not necessarily reflect the views of UCT, the NRF or the trustees of the UCT Council.

REFERENCES

- [1] A. Arnab and A. Hutchison, "Digital Rights Management - An overview of Current Challenges and Solutions," in *Proceedings of Information Security South Africa (ISSA) Conference 2004*, 2004.
- [2] —, "Extending ODRL to Enable Bi-Directional Communication," in *Proceedings of the 2nd International ODRL Workshop*, 2005.
- [3] —, "Requirement Analysis of Enterprise DRM Systems," in *Proceedings of Information Security South Africa (ISSA) Conference 2005*, 2005.
- [4] B. Rosenblatt, "DRM for the Enterprise," 2004.
- [5] B. Rosenblatt and G. Dykstra, "Integrating content management with digital rights management - imperatives and opportunities for digital content lifecycles," *Giantsteps Media Technology Strategies*, White Paper, 2003, URL: http://www.giantstepsmts.com/drm-cm.white_paper.htm.