Towards a Secure Application-semantic Aware Policy Enforcement Architecture

Srijith K. Nair, Bruno Crispo and Andrew S. Tanenbaum

Dept. of Computer Science, Faculty of Sciences, Vrije Universiteit,
1081 HV Amsterdam, The Netherlands

Abstract. Even though policy enforcement has been studied from different angles including notation, negotiation and enforcement, the development of an application-semantic aware enforcement architecture remains an open problem. In this paper we present and discuss the design of such an architecture.

1 Introduction

As networked and grid computing and web service architectures are gaining acceptance, computer systems are being transformed from standalone systems into a shared and more open environment. Policies defining the limits and working conditions of various parts of such a system constitute a pseudo-contract based environment of operation. Enforcement of these policies in a proper manner plays a crucial role in preserving the trust and integrity of the system.

Even though a lot of research has been carried out in the area of policy enforcement at various levels of abstraction, it still remains an open problem. Most of the policy enforcement approaches have been geared towards enforcement of users polices on a time-sharing machine administered by a central authority [1] . With the advent of cheap secure hardware [2] and technologies like secure booting [3] that are cheap enough to be deployed on individual user machines, we need to re-evaluate existing approaches towards policy enforcement. In this position paper we plan to describe briefly our contribution towards solving some of the issues of the policy enforcement problem.

In Sect. 2 we give examples of applications whose policies are hard or impossible to enforce with current solutions. Architecture for Zodac, a new secure policy enforcement system is presented in Sect. 3 and the security considerations of the system are discussed in Sect. 4. We then provide a brief overview of previous work in Sect. 5 and conclude in Sect. 6.

2 Policy Enforcement Examples

Policy enforcement is a critical part of any secure system. In this section we provide couple of scenarios and example applications whose functionality and integrity depends a lot on the presence of such a system.

Mobile Agents - Companies are opening up their services to allow third party generated code (agents) to run on their hosting platform and query their back-end databases [4]. Such mobile agents raise two main security issues - first, the safety of the platform running these untrusted agents and second, protecting the integrity and confidentiality of the data and code of the agent. When a mobile agent is sent off to a remote host, it could be associated with certain policies that define its working constraint. For example, an auction agent could be allowed to share the personal details of the owner, but not the highest bid it is allowed to place. The agent owner needs to be certain that the policies are enforced in an environment that he does not own or control. Similarly, the remote host will also be constrained by policies that define the limits of the service that it can provide. These policies too need to be enforced.

Emails - Corporate emails can be associated with various policies like Do not forward, Forward only within the Accounting Department, To be accessed within protected/safe environment only etc. It is paramount that these policies are enforced in a proper manner, without relying on the judgment or goodwill of the recipient.

Digital Content Usage - To exploit the emerging market for digital media (music, movies, e-book etc.) the content owners may decide to associate various kinds of policies with the content they distribute. For example, a digital audio clip could be associated with the policy that unless a payment has been made, only 30 seconds of it can be played or that it can be played only 5 times. The content owners need to be convinced that these policies will be enforced in the users end machine.

Privacy Data Protection - When a user shares his data with an external party, he may wish to associate a policy governing the usage of the data. For example, he may wish to state that none of the data he has provided should be shared with a third party. More complicated policies could take the form of sharing as long as owner gets paid 1c per transaction or that data should be destroyed after 1 year. The user needs to be certain that the external party has a system in place to enforce these policies. The external party needs to prove that a system exists that can guarantee the enforcement of the policies and then carry out the actual enforcement at its end.

3 Zodac Enforcement Architecture

Our aim in this paper is to describe a general purpose policy enforcement architectural framework, named Zodac, which is generic enough to be used in a wide range of applications, some of which are mentioned in the previous section.

3.1 Components

In order to build a generic framework, we have divided our architecture into components based on the functionality provided by them or on the levels of trust associated with it.

File system - The file system will be implemented as strongly typed with respect to the application. A file that is associated with an email client can only be accessed by a call from the email client and not a text editor. In addition, sticky policies are also associated with every file, defining other usage restrictions.

Application - While no inherent trust is associated with an application, they may have to be rewritten to make use of the layered structure defined in the architecture.

Application Policy Enforcer - Several policies can have attributes closely related to the applications semantics. These cannot be readily interpreted at a very low level. For example a policy related to Play cannot be interpreted directly at the operating system level. Applications calls that relate to these policies have to be intercepted by a higher up layer that can vet through the calls, interpret them and then translate them to systems calls that can be understood at a lower level. The Application Policy Enforcer (APE) does this job.

Base Policy Enforcer - Application semantic independent calls (like file open, read) are common between applications and hence can be intercepted and analysed at a common layer just above the operating system. The Base Policy Enforcer (BPE) is the name we give to this layer.

Policy Evaluator - This is the primary engine that makes decisions on which calls can be allowed as per policy and which of them have to be denied. Since the engine would need application semantic level knowledge to make certain policy decisions, the Policy Evaluator (PE) will be a per-application-type engine. For example, an email PE will be different from a PDF reader PE.

Plugin Extensions - These are extensions provided by trusted parties that provide additional functionality at the APE and the PE level. For example, a music file purchased from Apple iTunes [5] would have policies specific to iTunes that cannot be correctly analysed by a generic APE or PE for a media player. Hence, an extension can be provided by Apple to correctly interpret such policies.

3.2 Control Flow

This subsection describes the flow of control between the various components of the system when an application opens a file for processing. Figure 1 below gives a summary of the flow:

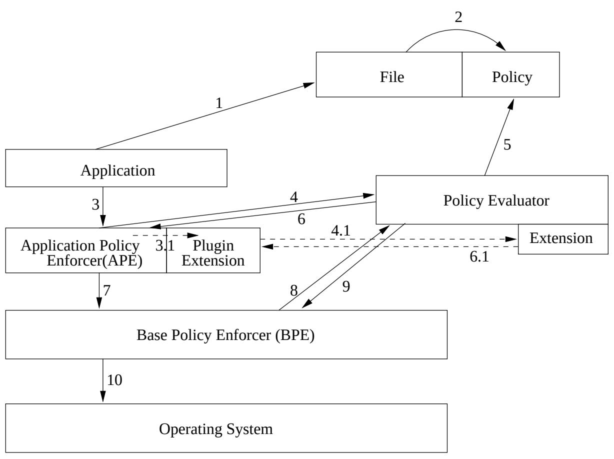

Fig. 1. Proposed enforcement system architecture's control flow

The application opens a file (1) and in turn opens the associated policy (2). Any system /application calls and operations on this file are trapped and forwarded to the APE (3). The APE may forward this call to a plugin extension (3.1) if the functionality requested is provided by a custom extension. The APE (or the extension) checks with the Policy Evaluator (4) (or its extension, 4.1) whether the call should be allowed or not. The evaluator checks the policy (5) and conveys the decision back to the APE or its extension (6/6.1). If the policy allows the call, the APE passes the call to the BPE layer (7), translating the higher level (application semantic) calls to system level calls. The BPE queries the Policy Evaluator again (8) to make sure that the calls it received is allowed by the policy. The evaluation result is sent back to BPE as denoted in step 9. If allowed, this system call is then sent to the operating system. The resulting reply/message from the OS is then sent back to the APE, which performs the appropriate action.

The additional check in steps 8 and 9 are required to provide a second level of enforcement so that in case a direct system level call is made to the BPE without being routed through the APE, it can be trapped and evaluated.

4 Security Considerations

The application running in user-space is not trusted and hence it can attempt to subvert the enforcement of the policy associated with a file. Our secure framework prevents such security breaches.

We assume the core operating system to be free of exploitable bugs. The operating system accepts only the application calls that have been properly vetted by the APE and the BPE. Jailing [6], reference monitors [7] or other similar implementations could be considered for the actual implementation of this enforcement. A windowing system similar to EROS Trusted Window System [8] or Nitpickers [9] could be used to provide a secure windowing environment to differentiate between different levels of confidentiality of windows.

The APE needs to translate application-calls into low level system-call. The PE (with the help of the extension) evaluates the policy. The policy language should be generic enough to specify most common attributes of usage for different applications and yet extendable enough to be useful for specific kinds of applications. A modification of XACML [10] and XrML [11] could be used as a starting point for such a language. Since the plugin extension systems integrity depends on the plugin provider, a form of trust relationship, maybe in the form of signed codes will have to be established between the provider and the system.

One of the most crucial parts of the enforcement system that is still under investigation is the memory and data management. While a single application can be compartmentalized to prevent data leaks and side channel attacks, the interaction between various applications is a vulnerable area for such attacks. The memory and data management component of the system has to be robust enough to withstand such attacks. We hope to resolve this in Zodac using a memory handle management system wherein the BPE acts as a manager deciding whether to allow or deny request or call for the handle to the applications memory by other applications.

5 Related Work

Most research work that tackle the issue of policy enforcement either concentrate on policy negotiation [12], the decision making process [13], limit themselves to architectures that does not handle application level semantics [14] or consider only the enforcement of policies on machines that are directly under the control of the user [1]. Though a lot of work has been done on access control policy enforcement [15, 16], that forms only a part of the system and the other parts like policies based on resource management, semantic level issues etc. are left out in such work.

6 Conclusion

In this paper we showed the need for a secure policy enforcement system and how previous research has not provided us one. We then introduced Zodac, a secure policy enforcement system and described the components of the system, detailing the passing of control between them. In the end the security issues associated with Zodac were discussed and couple of possible solutions was presented.

References

1. Multics. http://www.multicians.org/, May 2006
2. Trusted Computing Group. http://www.trustedcomputinggroup.org/, May 2006
3. Arbaugh, W. A., Farber, D. J., Smith, J. M.: A Secure and Reliable Bootstrap Architecture. In IEEE Security and Privacy Conference, May 1997, pp. 65-71

4. Alexa Web Search Platform. http://websearch.alexa.com/welcome.html, May 2006
5. Apple iTunes. http://www.apple.com/itunes/, May 2006
6. The Jail Subsystem, FreeBSD Architecture Handbook. http://www.freebsd.org/doc/en_US.ISO8859-1/books/arch-handbook/jail.html, May 2006
7. Anderson, J. P.: Computer Security Technology Planning Study. Technical Report ESD-TR-73-51, U.S. Air Force Electronic Systems Division, Deputy for Command and Management Systems, HQ Electronic Systems Division (AFSC), Bedford, Massachusetts, October 1972, Vol. 2, 5869
8. Shapiro, J. S., Vanderburgh, J., Northup, E., Chizmadia, D.: Design of the EROS Trusted Window System. In Proceedings of the 13th USENIX Security Symposium, pp. 165-178, 2004
9. Feske, N., Helmuth, C.: A Nitpicker's guide to a minimal-complexity secure GUI. 21st Annual Computer Security Applications Conference (ACSAC 2005), Tucson, Arizona, USA, 2005
10. Extensible Access Control Markup Language (XACML). OASIS standards, http://www.oasis-open.org/specs/index.php#xacmlv1.0
11. eXtensible Rights Markup Language (XrML). http://www.xrml.org, May 2006
12. Mobach, D. G. A., Overeinder, B. J., Brazier, F. M. T., Dignum, F.P. M.: A Two-tiered Model of Negotiation Based on Web Service Agreements. Proceedings of the Third European Workshop on Multi-Agent Systems (EUMAS'05), December 2005
13. Blaze, M, Feigenbaum J., Lacy, J.: Decentralized Trust Management. IEEE Symposium on Security and Privacy, Oakland, CA. May 1996
14. Erlingsson, U., Schneider, F. B.: SASI Enforcement of Security Policies: A Retrospective. Proceedings New Security Paradigms Workshop, Ontario, September 1999
15. Sandhu, R. S., Coyne, E. J., Feinstein, H. L., Youman, C. E.: Role-Based Access Control Models. IEEE Computer, Vol. 29.2 pp 38-47, US,1996
16. Graubert, R.: On the Need for a Third Form of Access Control. Proceedings of the 12th National Computer Security Conference, pp. 296-304, October 1989