FUNCTIONALITY-BASED APPLICATION CONFINEMENT

Parameterised Hierarchical Application Restrictions

Z. Cliffe Schreuders and Christian Payne
School of IT, Murdoch University, South St, Murdoch, Western Australia
c.schreuders@murdoch.edu.au, c.payne@murdoch.edu.au

Keywords: Application-Oriented Access Control, Application Confinement, Sandbox, Functionality-Based Application Confinement (FBAC), Role-Based Access Control (RBAC), Usable Security.

Abstract: Traditional user-oriented access control models such as Mandatory Access Control (MAC) and Discretionary Access Control (DAC) cannot differentiate between processes acting on behalf of users and those behaving maliciously. Consequently, these models are limited in their ability to protect users from the threats posed by vulnerabilities and malicious software as all code executes with full access to all of a user's permissions. Application-oriented schemes can further restrict applications thereby limiting the damage from malicious code. However, existing application-oriented access controls construct policy using complex and inflexible rules which are difficult to administer and do not scale well to confine the large number of feature-rich applications found on modern systems. Here a new model, Functionality-Based Application Confinement (FBAC), is presented which confines applications based on policy abstractions that can flexibly represent the functional requirements of applications. FBAC policies are parameterised allowing them to be easily adapted to the needs of individual applications. Policies are also hierarchical, improving scalability and reusability while conveniently abstracting policy detail where appropriate. Furthermore the layered nature of policies provides defence in depth allowing policies from both the user and administrator to provide both discretionary and mandatory security. An implementation FBAC-LSM and its architecture are also introduced.

1 INTRODUCTION

Traditional access control models such as Mandatory Access Control (MAC), Discretionary Access Control (DAC) and Role-Based Access Control (RBAC) are based on the paradigm of protecting users from one another (Department of Defense, 1985, Ferraiolo et al., 1995). Consequently programs typically run with all of the user's privileges. These models cannot differentiate between a program acting on the behalf of a user and a program using its privileges nefariously (Miller and Shapiro, 2003). As a result vulnerabilities and malware represent a serious threat as malicious code has unrestricted access to the user's privileges.

Existing application confinement schemes attempt to address this by limiting the privileges associated with processes, thereby mitigating the impact from vulnerabilities and malicious code. Several techniques have been developed to provide application-oriented access controls; however, these techniques do not provide abstractions which are easy for users to apply, and do not scale well to confine the numerous feature rich applications found on modern systems.

A new application-oriented access control model Functionality-Based Application Confinement (FBAC) is presented which provides separation of duties, defence in depth through layers of mandatory and discretionary application-oriented access controls and policy abstractions which are flexible, manageable and easy to conceptualize.

2 APPLICATION-ORIENTED ACCESS CONTROL MODELS

Existing application-oriented access control models assign privileges using monolithic self-contained non-hierarchical policy abstractions. This limits the scalability of these approaches as these abstractions can not adapt to the different security needs of applications.

Isolation sandboxes such as chroot, BSD Jails (Kamp and Watson, 2000), Solaris Zones (Tucker and Comay), and Danali (Whitaker et al., 2002) provide a single policy abstraction, the isolated container, which simply restricts contained applications to a limited namespace (Kamp and Watson, 2004) or virtual machine (Madnick and Donovan, 1973). Isolation requires significant redundancy as shared resources need to be duplicated (Krohn et al., 2005). It also inhibits the ability of applications to easily and securely exchange data as is commonly required.

Some application-oriented schemes mediate access to specified resources by simply assigning raw privileges to processes. These are either coarsely grained (such as with POSIX capabilities (Bacarella, 2002), and Bitfrost (Krsti and Garfinkel, 2007)) or finely grained (as with CapDesk (Miller et al., 2004), Polaris (Stiegler et al., 2006), TRON (Berman et al., 1995), Systrace (Provos, 2002) and Janus (Wagner, 1999)). Methods of mediating this type of access control include using capabilities (Wagner, 2006) or system call interposition (Goldberg et al., 1996). These privilege associations provide very little policy abstraction other than the granularity of the privileges assigned making the policy either inexpressive or extremely large and complex (Garfinkel, 2003). Translating high level security goals into finely grained policies is difficult, making these policies difficult to both construct and verify for correctness (Marceau and Joyce, 2005).

Models such as Domain and Type Enforcement (DTE) (Badger, 1996) which extends the type enforcement model (Boebert and Kain, 1985) , Role-Compatibility (RC) (Ott, 2002), and AppArmour (previously known as SubDomain) (Cowan et al., 2000) provide large inflexible policy abstractions which, although capable of grouping related privileges, cannot adapt to the various policy needs of feature rich applications. For example, although a DTE domain represents a policy abstraction, domains typically apply to a single application only (Marceau and Joyce, 2005). Additionally, there is significant overlap of privileges granted to compiled domain policies and yet domains are specified separately (Jaeger et al., 2003).

Although some implementations of these models allow a policy abstraction to be comprised of smaller parts, these parts are reduced to a monolithic policy abstraction either at system start-up or in advance, which limits their flexibility. One example of this is SELinux's DTE Domains which can be specified using macros in the m4 language which are compiled in advance into many lines of rules, thereby creating a single policy abstraction (a Domain) directly containing all the relevant privileges. Similarly, any abstractions in AppArmor profiles are compiled away at system start-up and applied as a raw list of privileges associated with the application. This approach means that any finer grained abstractions which may have been in place when constructing policy is not available when managing the privileges of a process.

DTE and RC policy abstractions define multiple restricted environments and allow propagating processes to transition between them. Specifying these transitions is often a complex and error-prone task. Programs need specific authorisation to label files as being accessible in different domains or roles, and users and programs both need permission in order to execute programs belonging in another domain or role (Hinrichs and Naldurg, 2006).

3 THE FBAC MODEL

Inspired by the Role-Based Access Control (RBAC) model (Ferraiolo and Kuhn, 1992), behaviour based process confinement research (Raje, 1999), programming language features such as subroutine parameterisation, and by applying a unique approach to defence in depth, Functionality-Based Application Confinement (FBAC) provides an expressive, finely grained, yet easy to apply and manage application confinement. In contrast to existing application-oriented models FBAC allows reusable and flexible policy abstractions to be defined which can be adapted to suit the needs of different applications with related security goals.

3.1 Functionality-based

The design of the FBAC model originated from observing the advantages that the RBAC model brings to the management of user privileges and FBAC employs an analogous paradigm for the confinement of individual programs. While in RBAC different users share common sets of privileges relating to their role within an organisation (Ferraiolo et al., 1995), in application confinement each category of application requires related sets of privileges corresponding to their intended behaviour (Raje, 1999). Recognizing this correspondence provides a convenient mechanism to both model the privileges that a program requires and for end users to assign a program the privileges it needs based upon the functionality the application performs. Therefore, while RBAC assigns privileges

to users according to their role, FBAC employs reusable and flexible policy abstractions known as *functionalities* describing the actions that an application may legitimately perform. For example, the shared functionality of different web browsers is reflected in their requiring a common set of privileges to carry out their tasks and forms the basis on which end users assign an application confinement policy.

Furthermore, applying an RBAC-like approach to application confinement also provides the benefit of separation of duties. Static separation of duty prevents conflicting functionalities or privileges from being assigned to the same application while dynamic constraints ensure that specified sets of privileges cannot be activated concurrently at runtime.

3.2 Hierarchical Policy

Unlike existing application confinement models, FBAC policies are constructed in a hierarchical fashion by employing a ANSI/NIST RBAC-like structure (Ferraiolo et al., 2001). This allows layers of abstraction and encapsulation with high-level functionalities describing the overall purpose of the application (for example, web_browser and email_client) and mid-level functionalities specifying the functional components which make these up such as http_client and pop3_client. These in turn are built from low-level abstractions describing the finely-grade privileges available on the system, for example file_r for reading from a file. This hierarchical structure improves the manageability of policy by encapsulating details while providing flexible abstractions for association with specific applications. This allows FBAC policies to be applied to multiple applications where these have shared functionality and provides improved scalability compared with existing finely grained application confinement models.

The hierarchical design of FBAC's policy abstractions allows small or large policy components to be easily activated or deactivated at runtime. This action may be initiated by the user, the administrator or even the software itself. For example, in the case of multi-functionality applications such as the Opera web browser which also incorporates e-mail, IRC, news reader and bittorrent client functionality, the user can actively control the privileges currently available to the program according to those functionalities being used at the time. This is analogous to a user under RBAC only activating the rolls relating to the part of their job description they are currently performing and allows the principle of least privilege to be enforced. This level of run-time policy control is not available with existing application-oriented access control models such as DTE or AppArmor where privileges are contained in a monolithic abstraction associated with the security context. For example, in DTE or AppArmor dropping the ability to send emails would involve transitioning to an entirely separate domain or profile.

3.3 Functionality Parameterisation

Based on the results of previous research which explored the use of parameterisation for application sandboxes (Raje, 1999), functionalities are parameterised to allow them to be applied to different applications with similar functionality; for example, two different web browsers. This allows application confinement policies to be customised to the specifics of each program (such as where it stores configuration files etc.) while maintaining the abstraction of the original policy specification.

FBAC functionalities are passed arguments in a fashion similar to subroutines in programming languages. Subsequently, the hierarchical relationship between functionalities allows arguments to propagate to any contained functionality. By specifying resource names as arguments functionalities can be reused within the hierarchy to grant access to various resources.

Functionality definitions may contain default argument values. This maintains abstraction and simplifies the process of assigning functionalities to applications in common cases while not restricting flexibility where customisation is required.

Although the MAPbox mechanism (Acharya and Raje, 2000) has previously employed parameterisation to support application confinement, FBAC overcomes limitations of this approach by allowing confinement of multipurpose applications and providing a more manageable policy structure than that of MapBox, where users assign a complex finely-grained list of privileges to each class.

3.4 Mandatory and Discretionary Controls

Another feature of the FBAC model is its ability to confine applications based upon the combination of policies specified by both users and administrators. While existing application oriented access controls are generally designed to be applied as either a discretionary control (such as Janus or TRON) or a

mandatory control (such as AppArmor or DTE), FBAC allows both mandatory and discretionary policies to be applied simultaneously. This allows users to ensure their applications execute with least privilege and protect themselves from malicious code while also allowing administrators to restrict applications to enforce system-wide security goals, confine users to specific programs or to manage user protection. Each of these policies is known as an FBAC *confinement* and may reuse functionalities from other confinements. The privileges of an application therefore depend upon the intersection of the privileges specified by the confinements which apply to it. This layered approach to application confinement is unique and provides defence in-depth while requiring the maintenance of only a single mechanism.

4 USING FBAC

The initial task of establishing functionalities involves the construction of functionalities which represent functional requirements of applications. Functionalities can contain other functionalities and can also contain direct privileges. The hierarchical and modular nature of FBAC policy eases management and maintenance. This initial functionality creation task involves the analysis of existing applications and requires some expertise and would normally therefore be done by a trusted third party.

However, subsequently users and administrators can restrict applications with FBAC by simply assigning the appropriate functionalities and providing any arguments necessary to satisfy parameters. This process is well suited to a GUI and mostly involves pointing and clicking. Familiarity with the FBAC-LSM policy language is unnecessary for ordinary users who can simply use graphical tools to confine applications.

Administrators can easily limit users to specific applications and specify what those applications can do. Users may then supply more restrictive parameters protecting their own resources from the application.

5 REPRESENTING POLICY

Figure 1 is an example policy representation of a simple functionality which provides an abstraction to read the contents and attributes of a file. The first line simply specifies the name of the functionality (`functionality [name]`). The directives which detail the functionality are enclosed in curly braces. Each directive ends in a semicolon. The first two directives are for a graphical tool to describe the purpose and level of detail of the functionality. Then a parameter named files is specified; its default value is to grant access to nothing (`parameter [parameter name] "[default value]"`). After the purpose of this parameter is described, two privileges are included, which permit access to the files described by the parameter. (`privilege [operation name] ["literal filename" or parameter name]`).

A functionality can also contain another functionality with the following syntax (`functionality [functionality name] ([optional parameter name=] [["literal filename" or parameter name], ...])`

Application profiles share a similar syntax, with a difference in the initial definition (`application [name]`) and contain a list of binaries which make up the application (`binarypaths [path]:[path…]`).

```
functionality files_r
{
functionality_description "read
access to these files";
lowlevel;
parameter files "";
parameter_description "allows these
files to be accessed as described";
privilege file_read files;
privilege file_getattr files;
}
```

Figure 1: Low-level FBAC-LSM functionality and privileges.

6 THE IMPLEMENTATION - FBAC-LSM

A prototype implementation of FBAC is near completion. FBAC-LSM is a Linux Security Module (LSM) (Wright et al., 2002) with accompanying policy tools. As Figure 2 illustrates, FBAC-LSM is comprised of a graphical Policy Manager tool which is used to maintain policy, the LSM which resides in kernel space and enforces security decisions, a Policy Server which feeds the policy into the LSM

via a virtual file system at system boot or on request, and a graphical Process Manager tool which can be used to activate or deactivate the functionalities associated with a running process. When an application attempts to access any mediated resource, after standard DAC rules apply, the LSM is consulted and the request is either allowed or rejected based on the FBAC policy as represented in the LSM. Figure 3 illustrates the simple task of selecting functionalities using the graphical Policy Manager.

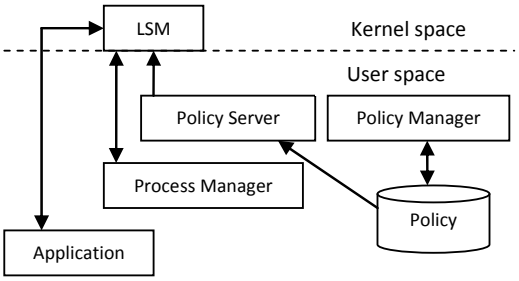

Figure 2: FBAC-LSM architecture.

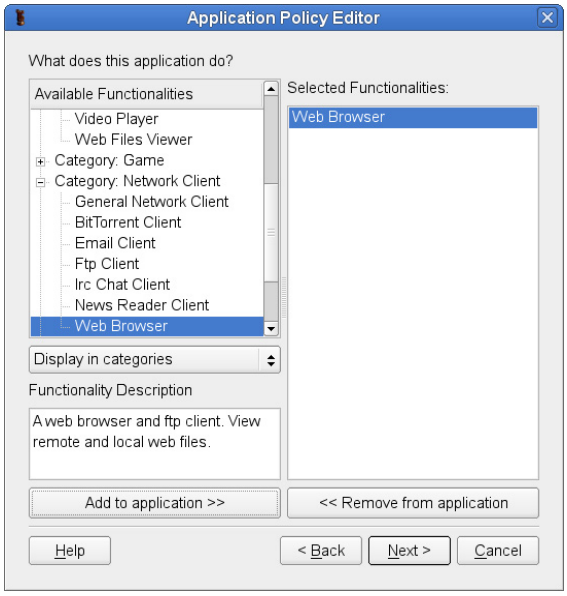

Figure 3: The graphical FBAC-LSM Policy Manager tool.

7 RESEARCH STATUS

FBAC-LSM has shown promising results and a hierarchy of functionalities that represent the functionalities required for some common applications, such as web browsers, has been developed. When the prototype system is complete a detailed study comparing the security and usability of the new system with existing systems such as SELinux and AppArmor will be presented and FBAC-LSM will be made available open source using the General Public Licence.

8 CONCLUSIONS

FBAC consolidates concepts from user-oriented access control, and application sandboxing research to provide an application-oriented access control model which confines applications in terms of the functions they perform. Policy is hierarchical, parameterised and multi-layered. This approach provides security and policy management benefits such as conceptual simplicity through abstraction and encapsulation, policy reusability and flexibility, improvements in scalability, separation of duties, dynamic process controls, and defence in depth. Preliminary results of the new model are promising and further study of the efficacy of the model in action is warranted.

ACKNOWLEDGEMENTS

The authors would like to thank Dr Tanya McGill for her guidance in the preparation of this paper. We also acknowledge the valuable feedback and comments of the reviewers.

REFERENCES

ANSI INCITS 359-2004. American National Standards Institute / International Committee for Information Technology Standards (ANSI/INCITS).

Acharya, A. & Raje, M. (2000) MAPbox: Using Parameterized Behavior Classes to Confine Applications. *Proceedings of the 2000 USENIX Security Symposium.* Denver, CO, USA.

Bacarella, M. (2002) Taking advantage of Linux capabilities. *Linux Journal,* 2002.

Badger, L. (1996) A Domain and Type Enforcement UNIX Prototype. *Computing Systems,* 9**,** 47-83.

Berman, A., Bourassa, V. & Selberg, E. (1995) TRON: Process-Specific File Protection for the UNIX Operating System. *Proceedings of the 1995 Winter USENIX Conference*.

Boebert, W. E. & Kain, R. Y. (1985) A Practical Alternative to Hierarchical Integrity Policies. *Proceedings of the 8th National Computer Security Conference***,** 18–27.

Cowan, C., Beattie, S., Kroah-Hartman, G., Pu, C., Wagle, P. & Gligor, V. (2000) SubDomain: Parsimonious Server Security. *USENIX 14th Systems Administration Conference (LISA)*.

Department of Defense (1985) *Trusted Computer Security Evaluation Criteria. DOD 5200.28-STD*.

Ferraiolo, D., Cugini, J. A. & Kuhn, R. (1995) Role-Based Access Control (RBAC): Features and Motivations. *Annual Computer Security Applications Conference.* Gaithersburg, MD, USA, IEEE Computer Society Press.

Ferraiolo, D. & Kuhn, R. (1992) Role-Based Access Control. *15th National Computer Security Conference.* Baltimore, MD, USA.

Ferraiolo, D. F., Sandhu, R., Gavrila, S., Kuhn, D. R. & Chandramouli, R. (2001) Proposed NIST Standard for Role-Based Access Control. *ACM Transactions on Information and System Security,* 4**,** 224–274.

Garfinkel, T. (2003) Traps and Pitfalls: Practical Problems in System Call Interposition Based Security Tools. *Proceedings of the 10th Network and Distributed System Security Symposium.* February ed. San Diego, CA, USA, Stanford University.

Goldberg, I., Wagner, D., Thomas, R. & Brewer, E. A. (1996) A Secure Environment for Untrusted Helper Applications: Confining the Wily Hacker. *Proceedings of the 6th USENIX Security Symposium.* San Jose, CA, USA, University of California.

Hinrichs, S. & Naldurg, P. (2006) Attack-based Domain Transition Analysis. *2nd Annual Security Enhanced Linux Symposium.* Baltimore, Md., USA.

Jaeger, T., Sailer, R. & Zhang, X. (2003) Analyzing Integrity Protection in the SELinux Example Policy. *Proceedings of the 12th USENIX Security Symposium***,** 59–74.

Kamp, P.-H. & Watson, R. (2000) Jails: Confining the Omnipotent Root. *Sane 2000 - 2nd International SANE Conference*.

Kamp, P.-H. & Watson, R. (2004) Building Systems to be Shared Securely. *ACM Queue,* 2**,** 42-51.

Krohn, M., Efstathopoulos, P., Frey, C., Kaashoek, F., Kohler, E., Mazieres, D., Morris, R., Osborne, M., Vandebogart, S. & Ziegler, D. (2005) Make least privilege a right (not a privilege). *Procedings of 10th Hot Topics in Operating Systems Symposium (HotOS-X).* Santa Fe, NM, USA.

Krsti, I. & Garfinkel, S. L. (2007) Bitfrost: the one laptop per child security model. *ACM International Conference Proceeding Series,* 229**,** 132-142.

Madnick, S. E. & Donovan, J. J. (1973) Application and Analysis of the Virtual Machine Approach to Information Security. *Proceedings of the ACM Workshop on Virtual Computer Systems.* Cambridge, MA, USA.

Marceau, C. & Joyce, R. (2005) Empirical Privilege Profiling. *Proceedings of the 2005 Workshop on New Security Paradigms***,** 111-118.

Miller, M. S. & Shapiro, J. (2003) Paradigm Regained: Abstraction Mechanisms for Access Control. *8th Asian Computing Science Conference (ASIAN03)***,** 224–242.

Miller, M. S., Tulloh, B. & Shapiro, J. S. (2004) The structure of authority: Why security is not a separable concern. *Multiparadigm Programming in Mozart/Oz: Proceedings of MOZ,* 3389.

Ott, A. (2002) The Role Compatibility Security Model. *7th Nordic Workshop on Secure IT Systems.*

Provos, N. (2002) Improving Host Security with System Call Policies. *12th USENIX Security Symposium.* Washington, DC, USA, USENIX.

Raje, M. (1999) Behavior-based Confinement of Untrusted Applications. TRCS 99-12. *Department of Computer Science.* Santa Barbara, University of Calfornia.

Stiegler, M., Karp, A. H., Yee, K. P., Close, T. & Miller, M. S. (2006) Polaris: virus-safe computing for Windows XP. *Communications of the ACM,* 49**,** 83-88.

Tucker, A. & Comay, D. Solaris Zones: Operating System Support for Server Consolidation. *3rd Virtual Machine Research and Technology Symposium Works-in-Progress*.

Wagner, D. (2006) Object capabilities for security. *Conference on Programming Language Design and Implementation: Proceedings of the 2006 workshop on Programming languages and analysis for security,* 10**,** 1-2.

Wagner, D. A. (1999) Janus: An Approach for Confinement of Untrusted Applications. Technical Report: CSD-99-1056. *Electrical Engineering and Computer Sciences.* Berkeley, USA, University of California.

Whitaker, A., Shaw, M. & Gribble, S. D. (2002) Denali: Lightweight virtual machines for distributed and networked applications. *Proceedings of the 5th USENIX Symposium on Operating Systems Design and Implementation***,** 195–209.

Wright, C., Cowan, C., Smalley, S., Morris, J. & Kroah-Hartman, G. (2002) Linux Security Module Framework. *Ottawa Linux Symposium.* Ottawa Canada.