A Policy Language for Abstraction and Automation in Application-oriented Access Controls

The Functionality-based Application Confinement Policy Language

Z. Cliffe Schreuders, Christian Payne, Tanya McGill
School of IT
Murdoch University
Murdoch, Western Australia
{c.schreuders, c.payne, t.mcgill}@murdoch.edu.au

Abstract—**This paper presents a new policy language, known as *functionality-based application confinement policy language* (FBAC-PL). FBAC-PL takes a unique approach to expressing application-oriented access control policies. Policies for restricting applications are defined in terms of the features applications provide, by means of parameterised and hierarchical policy abstractions known as *functionalities*. Policies also include metadata for management and the automation of policy specification. The result is a novel scheme for application confinement policy that reuses, encapsulates and abstracts policy details, and facilitates *a priori* policy specification: that is, without having to rely solely on learning modes for creating policies to restrict applications. This paper presents the policy language, and illustrates its use with examples. A Linux-based implementation, which uses FBAC-PL, has demonstrated that this approach can overcome policy complexity and usability issues of previous schemes.**

Keywords-application-oriented access control, policy abstraction, a priori policy specification, functionality-based application confinement, policy usability

I. INTRODUCTION

Access controls are typically designed to protect resources from users. In these schemes each process is restricted based on its user identity, and treated as if it is acting on behalf of the user. However, this user-oriented approach is insufficient, as processes do not necessarily act in the best interest of users: for example, as is the case with malware and software vulnerabilities. Application-oriented access controls mitigate this threat by restricting the actions of each application.

Application-oriented schemes include isolation-based containers or sandboxes, such as `chroot()`, BSD Jails [1], and Danali [2]. These schemes require duplicated and often redundant resources, and limit the sharing of resources between applications, which can interfere with natural workflows. Other rule-based schemes provide controls over shared resources allowing applications to access the same resources in a restricted manner. These schemes include TRON [3], Systrace [4], and domain and type enforcement (DTE) [5]. Two established security mechanisms that provide rule-based application-oriented access control on Linux are SELinux [6] and AppArmor (previously known as SubDomain) [7]. Due to policy complexity, these schemes typically rely on learning modes to generate policy. Arguably policy complexity and usability issues have limited the adoption of these schemes.

This paper presents a new policy language, known as functionality-based application confinement policy language (FBAC-PL), that can overcome many of the policy complexity issues with previous finely-grained rule-based application-oriented access controls, and can in many cases facilitate the creation of complete policies *a priori*: that is, without having to first run the program being confined. FBAC-PL was designed to express policies based on the functionality-based application confinement (FBAC) model [8, 9]. FBAC restricts programs in terms of the features they provide, using policy abstractions known as *functionalities*. Functionalities are hierarchical; that is, they can contain other functionalities for layers of abstraction and encapsulation. Functionalities are also parameterized, so that they can adapt to the specific needs of each application. A Linux security module (LSM)-based implementation, known as FBAC-LSM was developed, which represents policy using FBAC-PL[1].

II. FBAC POLICY LANGUAGE OVERVIEW

FBAC-PL expresses functionality-based application-oriented access control policies. FBAC-PL is a name-based policy language; that is, it authorises access based on the names of resources. FBAC-PL policy is centrally managed and can simultaneously specify mandatory and discretionary controls. Policies include access rules and metadata used for policy administration and automation.

FBAC-PL expresses FBAC policies in three different types of policy files:

- *confinements*, which specify the sets of application restrictions that apply to users and who is authorised to make changes to application policies; this defines the system wide configuration;
- *application policies*, which specify how applications are identified and how they are restricted; and,
- *functionalities*, which are used as modules for specifying application policies.

[1] FBAC-LSM is available as free open source software at: http://schreuders.org/FBAC-LSM

III. NAME-BASED PATTERN MATCHING

Two distinct approaches to access control mediation have emerged: label-based and name-based mediation. FBAC-PL and the FBAC-LSM implementation take a name-based approach. Using name-based mediation, resources are protected based on their names rather than via labels attached to objects. For example, access to files is mediated in terms of their pathnames rather than the labels associated with the files. Name-based protection provides the flexibility of centrally managed access control policy, as distinct from the management of files. Also, the needs of specific applications can be described in terms of resources that overlap with the needs of other applications, and policies can be defined by multiple users, without the overhead of managing multiple labels for each resource. For these reasons, FBAC is believed to be well suited to a name-based implementation. However, a label-based implementation (although more complex) would also be possible.

Resource descriptors in FBAC-PL take the form of simple patterns. Wildcard patterns provide support for describing multiple files, directories, IPv4 IPs, and ports using pattern matching techniques. Briefly, these patterns can describe resources in the following ways. Files and directories can have asterisks to match any valid characters in a pathname. "**" allows "/" to be included, which means that subdirectories can be included, while "*" does not. This is designed to be similar to AppArmor's wildcard file matching. IP matching is very simple where a direct IPv4 address can be used, or an octet can be replaced with an asterisk, which can represent any number. Ports can either be a particular number, all "*", or a range such as "6667-7000".

IV. SPECIFICATION KEY

FBAC-PL is specified using Backus-Naur Form (BNF) [10]. BNF is a common method for describing programming language syntax and the format of files or information. Many variations of BNF exist. The specific style used here is a form of Extended Backus-Naur Form (EBNF), similar to that used in the XML standard published by the World Wide Web Consortium (W3C) [11], and is used as described below.

Policies are defined as sets of rules in the form: *symbol ::= expression*. Where the *symbol* is nonterminal (variable) and the *expression* is defined in terms of symbols or terminals (literal values). Literal values that appear in expressions are underlined and italicised. The '↵' character represents a line feed. A '|' in an expression specifies an alternative valid value. Square brackets, '[' and ']', are used to group together parts of an expression to define the scope and precedence of alternatives and quantifiers. In EBNF, regular expression characters are used to denote quantification. The characters '*', '+' or '?' can follow groups, literals, or symbols. The '*' character specifies they can occur zero or more times, '+' indicates one or more times, and '?' indicates zero or one time. Symbols and literals in expressions are concatenated. Space and white space in policy is explicitly stated using the '*s*' or '*ws*' symbols, which are defined within the specification.

V. FBAC-PL SPECIFICATION IN BNF

The specification of FBAC-PL using BNF is as follows:

confinements_policy_file ::= conf_policy*
conf_policy ::= *application_confinement* s confinement_name ↵*{*↵ ws? active_default ↵ ws? application_policy_location ↵ ws? functionality_policy_location ↵ ws? applies_to ↵ ws? maintained_by ↵ ws? task_with_no_profile ↵*}*↵
confinement_name ::= astr
active_default ::= *active_state* s active
active ::= *active* | *inactive*
application_policy_location ::= *application_policies* s [file | directory]
functionality_policy_location ::= *functionality_policies* s [file | directory]
applies_to ::= [only_applies_to_users s user_list | does_not_apply_to_users s user_list | applies_to_all_users]
maintained_by ::= application_policies_maintained_by s user_list
task_with_no_profile ::= *task_with_no_profile* s no_profile_action
no_profile_action ::= unconfined | confine_with_restricted_profile | deny_execution
audit ::= ws? *audit* s audit_when
audit_when ::= *all* | *denied* | *none*
user_list ::= i_1, i_2, …, i_n

functionality_policy_file ::= func_policy*
func_policy ::= *functionality* s functionality_name ↵*{*↵ [ws? functionality_level ↵]? [ws? functionality_description ↵]? [ws? functionality_category ↵]? [ws? functionality_suggest_when ↵]? [ws? parameter_definition ↵]+ [ws? contained_functionality ↵]* [ws? contained_privilege ↵]* [ws? macro ↵]* *}*↵
functionality_name ::= astr
functionality_level ::= *highlevel;* | *lowlevel;* | *baselevel;*
functionality_description ::= "str";
functionality_catagory ::= category s [misc | file_editor | file_viewer | game | network_client | network_server | system_tools | platform];
functionality_suggest_when ::= *suggest_functionality* s [in_iconcategory | uses_lib];
in_iconcategory ::= *iconcategory* s "str"
uses_lib ::= uses_library s "str"
parameter_definition ::= parameter_declaration ↵[parameter_type ↵]? [parameter_automation ↵]?
parameter_declaration ::= *parameter* s parameter_name s suggested_value;
parameter_type ::= ws? *parameter_type* s directory;
parameter_automation ::= ws? *parameter_automate* s [find_path | *usedefault*];
parameter_name ::= astr
find_path ::= (searchforpathmatching | searchfordircontaining) s string_pattern
suggested_value ::= resource_list
contained_functionality ::= *functionality* s functionality_name*(* arg* *);*
contained_privilege ::= *privilege* s operation [s objects]?;
objects ::= $resource\_list_1$, $resource\_list_2$, ..., $resource\_list_n$;

application_policy_file ::= app_policy*
app_policy ::= *application* s application_name ↵*{*↵[ws? executable_path ↵]+ [ws? contained_functionality ↵]* [ws? contained_privilege ↵]* [ws? macro ↵]**}*
application_name ::= astr
executable_path ::= *executablepaths* $path_1$:$path_2$:…:$path_n$;
path ::= nwsstr

arg ::= ws? [parameter_name=]? argument
argument ::= resource_list | *<default>*
resource_list ::= "resource_descriptor" | *{*"resource_descriptor"$_1$: "resource_descriptor"$_2$:... : "resource_descriptor"$_n$*}*
resource_descriptor ::= file | directory | protocol | port | IPv4 | string_pattern
macro ::= *macro* [*permission_directory_path* s operation_list, s directory_list, s path_rule_list | *permission_path* s operation_list, s resource_list];
operation_list ::= "operation" | *{*"operation"$_1$: "operation"$_2$:... : "operation"$_n$*}*
directory_list ::= "directory" | *{*"directory"$_1$: "directory"$_2$: ... :"directory"$_n$*}*
path_rule_list ::= "str" | *{*"str"$_1$:"str"$_2$: ... :"str"$_n$*}*
operation ::= *file_read* | *file_write* ...
file ::= /str[/str*]? | *

directory ::= /str[/str*]?/ | *
protocol ::= *UDP* | *TCP* | *RAW*
port ::= porti | * | porti - porti
IPv4 ::= oct.oct.oct.oct | *
oct ::= ipi | *

s ::= (space)
ws ::= [? (space) | ___? (tab) | ~~?~~? (new line)] ws*
str : String
nwsstr : String with no whitespace
astr : Alphanumerical String (a-z & A-Z & 1-9 & - & _)
int : `Integer`
porti : Integer (1 – 65353)
ipi : Integer (1-255)

Although white space indicated by the symbol '*ws*' is optional, a tab is recommended for policy readability. In addition, after optional white space, any lines beginning with a hash ('#') are ignored and can be used to comment policy. FBAC-LSM is not strict about the order of the elements in the **conf_policy**, **func_policy**, and **app_policy** expressions as specified above.

VI. CONFINEMENTS

The confinements policy is stored in the directory `/etc/fbac-lsm` in the file `confinements.fbac`. This file is maintained manually by an administrator and defines and configures all the sets of rules that apply to the users specified within the file. This file is the system-wide configuration file for the FBAC-LSM mechanism. Figure 1 demonstrates a possible `confinements.fbac` file. In this example one confinement is specified: a discretionary control. The discretionary control applies to the user whose user identity (uid) is 1000 (which in this case corresponds to the user 'cliffe') and is maintained by that same user. Using FBAC-PL, adding additional mandatory or discretionary controls that are simultaneously enforced is straightforward, they are configured in this file.

```
application_confinement cliffes_dac_confinement
{
    active_state active

    application_policies "/etc/fbac-lsm/applications/cliff-
    es_dac/"
    functionality_policies "/etc/fbac-lsm/functionalities/"

    only_applies_to_users 1000
    application_policies_maintained_by 1000
    task_with_no_profile unconfined
    audit denied
}
```

Figure 1. Example Confinement Policy

VII. APPLICATION POLICIES

The policies for restricting applications are stored in the file or directory specified in a confinement policy. Users specified via the '*application_policies_maintained_by*' value in a confinement are authorised to add, edit or remove these application policies. Users are not required to be familiar with the FBAC-PL syntax as they specify policy using the policy manager, a graphical tool that creates the FBAC-PL policy and writes the policy files on the behalf of authorised users. Users are not allowed direct access to the application policy files.

Figure 2 shows an example application policy file, `konversation.fbac`, which was generated by the policy manager to restrict the application Konversation. Konversation is a graphical IRC client. The paths to its executables are specified, then the functionalities describing the security goals of the application are specified and parameterised.

```
application konversation
{
    executablepaths /opt/kde3/bin/konversation;
    functionality Standard_Graphical_Application
        (peruser_directory="/home/*/.kde/share/apps/konver-
        sation/",
        peruser_files="/home/*/.kde/share/config/konversat-
        ionrc",
        application_libraries_directory="",
        libraries_fileextension=<default>,
        config_directory="/home/*/.kde/share/apps/konversa-
        tion/",
        config_files="",
        read_only_directory="/opt/kde3/shar/apps/konversat-
        ion/");
    functionality Irc_Chat_Client
        (chat_IRC_servers=<default>,
        IRC_remote_port=<default>,
        save_received_files_in_directory="/home/cliffe/dow-
        nloads/",
        send_files_in_directory="");
    functionality Uses_Perl ();
}
```

Figure 2. Example Application Policy for Konversation

The three functionalities specified are *`Standard_Graphical_Application`*, which is the base-level functionality describing the type of user interface, the *`Uses_Perl`* platform functionality, and the *`Irc_Chat_Client`* functionality. All three were automatically suggested by the policy manager based on the iconcategory specified in the `konversation.desktop` file and the application's dependencies. Functionalities for applications can also be specified manually by users based on their expectations of the features provided by applications.

The parameters specified for the *`Irc_Chat_Client`* functionality authorise Konversation to connect to any IRC server on the default IRC port, and save files to a download directory. As shown in Figure 2, FBAC-LSM functionalities are passed arguments in a fashion similar to subroutines in programming languages. This allows the policy abstraction to easily adapt to the differing details of applications providing related features.

VIII. FUNCTIONALITIES

Functionalities are the building blocks of policy and are reused for multiple confinements and applications. Functionalities are stored in the source specified in the `confinements.fbac` file. The names functionalities use follow a naming convention, where high level and base functionalities use mixed case. For example, *`Web_Browser`*, and all lower level functionalities use all lower case, such as *`file_r`*.

Figure 3 shows an example of a functionality that represents a high-level program feature, *`Irc_Chat_Client`*. As shown in the figure, functionalities can include additional information used by the policy manager to manage and automate application policy construction. In the figure, bold text denotes policy that can be enforced by the LSM, while the information in italics is metadata only used by the policy manager for management and automation.

```
functionality Irc_Chat_Client
{
    functionality_description "An irc (chat)
    client.";
    highlevel;
    category network_client;
    suggest_functionality iconcategory "IRCClient";
    suggest_functionality uses_library "python-irclib";

    parameter chat_IRC_servers "*";
    parameter_description "the remote servers the program
    can connect to to chat with IRC and send files with
    DCC";
    parameter_type IP;
    parameter_automate usedefault;

    parameter IRC_remote_port {"6665-6669":"7000":"194":"9-
    94"};
    parameter_description "the local chat (IRC) port.
    Usually 6667 or nearby (6665-6669) or rarely 194 or 994
    (secure)";
    parameter_type port;
    parameter_automate usedefault;

    parameter save_received_files_in_directory "/home/*/do-
    wnloads/";
    parameter_description "the directories received files
    are saved to";
    parameter_type directory;

    parameter send_files_in_directory "/home/**/";
    parameter_description "the directories files to send to
    others are in";
    parameter_type directory;

    functionality IRC (chat_IRC_servers, IRC_remote_port);
    functionality IDENT ( );
    functionality DCC
        (chat_IRC_servers,
        save_received_files_in_directory,
        send_files_in_directory);

    functionality general_network__connectivity_awareness_-
    and_common_file_access ( );
}
```

Figure 3. Example High-Level Functionality Policy: Irc_Chat_Client

The definition starts with a description, and includes the "highlevel" directive to specify, for the policy manager, that it is a high-level functionality. This functionality is assigned a category, "network_client", which is used to group related functionalities to ease the process of selecting functionalities. As shown in the example, functionality suggestion directives can be used to specify attributes of applications that are likely to use this functionality. Given the presence of this example functionality, when creating a policy for an application, if the application has the "IRCClient" icon category specified in its '.desktop' file or it depends on the "python-irclib" library, the `Irc_Chat_Client` functionality is automatically suggested.

Parameters are also specified, which are used to adapt the functionality to the needs of specific applications. The definition of the parameter files also contains a default argument value that is used when the argument "<default>" is passed to a parameter. This is similar to a feature of programming languages such as Python, C++, and Windows PowerShell that enables subroutine parameters to have default values. This feature allows further abstraction in common cases without sacrificing flexibility. Parameters can also have methods for automating argument specification. For example, in the figure the line "`parameter_automate usedefault`" instructs the policy manager to use the default values when automating arguments. Other methods for automation of arguments exist, such as searching for directories matching patterns containing the application's name.

The hierarchical containment relationship between functionalities enables arguments to propagate to contained functionalities. For example, when an application policy includes the `Irc_Chat_Client` functionality, the value for the *chat_IRC_servers* parameter is passed as an argument. If an IP address is specified in the application policy, the `Irc_Chat_Client` functionality will consequently only grant access to browse web resources on the named host. This is achieved as a result of this information propagating from functionality to contained functionality until the information is used in the definition of a privilege.

IX. Conclusions

This paper has presented an application-oriented access control policy language, known as FBAC-PL. FBAC-PL can express policies for restricting applications based on the features they provide, and encapsulates policy details using reusable parameterised abstractions. FBAC-PL can include metadata that facilitates automation that is not available in other existing schemes. A Linux-based implementation, FBAC-LSM, has demonstrated the advantages of this new policy language. This new approach to policy poses unique opportunities to further improve the usability of application-oriented access controls.

References

[1] P.-H. Kamp and R. Watson, "Jails: Confining the Omnipotent Root," in *2nd International System Administration and Networking Conference (SANE 2000)* Maastricht, The Netherlands, 2000.

[2] A. Whitaker, M. Shaw, and S. D. Gribble, "Denali: Lightweight Virtual Machines for Distributed and Networked Applications," in *5th USENIX Symposium on Operating Systems Design and Implementation* Boston, MA, USA: USENIX Association, 2002, pp. 195–209.

[3] A. Berman, V. Bourassa, and E. Selberg, "TRON: Process-Specific File Protection for the UNIX Operating System," in *Winter USENIX Conference* New Orleans, LA, USA: USENIX Association, 1995, pp. 165-175.

[4] N. Provos, "Improving Host Security with System Call Policies," in *12th USENIX Security Symposium* Washington, DC, USA: USENIX Association, 2002.

[5] L. Badger, "A Domain and Type Enforcement UNIX Prototype," *Computing Systems*, vol. 9, 1, pp. 47-83, 1996.

[6] P. Loscocco and S. Smalley, "Integrating Flexible Support for Security Policies into the Linux Operating System," in *FREENIX Track: 2001 USENIX Annual Technical Conference* Boston, MA, USA: USENIX Association, 2001, pp. 29-42.

[7] C. Cowan, S. Beattie, G. Kroah-Hartman, C. Pu, P. Wagle, and V. Gligor, "SubDomain: Parsimonious Server Security," in *USENIX 14th Systems Administration Conference* New Orleans, LA, USA: USENIX Association, 2000.

[8] Z. C. Schreuders and C. Payne, "Reusability of Functionality-Based Application Confinement Policy Abstractions," in *10th International Conference on Information and Communications Security (ICICS 2008)* Birmingham, UK: Springer, 2008, pp. 206-221.

[9] Z. C. Schreuders and C. Payne, "Functionality-Based Application Confinement: Parameterised Hierarchical Application Restrictions," in *SECRYPT 2008: International Conference on Security and Cryptography* Porto, Portugal: INSTICC Press, 2008, pp. 72-77.

[10] E. K. Donald, "Backus Normal Form vs. Backus Naur form (Letter to the Editor)," *Communications of the ACM*, vol. 7, 12, pp. 735-736, 1964.

[11] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau, World Wide Web Consortium (W3C), W3C Recommendation (http://www.w3.org/TR/REC-xml/):"Extensible Markup Language (XML) 1.0 (Fifth Edition)," 2008.