SPCL: Structured Policy Command Language

Michael Locasto Matthew Burnside Chun Li Aron Wahl

Department of Computer Science
Fu Foundation of Engineering & Applied Science
Columbia University
{*locasto,mb@cs.columbia.edu*}
{*alfredli1,aronwahl@hotmail.com*}

Chapter 1. SPCL Overview

1.1 Introduction

The SPCL is designed to address a number of issues found in policy and permission systems. The notion of a policy is central to the design and operation of most complex systems. Because the concept of policy is a nebulous one, efforts to formalize it have suffered from both specialized domain requirements and the inherent complexity of policy decisions. The SPCL is an effort to provide a domain-independent language for specifying policy requirements, principals, rules, and actions in a lightweight syntax with semantics as close to natural expression as possible, in order to avoid esoteric and complex programming constructs. If need be, SPCL can be compiled to another form, such as that supported by the Keynote trust management system.

Current policy systems are either built into a system (making them hard to adjust or replace) or too course-grained to be effective and easy to use. There is a clear need for a policy language that greatly simplifies the task of policy specification and separates policy specification from policy implementation. In addition, policy is often specified by non-programmers, so any language should be sufficiently easy to learn while retaining powerful specification abilities for systems programmers or application architects.

It is important to note that policy is not merely access control. Policy is essentially about expressing a want or desire. Therefore, we can conclude that policy is about *specifying*, *enforcing*, and *revising* requirements. Any system built to handle policy should provide specification, enforcement, and revision mechanisms.

1.2 Background

1.2.1 What is Policy?

Policy is generally considered to be a plan or course of action intended to influence decisions or outcomes. A policy is a set of guiding principles whereby some outside agency can match its current environment or task with the guidelines in the policy and proceed with a course of action.

Policy decisions range from very simple (accept or reject access request N) to very complex (never let the dogs outside unless someone is home and their collars are on and the electric fence is on and it is not raining) to extremely complex (allow half of this customer's address space to advertise their prefix with us, but do not let

their traffic use us as a transit network, unless the lawyers say otherwise).

Policy is ubiquitous in computing systems, and is often implicitly coded into a system's structure by functional requirements, language features, and design decisions. Policy is not easily managed because it is so high-level and rarely translated to formal program semantics.

1.2.2 What are some examples of current policy mechanisms?

A language-specific attempt to provide a policy mechanism is the Java Policy and Permissions objects. While the Java approach is a fairly good attempt, it suffers from two major weaknesses: difficulty of use and lack of a fine-grained approach. The file used to control the policy rules in the currently running JVM is often ignored because it is an additional layer of complexity in the design, construction, and testing of a system. The default file controlling Java policy (if a SecurityManager is invoked in the current JVM) is located in the `JAVA_HOME/jre/lib/security/java.policy` file, and has a series of "grant blocks" that allow a set of classes to have a predefined set of permissions (essentially an arbitrary collection of statements permitting the set of classes to invoke certain language functions). The Java policy file syntax is roughly the following:

```
grant SOME_CLASSES{

  permission SOME_PERMISSION "args";
  permission ANOTHER_PERMISSION "args";

}
```

In addition, the default policy mechanism is very course-grained and although it is easily extensible, does not lend itself to quick analysis of any given software. Even though there are a number of permissions to legislate, this set of permissions is not provably complete or precise enough for many complex applications.

Another effort to specify a policy language is the Web Services Security Policy Language by IBM, Microsoft, Verisign, and RSA. However, this policy language is domain-specific and meant to provide an XML-grammar based mechanism for policy assertions for web services.

Yet another system that involves policy specification is KeyNote, a trust management and policy system. While KeyNote is a powerful mechanism, its policy specification syntax is not for the faint of heart.

1.3 SPCL Features

1.3.1 SPCL Highlights

SPCL's primary two features are an object-oriented-like command language (and compiler) and a PolicyEngine interpreter acting as an oracle to a number of other systems. SPCL's syntax is closest in appearance to a mixture of Java and SQL. The PolicyEngine should be thought of as a VM for policy.

A common pattern for specification is of the form:

```
{command} {principal} performs {action} on {object} when {conditions}
```

A policy is specified in the SPCL syntax, parsed, and then translated to a form suitable for input to the PolicyEngine. The specification for the PolicyEngine is platform-independent, so it can be implemented for any target platform. After parsing and construction, it then loads and interprets the policy. The PolicyEngine has a simple protocol for asking questions about the running policy (in SPCL) as well as loading new policies in, updating current policy, and retiring outdated policy.

1.3.2 SPCL Goals

The SPCL has three primary goals that correspond to the essential components of a policy mechanism. SPCL must support policy specification, enforcement, and revision.

First, in order to support **policy specification**, the SPCL should provide a syntax that is close to the natural language method of specifying policy rules and permission constraints. SPCL is necessarily a high-level language. Ease of use and ease of specification is expected to increase because of this goal.

In the absence of natural language processing support for policy (which may be rectified by research conducted by the Navy), this pattern of principals, actions, and objects very naturally suggests itself. SPCL, like SQL, is a language that seeks to be easy to use while remaining focused, formal, and powerful.

The important point here is the focus on the user of the system. The user is actually writing the policy as they express their desires. However, they should have to do a very small amount of work to express their policy formally, and this work should not involve strange syntax or traditional source code structures a 'programmer' would know how to manipulate. This requirement is aided by the simple command-oriented nature of policy. Most policies have little hierarchical structure; they specify a set of actions under given conditions.

Second, in order to support **policy enforcement**, the SPCL should have semantics that are not costly to interpret, have little or no side effect other than the intended goal of the policy, and are domain independent.

Third, to support **policy revision** SPCL should have some idea of system lifecycle. Most policy is implemented implicitly or statically in program code and is difficult and expensive to change. Policy needs to be dynamically updateable, and should respond to a lifecycle model. Knowledge of the lifecycles for both the policy and the systems the policy governs will allow the PolicyEngine to make informed and powerful decisions. Users must be able to create, alter, and retire their policy easily, without a costly impact on the rest of the system.

1.4 Example Policy Domains

1.4.1 Authentication & Authorization

Authentication and authorization is the most simple and common sort of policy implementation in systems today. SPCL is useful for this base case of binary accept or reject decisions, such as those faced during a login procedure or access request. An example policy for authentication is:

```
Reject any user who provides the wrong password.
```

```
Reject any user who has three successive bad login attempts.
```

1.4.2 Firewall Rules & Packet Filtering

Filtering IP packet traffic is another application of policy. The network administrator desires traffic into and out of her network to have certain characteristics. Packet filters are often simple tables of rules. Traffic is pattern matched against the table, and the specified action is performed. A common tool for packet filtering is the Linux tool *ipchains* and has syntax like the following:

```
ipchains -A input -j ALLOW -p tcp -s 0.0.0.0/0 -d www.mycompany.com 80
```

to express a policy of:

```
Allow anybody to access my webserver over TCP on port 80.
```

Note the natural structure of the command, however:

```
Add an 'ALLOW' rule to my 'input' policy for a principal '0.0.0.0/0' on my object
'www.mycompany.com' with the condition that the protocol is TCP.
```

1.4.3 BGP

BGP, or Border Gateway Protocol (v4), is the protocol used on the Internet to advertise routes between computers. BGP does not calculate the *optimal* path (shortest path) for traffic; rather, BGP calculates the *best* path for traffic, where best is defined by an arbitrary policy: legal and business agreements.

1.4.4 House Monitoring System

The application that SPCL focuses on for motivating examples is a home monitoring system. This system is a policy-aware control application for home appliances. Homes and families have many arbitrary rules that govern use of various appliances and tasks that need to be done in order for the home to run smoothly.

1.5 Example SPCL Syntax

```
/** This policy applies to the zone 'home'. */
zone home;

/**
 * Some comments about the source policy.
 * This is an illustrative example of SPCL Syntax.
*/
policy MyPolicy{

        default{
                DENY * {NOTIFY police;}
                ALLOW "channel 5" BY Mom,Dad,Ken ON tv WHEN (time=1700){NOTIFY Mom;}
        }
```

```
        group parents{
                ALLOW * ON *;
        }

        group children{
                DENY * ON *;
        }

        principal police{
                phone = "911";
        }

        principal Mom{
                public key = "ldap://192.168.4.240/mom/keys/pub.key";
                email = "mom@31north.home";
                alias wonderWoman, admin;
                groups parents,students;
        }
        principal Dad{
                ALLOW *; ALIAS father; GROUPS parents;
        }
        principal Ken{
                ALLOW "GET /index.html HTTP/1.1" ON webserver
                  WHEN (state="summer" AND time>1800){NOTIFY logger;}
                ALLOW "open door" ON fridge;
                ALLOW "play" ON outside
                  WHEN (state="sun shining" AND Ken.homework=true)
                        { if( time>1400 )
                                NOTIFY Police;
                          else
                                NOTIFY Mom;
                        }
                GROUPS students,children;
        }

        object fridge{
                url = "appliance://FF::AB:34:22/kenmore1/";
                actions{
                        rule rule1 = "ls [a..z]*";
                        rule1.meta-action = "NOTIFY Mom";
                        rule1.default-action = "DENY";
                }

        }

        object webserver{
                url = "http://www.thesmiths.home:80/";
                actions{
                        rule rule1 = "GET [a..z]*.html HTTP/1.[1|0]";
                        rule1.meta-action = "NOTIFY Mom";
                        rule1.default-action = "DENY";
                }
        }
}
```

1.6 SPCL Language Primitives

The language is a declarative-style specification language; it is not an imperative one. The language does allow for traditional branching control flow (if..then..else) even though its model of computation is primarily event-based. It has the following primitives (some of which directly translate to reserved words):

- zone - an area containing many policy definitions generally under 1 administrative domain.
- policy - a policy definition
- principal - an object in the system that acts autonomously
- object - an object in the system that is acted upon
- group - a group of principals
- action - a flexible but concrete notion of what a principal attempts to get permission for
- rule - a regular expression for a class or group of actions
- condition - an expression to be satisfied in order for the policy engine to respond to the user.
- meta-action - the action taken by the system in response to a principal's request to perform an action on an object
- constraint - reserved for future use
- requirement - reserved for future use
- assertion - reserved for future use
- demand - reserved for future use
- consequence - reserved for future use

The language operates with a data model that focuses on events that affect a three-tiered map of principals, groups, and objects. Events can query the state of the map (a relationship between map objects) or alter the state of the map (change the current policy), and three precedence levels: user, group, and default group. The policy engine will match rules beginning with the specific user, then against groups the user belongs to, and finally against the default group.

1.7 Conclusions

SPCL is a language meant to simplify the expression of arbitrary policy. The primary goals of SPCL is to formalize the expression of policy, separate the specification of policy from the implementation of the system the policy governs, and provide the system with a lifecycle-aware policy mechanism. Meeting these goals provides a policy mechanism that supports the specification, enforcement, and revision of policy.