Policy

Patrick McDaniel
AT&T Labs – Research
pdmcdan@research.att.com

Policy describes how environments dictate the behavior of applications and services. Security policies specify how relevant conditions mandate how, when, and/or to whom access to controlled resources is given. For example, the access rights defined by a UNIX file-system state a policy: read, write, and execute bits define the kinds of operations a user is permitted perform on the files or directories. The policy is *evaluated* by determining whether the user has sufficient permissions to perform the operation at the point at which access is attempted. The policy is *enforced* by the file-system by allowing or preventing the operation.

Security policy has historically been divided into two broad classes: provisioning policy and authorization (or access control) policy. Provisioning policies define how software is configured to meet the requirements of the local environment. As is illustrated by the UNIX file-system policy, authorization policies map entities and resources onto allowable action. The following considers these broad classes in serial.

One can view any kind of software configuration as provisioning policy. That is, any aspect of software behavior configured at run-time is policy. This is relevant to the current discussion where configuration affects how security is provided. For example, one of the central goals of the `ssh` remote access utility [1] is to provide confidentiality over the session. The `ssh` policy states which cryptographic algorithm (e.g., <u>3DES</u>, <u>Blowfish</u>). should be used to encrypt session traffic (e.g., as specified by host-local configuration file). The policy (i.e., encryption algorithm) dictates how the goal (i.e., confidentiality) is achieved. In general, the degree to which a user or administrator can influence security is largely dictated by the scope of the system's provisioning policies.

General-purpose policy management services support the creation, storage, and enforcement of policy. Note that while these services can be used to manage authorization policy, they have historically been targeted to provisioning policy. The IETF Policy Framework Working Group (PWG) has developed a widely adopted reference architecture and lexicon for policy management [2]. This framework defines a collection of components and operations used to manage the network (commonly referred to as *policy-based networking*). Depicted in Figure 1, the architecture defines four logical policy components; *policy editors*, *policy repositories*, *policy consumers*, and *policy targets*.

A policy editor provides interfaces for specifying and validating policy specifications. The policy editor is responsible for detecting (and potentially resolving) inconsistencies in the specification. For example, a `ssh` policy that mandates both <u>DES</u> and <u>3DES</u> be used for confidentiality is erroneous (only one algorithm should be specified). Such a policy would be flagged as errored, and where available, corrected using a resolution algorithm. The policy editor delivers validated policies to policy repositories as dictated by the environment. A policy repository stores the policies to be used by an administrative domains. The policy repository does not act on or interpret policy.

A policy consumer translates policy into action. The consumer acquires and evaluates the policy relevant to the current environment. The resulting action is communicated to the set of policy targets. Targets enforce policy by performing the actions that implement the defined semantics. For example, again consider the `ssh` policy. A consumer would interpret policy to determine which algorithm is to be used for confidentiality. `ssh`

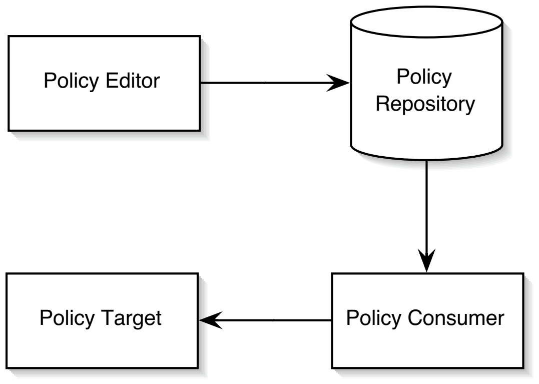

Figure 1: IETF Policy Framework working group architecture - architecture supporting the creation, distribution and enforcement of network management policies.

Role	Permissions			
	sell	*bid*	*clear*	*view*
Seller	yes	no	no	yes
Buyer	no	yes	no	yes
Auctioneer	no	no	yes	yes

Table 1: Example role-based policy for online auction application.

would then act as a policy target by configuring the `ssh` client and subsequently using it to encrypt the traffic (and thus enforce confidentiality).

Authorization policy describes to whom and under what circumstances access to resources is granted. These policies are further defined by an *authentication* policy and an *access control* policy. The authentication policy states how the identity of the requesting entity must be established. For example, an authentication policy for a UNIX system is the password: the user must provide the appropriate password at the login prompt to be allowed access to the system. How authentication is performed is largely defined by environment needs, and outside the scope of this section.

An access control policy maps an identity established during authentication and other information to a set of rights. Rights, often called permissions, defines the types of operations that can be granted, e.g. `read`, `write`, and `execute` on a UNIX file system. The structure and meaning of these policies are defined by their *access control model*. (there are many models, we choose to focus on one)

One popular model is the *role based* access control model. In this model, the policy defines collections of permissions called *roles* [3]. Users assume roles as they perform different tasks within the system. Hence, the set of rights is strictly defined by the set of rights allowed to the roles they have assumed.

The following example illustrates the use of role based access control policy. Consider a simplified sealed bid online auction application. Three entities participate in this application: an auctioneer, a bidder, and a seller. The bidder *bids* for goods *sold* by the seller. Once all bids are placed, the auctioneer *clears* the auction by opening the bids and declaring the highest bidder the winner. Once a winner is declared, interested parties can *view* the result. The access control policy for the online auction defines four permissions (described above), bid, sell, clear, and view, and three roles, seller, buyer, and auctioneer. Table 1 describes a policy that assigns the permissions to roles.

The auction policy is enforced at run-time by evaluating the permissions associated with the roles that they

```
KeyNote-Version: "2"
Authorizer: "DSA:4401ff92"   # the Alice CA
Licensees: "DSA:abc991"      # Bob DSA key
Conditions: ((app_domain == "ssh") && (crypt == "3DES"));
Signature: "DSA-SHA1:8912aa"
```

Figure 2: KeyNote Credential - specifies a policy stating that Alice allows Bob to connect via SSH if 3DES encryption is used.

have assumed. In any role-based system, a user assumes roles through a explicit or implicit software process. The example online auction maps user accounts to roles. For example, the *sally* account is mapped to the seller role. When a user logs into the application as *sally*, she automatically (implicitly) assumes the *seller* role. From that point on, she is free to perform any action allowed by that role. Other accounts are similarly mapped to the *buyer* and *auctioneer* roles, and governed by the associated policy.

One might ask why we do not just assign a user the union of all rights assigned to each role they may assume. Firstly, there may be legitimate reasons that a user be explicitly prohibited from simultaneously assuming more than one role. For example, in the above example it may be important that a user be prevented from assuming both auctioneer and bidder roles. If a user were to assume both roles, it could “cheat” by viewing all bids before placing its own.

The second reason role based models are useful is convenience. Roles provide a powerful abstraction for dealing with the potentially many rights that must be managed. In separating the access control from the user, one simplifies the process of evaluating. Moreover, this eliminates the need to alter access control policy each time a entities position changes.

Many access control models have been defined, studied, and ultimately used in real environments. Each model defines a unique way of viewing the relations between protected artifacts, actions, and the entities that manipulate them. Because applications and environments view these concepts in vastly different ways, it is unlikely that any one model will be universally applicable. Hence, like the use policy itself, the selection of a model is a function of taste and system requirements.

Trust Management (TM) systems blur the lines between authorization and provisioning policy. TM policies, frequently called *credentials*, define statements of trust. More specifically, they state that an entity is trusted to access a particular resource under a specified set of conditions. A entity supplies the appropriate set of credentials when accessing a protected resource.[1] If the supplied credentials are authentic and consistent with a *local policy*, the entity is permitted access.

Note that the TM policy alters the traditional policy flow: each entity accessing the resource supplies those policies that are needed to perform an action. This eliminates the need for the policy consumer to discover and acquire policy. It is incumbent upon the user to supply the set of credentials that allow access. Hence, TM systems enable policy in creating widely distributed systems.

Provisioning policy is specified in TM systems through the access criteria. Hence, TM systems do not specify provisioning directly, but mandate how the environment provisioning must be provisioned to allow access. This model of policy again departs from traditional uses: the policy infrastructure passively assesses whether the environment is correctly provisioned, rather than actively provisioning it.

The KeyNote system [4] provides a standardized language for expressing trust management policies. KeyNote credentials have three cryptographic components: an *authorizer*, a *licensee*, and a *signature*. The authorizer identifies the authority issuing the policy. KeyNote authorities are cryptographic keys. The signa-

[1]Policies can define access to actions, rather than resource. However, the evaluation of such policies relating to action is identical to resource-oriented policy.

ture creates a cryptographically strong (and verifiable) link between the policy and the authorizer. The licensee is to the entity to which the policy refers, e.g., the entity to be allowed access.

KeyNote credentials express a *says* relation: an authority says some entity some aspect has some rights under a set of conditions. To illustrate, the KeyNote credential defined in Figure 2 expresses a policy governing `ssh` access. In this credential the authorizer authority states that an licensee entity has the right to access the host only if 3DES is used to implement confidentiality. Note again that both the authorizer and licensee are not really entities, but keys. Hence, any entity that has access to the private key of the licensee, can use this credential to gain access to the host. Moreover, any entity with access to the private key of the authorizer can create and sign such credentials.

On first viewing, the credential in Figure 2 may appear to be ambiguous. The credential does specify which hosts are to be governed. This ambiguity is the source of much of the power of trust management. This credential is only accepted by those systems which have a *local policy* that (directly or indirectly) states that the authorizer has dominion over `ssh` access.

Note that the local policy need not specifically identify the authority in the above credential. The local policy can simply state the authorities it trusts to make policy decisions. Further credentials (typically supplied by the user) that express the delegation by the trusted authorities over `ssh` access are used to construct a delegation chain. Hence, access is granted where a chain of credentials beginning at the local policy and terminating at the access granting credential can be found.

Computing environments are becoming more fluid. Increasing requirements for software systems to be more open, and at the same time, more secure place unique demands on the supporting infrastructure. New environments and changing threats mandate that the kinds of security provided be reexamined and reconfigured constantly. Consequently, systems need to be more flexible and adaptive. Policy is increasingly used as the means by which correct behavior is defined.

Policy is not a panacea. While significant strides have been made in the construction, distribution and use of policy, many areas require further exploration. For example, we know little about the security implications of enforcing several policies (i.e., composing policies) within the same domain. Future policy systems need to find techniques that identify and mitigate interactions between policies. A more systemic area of investigation is scale: how do we deploy policy systems that are feasible in networks the size of the Internet. It is the answers to these questions, rather than the specifics of policy construction, that will determine the degree to which policy systems will be adopted in the future.

References

[1] Tatu Ylonen. SSH - Secure Login Connections Over the Internet. In *Proceedings of 6th USENIX UNIX Security Symposium*, pages 37–42. USENIX Association, June 1996. San Jose, CA.

[2] Internet Engineering Task Force. Policy Framework Working Group, November 2002. `http://www.ietf.org/html.charters/policy-charter.html`.

[3] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-Based Access Control Models. *IEEE Computer*, 29(2):38–47, 1996.

[4] M. Blaze, J. Feignbaum, J. Ioannidis, and A. Keromytis. The KeyNote Trust Management System - Version 2. *Internet Engineering Task Force*, September 1999. RFC 2704.