Partitioning Trust in Network Testbeds

Gary Wong, Robert Ricci, Jonathon Duerig, Leigh Stoller, Srikanth Chikkulapelly
University of Utah, School of Computing
{gtw, ricci, duerig, stoller, srikanth}
@cs.utah.edu

Woojin Seok
Korea Institute of Science
and Technology Information
wjseok@kisti.re.kr

Abstract

Traditionally, testbeds for networking and systems research have been designed as monolithic facilities: they contain a single root of trust. The resources in the facility are assumed to be administered by a single entity or a set of mutually-trusting entities. All user management, including vouching for users' identities and taking responsibility for their actions, is done using a flat trust structure or a simple hierarchy with the facility itself as the root. This design is not a good match for testbeds that are composed of multiple autonomous facilities, or in which different parts of the testbed operate under different trust models.

In this paper, we argue that partitioned trust is increasingly important in large scale and security-sensitive testbeds. We present a design that accomplishes this partitioning by using multiple trust roots. The trust domains created by these roots may decide, independently, how much trust to place in each other, and can apply policies based on the domain or principal that originates a request. The domains could represent separately administered facilities (as in a federated testbed), or they could represent sections within a single facility that run with different trust models (for example, with differing levels of security.) We have implemented this design in ProtoGENI, a control framework for federated testbeds; we include details of this implementation and share experiences from using it in an active deployment with hundreds of users.

1. Introduction

Distinct trust domains arise for a number of reasons in network experimentation environments. *Federated* facilities, in which locally autonomous testbeds pool their collective resources to form a large and loosely coupled environment, lead to complex systems with different parts owned and operated by different organizations. Other facilities are managed by a single entity, but encompass resources that operate at different security levels and therefore with complex trust relationships.[1] In general, as testbeds grow in size and scope, their designs must solve distributed systems problems, and concerns such as fault tolerance, decentralization, coordination and trust become more prominent. In all of these cases, we argue that it is important that the testbed be capable of treating users and resources from different trust domains differently.

However, current testbeds [22, 14, 13] operate within a single trust domain, even if different users are granted different permissions. The fundamental problem caused by this concentration of trust is that every facility provided by the testbed must be trusted equally. This model is not scalable to large sets of resources and users, nor is it suitable for support of operations which must conform to strict isolation requirements.

By partitioning trust, we propose to overcome these limitations; our model divides a testbed (or federation of testbeds) into multiple trust domains. Each object within the system belongs unambiguously to a single domain, and each facility has authority only over those objects which fall within its domain.[2] This tight binding is an essential requirement, and guarantees important properties beyond those merely arising from multiple trust anchors. For instance, the trust model used by Web browsers when communicating over TLS/SSL [2] makes use of many trust anchors — browsers typically trust hundreds of root CA certificates — but trust is not *partitioned*, and any CA is permitted to sign any certificate. Therefore, no guarantees about the security of the composite system can be made beyond that of the least trusted CA.

[1] As an example, consider a facility in which part of the testbed has strong safeguards in place to prevent exfiltration of malware, and part of the testbed has no such safeguards. An experiment vetted to run in the lower security environment may be trusted to run in either; an experiment that is authorized to run in the protected part of the testbed may be prohibited from running in the part that lacks exfiltration protection.

[2] Note that we make a subtle, but important, distinction between having authority over an object and making assertions about it; only authorities can attest to the identity of objects, while assertions (statements relating to that object) may be made by any party, and may be accepted by any entity that trusts the asserter.

Our model permits a hierarchical structure (that is, "sub-partitions" of trust). It is also transparent: when an object from one domain interacts with an object from another domain, both objects' true identities are exposed, so that policies can be based on the object's full identity, the domain to which it belongs, or both.

We begin by summarizing related work in this area in Section 2, and then give an overview of our design in Section 3. The details of our trust model, and the authentication and authorization mechanisms used to implement it, are presented in Section 4. We close in Section 5, discussing lessons we have learned during design, implementation, and deployment in a production federated testbed.

2. Related Work

Our design is related to, and motivated by, recent work on federated testbeds. In a federated model, each member of the federation maintains independent responsibility for maintaining its resources and local control over usage policies. One of the primary drivers of this interest in federation is the GENI [6] project; the trust model that we present in this paper is intended to fit into the GENI architecture [7].

The Emulab-PlanetLab portal [21] provided Emulab users with access to PlanetLab resources; however, it used a simplistic trust model in which all users on Emulab were "proxied" through a single PlanetLab account. This imposes significant limitations on the transparency of federated operations: from the point of view of the resource provider, activity is observed at the granularity of the proxy, not of the ultimate *user* of the proxy. Consequently, it is capable of enforcing only similarly coarse-grained policies. We discuss this federation further in Section 5. The DETER Federation Architecture [4] has a more sophisticated model of federated experiments, but similarly proxies all requests through a single account.

PlanetLab runs a federation involving multiple centralized entities (PLCs), including one in the United States, another in Europe, and another in Japan. This federation has historically used a tightly-coupled strategy that does not partition trust and replicates some testbed state globally, making such partitioning difficult. As part of the GENI project, this federation is evolving along a similar path to the one we present in this paper. PlanetLab has designed CERTDIST [16], a system for securely distributing the results of authorization decisions; such a distribution mechanism is complementary to our design.

3. Architecture Overview

Our design partitions trust within a testbed by:

- Partitioning the namespace for objects, so that each object belongs unambiguously to a particular trust domain.
- Associating each trust root with one partition of the namespace, so that there is a unique "owner" for each object.
- Allowing namespaces to be recursively partitioned, so that trust domains can make fine-grained divisions within themselves.
- Making object identities verifiable by entities outside their trust domains: they can be cryptographically traced back to trusted roots. This enables actors in the system to make secure, informed decisions about objects that come from other domains.
- Keeping authentication and authorization separate, so that they can be managed by different entities.

Together, these design decisions create a system in which trust is decentralized. Each domain can independently apply different levels of trust or policies to other domains, and the ill effects of bad behavior by any root are confined to the objects within its domain.

In practical terms, these properties can be realized by naming objects with a multi-level scheme, such as the one used for domain names, and by creating a "forest" of hierarchical certificate authorities (CAs) reflecting the hierarchy of the namespace. Statements of identity and authorization can be cryptographically signed, and traced back to a unique root CA. We provide a description of one such concrete implementation, the ProtoGENI authentication and authorization system, later in this paper.

3.1. Cooperating Services

A testbed is built from a number of logical services, which cooperate to manage the resources and users in the system. Our design includes four such services:

Resource Providers (RPs) are responsible for managing the physical resources (such as computers or network links) on which users will create network topologies and run experiments. Resource Providers are responsible for the allocation of their resources, and may make decisions about who is authorized to use them.

Identity Providers (IdPs) attest to the identity of users by forming names that uniquely identify them, then issuing those users certificates that they can use to prove their identities to other services.

Responsible Authorities (RAs) take responsibility for users' actions; because users may conduct many different activities on a testbed, different RAs may vouch for different activities a user performs.

Trusted Introducers (TIs) are optional components which can ease scalability problems for large federations of testbeds by certifying service providers to each other.

There is considerable scope for a variety of policies within this model: although the logical structure of the facilities is rigid, the choices of which policies to implement (and the modules in which to implement them) are determined by the needs of the testbed. In some cases, implementations might omit certain types of services or combine multiple logical services into a single module.

The distinction between Identity Providers and Responsible Authorities is worth further discussion. Keeping these entities (logically) separate allows for flexible policies in which identity and accountability are managed independently. For example, a user's identity may be warranted by an entity external to the testbed, such as the organization that employs them or the school they attend. This entity might maintain identity information for all of its employees or students, and it may be inappropriate to grant all of these individuals access to the testbed. To control access to the testbed, then, another level of authorization is required; this is the role filled by the Responsible Authorities.

Each request for resources must be endorsed by an RA, which agrees to take responsibility for the actions the user takes with the granted resources. Furthermore, Resource Providers may want to permit different levels of access to users depending on what activities they are performing and who is responsible for their actions. RPs may treat requests for resources differently if the request is endorsed by a professor as part of a class, an industry partner as part of R&D activities, or by a national agency as part of a high-priority research project. Each resource request is endorsed by one Responsible Authority, so that even if a user has multiple roles, it is unambiguous who claims responsibility for their actions in a given experiment.

3.2. Constructing a Trust-Partitioned Testbed

At a high level, to form a testbed under this framework, parties form trust relationships with one another: Resource Provider *A* may agree to trust Identity Provider *B*'s statements about what users it has, and Responsible Authorities *C* and *D*'s statements about the activities they vouch for (see Figure 1). Note that this does not preclude *A* from having local allocation policies: just because it recognizes *B*'s users does not mandate it to satisfy all requests for resources that they might make. Arrangements regarding "fair sharing," etc. can be made as part of a federation agreement. Trust relationships need not be symmetric: *A* may choose to trust *B* even if that trust is not reciprocated.

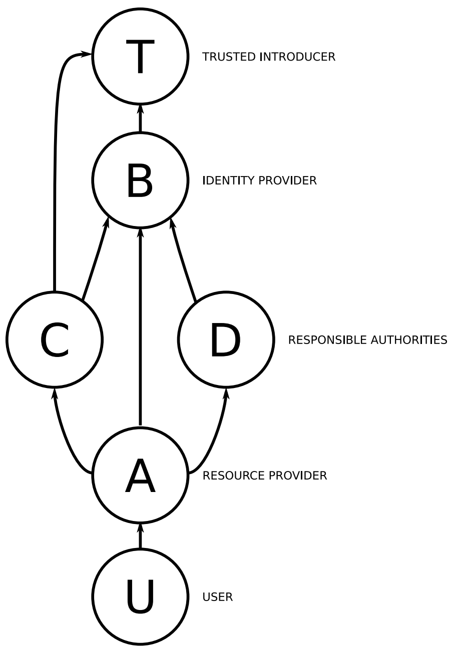

Figure 1. An example of a small federated system. Arrows indicate trust relationships the individual principals have decided upon.

Of course, establishing trust in this pairwise fashion does not scale well to large numbers of domains. In general, trust is not transitive; that is, *A* trusting *B* and *B* trusting *C* does not imply that *A* trusts *C*. In a facility where it is desirable that most testbeds establish at least a minimal level of trust with one another, a *Trusted Introducer* can make this process more convenient: TIs allow members of the facility to publish the certificates that are used to establish trust, and to discover the certificates of other members. An introduction does not mandate a specific trust relationships: as described in Section 4.1, any party may choose not to trust some certificates that it learns through such an introduction, or may choose to trust additional certificates not provided by the TI. Use of a TI does not mandate specific policies; a Resource Provider that recognizes a request vouched for by a trusted Responsible Authority, for example, may still have policies that prevent allocating resources to it. In almost any practical testbed, federates joining, leaving, or changing their certificates is a rare event, so these certificates can be cached for long periods of time, and the federates need only interact with Trusted Introducers on an infrequent (daily or weekly) basis.

3.3. Using a Trust-Partitioned Testbed

To begin using a testbed built under this model, a user first obtains a name and certificate from an Identity Provider. Depending on the particular testbed, this may be a service run by the user's employer or school, such as Shibboleth [11], or it may be a service associated with the testbed itself. The IdP gives the user proof of this identity (in practice, a certificate signed by the IdP), which the user may present to other parties.

This identity is, by itself, not enough to use the testbed; the user must find a Responsible Authority willing to vouch for his or her actions on the testbed. For a testbed that supports coursework, this RA may be the instructor of a class. For a researcher, it may be the PI of a project that he or she is working on. Industrial testbeds may designate project leaders or managers as RAs; testbeds with formal approval processes for experiments may appoint bodies to vet experiments. The user presents the proof of their identity issued by the identity provider; thus, the RA must trust the user's IdP. If the RA agrees to vouch for the user's activity, it issues the user a signed statement to this effect. This statement is specifically bound to the user's identity.

The user may use the testbed in more than one capacity; for example, he or she may be taking more than one class, or be participating on more than one research project. In this case, he or she will get an endorsement from more than one RA.

The user now approaches one or more resource providers, providing the signed documents from the IdP and RA. In making this request, the user presents an endorsement from a single RA, even if he or she has several; this ensures that there is a single party that is unambiguously responsible for the use to which the resources will be put. The Resource Providers consult their own policies, availability, etc. and decide whether or not to grant the user's request.

3.4. The Model as Implemented in ProtoGENI

To give a concrete example of our model, and to demonstrate its usefulness through deployment in a real facility, we now describe how we have implemented it in the ProtoGENI federated testbed.

The basic architecture of ProtoGENI is based on the "Slice-based Federation Architecture" (SFA) [15], which has been developed by the GENI community. The SFA is so named because it centers around partitioning the physical facility into "slices," each of which can be running a different network architecture or experiment inside. Physical resources, such as PCs, routers, switches, links, and allocations of wireless spectrum are known as "components;" when a user allocates resources on a component, the set of resources they are given comprises a "sliver." This sliver could be a virtual machine, a VLAN, a virtual circuit, or even the entire component. Each sliver belongs to exactly one slice: in essence, a slice is a container for a set of slivers.

ProtoGENI has three main types of principals:

Slice Authorities (SAs) are responsible for creating slice names and granting users the necessary credentials to manipulate these slices. They correspond to both the "Identity Provider" and the "Responsible Authority" of our facility model, as a Slice Authority both warrants user identities (by issuing certificates to users) and takes responsibility for those users' actions within slices (by issuing slice names for users.) A Slice Authority may be an institution, a research group, a governmental agency, or other organization that can provide accountability for its users.

Component Managers (CMs) are ProtoGENI's Resource Providers. When a user wishes to allocate a sliver of a component, he does so through its Component Manager. These allocations may be physical hosts, virtual machines, routers, switches, links, allocations of wireless spectrum, etc. An individual component manager may manage a collection of components, called an *aggregate*; in practice, each facility in ProtoGENI runs a single component manager that manages all of its resources, and the largest aggregates contain hundreds of nodes and thousands of links.

Users access components from the federated testbed to run an experiment or a service. A user has an account with a Slice Authority, called that user's "home" SA; this Slice Authority vouches for the identity of the user and issues slice names to the user. The user is, however, free to create slices using any Slice Authority that, according to its own policies, is willing to be responsible for that user's actions.

Principals and many other objects in the system are uniquely named by Uniform Resource Names (URNs); to create an object, a URN is created to name it. The URN scheme that we use [20] is hierarchical — each authority is given its own namespace, which it can further subdivide if it chooses. To maintain trust partitioning, each authority is prohibited, through mechanisms described in Section 4.2.1, from creating URNs outside of its namespace. An example of a ProtoGENI URN is:

```
urn:publicid:IDN+emulab.net+user+jay
```

Because the URN contains the identity of the authority that issued it (in this example "`emulab.net`"), it is possible to tell which authority "owns" the object without resorting to a lookup service.

4. Authentication and Authorization

When different parts of a system may be owned and operated by different organizations, or need to enforce custom local policies, sophisticated requirements for authentication and authorization infrastructure arise. Our approach is to adopt a decentralized architecture, to support disconnected operation, and to decouple authentication from authorization whenever possible.

The authentication system is based on the IETF PKIX model [1], while the authorization mechanism involves the presentation of cryptographically signed *credentials* (which behave analogously to X.509 Attribute Certificates [5]). When a principal presents a certificate or credential, it presents all of the signatures required to link the certificate or credential with one of the trust roots; as a result, no direct communication with the certifying party is required to validate certificates and credentials. Together, these primitives allow the warranting of identities, the granting and delegation of permissions, and the verification of identity and privilege. Most importantly, all of these operations may be performed by different principals, who need no direct knowledge of each other. Very little global policy is imposed, other than conformance to uniform naming and data representation schemes.

4.1. Trust

It follows from the basic principle of partitioned trust that domains have the ability to selectively trust each other. Each principal in the system has the ability to decide which certificates and signatures to accept, and the set of entities from which to honor requests.

4.1.1. Certificate Authorities. We deliberately refrain from imposing a single hierarchy on the trust structure (as used in PEM [8], for instance), as that model is inadequate to support local fine-grained trust decisions. On the other hand, an entirely decentralized public key distribution mechanism (such as PGP's "web of trust" [18]) is highly flexible, but tends to introduce barriers to new principals entering the system, as their certificates are unlikely to be accepted by others until they are able to obtain signatures from a sufficient number of existing authorities.

Our public key infrastructure treats each domain as a trust anchor, with its own self-signed CA certificate. Each CA may form subsidiary namespaces and issue corresponding CA certificates over them. This approach yields significant flexibility in trust decisions (at the granularity of domains), although it does introduce the problem of distributing the set of root certificates throughout a federation. When the system makes use of Trusted Introducers, a TI publishes a bundle of certificates from known domains as a convenience, but it is important to note that this does not detract from any site's autonomy: each domain is free to add to or delete from the TI's list of certificates (or even ignore it entirely). A TI also aggregates certificate revocation lists [1] from the same set of domains, though nothing prevents domains from communicating their CRLs to each other directly.

Since CA certificate bundles are not expected to change frequently, they can be cached. CRLs are more problematic, since the consequences of a stale revocation list which does not contain a recently revoked certificate could be severe. Therefore, domains may choose to obtain current information about certificate validity from each other (e.g. through the use of OCSP [12]).

4.2. Authenticating Identities

Public key certificates are issued to each principal in the system, including users, services, and components. They must be signed by the authority corresponding to the namespace in which the principal's name belongs: since these namespaces do not overlap, there always exists exactly one certificate authority whose signature will be accepted on any valid certificate.

All requests are made over TLS [2] channels, and both the client and server must authenticate to each other. (This implies that if either peer has decided not to trust the other's CA, then no communication or operation will be possible.)

We have ensured that certificates are self contained: certificates are always presented in conjunction with any intermediate CA certificates, so that any verifier can determine the validity of any certificate with no information other than the set of trust anchor certificates and current CRLs. (TLS and PKIX already provide this property, and we have been careful to preserve it in the conventions and certificate extensions we have added.)

4.2.1. Issuing Names. Formal structure within names is essential for partitioning trust, so that an unambiguous trust root can be identified for any named object. ProtoGENI has adopted the proposal by Viecco et al. [20] for uniform naming conventions throughout GENI. This name scheme tightly binds each object's name to a particular authority, which is necessary to maintain clear definitions of trust boundaries. By refusing a CA's signature on any certificate whose subject name lies outside the CA's namespace, we are able to guarantee the important property that any valid object name corresponds to exactly one trust root. Although there are certain limitations to this model (for instance, if a principal wishes to associate with a different CA for any reason, then it is forced to change its identity), the benefits are significant: first, each domain has great flexibility in choosing the set of peers with whom it will operate; second, we achieve a reasonable level of fault containment, since even extremely severe faults (e.g. malicious CAs or Byzantine failure of a CA) are unable to affect objects outside their assigned namespace. This property is extremely important, as a single key compromise in a PKI system without naming restrictions can leave the entire system vulnerable [17]. A series of attacks against root CAs in mid-2011 attracted widespread publicity and has required extensive software patches to terminate trust in the affected CAs [3].

4.3. Authorizing Operations

Credentials are used in conjunction with our public key infrastructure to allow secure validation of permissions: X.509 certificates prove that a key is bound to a principal, and credentials prove that permissions have been assigned to that principal.

It is important to note that our credentials are issued as the result of authorization decisions, and could thus be considered to represent *capabilities*. Another useful class of statement about principals are *assertions*, which can be used as *input* to policy decisions; assertions may take the form of statements such as "*X* is a student," "*Y* has a Top Secret security clearance," etc. ABAC [10] (Attribute Based Access Control) combines a system for making signed assertions with a system of formal logic to reason about authorization decisions. Under separate work by the GENI ABAC team, ABAC is also being integrated with ProtoGENI.

Almost all services must verify *both* certificates and credentials: relying on either alone is inadequate. (There are a small number of exceptions, such as a ProtoGENI user's own facility, which records state concerning the users privileges over objects under its control, and consequently is able to issue credentials on the basis of a successful key challenge alone.)

While we were able to reuse existing standards to implement our authentication and certificate scheme, we chose not to use X.509 Attribute Certificates for our credential format. This is partly because current X.509 implementations of certain features we require tends to be weak[3], and partly because we require so many dedicated attributes in our credentials that any existing generic attribute certificate format would require a great deal of extension in any case. Instead, we represent our credentials as signed statements in the form of enveloping XML signatures [23]. As with public key certificates, we ensure that credentials are self-contained: all credentials must include certificate chains sufficient to verify their enclosed signatures according to the IETF PKI certification path validation algorithm.

[3]For instance, section 1.2 of RFC 5755 [5] explicitly recommends *against* implementing AC path delegation, which would be necessary for our delegation mechanism.

The mechanism of conveying permission information indirectly (through the use of signed statements) was deliberately chosen to minimize direct communication between the issuer and the verifier of privileges. The benefits of our approach are increased fault tolerance (privileged operation can proceed even if the privilege granter is temporarily unavailable), and reduced communication cost at the time of service requests. The fact that knowledge is distributed does imply that it is impossible to guarantee that any single entity holds complete information about the authority of any principal: each party is responsible for proving its own privilege. Presenting credentials is therefore optional, and an important consequence of this fact is that credentials must only ever be issued to *increase* permission: the mechanism is not directly suited for communicating restrictions one might wish to enforce.

4.3.1. Granting Credentials. Credentials are initially issued by the authority over which privileges are being conveyed. The authority generates an XML document containing:

- A `target`, which is some ProtoGENI object identified by URN over which the credential applies. In theory, any named GENI object could be a target, although in practice, targets are usually slices, slivers, or Component Managers.
- An `owner`: the only principal who may present the credential. Owners are also specified with URNs. Although the URN could potentially address any type of object, owners are typically users, and sometimes authorities or slices.
- A type-dependent description: for privileges (the typical case of credentials), this description is a list of pairs of identifying strings and a flag indicating whether that privilege may be delegated.
- Optional extensions.
- Parent credentials (which are required for delegated credentials).
- Signature(s) covering all of the above. Credentials must ultimately be signed by the authority identified in the target URN; no other signature will do.

A simplified example of such a credential can be found in Figure 2.

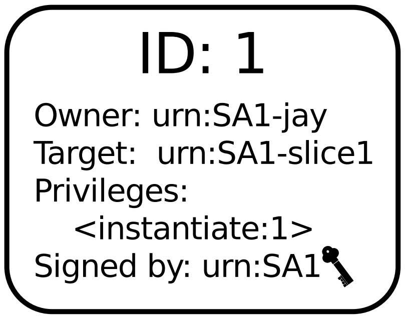

Figure 2. Example credential, showing target, owner, a set of permissions, and a signature. Note that the URNs in this figure are abbreviated.

4.3.2. Delegation of Credentials. If a credential allows it, the owner may *delegate* the corresponding permissions (or a subset of them) to another principal. As shown in Figure 3, the original credential is embedded in a new credential, which specifies the new permissions (no greater than the original); the new owner; a full copy of the original (including its parents, if any); and the old owner's signature, covering all the above. Because the delegated credential contains the original credential, it is self-contained: no additional information is required to trace it back to its original trust root.

Delegation is the primary means by which permissions are shared in ProtoGENI: the facility is general enough that it can replace user groups, access control lists, and other mechanisms for granting access rights to other users. One significant advantage of delegation is that it is highly decentralized: the issuer, original holder, delegate, and eventual verifier of delegated credentials are typically different parties, and no direct communication is ever established between any of them other than the pairwise exchanges as the credential is passed from one holder to the next.

ProtoGENI is designed to support liberal delegation policies. If the system discourages delegation (by providing inadequate tools, or restrictive policies), users will be tempted to "roll their own" delegation by sharing private keys, which is extremely undesirable.

5. Experiences

The primary indicator of the success of our design is that ProtoGENI is a running, active system; the current federation contains sixteen Component Mangers, and over 200 users have created more than 3000 slices. The authentication and authorization system seems sufficiently intuitive to users and powerful enough for facility administrators; the only reports we have received have been a few instances of confusion on the part of users as to the difference between certificates and credentials, and the necessity of having both.

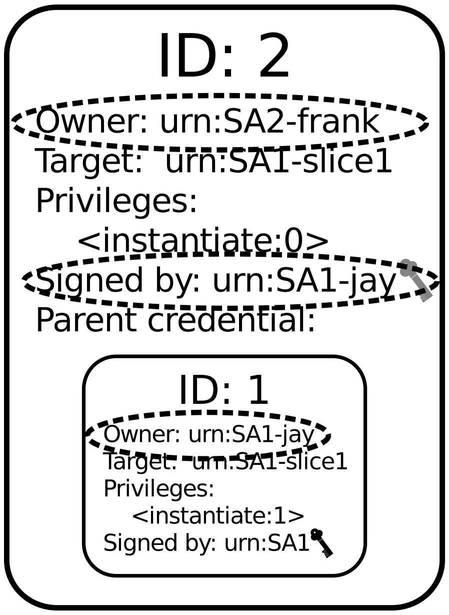

Figure 3. A delegated credential. Note that the credential from Figure 2 is embedded as the "parent," that the delegated credential has a new owner, and the signature on the delegated credential is that of the original owner.

ProtoGENI has been open to users and tool developers throughout its development. This allowed our experiences with actual users and experimenters to guide our design decisions. Described below are some of the lessons we have learned from seeing our system used by others.

Experience from Earlier Federations: This is not the first federation of testbeds that the we have attempted to build. Our earlier effort [21] was a project to modify the Emulab testbed to act as a "gateway" to the PlanetLab facility. This earlier federation was not entirely successful, and the design we present here has taken a number of lessons from the mistakes and difficulties we encountered during that process.

An essential flaw with our older Emulab-PlanetLab gateway was that PlanetLab and Emulab did not share any trust mechanisms. There was no way for an Emulab user to authenticate themselves to PlanetLab or vice versa. In order to work around this limitation, all resources created on PlanetLab were set up under a single PlanetLab user account. PlanetLab thus had no direct way to determine which Emulab users were responsible for activity run through the portal. This opacity meant that there was no way for PlanetLab to individually blacklist any Emulab user who had abused their authority. It also meant that that PlanetLab had to trust Emulab to enforce one of PlanetLab's own policies, namely that users must belong to an institution that has joined the PlanetLab Consortium. Finally, this lack of visibility into individual accounts in effect made the Emulab facility itself responsible for all of its users activities, taking on all of the responsibilities that, in our newer architecture, are distributed among Identity Providers, Responsible Authorities, and the users themselves.

The problems we faced in our first attempt to federate with PlanetLab underscore the necessity of having a single federation-wide identity. They also shows how important it is to separate Identity Providers from Resource Providers. By sharing authentication mechanisms and agreeing to a common API, we have federated PlanetLab and ProtoGENI in a matter of weeks rather than months, and no important security properties were compromised.

UUIDs: Our initial strategy was to use UUIDs [9] as identifiers for objects such as components and users: because they are essentially large random numbers, UUIDs can be generated by any party with a high confidence that they will be unique. They are also opaque, meaning that clients do not have to parse or interpret them. However, we discovered that using a flat namespace for all objects had a number of drawbacks.

First and foremost, there is no notion of an identifier "belonging" to any particular authority. As discussed in Section 4.2.1, the ability to follow the same chain of delegation in an identifier and the certificates that pertain to that identifier allows us to minimize the trust placed in each authority. With this ability, it is not possible for one authority to issue certificates for identities issued by another authority. In a flat namespace, no such separation is possible; all authorities must, in essence, be trusted to sign certificates for any identity in the system.

A flat-namespace, multiple-CA system is essentially the system used by web browsers, and concerns about such a model have recently become prominent [19, 17, 3]. For example, a single malfunctioning, compromised, or malicious CA can sign certificates for any website, effectively subverting the security of the entire "secure" web. We wanted to avoid placing such complete trust in every root, and a move to hierarchical URN identifiers allowed us to do so.

A flat namespace also requires a lookup service to resolve the authority issuing an identifier. For example, to look up information about a component, one would need to first resolve the UUID to a Component Manger, then contact that CM to ask about the component. While decentralized resolvers for flat namespaces do exist (such as DHTs), we saw that including the authority in the identifier has the additional benefit of eliminating the first lookup step, so that operations require contacting only the entities directly involved.

Local policies: In the current ProtoGENI federation, few Component Managers are exercising their ability to implement local policies, and relatively simple mechanisms have been sufficient. A policy to block external users was requested fairly early in the history of the federation, when the administrators of a testbed owned and operated by a university desired the ability to reserve all components for local use during critical class periods. Another site, operating a highly heterogeneous testbed, wished to reserve some hardware for their own use, while sharing the remainder with the federation: this became the motivation for a feature which has been added to allow different policies to apply to different component types. These basic facilities have been adequate to address the policy needs of all subsequent testbeds in the federation. While we considered designing a more sophisticated policy description engine, capable of communicating restrictions between federates and assisting users to form requests compliant with remote policies, our experience operating ProtoGENI suggests that such formalisms are not necessary in small to moderate sized federations.

Root Key Management: One of the facilities federated with ProtoGENI needed to re-generate its certificate to correct some of the fields contained therein. Initially, this facility created an entirely new certificate, using a new private key. Because its key had changed, all certificates and credentials signed by that facility (and transitively, delegations of those credentials) became invalid. Since ProtoGENI credentials may be delegated without intervention from the original issuer, it was not feasible for the facility to track down every delegated credential to be re-signed. When the potential magnitude of these effects became clear, the facility simply revoked the new certificate they had created, and made a new one using their original private key. From this experience, we added as a "best practice" the policy that federates should not change their private keys for minor certificate re-issues. Of course, if a certificate must be re-generated because it is suspected that the private key may have been compromised, adopting a new private key is unavoidable.

This incident, as problematic as it was for the facility in question, demonstrated an important property of our design: because trust is partitioned in the ProtoGENI federation, no other federates were affected. While it resulted in a temporary inconvenience for users of one facility, operation in the rest of ProtoGENI continued as normal.

6. Conclusion

As testbeds become larger, more complicated, and more security-sensitive, trust models with a single root are no longer appropriate. We have described the principle of partitioned trust, the properties it provides in testbeds, and its realization in ProtoGENI in particular. The result is a loosely coupled system which partitions a testbed into a set of trust domains; each trust domain can independently decide how much to trust other domains and can apply policies to entire domains or individual principals within them.

Acknowledgments

Many people have been active participants in the GENI design process, which arrived at the basic design in to which our system is intended to fit. While the total number of contributors to this process is large, we would like to specifically acknowledge the chairs of the GENI Facility Architecture Working Group and heads of the GENI control frameworks: Larry Peterson (PlanetLab), John Wroclawski (TIED), Jeff Chase (ORCA/BEN), and Max Ott (OMF). Other major contributors to the design of GENI's authentication system have included Steve Schwab and Ted Faber.

This material is based upon work supported by the National Science Foundation under Grant No. 0723248 and the GENI Project Office.

References

[1] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. Internet X.509 public key infrastructure certificate and certificate revocation list (CRL) profile. Request for Comments 5280, IETF, May 2008.

[2] T. Dierks and E. Rescorla. The transport layer security (TLS) protocol version 1.2. Request for Comments 5246, Internet Engineering Task Force, Aug. 2008.

[3] K. Dilanian. Cyber-attack in Europe highlights internet risks. *Los Angeles Times*, 9 Sept. 2011. `http://articles.latimes.com/2011/sep/09/world/la-fg-cyber-attack-20110910`.

[4] T. Faber, J. Wroclawski, and K. Lahey. A DETER federation architecture. In *Proc. of the DETER Community Workshop on Cyber Security Experimentation and Test (CSET)*, Boston, MA, Aug. 2007.

[5] S. Farrell, R. Housley, and S. Turner. An internet attribute certificate profile for authorization. Request for Comments 5755, Internet Engineering Task Force, Jan. 2010.

[6] Exploring networks of the future. `http://www.geni.net/`.

[7] GENI Planning Group. GENI facility design. GENI Design Document GDD–07–44, GENI, Mar. 2007. `http://geni.net/GDD/GDD-07-44.pdf`.

[8] S. Kent. Privacy enhancement for internet electronic mail: Part II: Certificate-based key management. Request for Comments 1422, Internet Engineering Task Force, Feb. 1993.

[9] P. Leach, M. Mealling, and R. Salz. A universally unique IDentifier (UUID) URN namespace. Request for Comments 4122, Internet Engineering Task Force, July 2005.

[10] N. Li, J. C. Mitchell, and W. H. Winsborough. Design of a role-based trust management framework. In *Proc. of the 2002 IEEE Symposium on Security and Privacy*, May 2002.

[11] R. L. Morgan, S. Cantor, S. Carmody, W. Hoehn, and K. Klingenstein. Federated security: The Shibboleth approach. *EDUCAUSE Quarterly*, 27(4):12–17, 2004.

[12] M. Myers, R. Ankney, A. Malpani, S. Galperin, and C. Adams. X.509 internet public key infrastructure online certificate status protocol — OCSP. Request for Comments 2560, Internet Engineering Task Force, June 1999.

[13] M. Ott, I. Seskar, R. Siraccusa, and M. Singh. ORBIT testbed software architecture: Supporting experiments as a service. In *IEEE Tridentcom*, pages 136–145, Trento, Italy, Feb. 2005.

[14] L. Peterson, A. Bavier, M. E. Fiuczynski, and S. Muir. Experiences building PlanetLab. In *USENIX OSDI*, pages 351–366, Seattle, WA, Nov. 2006.

[15] L. Peterson, R. Ricci, A. Falk, and J. Chase. Slice-based federation architecture. `http://groups.geni.net/geni/wiki/SliceFedArch`, June 2010.

[16] S. Sevinc, L. Peterson, T. Jim, and M. Fernández. An emulation of GENI access control. In *Proc. of the 2nd Workshop on Cyber Security Experimentation and Test (CSET '09)*.

[17] C. Soghoian and S. Stamm. Certified lies: Detecting and defeating government interception attacks against SSL. `http://files.cloudprivacy.net/ssl-mitm.pdf`.

[18] W. Stallings. The PGP web of trust. *BYTE*, 20(2):161–162, Feb. 1995.

[19] M. Stevens, A. Sotirov, J. Appelbaum, A. Lenstra, D. Molnar, D. A. Osvik, and B. de Weger. Short chosen-prefix collisions for MD5 and the creation of a rogue CA certificate. In *CRYPTO 2009*, Aug. 2009.

[20] C. Viecco. Use of URNs as GENI identifiers. `http://gmoc.grnoc.iu.edu/gmoc/file-bin/urn-proposal3.pdf`, June 2009.

[21] K. Webb, M. Hibler, R. Ricci, A. Clements, and J. Lepreau. Implementing the Emulab-PlanetLab portal: Experience and lessons learned. In *USENIX WORLDS*, San Francisco, CA, Dec. 2004.

[22] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An integrated experimental environment for distributed systems and networks. In *USENIX OSDI*, pages 255–270, Boston, MA, Dec. 2002.

[23] XML signature syntax and processing. `http://www.w3.org/TR/xmldsig-core/`, June 2008.