G E N I

Global Environment for Network Innovations

GENI Security Architecture
Spiral 2 Draft 0.5

Document ID: GENI-SEC-ARCH-Draft-spiral2

March 15th, 2010

Prepared by:
Alefiya Hussain and Stephen Schwab
SPARTA, Inc. dba Cobham Analytic Solutions

Table of Contents

Table of Figures

1. Document Scope

This section describes this document's purpose, its context within the overall GENI document tree, the set of related documents, and this document's revision history.

1.1 Purpose of this Document

The goal of this document is to incrementally track, revise, and extend the security design of the GENI control frameworks and the related projects while identifying the key security decisions and implementations within each control framework and their impact on the design of the control framework, related projects, and federation efforts. Further we also plan to include detailed discussions of the security aspects of many spiral 2 prototypes and integration efforts.

Some of this material is drawn from the Spiral 1 Security Architecture documents.

1.2 Context for this Document

1.3 Related Documents

The material in this document is drawn from the following documents listed below.

Document ID	Document Title and Issue Date
GENI-SE-SY-SO-02.0	"GENI System Overview", September 29, 2008. http://www.geni.net/docs/GENISysOvrw092908.pdf
GDD 06-10	"Towards Operational Security for GENI," by Jim Basney, Roy Campbell, Himanshu Khurana, Von Welch, GENI Design Document 06-10, July 2006. http://www.geni.net/GDD/GDD-06-10.pdf
GDD 06-23	"GENI Facility Security," by Thomas Anderson and Michael Reiter, GENI Design Document 06-23, Distributed Services Working Group, September 2006. http://www.geni.net/GDD/GDD-06-23.pdf
SANS	SANS Institute- Glossary of Security Terms. http://www.sans.org/resources/glossary.php
GENI-SE-CF-PLGO-01.2	PlanetLab GENI Control Framework Overview http://groups.geni.net/geni/attachment/wiki/PlanetLabGeniControlFrameworkOverview/011409%20%20GENI-SE-CF-PlanetLabGENIOver-01.2.pdf
GENI-SE-CF-PRGO-01.3	ProtoGENI Control Framework Overview http://groups.geni.net/geni/attachment/wiki/ProtoGeniControlFrameworkOverview/011409%20%20GENI-SE-CF-ProtoGENIOver-01.3.pdf
GENI-SE-CF-ORGO-01.2	ORCA GENI Control Framework Overview http://groups.geni.net/geni/attachment/wiki/OrcaGeniControlFrameworkOv

	erview/011409%20%20GENI-SE-CF-ORCAGENIOver-01.2.pdf
GENI-SE-CF-RQ-01.3	GENI Control Framework Requirements http://groups.geni.net/geni/attachment/wiki/GeniControlFrameworkRequirements/010909b%20%20GENI-SE-CH-RQ-01.3.pdf
GDD 06-24	"GENI Distributed Services," by Thomas Anderson and Amin Vahdat, GENI Design Document 06-24, Distributed Services Working Group, November 2006. http://www.geni.net/GDD/GDD-06-24.pdf
N/A	"GMC Specifications," edited by Ted Faber, Facility Architecture Working Group, September 2006. http://www.geni.net/wsdl.php
GDD 06-23	"GENI Facility Security," by Thomas Anderson and Michael Reiter, GENI Design Document 06-23, Distributed Services Working Group, September 2006. http://www.geni.net/GDD/GDD-06-23.pdf
GDD 06-10	"Towards Operational Security for GENI," by Jim Basney, Roy Campbell, Himanshu Khurana, Von Welch, GENI Design Document 06-10, July 2006. http://www.geni.net/GDD/GDD-06-10.pdf
N/A	“Slice Based Facility Architecture,” Draft v1.02, November 3, 2008, by Larry Peterson, et.al. http://svn.planet-lab.org/attachment/wiki/GeniWrapper/sfa.pdf
N/A	SHARP: An Architecture for Secure Resource Peering, 2003, by Yun Fu, Jeffrey Chase, et.al. http://www.cs.ucsd.edu/~vahdat/papers/sharp-sosp03.pdf
N/A	Sharing Networked Resources with Brokered Leases, 2006, by David Irwin, Jeffrey Chase, et.al. http://portal.acm.org/citation.cfm?id=1267377
N/A	ORCA Technical Note: Guests and Guest Controllers, 2008, by Jeff Chase http://www.cs.duke.edu/nicl/pub/papers/control.pdf
N/A	ORCA references: http://nicl.cod.cs.duke.edu/orca/
N/A	ORBIT Testbed Software Architecture: Supporting Experiments as a Service Maximilian Ott, Ivan Seskar, Robert Siraccusa, Manpreet Singh http://www.orbit-lab.org/wiki/Orbit/Documentation/Publications
N/A	ORBIT Measurements Framework and Library (OML): Motivations, Design, Implementation, and Features, Manpreet Singh, Maximilian Ott, Ivan Seskar, Pandurang Kamat http://www.orbit-lab.org/attachment/wiki/Orbit/Documentation/Publications/final-oml-paper.pdf
N/A	Overview of the ORBIT Radio Grid Testbed for Evaluation of Next-Generation Wireless Network Protocols D. Raychaudhuri, I. Seskar, M. Ott, S. Ganu, K. Ramachandran, H. Kremo, R. Siracusa, H. Liu and M. Singh http://www.orbit-lab.org/attachment/wiki/Orbit/Documentation/Publications/Orbit_WCNC_05_final.pdf
N/A	GENI Engineering Conference III – Presentations http://groups.geni.net/geni/wiki/CFWGGEC3

N/A	DETER Federation Daemon (fedd) http://fedd.isi.deterlab.net/
N/A	Access Control for Federation of Emulab-based Network Testbeds, Ted Faber and John Wroclawski, In Proceedings of the CyberSecurity Experimentation and Test (CSET) Workshop, San Jose, (July 2008) http://www.usenix.org/events/cset08/tech/full_papers/faber/faber.pdf
N/A	A DETER Federation Architecture, Ted Faber, John Wroclawski, Kevin Lahey, Proceedings of the DETER Community Workshop on Cyper Security Experimentation and Test, Boston, MA, (August 2007). http://www.usenix.org/events/deter07/tech/full_papers/faber/faber.pdf
WJ03a	Automated Trust Negotiation Technology with Attribute-based Access Control, W. Winsborough and J. Jacobs, In Proceedings of the DARPA Information Survivability Conference and Exposition, 2003, Vol. 2 pp 60-62, April 22-24, 2003.

1.4 Document Revision History

Revision No	Date	Revision By	Summary of Changes
0.1	Jan 31 2010	Alefiya Hussain	Initial Draft
0.2	March 14 2010	Stephen Schwab	Revised, Posted for GEC7

2. The GENI Security Architecture

This section discusses the GENI security architecture as interpreted by the authors within the framework promulgated by the GENI Project Office (GPO). Given the spiral rapid prototyping model employed by the GENI community to create working end-to-end prototypes *and* to reach consensus on requirements and design choices for the long-term GENI efforts, a security architecture is necessarily a fluid, evolving set of concepts and ideas. This document attempts to serve a critical role in the GENI ecosystem, by (1) capturing the essential elements of the security issues posed by GENI; (2) tracking the evolution of thinking regarding problems, solutions, and future challenges within the context of the development and prototyping spiral projects; and (3) documenting a succinct set of guidelines, policies, and mechanisms that are appropriate to offer as pragmatic design choices in the context of GENI.

2.1 Terminology and Overview

The GENI testbed initiative is an exciting development for networking architecture, protocols and service design as the infrastructure enables long-running realistic experimentation that allows end users to opt-in to test the proposed experimental systems. Thus GENI has more sophisticated security requirements than the traditional Internet architecture.

It is worth considering for a moment how securing GENI differs from securing "the Internet". Ideally, one might pre-suppose that GENI and the Internet are both built out of elements (e.g. end-systems and network gear, a.k.a. boxes) that speak various protocols and are configured to do so by local or remote operators. At this level of abstraction, all that is needed is a means to authenticate individual operators and authorize their various commands and configuration changes on each box, plus incorporation of sufficiently robust security features within each distinct protocol layer, e.g. secure ARP, secure routing, secure naming, secure transport, secure QoS, etc.

From this viewpoint, all the problems of Internet security are "merely" because of the inertia of maintaining backwards compatibility with the installed base, deployed protocols, and customary organization and configuration of the existing Internet. If only we had a clean-slate network deployment, everything could be revisited and done securely. Since GENI could be such a clean-slate network deployment, according to this line of reasoning, it is straightforward to design in all the necessary authentication, authorization and security protocols and assure ourselves of an ideal, trustworthy system.

Unfortunately, the situation is not so simple. GENI, while affording the possibility to create a clean-slate network architecture within an experimental slice, bootstraps itself using clearinghouses, control frameworks, component managers and slice and management authorities that rely heavily on Internet protocols. So while GENI may not always be tied to the Internet architecture forever, during the prototyping spirals at least, GENI security must consider all the insecurities inherited from the Internet. (As an aside, deploying GENI entirely above a collection of encrypted VPN tunnels is feasible – but probably not sufficient to enable the sorts of user opt-in experiments that are desirable.)

Moreover, it is far from clear that the state-of-the-art in network security would be sufficient to build and deploy, at the scale envisioned for GENI, a suite of protocols and complementary authentication and authorization technology to enable a cost-constrained, trustworthy GENI ecosystem. For example, corporate and government PKI and authenticated identity rollouts are notoriously expensive and difficult to maintain – can GENI drive down the cost to manage such a large scale authentication and authorization system, without compromising on security goals?

Additionally, GENI's key strategy is growth via federation which allows incorporating existing facilities into the overall GENI ecosystem and adding new technologies as they mature, thus allowing GENI to be nimble and not commit to a single technology at the start. However, this strategy will cause heightened concerns from users and network operators about security as enforcing security properties in such an environment is difficult, particularly since the requesters and resources will typically be managed by different authorities and may have different authorization mechanisms.

Some of the most challenging aspects of securing GENI networks concern the authentication support for authorization. Authorization decisions require the authentication of the entity making a request. Authentication normally implies the use of cryptographic techniques. But the application of existing cryptographic techniques to the GENI networks environment presents certain challenges.

The identification of the principal itself in the GENI networks may be challenging. Current Internet interactions are typically client-server, where the explicit individual identity of the client and server are important. However, in a GENI network if we move away from individual identities to attribute based identities and access control (X.509 Attribute Certificates, KeyNote and PolicyMaker) the aspects of the principal's attributes that are important may change radically as it interacts with different components in the GENI network. This is a stark change from the traditional Internet client-server model where both have a common understanding of the identities or attributes that are important. For example, within the principal's network, the individual's identity or company role may be important. But beyond the immediate network of the principal, it is not likely that the individual identity of the end user will be important. Aggregate security attributes will be more likely to be used, which may be labels, groups, etc. Furthermore, the aggregate attributes may themselves differ in different domains. Consequently, there may be multiple and varying principal identities or attributes that are important.

Also, secure protocols often rely on a well-defined notion of end-system address as a pre-requisite for negotiating and establishing an authenticated communication channel. If a GENI slice can re-define the very abstraction of end-system address, it may be difficult to reuse older authentication protocols in a secure manner.

The GENI project office envisioned a centralized entity called the *clearinghouse* that could serve as a central catalog of all GENI resources where a researcher can search for the resources he needs, authenticate himself, reserve a sliver on them, and start to experiment using some from of GENI money or GENI points. This vision of the clearinghouse as a GENI portal is extremely powerful as it provides the researcher with a clean and familiar interface to assemble complex experiments using a vast range technologies across various testbeds. The GENI money could be assigned to a researcher based on a vetting process and allows the process to be more objective and reward based possibly on the researcher's past GENI history.

2.2 The Threat Model

GENI's scale, widespread deployment, and visibility will make it an inviting target for attack and thus careful attention must be paid to security in its design. In our view, security considerations need to permeate every control framework and interface to be defined in GENI. The text in this section is drawn from GDD 06-23 and discussions at previous GENI Engineering Conferences. We begin with a diagram that illustrates how to frame our thinking about GENI and the threats facing the system.

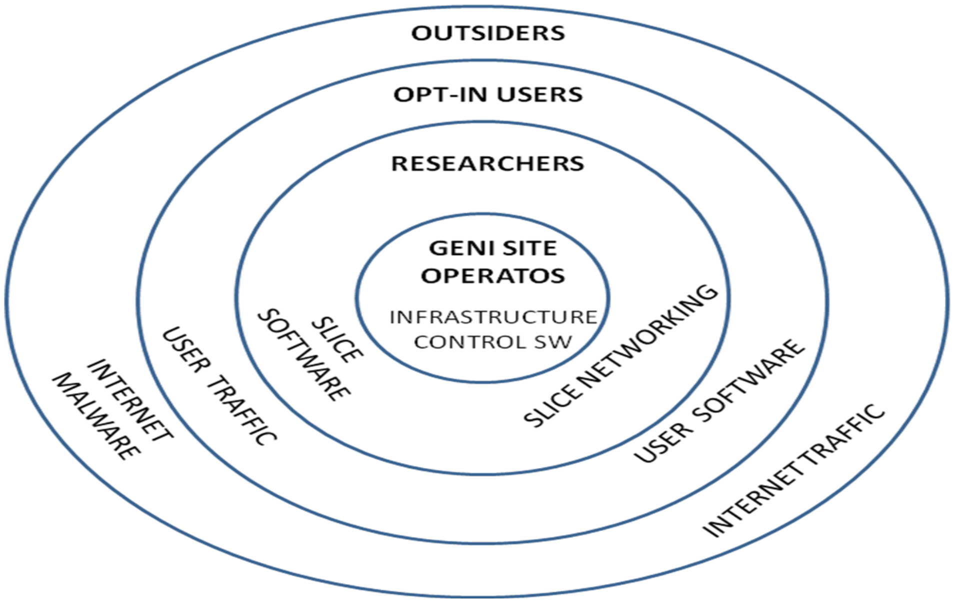


Figure 1. GENI Threats. The illustration presents rings of threats. At the center is the infrastructure with the greatest privilege. Working outwards are rings including GENI researchers, opt-in users making use of GENI experimental slices, and finally outsiders.

In terms of modeling threats, the GENI Infrastructure includes Clearinghouses, Control Frameworks, Component Managers, Aggregate (Component) Managers, Slice and Management Authorities, and everything else that supplies resources or facilitates the management of users or resources within the GENI ecosystem. This is the base layer of GENI, analogous in some ways to an operating system, albeit different in other respects. We include threats to this infrastructure within the center ring – namely the privileged GENI operators who interact with the various GENI elements, and the software running on all these GENI elements. (Without loss of generality, we have labeled this control framework software, but for clarity state that potentially any software running on a GENI element that is part of the infrastructure is a threat, e.g. if any GENI operator or software running on a GENI infrastructure element is malicious or compromised, then there are serious consequences for the portion of the GENI ecosystem within their (or its) purview.)

As we work outwards, GENI slices, including the GENI researchers, the software running within that slice, and the networking behavior including traffic implemented within that slice is a potential threat. Ideally, the consequences would be less serious if a threat at this level attacks GENI than a threat at the infrastructure level. Threats at this level should be eliminated once the slice is terminated. A goal of our security architecture is to ensure that this situation actually occurs in practice, when GENI control frameworks are deployed and operated in the real world.

Continuing outwards, opt-in users, with even less privileges should pose an even lower risk to GENI if they turn out to be malicious. We consider the users' network traffic, and the users' software also to be at this threat level. Note that the users' software may be executing on their end-system, and might be supplied by the GENI Researcher, might be part of their standard OS and application suite, or may be a combination of both. Since the software can act with all the powers wielded by the GENI opt-in users, it must be considered indistinguishable from the GENI opt-in users, at least in terms of what threat it may pose within our model. Lastly, GENI is of course connected to the Internet, including whatever endemic Internet malware and traffic is present.

Considering this threat model, we recognize that there are three broad classes of attacks that must be addressed by the GENI Security Architecture and by its operational procedures. First, external attacks may be launched by outsiders on the GENI infrastructure, either as a denial-of-service attack, or simply to gain control of GENI resources. Second, and related, we need to contain and prevent the impact of accidentally or maliciously misbehaving GENI experiments on the outside world; similarly, we must limit the impact of attackers posing as legitimate GENI researchers. Third, we need a level of isolation between experimental slices, so that GENI cannot be surreptitiously or intentionally used by one researcher to disrupt another slice. We discuss these three types of attacks in this section by providing a list of specific threats that the GENI security architecture must address.

For the moment, we are deferring consideration of a fourth threat, that of a malicious insider within the GENI infrastructure itself, and instead consider this set trustworthy. While GENI will initially have a small community of operators and sites, and rely on non-technical means to address this issue, we believe that as GENI scales and federates with large numbers of other systems, this threat will need to be re-evaluated.

The threats are listed according to one estimate as to the relative frequency of that particular type of problem; for example, accidentally misbehaving experiments are likely to be a somewhat frequent occurrence on a platform designed to support experimental investigation, while determined attacks against the GENI software are relatively less likely, but more serious. Fortunately, many of the same technical solutions can be applied to both root causes. Note that the threats we list below are not intended to be completely mutually exclusive: systematic attacks against GENI may combine multiple elements, and thus the facility needs to be able to deal with all of these types of problems simultaneously.

- Containing runaway experiments that cause unwanted traffic. Experience with past control frameworks such as PlanetLab and Emulab suggests that unintentional misbehaving experimental code will be a common occurrence on GENI. We believe a process is needed to assign and enforce specific, minimal privileges appropriate to each experiment in addition to limiting experimental behavior such that all unwanted traffic can be eliminated from the network once the experiment slice is terminated.. Hence a novice user's mistake will not have global consequences on the Internet. This would require a rapid "kill switch" to enable operations staff to quickly suspend the misbehaving experiment .

- Isolating runaway experiments that disrupt the execution environment for other experiments within GENI, e.g., by exhausting disk space or file descriptors. These issues can be handled by providing stronger isolation between experiments and by monitoring shared resources for unexpected usage patterns. The GENI facility must also ensure that hosting organizations are not put at significant risk for contributing resources to GENI, and the GENI effort must take measures to convince hosting organizations that problems are rare and dealt with promptly.
- Containing the misuse of an experimental service by an end user, for example, one example experimental service conceived for GENI is to run a virtual ISP supporting a novel internal architecture. Such an experimental ISP might be used by a malicious user to launder illegal packets. We expect this set of concerns to be addressed by establishing GENI-wide standards for experiments offering packet delivery services (or their equivalent) to end users. For example, GENI might require that an experimental ISP provide basic monitoring or tracing tools for law enforcement enquires.
- Preventing and detection of theft or corruption of an experimenter's credentials to use GENI. Unfortunately, it is well-known within the security community that users are often careless with the keys used for authentication, if only because key compromises are silent until it is too late. Carefully calibrating privileges to match the experimenter's sophistication is one avenue (e.g., users likely to be careless with their keys would be given more limited privileges); another is to use technical means discussed in subsequent sections to make it more difficult for attackers to gain access to user keys. Also, since end host corruptions are endemic on the Internet today, we need to make it easy for the GENI operations staff to revoke and replace end user keys and privileges after such break-ins. Even so, this is perhaps the most likely avenue for malicious attacks against GENI.
- Denial of service attacks against the GENI infrastructure. GENI should fail "off" to avoid providing an avenue for an attacker to take control, and then use denial of service to prevent the operations staff from taking countermeasures. Technically, this can be accomplished by requiring privileges to be frequently refreshed.
- Direct attacks against vulnerabilities in the GENI management software. GENI is a complex distributed system, and therefore special care must be taken to avoid vulnerabilities in its implementation. One step is the explicit modeling of trust relationships between GENI components as described below. Another important step is to observe that the software development processes adopted for GENI software are critical to the security of the GENI facility.

- Privacy of experimental data and the privacy of management policy. Preventing unauthorized access to information stored in GENI can be accomplished using the flexible access control architecture described later in the document. However, preventing all forms of information leakage while an experiment is running is an open research challenge.

2.3 The Trust Model

The GENI Security Architecture will assume that the common security practices will be in place. For example, it is important to actively manage all GENI hardware, e.g., to proactively keep all operating system software up to date with known security patches. This means that any changes GENI makes to host software is minimal, so that patches can be applied quickly. Another important step is that components should be configured with the minimal number of open ports. Also, it is important to instrument the GENI hardware to discover problems quickly, that is, enabling continuous monitoring for anomalous node behavior by GENI operations. (This is of course made more complicated by the fact that the experimental architectures and services running on top of GENI may be by their very nature, anomalous!) Once anomalous behavior is detected, it is imperative that it is analyzed and fixed rapidly. The emergence of trusted computing hardware and the integrity measurement architectures should provide a mechanism for GENI operations staff to reset every node in GENI to a known, good state.

As stated in the earlier GENI Facility Security document, GDD 06-23:

> Additionally, the GENI security architecture also assumes good software development processes are used for all software that is deployed on the GENI facilities. It is well-known that poor software quality is the source of numerous types of serious security vulnerabilities in practice (e.g., buffer overflows and format-string vulnerabilities). We believe it is imperative that sound software development processes be adopted by the GENI community so as to eliminate, to the extent practicable, these types of vulnerabilities. While specifying software development processes is outside the scope of this document, an example might be that all GENI-defined interfaces and protocols be adopted only after an open, public review of potential security vulnerabilities, that changes to interfaces be made only through a similar formal process, and that conformance tests be generated (ideally, automatically) from a formal specification of the interface. We also suggest, where practical, all GENI software should be implemented to be type-safe, using tools such as CCured or languages such as Java. In cases where type-safety is impractical, as in modifications to an existing operating system implemented in C, standard practices such as software verification tools and test suites can be used to reduce the likelihood of vulnerabilities. We also believe that serious consideration

should be given to requiring that source code produced for GENI be made public, so as to allow for independent security analysis. However, we do not believe it is a cost-efficient use of GENI resources to require every aspect of the management software to be robust to arbitrary malicious attacks by privileged insiders (so-called Byzantine attacks). Rather, we intend to rely on detection, confinement and resetting to a known good state to correct intrusions when they occur.

A GENI researcher should not have to trust all the nodes, network environments, and other end users of the GENI network. There are few ways to assure the researcher that their data will be protected from attacks (exposure, unauthorized use or modification) by the node or the network environment where the data is processed in the clear. The researcher may apply end-to-end cryptographic protections against these attacks and not make the node privy to the cryptographic keying material, so that the data is never represented in clear-text on the node. While end-to-end cryptographic protection limits the damage that the node can cause to the data, it also limits the network services that can be performed. When considering protection against unauthorized access, or use attacks on the end user's data from other end users or slices in the infrastructure, the situation is a bit more reassuring. The nodes in the GENI environment can provide enforcement of the researcher's authorization policy, as long as they have the ability to authenticate the principals associated with each experiment and are provided the researcher's policy. However, note that in both cases, we are ultimately driven toward a model of explicit trust – researchers need the flexibility to explicitly describe which resources in the GENI substrate they trust, and to what degree, because technical means alone can not ensure that all substrate resources are trustworthy.

Similarly, it should not be necessary for the components or component managers to trust the rest of the GENI substrate that it is connected to. It would certainly be unwise to design the system so that it must trust all researchers and all adjoining interconnected GENI components. The GENI architecture grants the Component Manager (CM) the authority to start and manage slices locally. All requests from the CM for slice services will be on the behalf of the experimenter to provide services for an experiment. The component implicitly trusts the CM to adhere to the authorization and access control policies when requesting services. A component owner pre-establishes resource allocation policies regarding how the component's resources are assigned to GENI researchers. In summary, explicit models of trust, represented by entities within the GENI ecosystem, seem necessary to provide for local decision making over a large set of components and their owners.

In this version of the GENI security architecture we have focused on identifying and documenting the authorization and the authentication enforcements within the various clusters and related projects. These ideas are further explored in the subsequent sections.

3. The GENI Control Frameworks

In this section we discuss the various control frameworks that are part of GENI, focusing primarily on the security aspects and challenges we will face when they are intergrated with the other projects. For completeness, we include a brief description of the operational aspects of the control frameworks. This section borrows heavily from the following documents: GENI Control Framework Requirements GENI-SE-CF-RQ-01.3, ProtoGENI Control Framework Overview GENI-SE-CF-PRGO-01.3, PlanetLab Control Framework Overview GENI-SF-CF-PLGO-01.2, ORCA GENI Control Framework Overview GENI-SE-CF-ORGO-01.2, ORBIT talks and TIED talks at GECs.

3.1 ORCA

Point of Contact: Jeff Chase (chase@cs.duke.edu)

Open Resource Control Architecture (ORCA) is a software framework and open-source platform to manage a programmatically controllable shared substrate. It can be viewed as a service-oriented *resource control plane* hosting diverse computing environments (*guests*) on a common pool of networked hardware resources such as virtualized clusters, storage, and network elements.

An Orca deployment is a dynamic collection of interacting control servers (actors) that work together to provision and configure resources for each guest according to the policies of the participants. The actors represent various stakeholders in the shared infrastructure: substrate providers, resource consumers (e.g., GENI experimenters), and brokering intermediaries that coordinate and federate substrate providers and offer their resources to a set of consumers. Orca is based on the foundational abstraction of resource leasing. A lease is a contract involving a resource consumer, a resource provider, and one or more brokering intermediaries. Each actor may manage large numbers of independent leases involving different participants.

The Orca actor protocol operates on five kinds of objects: slices, leases, slivers, pools, and principals. Each object is named by a unique identifier, which is an RFC 4122 GUID. Each object has an attached property list of named attributes. The properties are passed to actors and plugins that operate on the object, which may add to the property list or modify it in well-defined ways. Each actor maintains state pertaining to the leases it knows about. Each lease is initiated by a service manager or slice manager (SM), and must be approved (ticketed) by a broker and granted by an aggregate manager (AM). The GUID for an object is assigned by the actor that creates it. Slices may also have user-assigned symbolic names for convenience.

An actor determines what rights to assign to a principal based on security assertions attached to endorsements it receives for that principal's public key. There are three types of security assertions of interest:

1) *Security Attributes*. A security attribute is a property of the principal that may be queried by an authorization policy. Example: "This principal is one of Chase's students." Attributes are a general basis for Attribute-Based Access Control (ABAC). In an implementation integrating Shibboleth deployments, a primary role of Shibboleth Identity Providers (IdPs) is to certify attributes for an authenticated identity.
2) *Delegations*. A principal may delegate a subset of its rights to another principal by issuing an endorsement specifying the rights to be delegated. A delegation chain rooted in a trust anchor is proof that the endorsed principal has specific rights.
3) *Resource Contracts*. Resource server actors (brokers and authorities) may delegate specific rights to specific resources at specific times to a specific principal by issuing resource contracts (tickets or leases) to that principal. This class of endorsements includes tickets and leases in SHARP-derived systems such as Orca.

Every operation is requested on behalf of some principal (the subject) and operates on an object. The authorization policy approves or denies each requested operation based on the subject, the object, and the nature of the operation. Orca as an architecture supports flexible authorization policies in the resource servers (authorities or AMs and brokers), based on external endorsing trust anchors such as IdPs, Slice Authorities, Management Authorities, GENI facility management, etc. An actor may receive endorsements, credentials, and delegations attached to a request, or it might fetch them on demand using some form of distributed storage and recovery service. Orca provides a bare-bones authorization policy based on simple ACL rules.

Figure 2 summarizes the identity and authentication processes within ORCA. The GENI ORCA control framework includes (is slated to include?) one or more Identity Providers, based on Shibboleth technology, which vouch for principals. They provide attributes for certain principals, for example, researchers. The user creates an identity by acting from a server utilizing a browser or acting from a server utilizing a set of helper tools, such as the Experiment Control Tools.

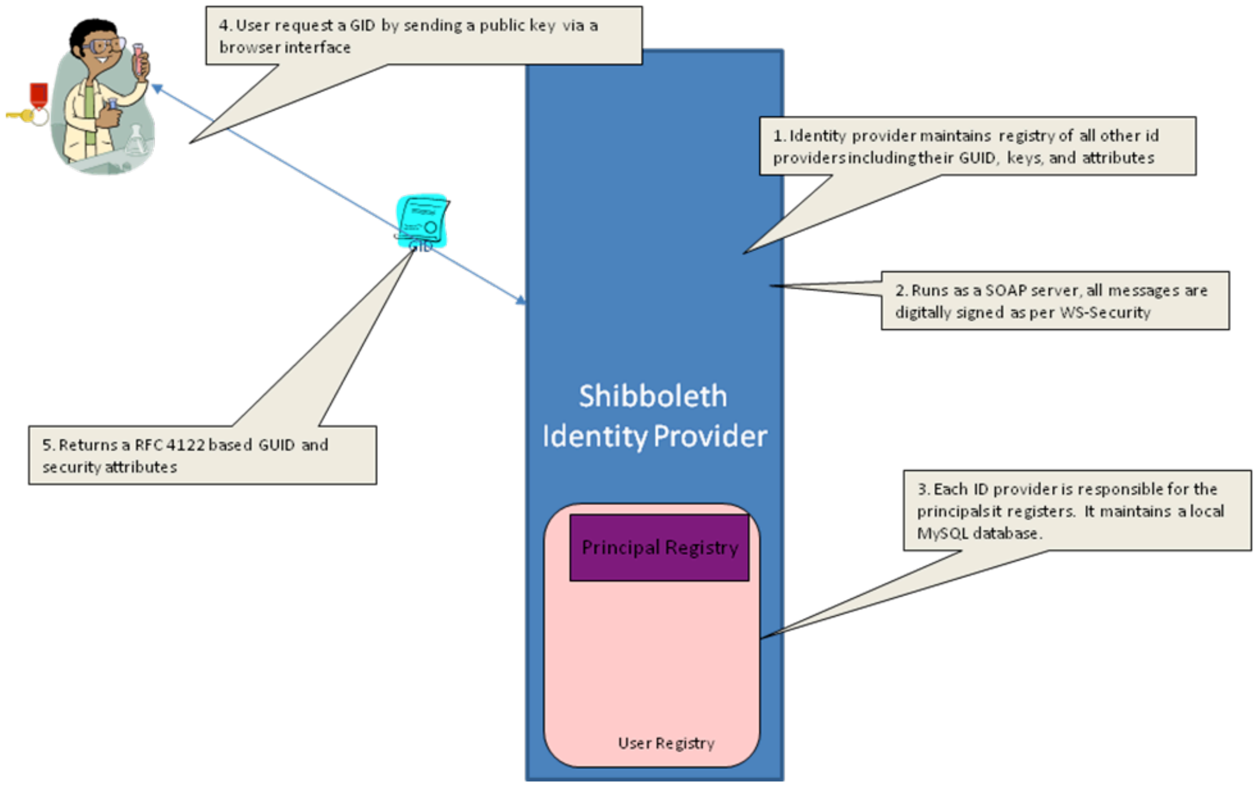

4. User request a GID by sending a public key via a browser interface
1. Identity provider maintains registry of all other id providers including their GUID, keys, and attributes
GID
2. Runs as a SOAP server, all messages are digitally signed as per WS-Security
Shibboleth Identity Provider
5. Returns a RFC 4122 based GUID and security attributes
3. Each ID provider is responsible for the principals it registers. It maintains a local MySQL database.
Principal Registry
User Registry

Figure 2. Identity and Authentication Mechanisms within ORCA

An important point to note about ORCA based on Shibboleth and Shirako philosophies, is that any kind of service provider does not really care about identity, but only security attributes associated with the identity and endorsed by an identity provider. Actual real identity is just one possible attribute but is not necessarily required. ORCA envisions that ultimately GENI may require binding identities to the real world identity, but it may not be necessary to mandate it. Early binding of identity could complicate the acceptable levels of indirection within GENI. For example, Jeff Chase in his comments on the CF Requirement document states "if Duke says the operation is being done on behalf of a CS faculty member, but does not say who, and an abuse is committed, is it sufficient to allow/require the institution to divulge identity only after the fact, that is, after evidence of the abuse has been presented?"

The ORCA GENI control framework, authorization and access control are currently based on digitally signed messages (WS-Security) and the Java Cryptography Architecture (e.g., keystore files). Access control is through tickets issued by the domain authorities to brokers who are responsible for delegating control over resources as shown in Figure 9. Every actor is identified by a GUID and possesses a keypair for authentication. Each actor has access to a registry of the GUIDs and public keys of other actors that are known to it. Actors sign their messages with their private keys, and authenticate messages based on their knowledge of the sender's public key.

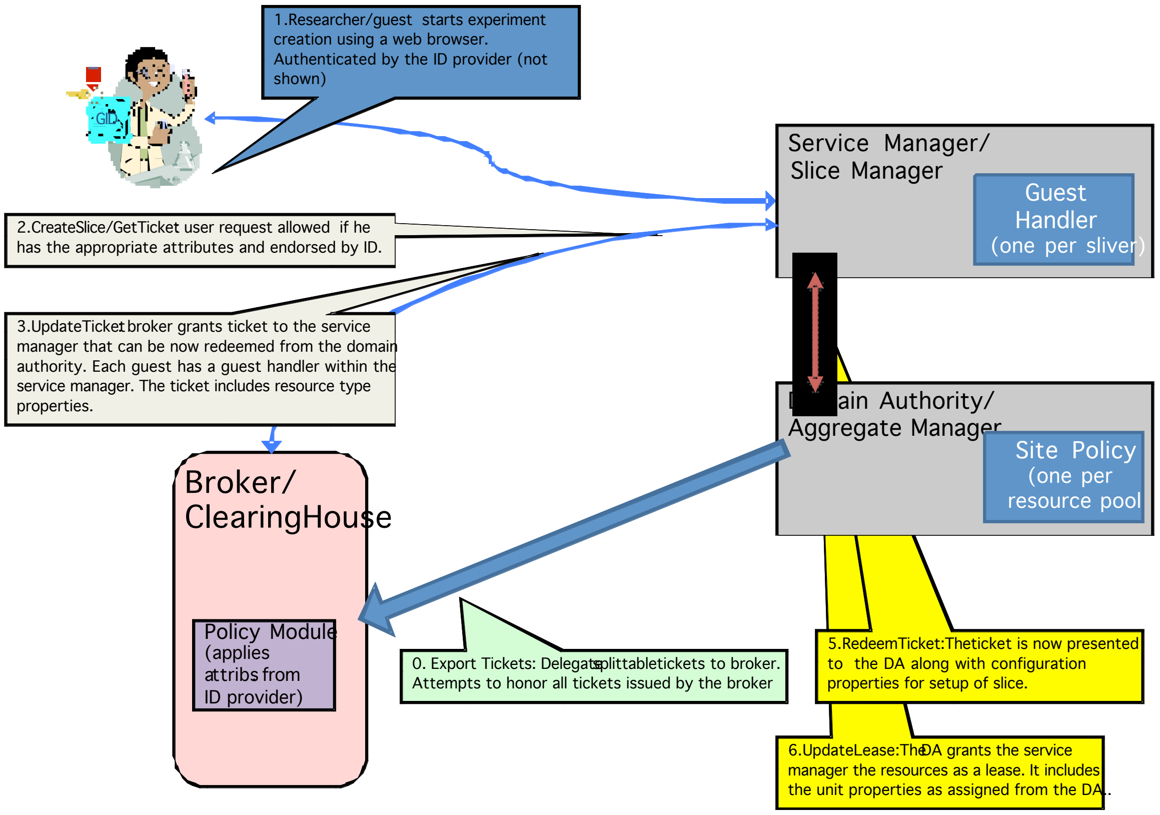

1.Researcher/guest starts experiment creation using a web browser. Authenticated by the ID provider (not shown)
GID
Service Manager/ Slice Manager
Guest Handler (one per sliver)
2.CreateSlice/GetTicket user request allowed if he has the appropriate attributes and endorsed by ID.
3.UpdateTicket: broker grants ticket to the service manager that can be now redeemed from the domain authority. Each guest has a guest handler within the service manager. The ticket includes resource type properties.
Domain Authority/ Aggregate Manager
Site Policy (one per resource pool)
Broker/ ClearingHouse
Policy Module (applies attribs from ID provider)
0. Export Tickets: Delegate splittable tickets to broker. Attempts to honor all tickets issued by the broker
5.RedeemTicket: The ticket is now presented to the DA along with configuration properties for setup of slice.
6.UpdateLease: The DA grants the service manager the resources as a lease. It includes the unit properties as assigned from the DA.

Figure 3: Slice Creation process in ORCA

3.2 ORBIT

Point of Contact: Ivan Seskar (seskar@winlab.rutgers.edu)

Open Access Research Testbed for Next-Generation Wireless Networks (ORBIT) radio grid testbed is developed for scalable and reproducible evaluation of next-generation wireless network protocols. The ORBIT testbed consists of an indoor radio grid emulator for controlled experimentation and an outdoor field trial network for end-user evaluations in real-world settings.

Orbit uses the login account information along with public and private keys for identification and authentication within the testbed. Once a user is authorized, they are permitted to control all aspects of their experiment and to access all the experiment data files.

3.3 ProtoGENI

Point of Contact: Rob Ricci (ricci@flux.utah.edu)

ProtoGENI is essentially a control framework that is based on the Emulab production systems and subsystems enhanced for the unique challenges faced in the GENI environment. The design is based on the knowledge that all entities that ProtoGENI will authenticate have unique global identifiers. ProtoGENI implements a single Public Key Infrastructure (PKI) server which covers authentication of all registries, aggregates and principals. This PKI provides all necessary certificates, and allows verification to be done using a limited number of root certificates. Since it is in a prototype state, it assumes the number of trusted "roots" will be small and can exchange root SSL certificates out of band to populate a certificate directory that can be used for verifying client certificates when they are presented as shown in Figure 4. The ProtoGENI GID consists of a UUID and Human Resolvable Name (HRN) all implemented in the DN of the SSL certificate. The SSL certificate is issued by home Emulab that authenticates the entity in GENI. The DN also includes the email address of the users.

Authentication of the entity is done by the clearing house, on basis of the SSL certificate that is signed by the home Emulab. Authentication implies no permissions, the SSL certificate just indicates the identity of the entity.

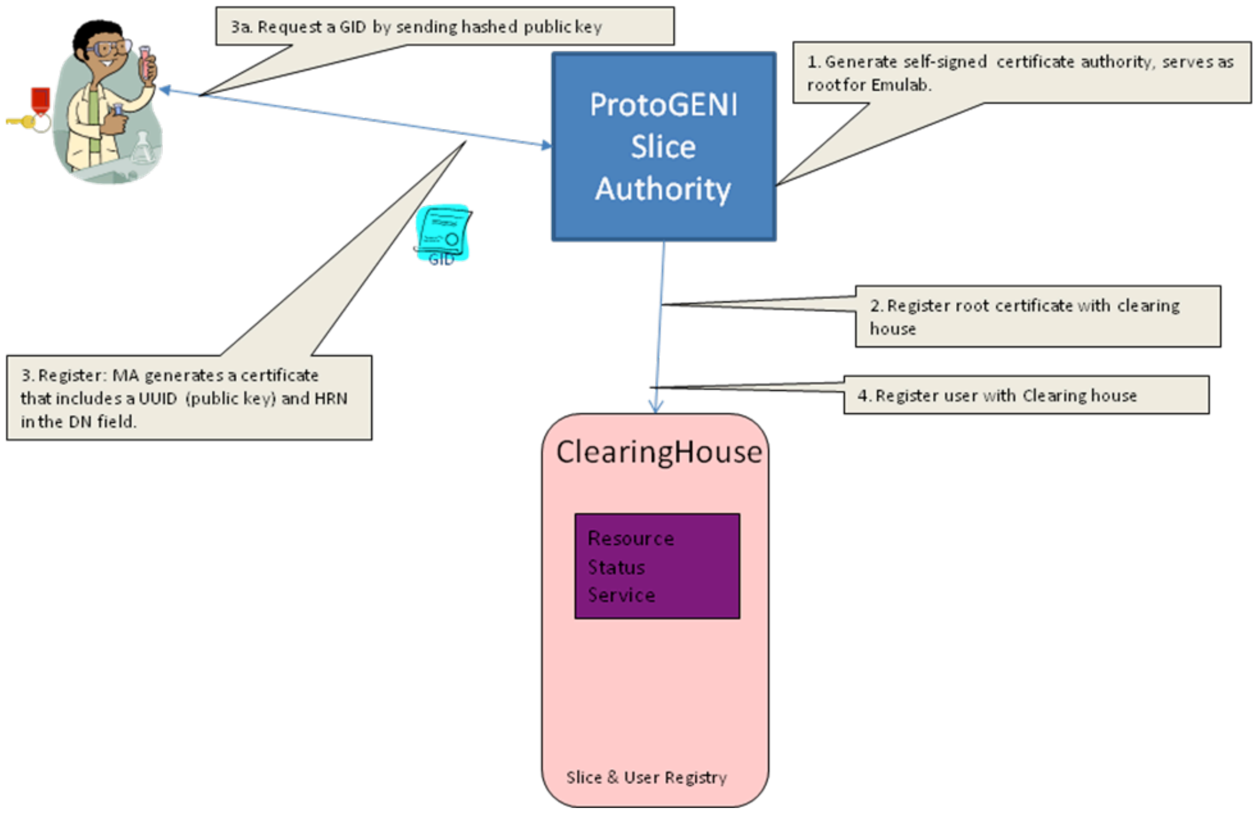
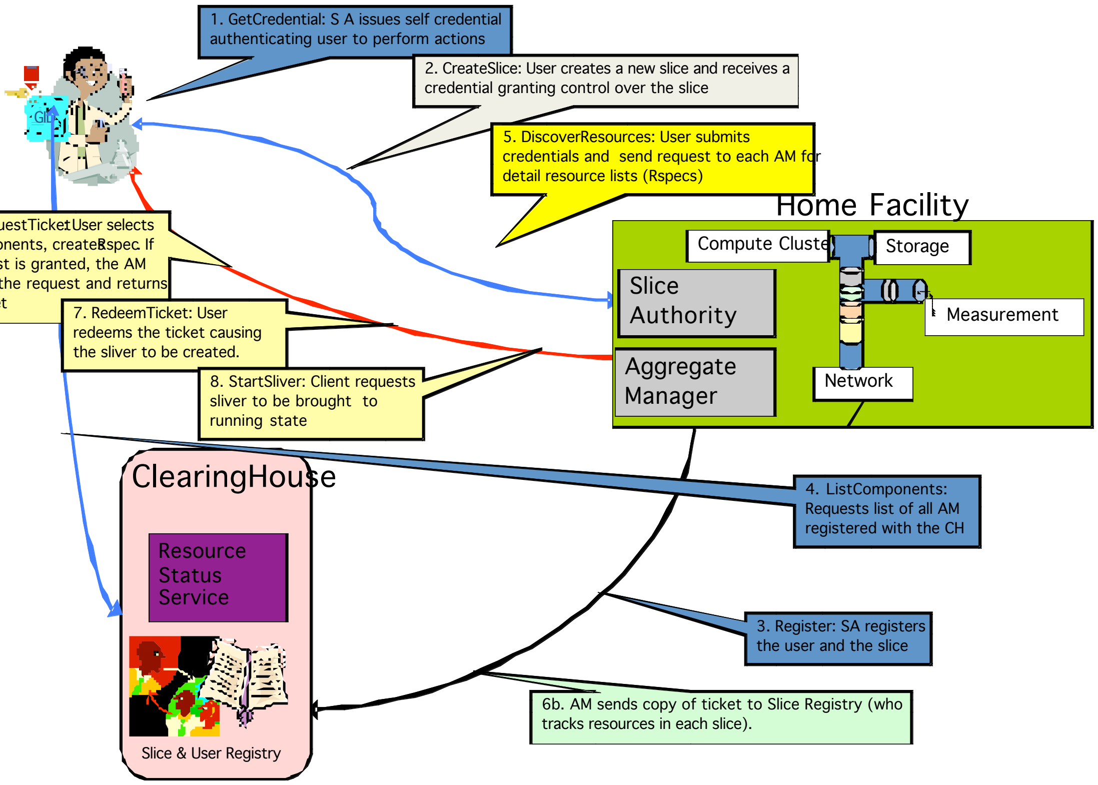

3a. Request a GID by sending hashed public key
1. Generate self-signed certificate authority, serves as root for Emulab.
ProtoGENI Slice Authority
GID
2. Register root certificate with clearing house
3. Register: MA generates a certificate that includes a UUID (public key) and HRN in the DN field.
4. Register user with Clearing house
ClearingHouse
Resource Status Service
Slice & User Registry

Figure 4: Identity and Authentication Mechanisms in ProtoGENI.

ProtoGENI is currently transitioning from the UUID-based identifiers to URN-based identifiers specifically to separate out identity and authentication. Each principal object in ProtoGENI will have a unique URN associated with it. The authority that issued the URN may issue certificates binding authentication material to that URN, that is for example supply the object's public key for authenticating the SSL session. With the identity and authentication functions separated, a service S will authenticate that "the requester is user Joe in the assertion" that is the assertion will contain Joe's identifier (his URN), and additionally Joe will present an authentication certificate that will essentially say "Joe's URN (the same one that was in the credential) is associated with public key X". The authentication certificate must be signed by the authority that issued Joe's URN. Service S will then challenge Joe to be sure he has the associated private key.

Authorization in the ProtoGENI system is initiated by the exchange of credentials that facilitate resource authorization and access control by aggregates as shown in Figure 3. The credentials are certified by the appropriate authorities (slice or aggregate managers) and objects (aggregates, components and slivers) to give them some intrinsic value. These are then certified by an authority or object by signing the token using its own private key, followed by signatures from its responsible authorities, up to the root authority. In the current implementation, there is always only one signature. The Public Key Infrastructure (PKI) that is used to authenticate principals provides all of the keys and other structure to sign and verify credentials. The aggregate that receives this token can then verify it using a set of root certificates.

The slice in ProtoGENI currently is defined as a set of slivers spanning the home Emulab facility along with the project and users associated with the project. The users are authorized and have access to the slivers so that they can run an experiment on the substrate.



1. GetCredential: S A issues self credential authenticating user to perform actions
2. CreateSlice: User creates a new slice and receives a credential granting control over the slice
5. DiscoverResources: User submits credentials and send request to each AM for detail resource lists (Rspecs)
Home Facility
Compute Cluster
Storage
Slice Authority
Measurement
Aggregate Manager
Network
6.RequestTicket: User selects components, creates Rspec. If request is granted, the AM signs the request and returns a ticket
7. RedeemTicket: User redeems the ticket causing the sliver to be created.
8. StartSliver: Client requests sliver to be brought to running state
ClearingHouse
Resource Status Service
4. ListComponents: Requests list of all AM registered with the CH
3. Register: SA registers the user and the slice
Slice & User Registry
6b. AM sends copy of ticket to Slice Registry (who tracks resources in each slice).

Figure 6. Authorization during Slice Creation in ProtoGENI

Figure 5 outlines the slice creation process; the register stage consisting of steps 1-3 where a slice exists in name only and is bound to a project and users; the instantiate stage consisting of stages 4-6 where a slice is initialized on a set of components and resources are assigned to it and finally the activate stage consisting for stages7-8, where the slice is booted and the experiment is active on behalf of the user.

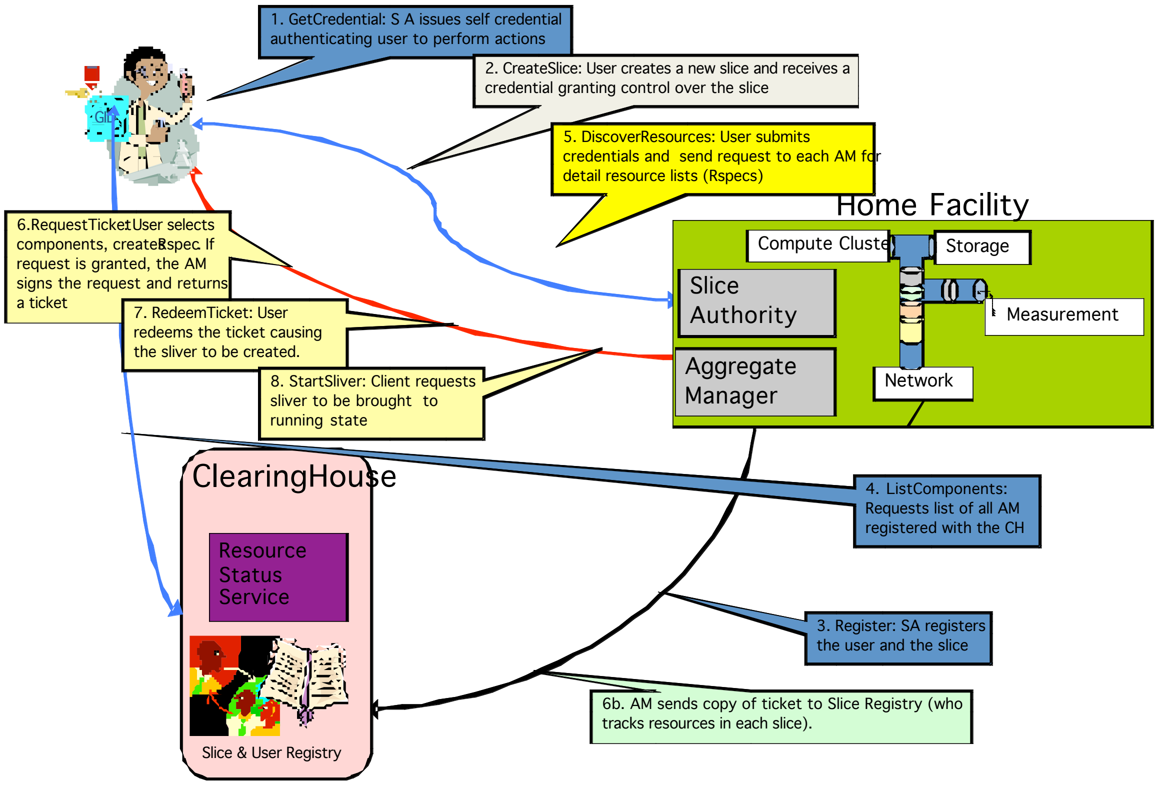

1. GetCredential: S A issues self credential authenticating user to perform actions
2. CreateSlice: User creates a new slice and receives a credential granting control over the slice
5. DiscoverResources: User submits credentials and send request to each AM for detail resource lists (Rspecs)
Home Facility
Compute Cluster
Storage
Slice Authority
Measurement
Aggregate Manager
Network
6.RequestTicket: User selects components, creates Rspec. If request is granted, the AM signs the request and returns a ticket
7. RedeemTicket: User redeems the ticket causing the sliver to be created.
8. StartSliver: Client requests sliver to be brought to running state
ClearingHouse
4. ListComponents: Requests list of all AM registered with the CH
Resource Status Service
3. Register: SA registers the user and the slice
6b. AM sends copy of ticket to Slice Registry (who tracks resources in each slice).
Slice & User Registry

Figure 6. Authorization during Slice Creation in ProtoGENI

The ProtoGENI suite thus uses certificates and credentials to authenticate entities. This approach combines identity and authentication mechanisms.

3.4 PlanetLab GENI

Point of Contact: Scott Baker (smbaker@gmail.com)
Andy Bavier (acb@cs.princeton.edu)

PlanetLab is a system that allows researchers to conduct experiments on hosts located at various locations around the world, by providing a global research network that supports the development of new network services, distributed storage, network mapping, peer-to-peer systems, distributed hash tables, and query processing. The PlanetLab prototype is based on the geniwrapper module. The current implementation consists of PlanetLab Central (PLC) that bundles together an aggregate manager, a slice manager, and a registry server. Individual PlanetLab nodes correspond to components and run a component manger. The PlanetLab prototype maintains all authoritative state at PLC. Individual nodes maintain only cached state that will be updated when a node fails or reboots.

PlanetLab also implements a single Public Key Infrastructure (PKI) server which covers authentication of all registries, aggregates and principals. This PKI provides all necessary certificates. The GID consists of a UUID and Human Resolvable Name

(HRN) implemented in the subject-alt-name field of the SSL certificate. The SSL certificate is issued by the authority that is responsible for the entity. It authenticates the entity in GENI by signing the certificate as shown in Figure 7: Identity and Authentication mechanisms in PlanetLab. The geniwrapper (http://svn.planet-lab.org/wiki/GeniWrapper) uses two crypto libraries: pyOpenSSL and M2Crypto to implement the necessary cryptographic functionality and the X.509 certificates, while public-private key pairs are implemented by the Keypair class.

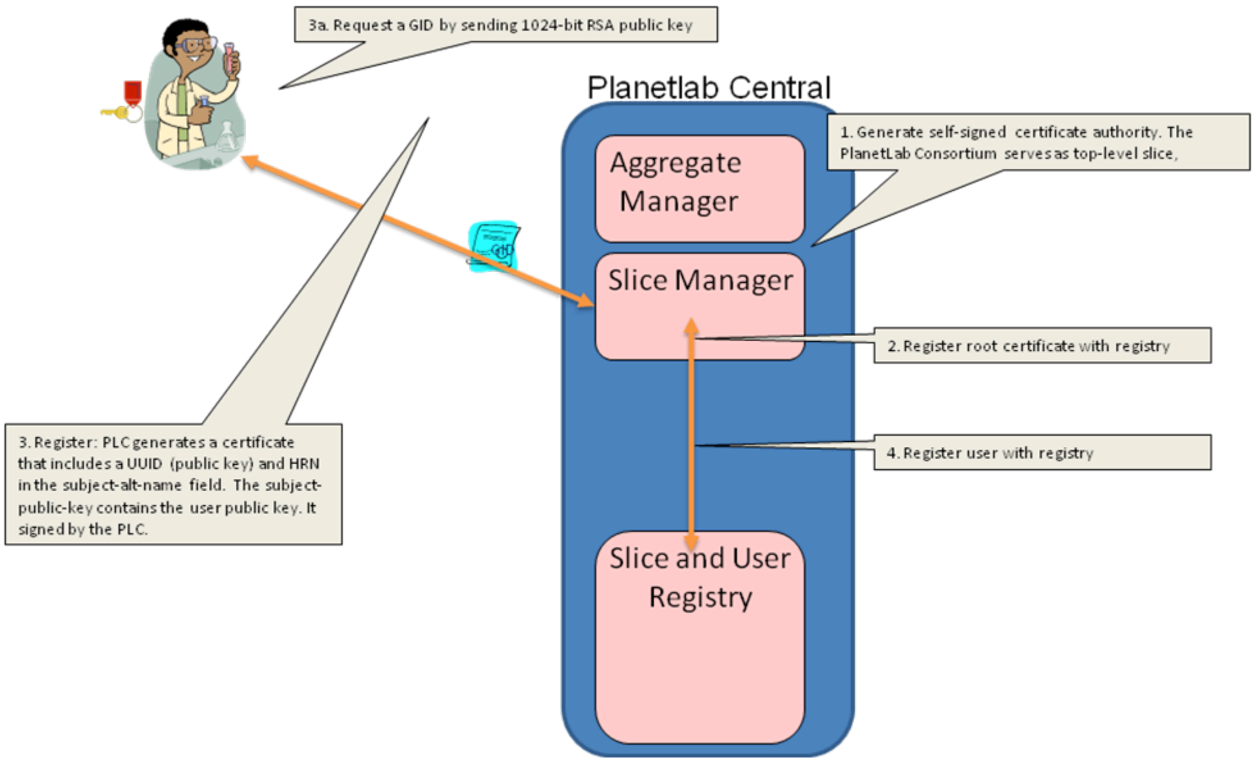
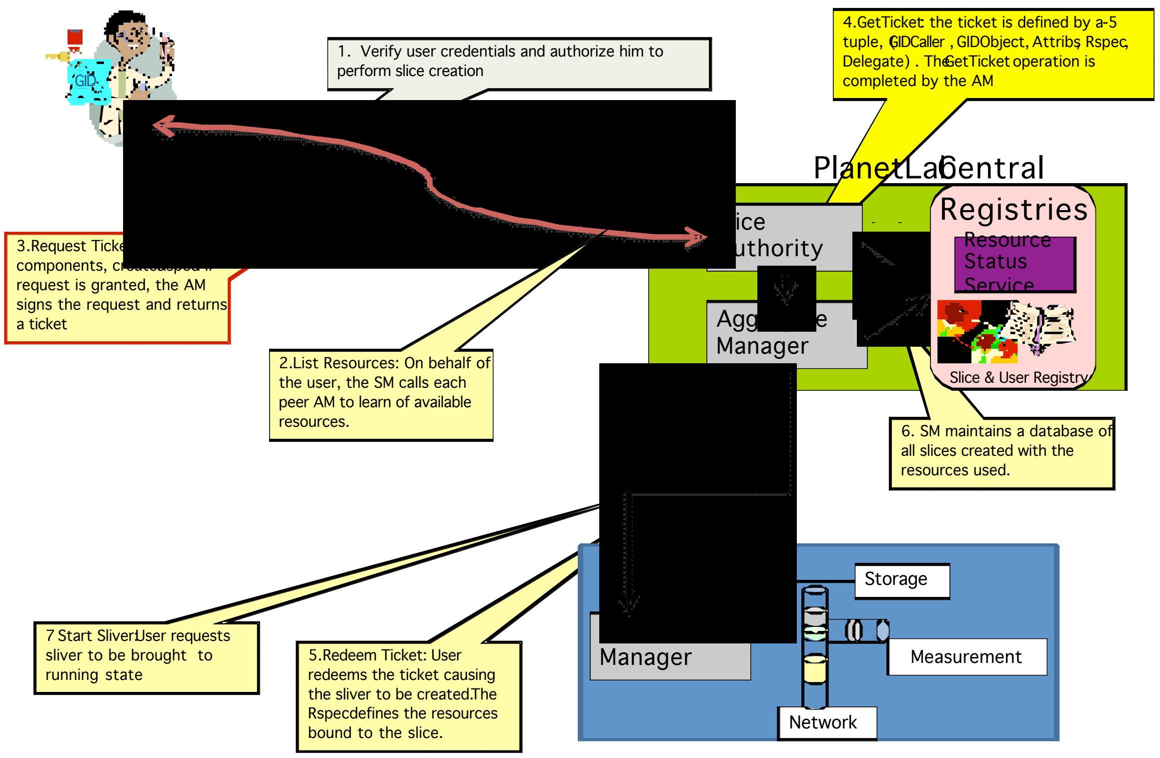

3a. Request a GID by sending 1024-bit RSA public key
Planetlab Central
1. Generate self-signed certificate authority. The PlanetLab Consortium serves as top-level slice,
Aggregate Manager
Slice Manager
2. Register root certificate with registry
4. Register user with registry
3. Register: PLC generates a certificate that includes a UUID (public key) and HRN in the subject-alt-name field. The subject-public-key contains the user public key. It signed by the PLC.
Slice and User Registry
GID

Figure 7: Identity and Authentication mechanisms in PlanetLab

All subsequent actions in the PlanetLab prototype contain a credential that consists of the GID of the caller, which in turn contains the public key of the caller. The PLC ensures that this public key matches the public key that is being used to decrypt the HTTPS connection's session key, thus ensuring the caller must possess the private key that corresponds to the GID and hence authenticating the user.

Authorization in the PlanetLab system is initiated by the exchange of credentials that facilitate resource authorization and access control by aggregates. Figure 8: Authorization during Slice creation process in PlanetLabFigure 8 shows the slice creation process in PlanetLab.



1. Verify user credentials and authorize him to perform slice creation
4.GetTicket: the ticket is defined by a 5-tuple, (GIDCaller , GIDObject, Attribs, Rspec, Delegate) . The GetTicket operation is completed by the AM
GID
PlanetLab Central
ice
uthority
Registries
Resource
Status
Service
Agg e
Manager
Slice & User Registry
3.Request Ticket: components, c... request is granted, the AM signs the request and returns a ticket
2.List Resources: On behalf of the user, the SM calls each peer AM to learn of available resources.
6. SM maintains a database of all slices created with the resources used.
Storage
Manager
Measurement
Network
7 Start Sliver: User requests sliver to be brought to running state
5.Redeem Ticket: User redeems the ticket causing the sliver to be created.The Rspec defines the resources bound to the slice.

Figure 8: Authorization during Slice creation process in PlanetLab

Once a user credentials are validated by the slice manager, the user can initiate the slice creation by invoking the GetTicket operation. A ticket in PlanetLab is a five-tuple consisting of (GIDCaller, GIDObject, Attributes, RSpec, Delegate) where GIDCaller is the GID of the principal performing the operation, GIDObject is the GID of the slice to which the ticket is bound, attributes is the set of PlanetLab attributes and RSpec is the set of resources bound to the slice. Once the ticket is generated for the user, it can then be redeemed at the respective aggregate managers.

In PlanetLab users invoke the **sfi** command to manage their slices. **sfi** manages a set of credentials on behalf of the user to invoke various slice or registry operations. There are essentially three types of credentials: user credential that enables retrieving information in the registry, the slice credential to control and terminate the slice, and if the user also serves as PI for a research organization, an authority credential that authorizes him to register nodes, slices, and users in the registry. Typically there is one user and authority credential, there may be multiple slice credentials.

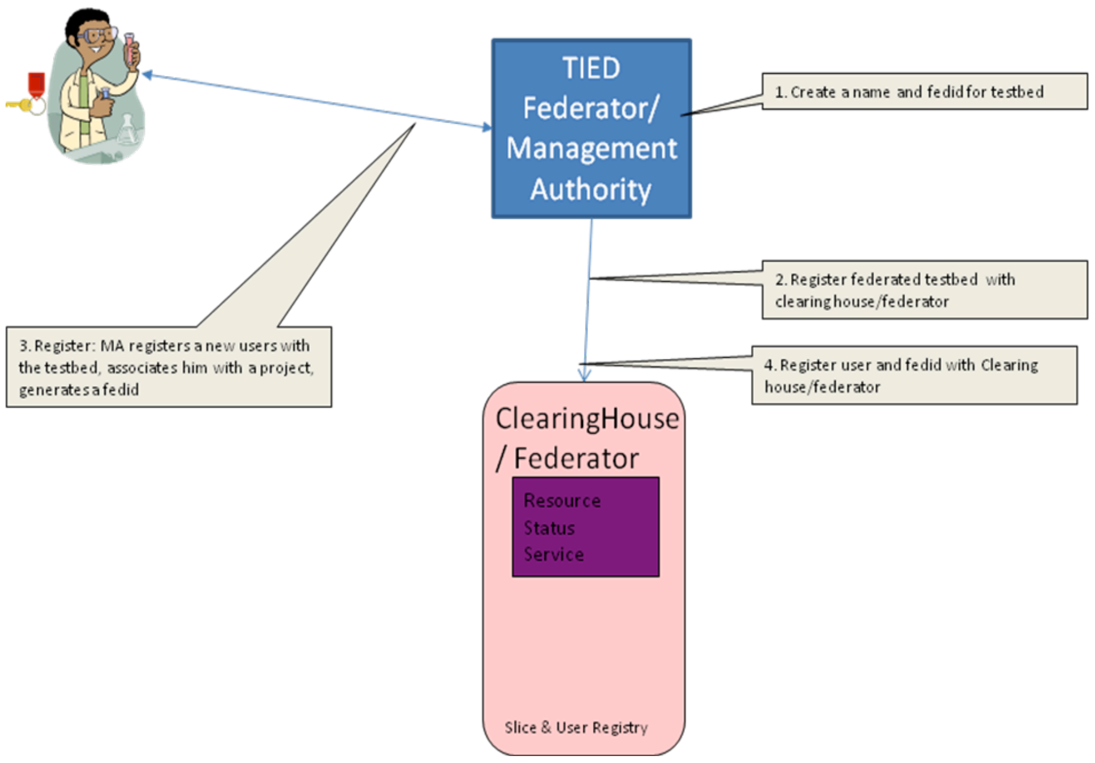
4. The Federated Suites

4.1 TIED

Point of Contact: Ted Faber (faber@isi.edu)

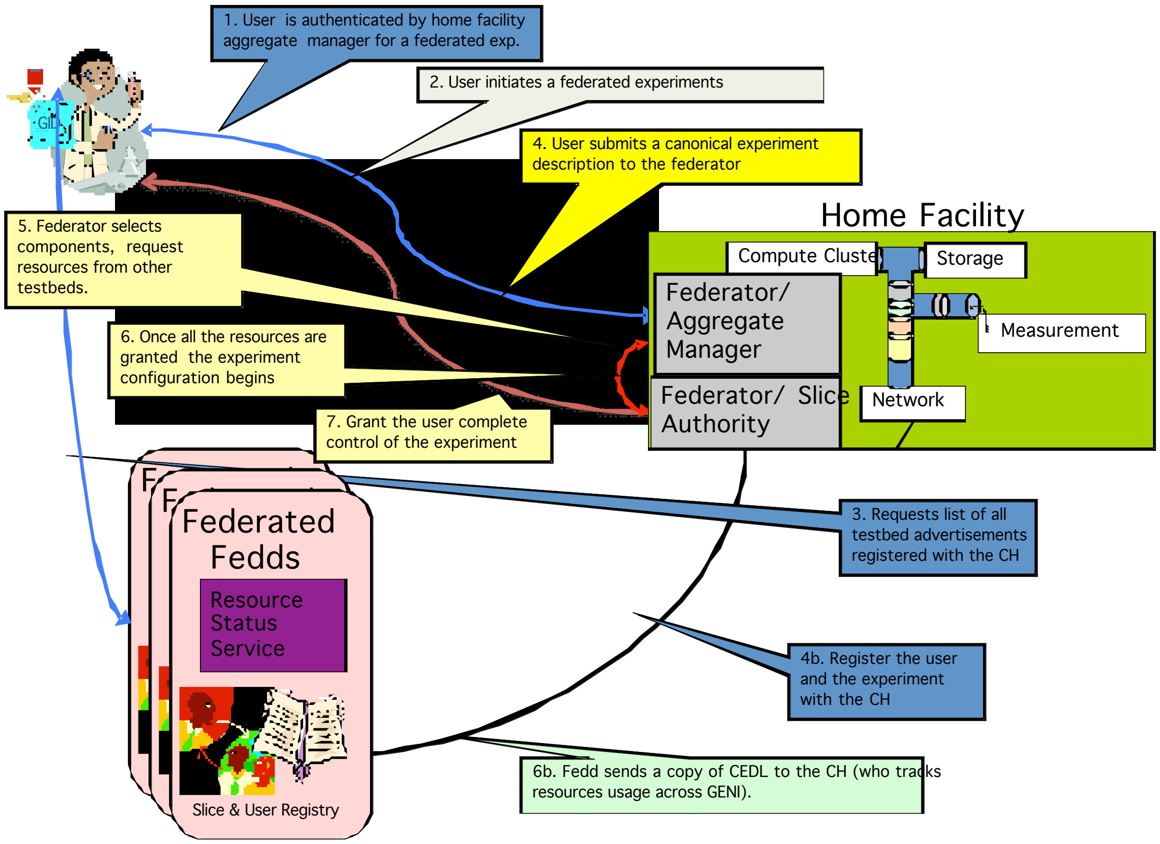
Trial Integration Environment built on DETER (TIED) is a testbed based on Emulab software that is specifically enhanced for security research by providing test suites, methodologies and tools for network security tests. TIED allows on-demand creation of experiments spanning multiple independently controlled facilities enabling federated experiments to create a coherent distributed environment, manage federated resources by applying appropriate security mechanisms, and provide a unified runtime environment to the researcher and experiment. The TIED federator (fedd) translates experiment requirements encoded in a canonical experiment description language and maps them to a federated experiment across multiple testbeds transparently for the experimenter.

Each users, projects and testbeds has a globally unique name. Typically in Emulab, projects are created by users within projects and those attributes determine what resources can be accessed. TIED generalized this idea into a testbed, project, user triple that is used for access control decisions. A requester identified as ("DETER", "proj1", "faber") is a user from the DETER testbed, proj1 project, user faber. Testbeds contain projects and users, projects contain users, and users do not contain anything. Testbeds make decisions about access based on these three level names. For example, any user in the "emulab-ops" project of a trusted testbed may be granted access to federated resources. It may also be the case that any user from a trusted testbed is granted some access, but that users from the emulab-ops project of that testbed are granted access to more kinds of resources. TIED also defines federation identifiers. They are 160-bit SHA-1 hash of the public key to avoids collisions when federating. A triple name can be replaced by a fedid as follows (fedid:1234, “proj1”, “faber). Figure 6 shows identification and authentication in TIED. Basically, authentication is at the home testbed, using priv-pub key pairs as shown below.


TIED Federator/ Management Authority
1. Create a name and fedid for testbed
2. Register federated testbed with clearinghouse/federator
3. Register: MA registers a new users with the testbed, associates him with a project, generates a fedid
4. Register user and fedid with Clearing house/federator
ClearingHouse / Federator
Resource Status Service
Slice & User Registry
Figure 9: Identification and Authentication in TIED

Authorization and access control within the TIED control framework is managed at the project level, that is, projects control resource access, each user's project membership level determines access to project resources as shown in Figure 7. Once a fedd has decided to grant a researcher access to resources, it implements that decision by granting the researcher access to an Emulab project with relevant permissions on the local testbed. The Emulab project to which the fedd grants access may exist and contain static users and resource rights, may exist but be dynamically configured by fedd with additional resource rights and access keys, or may be created completely by fedd. Completely static projects are primarily used when a user wants to tie together his or her accounts on multiple testbeds that do not bar that behavior, but do not run fedd.

Whether to dynamically modify or dynamically create files depends significantly on testbed administration policy and how widespread and often federation is conducted. In Emulabs projects are intended as long-term entities, and creating and destroying them on a per-experiment basis may not appeal to some users. However, static projects require some administrator investment per-project. The TIED authorization framework is built on the assumptions that the federated testbeds will be decentralized with alliances changing frequently. However, it is also necessary to support multiple trust models, (for example, hierarchical PKI, PGP web of trust) and explicit decision making in TIED-based testbed federations.



1. User is authenticated by home facility aggregate manager for a federated exp.
2. User initiates a federated experiments
4. User submits a canonical experiment description to the federator
Home Facility
Compute Cluster
Storage
Federator/ Aggregate Manager
Measurement
Federator/ Slice Authority
Network
5. Federator selects components, request resources from other testbeds.
6. Once all the resources are granted the experiment configuration begins
7. Grant the user complete control of the experiment
Federated Fedds
Resource Status Service
Slice & User Registry
3. Requests list of all testbed advertisements registered with the CH
4b. Register the user and the experiment with the CH
6b. Fedd sends a copy of CEDL to the CH (who tracks resources usage across GENI).

Figure 10: Authorization during experiment creation in TIED

TIED is currently prototyping attribute based access control as it will allow fine grain control along with support to scale to thousands of users and experiments in TIED. Essentially in the current prototype, a principal's identity is established by local authorities using local techniques, principal's attributes are determined locally and established by digitally signed credentials. The attributes and rules then drive a reasoning engine that determines authorization decisions.

ABAC facilitates authorization decisions by providing rules under which actors in the system, called principals, prove that they have certain attributes necessary for accessing resources. Which attributes are required for a given resource is a matter of policy outside the system. ABAC can represent delegation of various forms in scalable and separable ways that can be reasoned about formally. In ABAC, principals can be an individual (researcher, user) or larger authority (GPO, university). Principals can use a range of systems to authenticate themselves. A principal can be the subject of authorization decisions and have attributes asserted about it by other principals.

An attribute is a property of a principal created by the assertion of another principal. The University of Southern California (a principal) may assert that Ted Faber (a principal) is a staff member (attribute). The attributes are scoped by principal, that is if USC asserts Ted Faber is staff, that is one attribute, if ISI also asserts that Ted Faber is staff that is a second attribute. Assertions are represented as a digitally signed statement, called a credential. A given principal may also assert rules about how attributes relate. The GPO may assert that all USC GENI staff are also GPO prototypers. That delegates authority to USC to add to GPO prototypers. In this case the delegated attribute (GPO prototypers) is given to principals who also possess the delegating attribute (USC GENI). Finally, a principal may delegate at one remove. The GPO may assert that any NSF PI (any principal that the NSF has asserted a PI attribute about) can designate a principal as a GENI user and that user will be a GPO GENI user. The NSF can affect GPO GENI users by creating or deleting PIs; that is, by adding or removing assertions that a particular principal is a PI. Individual PIs can also directly designate local GENI users that are also GPO GENI users as above.

In this case, the delegated attribute (GPO GENI user) is delegated to principals who possess a one (or more) of a set of attributes (*P* GENI user for many *P*). That set is defined in terms of an authorizer attribute (NSF PI). Any principal with the authorizer attribute can assign the delegated attribute by assigning their local version of the delegating attribute (*P* GENI user where *P* has the NSF PI attribute). This links the authorizer attribute to the delegating attributes, and is often called a linked attribute. Each of these delegations is expressed as an ABAC credential: a signed assertion that can be used in a proof. Because each of these is a signed assertion of a fact or delegation of authority, connecting them in following the rules above corresponds to collecting those signed credentials, which establishes a trust relationship. ABAC credentials allow principals to negotiate directly about what they consider adequate proof.

Until an authorization decision needs to be made, all of these credentials can be kept locally and brought together to make the decision. Principals can also pass them around so they are available when needed. For example, when the NSF designates a PI, it may send that PI the signed attribute so that the PI can use it in authorization requests.

4.2 PlanetLab Fed

Point of Contact: Larry Petterson (llp@cs.princeton.edu)

This effort will integrate PlanetLab (PLC), PlanetLab Europe (PLE), PlanetLab Japan (PLJ)-along with other testbeds in Korea, Brazil, Europe, Japan, and the US-into an international federated research infrastructure. They will deploy and use the federation mechanisms developed as part of the PlanetLab cluster and focus on the policy issues that arise when autonomous organizations federate their networks together.

5. Other projects with Security Requirements in Spiral 2

GENI is being designed and prototyped by the research community, with project management and system engineering provided by the GENI Project Office (GPO). Each GENI spiral lasts for a 12 month phase. Spiral I's primary goal was to develop, integrate, and attempt to operate very rudimentary, end-to-end working prototypes, as rapidly as possible, then co-evolve them with the community's evolving research vision. Several new projects are starting in Spiral 2 to fill in several critical "missing pieces," including: security requirements and architecture, experiment workflow tools and user interfaces, and prototypes for instrumentation and measurement. Additional projects will build upon Spiral 1 achievements to date, including support for international and commercial federations and several early "shakedown" experiments that will prove critical in guiding system design. This section discusses ten projects that SECARCH has closely looked at at this stage in Spiral 2 and their security implications.

5.1 Embedded Real Time GENI

Point of Contact: TBD

The Embedded Real Time Measurement Framework for GENI includes the technology to support cross-layer communications, specifically, the ability to incorporate a diverse set of real-time measurements in networking protocols. They are targeting architectural experimentations across diverse heterogeneous technologies by supporting real-time cross-layer communications and measurements. The objective is to develop networking capabilities within the GENI infrastructure that enable deeper exposure of cross-layer information and user access to real-time measurements.

They have developed a set of specifications for enabling real-time measurements within the substrate and specifications for networking protocols based on the GENI requirements for real-time user-accessed cross-layer measurements is also currently under development. They identify a set of specifications for the implementation of a unified and integrated measurement framework called UMF, which is the Unified Measurement Framework, with the goal of limiting the hardware and software overhead and complexity associated with accessing measurement data.

They have developed a measurement framework based on GENI real-time measurement requirements and other resources within the GENI prototyping activities. It discusses the number of software architectures dedicated to network measurements which could serve as an interface between a unified measurement framework (UMF), the control framework, and the GENI experimenter. Their prototype assess several network management protocols and data exchange formats for exchanging measurement and control information between the substrate's performance monitors and the UMF, and between the UMF and the GENI control frameworks.

The UMF serves as a means for gathering physical layer measurements and conveying the data to the GENI researchers in an aggregated, unified way. They have also evaluated the corresponding networking protocol and management languages that will be essential in our implementation of the UMF. They have joined the ORCA Cluster (D) and plan to integrate UMF within the ORCA-BEN aggregate. This will enable real-time measurement experimentation using the existing networking elements (NEs) embedded in the BEN metro-area optical network.

In the scope of Spiral 2, the first major task is to design an implementation of the UMF that can be integrated within the ORCA Cluster. However, this design should be general enough such that it can be easily extended to other GENI control frameworks in the future. Thus ERM uses the NetFPGA Cube, which is an integrated system composed of a general purpose processor, in addition to the proprietary NetFPGA hardware. The UMF comprises of both a software component (run on the general purpose processor), as well as a hardware component (run on NetFPGA card).

5.1.1 ERM integration with ORCA

The main goal of the UMF is to present a uniform view and an abstraction of the measurement capabilities within a substrate and make them accessible to, and sliceable by, a control framework. As such, the UMF is required to interface with both the GENI control framework, as well as to a set of NEs within the GENI network substrate. Figure 4 shows an architectural flowchart of how the UMF interfaces specifically to the ORCA control framework and its NEs. The green dotted lines depict the flow of measurement commands, such as the signal monitoring commands, downstream from the ORCA control framework, through the UMF, down to the underlying NEs. The blue solid lines show the flow of retrieved measurement data from the NEs, up to the UMF and the ORCA control framework to be processed or stored.

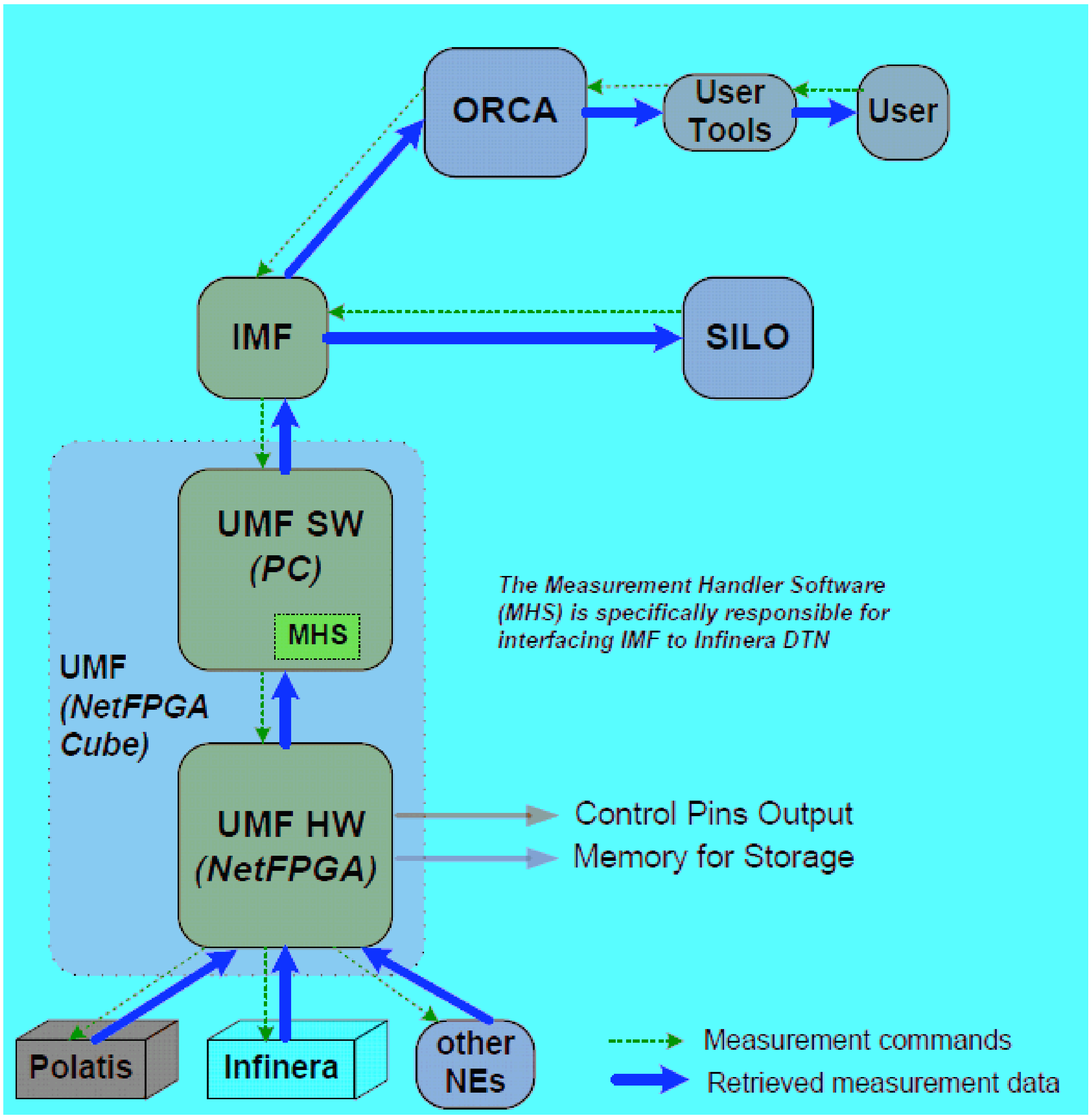


Figure 11. ERM Integration with ORCA--BEN

The UMF interfaces with the ORCA control framework via the integrated measurement framework (IMF). The ORCA control framework is in charge of managing the network resources and allocating slices of the available resources to the GENI researchers. ORCA receives the measurement data from the IMF and can choose to store it locally, or send it to the GENI users' external tools for further processing, or storage. The IMF is also responsible for interfacing the UMF to the Services Integration, controL, and Optimization (SILO) framework, which is an infrastructure for a non-layered internetworking architecture in which complex communication tasks are accomplished by combining functional blocks in a configurable manner. The cross-layered experimentation capability of SILO can be combined with the unified measurement capability of UMF to enable substrate measurement as a service in a custom protocol stack (i.e. a silo).

The first UMF prototype is implementing using the NetFPGA Cube, which is an integrated system composed of a general purpose processor, in addition to the proprietary NetFPGA hardware. The UMF comprises of both a software component (which runs on the general purpose processor), as well as a hardware component (which runs on the NetFPGA card). Next we outline the functionalities of the software and hardware components along with the security implications.

The software component of the UMF is responsible for interfacing with the IMF and sends measurement command information to the UMF, for example, slice allocation information, ID of NE to make the measurement, a measurement metric and rate such as power, bit-error rate (BER), final destination of the retrieved measurement data for processing or storage (ORCA, SILO, User tools, or UMF)

Thus the IMF is in charge of interfacing with the ORCA and SILO frameworks, and the applications running on them. It then communicates with the UMF which measurement metric the higher level applications wish to retrieve and what NE to use. Then, the UMF is responsible for actually interfacing with all available NEs, by sending vendor specific commands to them and receiving measurements from them. Once the UMF SW receives the measurement information from the NEs, it can store it locally or forward it up to the ORCA, SILO, or User tools for processing and/or storage. This design provides flexibility of design and may make it easier to secure as the prototype evolves to include other control frameworks. This design, however, still faces the same challenges in terms of access control, privacy, and security of measurement data.

The UMF HW is implemented in a specialized NetFPGA to provide timing-sensitive processing. If the measurement command information sent to the UMF HW by the UMF SW specifies a measurement to be made only once, then there is no time sensitivity, the vendor-specific measurement command sent from the UMF SW can be directly forwarded to the actual NE. However, if the measurement metric update rate is some fixed time interval, then the UMF HW will be in charge to keeping count of this time interval while repeatedly sending out the measurement command to the actual NE. The UMF HW will also manage or store the upward flow of measurement data should the NE be set to stream measurement information. Upon receiving the retrieved measurement data from the NE, the UMF HW can directly process the data and control/actuate some local hardware using the special I/Os available to the NetFPGA (such as GPIO, Ethernet, Serial, etc). This is only done if it is specified that the UMF HW should perform the processing. Otherwise, the UMF HW forwards the retrieved measurement data upstream for processing or storage.

5.2 Instrumentation Tools for a GENI prototype

Point of Contact: James Griffioen (griff@netlab.uky.edu)

This project provides instrumentation capabilities that give the GENI users the ability to monitor and better understand the runtime behavior of their experiments. These capabilities are currently being integrated within the ProtoGENI cluster. Much of the work on testbeds is primarily focused on creating, setting up, and running an experiment but this is only the first of many steps in an experiment. The real challenge can often be monitoring and analyzing the behavior of an experiment as it is a very involved, time consuming, and usually a manual process that is repeated many times. It requires setting up a monitoring environment.

While it is relatively easy to setup a monitoring and analysis environment for a single testbed, scaling it to federated testbeds and aggregates that span across the Internet can be extremely challenging due to bandwidth and storage constraints. In the current prototype, each experiment has its own *measurement controller* (MC) that controls and collects all packets and network states from the measurement points in that experiment. *A measurement point* is a simple packet sensor or a node and network state sensor. Experiments that span multiple aggregates have a minimum of one measurement controller for each aggregate. The measurement controller is implemented as an additional sliver in the experiment, either as an additional node or a virtual node. As shown in the figure below, the measurement controllers do not depend on the experiment links for connectivity, they has their own out-of-band connectivity with each of the measurement points. The measurement controllers are also highly provisioned nodes, and thus have plenty of storage and network connectivity resources to support all the monitoring activity with the experiment.

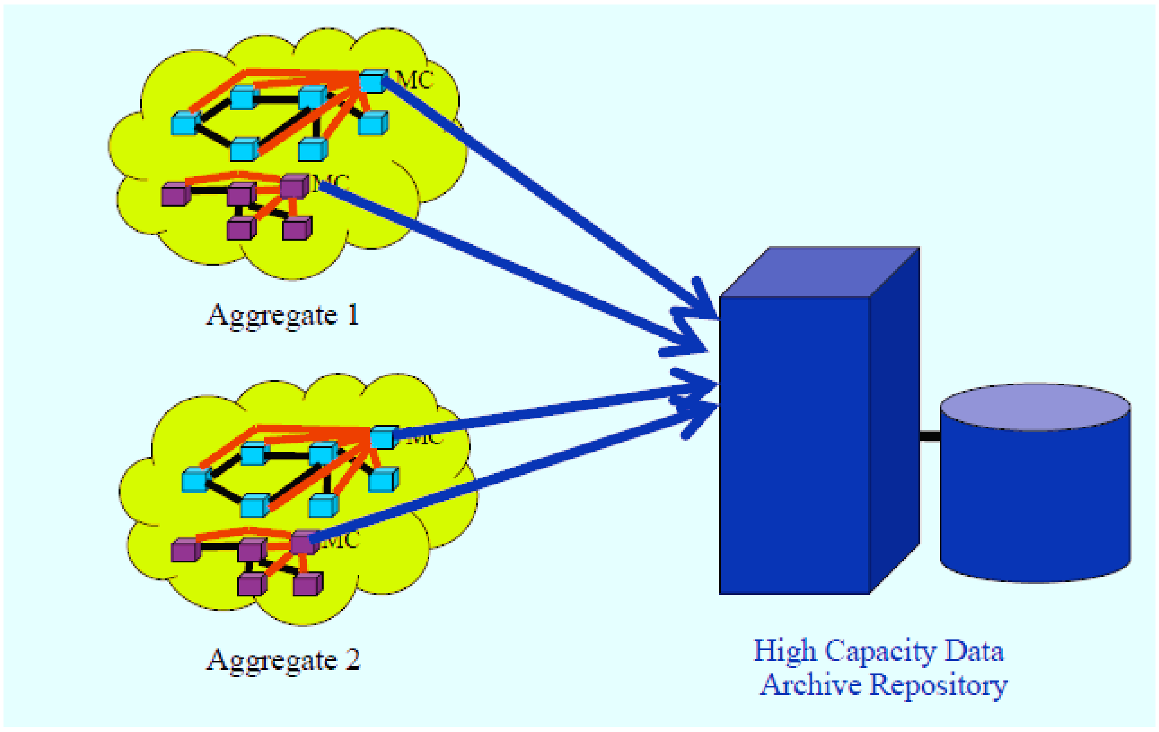


Figure 12: Instrumentation tools for monitoring and analyzing a GENI experiment. Each aggregate has a measurement controller that correlates statistics from multiple measurement points within the experiment.

The measurement point captures packet state information using tcpdump and node state using process daemons. Additionally, there is a control daemon to enable and disable monitoring. The first prototype accessed the experiment topological information directly from the Emulab database. However for integration with ProtoGENI, experiment topological information is accessed through XML-RPC calls on the clearinghouse and component managers and the monitoring is done from an SNMP daemon that has been specially configured to work inside Emulab's virtual nodes. Further the project uses experiment-specific (i.e., slice-specific) measurement nodes, thus creating a local measurement system within each experiment. This design matches the standard usage model in which users are primarily interested in collecting information about their own experiment. It allows users to keep measurement data private and local, but still allows data to be made public if desired.

5.3 Million Node GENI

Point of Contact: Justin Cappos (justinc@cs.washington.edu)

The million-node GENI project provides an end host deployment platform called Seattle that enables networking and distributed systems research by users contributing resources as the second type of opt-in mechanism. Seattle's architecture is comprised of three components. First, at the lowest level the sandbox component called the *vessel* guarantees security and resource control for an individual program. Programs are written to the Seattle API in a subset of the Python programming language (restricted python). This API provides portable access to low level operations (like opening files and sending messages). Vessels prevent the program running in them from performing unsafe actions (like opening the computer user's files) Vessels also have a specified number of resources they are allowed to consume. The vessel restricts the program from consuming more than the allowed number of resources.

Second, at a higher layer, the *node manager* determines which sandboxed programs get to run on the local computer. A public key infrastructure is used to authorize control over programs running within the vessels. The node manager restricts access to the vessels to only authorized parties. For example, every vessel has an owner and a set of users. The owner can change the set of users, change ownership to another party, split the vessel into multiple vessels, and other similar operations. Users are allowed to upload files to the vessel, start and stop programs, read the vessel's log, and other simple operations. The node manager also ensures that the total amount of consumed resources does not vary as vessels are split and joined. Lastly, the experiment manager lets students control their program instances across computers. Seattle programs are portable as students' code can run across different operating systems and architectures without change. An experiment manager locates vessels that the user controls and interacts with the node manager to control those vessels. For example, an experiment manager may take a user's command to deploy foo.py everywhere and go to contact all of the vessels the user owns on each of the node manager.

5.4 Enterprise GENI

Point of Contact: Rob Sherwood (rob.sherwood@stanford.edu)

The Enterprise GENI project is focuses on how GENI can be deployed on local networks, such as campus and enterprise networks, and to develop a kit that allows the easy deployment of OpenFlow in other networks. Specifically, in addition to deploying Enterprise GENI in a campus network, it will integrate the OpenFlow network with a GENI control framework, for example, PlanetLab, by an Aggregate Component Manager and provide access to Enterprise GENI testbeds to other GENI users. It also aims to define an Enterprise GENI deployment kit for research and potential commercial transition.

OpenFlow is a way for researchers to run experimental protocols in regular operational networks. The OpenFlow design has an internal flow-table and a standardized interface to add and remove flow entries on a Ethernet switch. It allows researchers to rapidly evaluate their ideas and protocols in real-world traffic settings and OpenFlow could serve as a useful campus component in a large-scale testbed like GENI. OpenFlow provides an open protocol to program the flow table in different switches and routers. A network administrator can partition traffic into production and research flows and researchers can control their own flows by choosing the routes their packets follow and the processing they receive. In this way, researchers can try new routing protocols, security models, addressing schemes on the same network with the production traffic is isolated and processed without any changes.

An OpenFlow switch consists of at least three parts: (1) A Flow Table, with an action associated with each flow entry, to tell the switch how to process the flow, (2) A Secure Channel that connects the switch to a remote control process (called the controller), allowing commands and packets to be sent between a controller and the switch using (3) The OpenFlow Protocol, which provides an open and standard manner for a controller to communicate with a switch.

An entry in the Flow-Table has three fields: (1) a 10-tuple packet header that defines the flow, (2) the action, which defines how the packets should be processed, and (3) statistics, which keep track of the number of packets and bytes for each flow, and the time since the last packet matched the flow.

There are three basic actions that are currently supported within the OpenFlow switch:

1. Forward this flow's packets to a given port (or ports). This allows packets to be routed through the network. In most switches this is expected to take place at line rate.
2. Encapsulate and forward this flow's packets to a controller. Each packet is delivered to a Secure Channel, where it is encapsulated and sent to a controller. This mechanism is typically used for the first packet in a new flow, so a controller can decide if the flow should be added to the Flow Table. Alternately, in some experiments, it could be used to forward all packets to a controller for processing, albeit with potential performance impacts and limitations.
3. Drop this flow's packets. This can be used for security, to curb denial of service attacks, or to reduce spurious broadcast discovery traffic from end-hosts.

For support within the GENI testbed, Enterprise GENI first developed a Network Virtualization Software that allows experimental use of the production networking infrastructure. The two basic components of this infrastructure are a FlowVisor and the Aggregate Manager. The FlowVisor virtualizes a physical OpenFlow switch into multiple logical OpenFlow switches, which can be controlled and operated by different experimenters. The FlowVisor basically appears to a network of OpenFlow switches as a single controller running as open-source software on a Linux PC. The switches continue to run an unmodified OpenFlow Protocol. The FlowVisor is critical to allowing multiple experimenters to run independent experiments simultaneously in one physical campus network.

The FlowVisor consists of two main parts:

1. A policy engine that defines the logical switches (e.g. "all http traffic", "A's traffic", "Network administrator's experimental traffic between 12midnight and 3am")
2. An OpenFlow mux/demux that implements the policy by directing OpenFlow commands to/from an OpenFlow controller and its network of logical OpenFlow switches.

The Aggregate Manager is built on top of the FlowVisor, and can be defined as an OpenFlow controller that controls a subset of the network resources, as defined by the local administrator.

5.4.1 Enterprise GENI and PlanetLab Integration

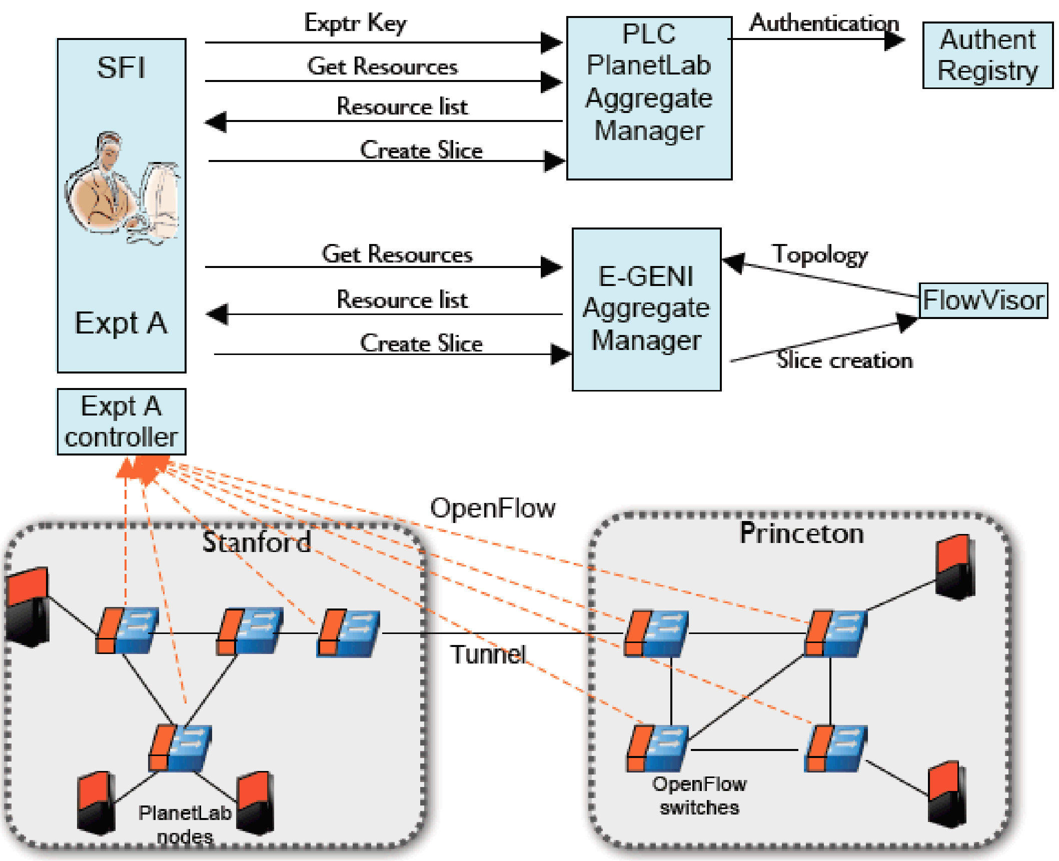


Figure 13: PlanetLab and Enterprise GENI integration Sept 2009

The high level idea of the integration between PlanetLab GENI and Enterprise GENI is to use the same control framework for the creation of a slice across both the substrates. Thus, a user can now create and run an end-to-end experiment across both the PlanetLab GENI and the Enterprise GENI substrates using the PlanetLab SFI interface tool. The computing resources are provided by PlanetLab whereas the networking resources are provided by Enterprise GENI for the experiment.

As shown in the figure above, first the user is authenticated using an experimenter's key with the authentication registry contacted through the PlanetLab Aggregate manager. Now the user can request a list of resources available of the PlanetLab substrate using the *get_resources* operation and identity the resources required for the experiment. Once the resources are identified by the user using an RSpec, the user can invoke the *create_slice* operation on the PlanetLab Aggregate manager so that a slice is created on the individual nodes and the resources are assigned to the experiment. The user can now log into these nodes and start running the experiment on it. However, the networking backend of the experiment is still not complete and thus any traffic generated by the nodes will be dropped.

In order to allocate the networking resources, the experimenter uses the same SFI tool to create a slice on the E-GENI aggregate manager. First the experimenter invokes a *get_resources* operation on the E-GENI aggregate manager that in turn returns the list of all the networking resources available within the testbed. The user then identifies the switches and the interconnects required for the experiment, and creates and RSpec defining the resources and sends it to the E-GENI aggregate manager in a *create_slice* operation. The E-GENI aggregate manager forwards that request to FlowVisor that in turn allocates the slices on the switches in the network. Now all the individual switches can be controlled using the OpenFlow controllers the user is going to run for the experiment. Once the experimenter starts the controller and the experiment, all the traffic generated from the slice is routed to the controller and then the controller handles the traffic as setup by the user.

Thus the extended SFI interface creates an end-to-end experiment across both the PlanetLab and the E-GENI substrate by controlling the computing resources and the networking resources using the same user interface.

5.5 GMOC

Point of Contact: Jon-Paul Herron (jph@grnoc.iu.edu)

The scope of this project is to facilitate the sharing of operational and experimental information among GENI experimental components. The GENI Meta Operations Center (GMOC) has both technical development and operational requirements and would need a well-defined protocol to help generalize the operational details of GENI prototypes and for the providers of prototypes to send those details to an operational data repository. These requirements suggest a modular approach, with a generalized protocol rather than a restricted set of hardware and software that GENI prototype participants would be required to run. In other words, it would be largely up to the GENI Spiral project investigators to decide what data to share and how to collect this data from their prototype infrastructure as shown in the figure below. The GMOC would provide the standardized format for this data and the systems required to share, monitor, display, and act on this data. In addition, the GMOC could be used to help provide a repository for data collections passing into and out of GENI prototypes for the purpose of discovering and isolating prototypes that have caused problems. This might require additional instrumentation at the connection points and substrate elements between prototypes.

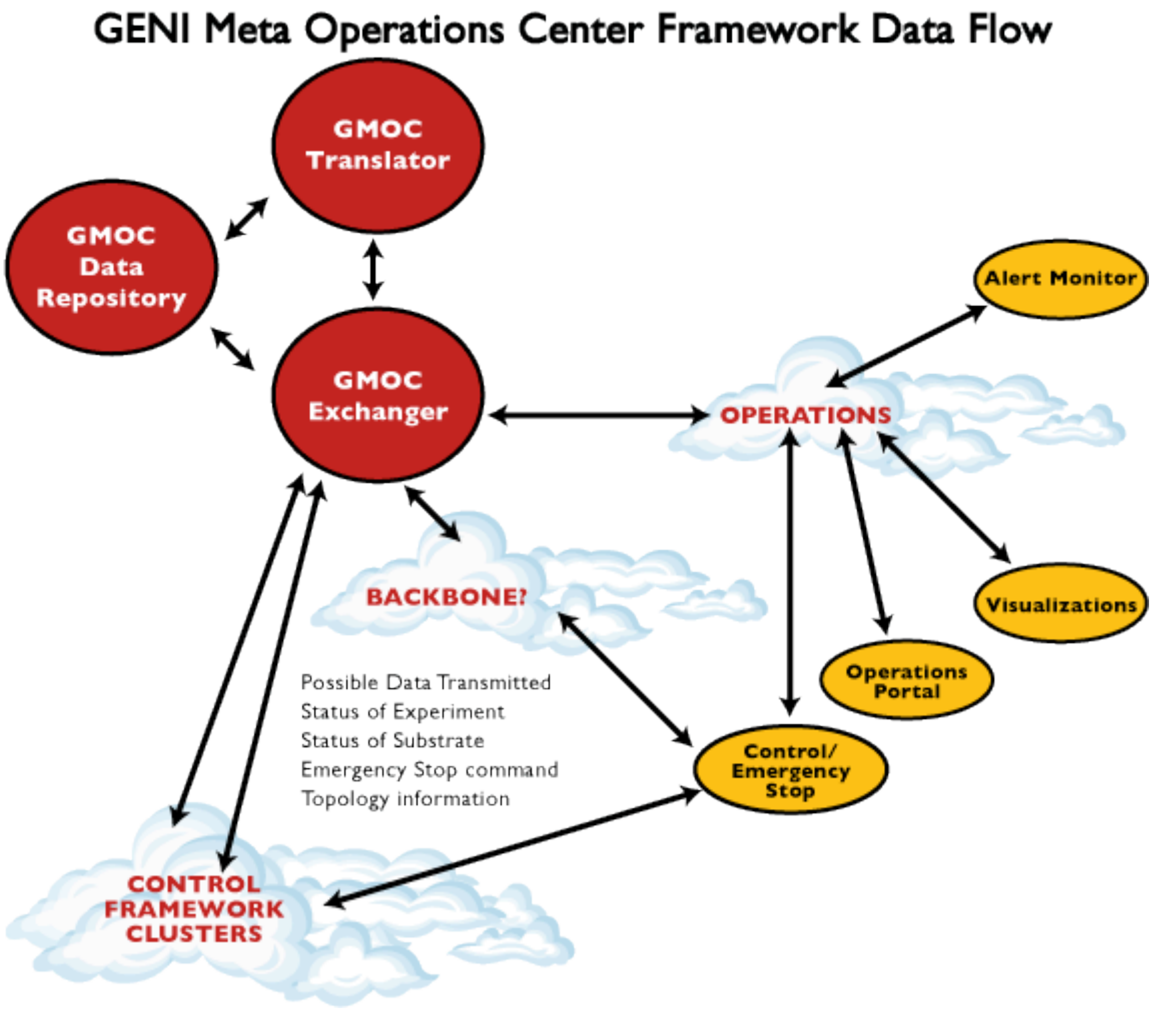


Figure 14: The GMOC Project Data Flow

This would be accomplished with the help of the other prototypes that are part of GENI Spirals. The GMOC will work with these other projects to develop the operational data formats, processes, and systems needed for a consistent and useful suite of GENI infrastructures. During the project, participants will investigate how a Meta Operations Center might interact with various prototype participants to accomplish operations functions.

The GMOC and SECARCH project are also currently discussing the requirements for a policy framework for the security and privacy of measurement data collected by the GMOC project. The GMOC project is focused on gathering operational and experiment data from components, aggregates and their interconnections within GENI to provide information that will aid in management and emergency shutdown functions. We envision during the initial prototyping stages, the security mechanisms for such a data repository will not be as critical, as in most cases it will be generic monitoring data which may not have privacy requirements and could be accessible to everyone in the GENI ecosystem. But as the GMOC starts to monitor and collect data that comes from within experiment slices, we will need to define privacy of data and usage policies. We have already started a discussion on what are the possible implications and requirements to ensure the privacy of such data.

5.6 CMULab

Point of Contact: David Anderson (dga@cs.cmu.edu)

This project builds upon CMU's existing cluster, neighborhood wireless/broad-band, and wireless emulation testbeds to identify and build prototypes of the authentication, resource arbitration, and node management primitives needed to deal with a very diverse set of resources. The CMULab project will integrate its testbeds with the ProtoGENI control framework. The CMUlab project will extend the Emulab network testbed, adding/extending support for a new types of nodes. Traditional Emulab nodes reside in a machine room near (in a network sense) the management systems and the traditional way of handling those nodes is built around this assumption. The project extends that model to allow for nodes that reside, as individual PCs, on non-controlled networks, utilizing the internet to reach the management system.

Since the CMULab uses a range of heterogeneous devices that may not be located in close proximity to the traditional Emulab control and data (boss, ops, and user servers) path model, it is facing new challenges in ensuring it the control and data paths can operate reliably. The devices may reside behind a NAT, and in a typical experiment their notion of control and data planes will often involve the same physical network interface. The nodes will not be able reimage themselves for the same reasons standard emulab nodes will as reimaging is too time and bandwidth expensive, and these nodes must be considerate of the network on which they reside. Also, ensuring the security and privacy of the control and data paths is a new challenge in this project.

The CMUlab project is in the process of developing software to create, allocate keys, start and stop, OpenVPN configurations to completely automate VPN operation for their diverse set of nodes. The goal is to integrate this software into Emulab and into ProtoGENI. They are also attempting to define rspec designs to encapsulate and configure bridging of these VPNs and the SEC-ARCH project is engaging with them at the GENI conferences to understand the security implications of the various design choices.

5.7 GpENI

Point of Contact: James Sterbenz (jpgs@iitc.ku.edu)

The project Great Plains Environment for Network Innovation aims to integrate its state of the art fiber optic network into the PlanetLab cluster. GpGENI has four university connections, University of Kansas, Kansas State University, University of Missouri-Kansas and University of Nebraska-Lincoln. Additionally, it also has three research network connections to the Great Plains Network, KanREN (Kansas

Research and Education Network) and MOREnet (Missouri Reasearch and Education Network) and two industry participants in Ciena and Qwest.

The project aims to develop a system with programmability at all layers that has open access to all in the research and experimentation community. This project is currently connecting the various participants with fiber and routers. The SEC-ARCH project is engaging with them at the GENI conferences to understand their progress and the security implications as their network comes online.

5.8. Instrumentation and Measurement for GENI (GIMS)

Point of Contact: Paul Barford (pb@cs.wisc.edu)
Joel Sommers (jsommers@colgate.edu)

The primary objective of the Instrumentation and Measurement project is to develop, test and deploy a prototype implementation of the *network* instrumentation and measurement systems that will eventually be widely available in the GENI infrastructure. The specific components of the instrumentation and measurement system include sensor nodes that will passively gather packets from links in a network, a measurement data repository and the experimental interfaces that enable users to specify the passive data components that will be gathered for their experiments. Another key component is the security and access control mechanisms that address the important issues of how the instrumentation and measurement systems can be accessed by GENI users and operators, how users are given different levels of access to packet capture capabilities in different environments, how fields in packets are anonymized and how the instrumentation and measurement systems themselves are made secure.

Conceptually the system can be thought of in terms of user activity and control flow that results in data flow as show below. Users specify experiments and their associated measurement configurations through an interface in the control framework (create slice). Among other things, this results in filters deployed on measurement systems and storage allocation. As experiments are run, packets are captured and at the conclusion of the experiment (destroy slice), the filters and storage are deallocated. The interface and aggregate manager facilitate the flow of control data and maintain the current state of the measurement systems.

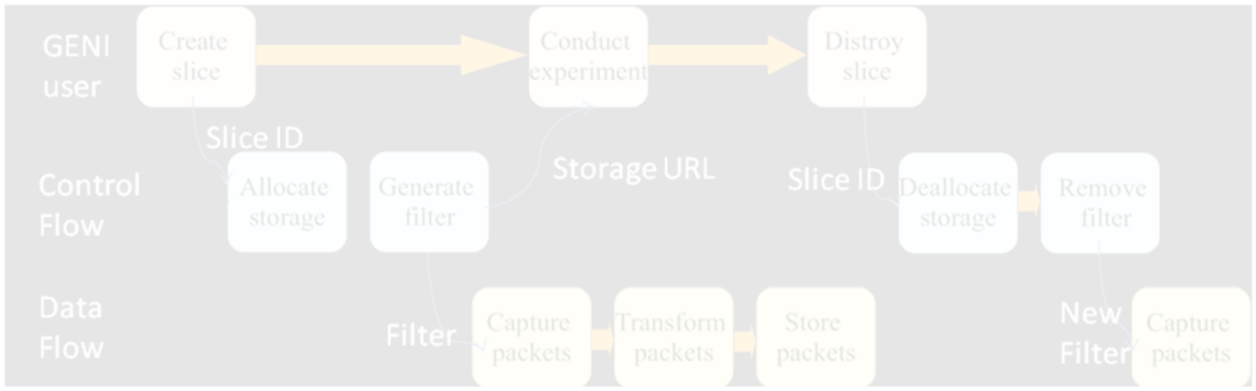


Figure 15: Control and Data Flow within the Instrumentation and Measurement Project

There are significant issues of security and privacy. The sensors, collection/synthesis and archival systems must only be accessible by authorized users, and the authorized users must only be allowed access only a specific proportion of the resources connected with their slice on the measurement systems. The storage repository should also be secure, GIMS has a good plan for anonymization, and the interfaces are still underspecified. Further, all of the measurement systems themselves must be secured against attack and the data that is collected will not compromise or violate the privacy of individuals or organizations that source/sink the data.

There is a constant tension between desired visibility into measurement data and privacy that is faced by multiple projects within GENI and is further discussed in Section 7.

5.9 Digital Object Registry Services

Point of Contact: Giridhar Manepalli (gmanepalli@cnri.reston.va.us)

This project attempts to adapt the Handle System and/or the CNRI Digital Object Registry to create a clearinghouse registry for principals, slices, and components in the ProtoGENI control framework. The project will also analyze ways in which the Handle System and the Digital Object Registry could be used to identify and register GENI software, including experimenter's tools, test images and configurations, and test results. Finally, they will define the operational, scaling, security, and management requirements, plus recommended design approaches, for implementing GENI clearinghouse and software registry services. CNRI has developed an initial technical approach to creating a clearinghouse registry,

The Handle System provides a unified, distributed, and secure identification system that can be directly used to identify all discreet resources and entities within GENI, for example, principals, slivers, aggregates, and components. One of the key features of the handle system, of particular interest in this context, is the ability to associate any data, or references to data, with each handle identifier, either directly within the handle record or through a pointer to external data. This allows the rapid and reliable resolution of any of a large set of identifiers to the current state data for the identified entity. Another critical functional aspect of the Handle System, beyond its ability to provide distributed storage and retrieval, is its secure administration mechanisms, which guarantee that only authorized entities can appropriately create, modify or delete handle records. This is of special importance within the GENI environment, as much of the information will have to be locally managed but globally accessible and trusted. For relatively small data items, such as the latest version number, the Handle System could be used directly to serve part of the GENI clearinghouse function. In other cases it will provide one or more pointers to external data, in registries or not.

The CNRI Digital Object Repository provides an object storage and retrieval protocol that can be used to provide a common overlay across multiple back-end storage systems.

It can be used to provide storage capacity beyond that of the handle system while providing the same level of distributed access and security. This technology could provide a standard distributed storage mechanism for experimental data sets, code modules, documentations and any other resources. The digital object generic registry system could be used for registering, indexing and providing search capability over any XML data. More specifically, this system could be to use it to validate, register, index and provide search capability over Rspecs, principals, and code within the GENI framework. A more dynamic application of this registry system could be extended to registering slices with the intent to record or share slice information with other researchers. Multiple instances of the digital object registry can be federated to provide a single search and access point across multiple registries.

For the clearinghouse, the project will utilize the registry technology and the handle system technology to provide a principal, component and slice registry. The principal registry will be the least dynamic of the three but will require the highest level of security to make sure that each principal's identification, authentication and authorization are kept up to date within the GENI framework. This registry will primarily leverage the handle system but could also use the registry for user discovery. The handle system would be used to identify, consistently and authoritatively define policies and permissions within the entire GENI framework.

5.10 GENI LEFA

Point of Contact: Kenneth J. Klingenstein (kjk@internet2.edu)
Steven Carmody, (Steven_Carmody@brown.edu)

The LEFA project is exploring the role of existing trust management infrastructure and systems in GENI, and in particular the Shibboleth In-Commons service deployed widely throughout academic institutions. Shibboleth supports the abstractions of identity provider (IdP) and service provider (SP). End-users (GENI researchers in our context) authenticate to the IdP's; typically there is an IdP at each campus that would already have a well-known and established authenticated identity for each member of the University community, including faculty, post-docs, graduate and undergraduate students. IdP's in turn pass attributes, such as the role as faculty or student, as well as specific course enrollments or research project participation; to the service provider which is enforcing an authorization policy. SPs may guard access to control framework resources such as those provided by aggregate managers to acquire slices on physical nodes or wide-area network infrastructures. SPs would base their authorization policy decisions on the *attributes* supplied by IdPs, rather than on specific identities of each individual.

Another important point of the LEFA approach is that management and administration of identities, keying material and public key certificates is essentially hidden behind the Shibboleth mechanisms. For end-users and GENI operators, this is potentially a very significant advantage, as this management and administrative burden is amortized over all the uses of Shibboleth identities on each campus, rather then being borne by the GENI infrastructure, and primarily the control frameworks or clearinghouses.

6. Security Guidelines and Policies

This section is a placeholder, to be expanded and presented at GEC 8, 9 and 10.

Specifically, best practices for projects contributing resources to GENI, including wired and wireless aggregates and components, their managers, as well as control frameworks and clearinghouses, will be defined based on *input* solicited from across the GENI community. In addition, best practices will be based on study, observations, and insights gained from researchers and campus organizations operating open network testbeds.

It is clear that some tensions exist between traditional IT operating practices and the overarching goals of the networking research community. From an operational mindset, networking research poses risks of misuse of available shared networking resources, and potential for spillover of traffic or collateral impact on non-research networking infrastructure and services. Security guidelines and policies in GENI therefore must aim to "level-up" all participating elements of the GENI research infrastructure. By doing so, participants, and especially campuses, can integrate and deploy GENI infrastructure within their networking infrastructure with the confidence that doing so will support researchers and CIOs in using, managing and securing GENI.

7. Security Mechanisms

Over the course of the development and prototyping spirals, we anticipate documenting a substantial set of security mechanisms that play a role in securing the GENI system. We define a core set of mechanisms, which well-designed secure systems will need to incorporate, either through implementation or integration within their solutions. The five mechanisms listed below are identity, authentication, authorization, access control, and audit.

7.2 Identity

Identity is defined as who someone or what something is, for example, the name by which something is known. Traditionally, identity requires identifiers—strings or tokens that are unique within a given domain, (that is globally or locally within a specific network, directory, application). Identifiers are the key used by the parties to an identification relationship to agree on the entity being represented. Identifiers may be classified as resolvable or non-resolvable. Resolvable identifiers, such as a domain name or e-mail address, may be referenced into the entity they represent, or some current state data providing relevant attributes of that entity. Non-resolvable identifiers, such as a person's real-world name, or a subject or topic name, can be compared for equivalence but are not otherwise machine-understandable.

In a federated environment such as GENI, an identity could be a union of a principal's, information stored across multiple distinct identity management systems. The databases could be joined together by the use of a common token. A principal's authentication process will thus occur across multiple networks or even across several organizations.

The GENI Management core [GENI-SE-SY-SO-02.0] defines unambiguous identifiers—called *GENI Global Identifiers* (GGID)—for the set of objects that make up GENI. GGIDs form the basis for a correct and secure system, such that an entity that possesses a GGID is able to confirm that the GGID was issued in accordance with the GMC (GENI Management Core) and has not been forged, and to authenticate that the object associated with the GGID is the one to which the GGID was actually issued.

Specifically, a GGID is represented as an X.509 certificate that binds a Universally Unique Identifier (UUID) to a public key. The object identified by the GGID holds the private key, thereby forming the basis for authentication as discussed in the next section. Each GGID (X.509 certificate) is signed by the authority that created and controls the corresponding object; this authority must be identified by its own GGID. There may be one or many authorities that each implement the GMC, where every GGID is issued by an authority with the power and rights to sign GGIDs. Any entity may verify GGIDs via cryptographic keys that lead back, possibly in a chain, to a well-known root or roots. Every entity within GENI will have a GGID for accountability and these identities will map to real world identities such as email and physical location address. A principal may have multiple identities.

[n.b. The definition of GGIDs, GIDs, or even public-key based identities is subject to change or evolution within the GENI control framework.]

7.3 Authentication and Authorization

Authentication verifies the identity of an entity in GENI. It is a key aspect of trust-based identity attribution, providing a codified assurance of the identity of one entity to another. Traditionally, authentication and identification mechanisms rely on maintaining a centralized database of identities, making it difficult to authenticate users in different administrative domains across federated networks. Each federated network keeps track of its users in a users account database and hence granting access to resources across networks is challenging. Each control framework may have its own mechanism of authentication at the early spiral prototypes.

Authentication methodologies include public-private (asymmetric) key pairs, the provision of confidential information such as a password, or utilizing encryption methodologies. The use of a Public Key Infrastructure (PKI) will allow establishing strong identities for facility users. Although PKIs are hard to bootstrap, GENI has a natural advantage since every site will have a local administrator who can establish and vouch for the credentials for each specific GENI research user and physical device. Authentication is required for both the network (local site) facility itself, to grant access to applications and services and provide a basis for resource isolation, but also for applications and users. A flexible and accessible public-key or other authentication service, along with the software libraries and resources to manage it, will facilitate the operation of GENI and the development of a large range of applications on top of it. This service must include the development of libraries to allow a variety of applications to use the service and the development of guidelines for how and when applications should use the service.

Even though GENI will allow an entity to have multiple identities, authentication is still required in order to verify that the identity presented for a particular GENI operation is a valid registered identity. The authentication in this case is of the GGID itself, and not of the entity represented by it.

As mentioned in section 6.1, a GGID binds a Universally Unique Identifier (UUID) to a public key. The object identified by the GGID holds the private key, thereby forming the basis for authentication. Each GGID is signed by the authority that created and controls the corresponding object; this authority must be identified by its own GGID. A name repository maps strings to GGIDs, as well as to other domain-specific information about the corresponding object. There may be multiple name repositories. Depending on the entity, the domain-specific information can be any of the following: (a) the URI at which the object's manager can be reached, (b) an IP address, (c) a hardware address for the machine on which the object is implemented, (d) the name and postal address of the organization that hosts the object.

[n.b. Authentication and Authorization systems based upon previously deployed identity and trust management services, such as Shibboleth In-Commons, may also play a role, and perhaps a preferred role, in GENI.]

Authorization is the process of allowing access to resources only to those permitted to use them. In GENI the resources include data, slices, component devices, network bandwidth, and functionality provided by services. The problem of authorization is often thought to be identical to that of authentication; however, more precise usage describes authentication as the process of verifying a claim made by a entity that it should be treated as acting on behalf of a given principle (person), whereas authorization is the process of verifying that an authenticated subject has the authority to perform a certain operation. Authentication, therefore, must precede authorization and many times the term authorization is used to mean the combination of authentication and authorization.

Authorization is traditionally implemented as permissions, such as an *access control list* or a *capability*. Authorization determines the access control rights of an entity, that is, is user X allowed to access resource R? The traditional way of performing authorization is to lookup a user's rights in an access control matrix, which has rows that represent users and columns that represent resources, The value in the matrix represents the read/write/execute or other access permission set. The columns in an access control matrix represent the access control lists (ACLs) and the rows represent capabilities. An ACL is associated with every resource in the system, and lists all entities that are authorized to access the object along with their access rights. The identity of an entity must be known before access rights can be looked up in the ACL. Thus, authorization depends on prior authentication and systems that rely on ACLs for authorization must use a decentralized authentication mechanism to work across administrative boundaries. Capabilities correspond to rows of the access control matrix and thus a capability is an unforgeable token that identifies (names) one or more resources and the access rights granted to the holder of that capability. Any user that possesses a capability can access the resources listed in the capability with the specified rights. In contrast to ACLs, capabilities do not require explicit authentication. However, it is typically the case that an initial set of capabilities is distributed only to an entity after authentication to some trusted service that mints these capabilities.

Capabilities can be transferred among entities, which make them suitable for authorization across organizational boundaries. Because capabilities explicitly list privileges over a resource granted to the holder, they naturally support the property of least privilege. However, because possession of a capability conveys access rights, capabilities must be carefully protected against theft (e.g. unauthorized transfer). In addition, capabilities may make it more difficult to perform auditing or forensic analysis. Especially for large-scale decentralized systems such as GENI where the logs themselves or the meaning of the information contained in the capabilities may be spread across several networks, collecting all the necessary information may involve considerable effort.

Permissions are traditionally based on the principle of least privilege discussed above where an entity is granted specific permissions that they need to do their jobs and no more. Exceptions to this principal may allow some “trusted” principals that are granted unrestricted access to resources, such as for monitoring usage on the network. Anonymous or guest entities that are not required to authenticate themselves are given very few permissions, although even a limited degree of access may be problematic. Pseudo-anonymity of various types may be used instead of truly anonymous access, although we defer this point to future prototyping spirals.

The main function of the GENI control frameworks is to allow the authorization and assignment of resources from multiple GENI or federated aggregates to GENI researchers following pre-established policies. This will involve the interaction of a variety of elements, such as, the researcher, the designated slice, the aggregate, (including its resource availability and local policies), policies associated with other entities, (such as the GENI clearinghouse or an intermediate broker), policies based on other parameters, such as researcher/slice lineage and status, and lastly, resource availability.

In all cases, a decision to grant a resource is made as a request from a researcher to an aggregate. In a simple case, supported by the current control framework architecture, an aggregate can check the slice lineage of a request against a local list of supported slices. However, ideally the control framework architecture should support richness in resource allocation and policy mechanisms. In particular, there should be a way to include policies that are associated with a clearinghouse or an intermediate broker.

The GENI control framework makes use of exchange of tokens (called credentials or tickets) to authorize principals within GENI. These tokens are then used to permit access to registries and authority services and are also used to authorize resource assignment and management. Further, tokens must be signed (certified) by the appropriate authorities and objects (principals, aggregates and slices) to associate value to them in the GENI network. This approach to authorization is very flexible, allowing entities to be widely dispersed and even disconnected for a short period within the GENI network.

Various resource allocation and policy mechanisms will be explored in Spiral 2 implementations and are discussed in the subsequent sections. The above authorization approach is widely used within the ORCA control framework.

7.4 Access Control

The core of our proposed security architecture for GENI is a pervasive and unified access control infrastructure. *Access control* refers to the mechanism used to reach a yes-no decision as to whether an access request should be granted. The decision is typically reached by a resource monitor based on security policy defined for the resource. The goal of the ABAC architecture we propose in Section 8 is to provide a unified and yet flexible mechanism for resource monitors to reach such decisions. Access control is often intimately tied to authentication and authorization as discussed above, however, we propose separating the entities authentication mechanisms from access control especially for components. We propose using access control methods that are not based on the public—private key pairs to provide additional flexibility that may be useful for certain classes of components that may not have the resources to support PKI.

All access rights for slices originate with a Slice Authority (SA) and it is responsible for approving the research users associated with the slice. All rights regarding component resources originate at the Management Authorities (MA). The MAs define the resource allocation policies for the components they manage and approve all research users that operate those components. Each component implements a resource allocation policy that determines how many resources, if any, to grant each slice. A researcher that is granted the instantiate capability for a given slice can be viewed as having the right to ask for resources from the component—the credential essentially confirms that some slice authority vouches for the slice—but it is up to the component to decide if it is willing to host the slice, and if so, how many resources to grant it.

7.5 Audit

Auditing actions post-mortem is often required by forensic investigators or other parties to ascertain precisely what happened *after* a security related incident. However, audit mechanisms depend on the logging of sufficient information to tie together the identity (and authentication mechanism or series of mechanisms) used to acquire the login or rights sufficient to wield that identities privileges; the recording of which operations were performed by which clearinghouses, slice or management authorities, aggregate and component managers, and individual components – including where feasible details of what parameters were supplied, and the results of each operation invocation.

Additionally, integrity of audit logs is necessary, both to prevent tampering with audit logs and to ensure that logs are managed in a way that safeguards them and retains them for sufficiently long durations so that they can be consulted after a security incidents, e.g. to determine the earliest point in time at which a break-in or other problem first occurred.