GENI

Global Environment for Network Innovations

GENI Security Architecture Spiral 1 Draft 0.5

Document ID: GENI-SEC-ARCH-0.5

June 30th, 2009

Prepared by:
Alefiya Hussain and Stephen Schwab
SPARTA, Inc.

Table of Contents

1. Document Scope

This document is entitled GENI Security Architecture. It is a draft, intended to be a living document more in the spirit of an IETF Internet Draft rather than a Request For Comments (RFC) document that defines a protocol or standard. Over the course of the first three GENI development and prototyping spirals, this document will track and define, and at times lead, the state of security architecture, design, implementation and issues on the collective mind of the GENI community. As a reader, your comments, criticisms and suggestions are welcome and essential to progress.

GENI is an evolving system, whose state is captured most recently within the GENI System Overview, GENI-SE-SY-SO-02.0. The GENI Control Framework Requirements and Slice Based Facility Architecture documents describe the current abstractions and architecture underlying the lowest layer of the system. While the scope of the GENI Security Architecture is intended to be broad, our attention is focused on this lowest layer for the moment, because it is essential to understand the tradeoffs and concerns facing the prototyping efforts as they ready their spiral 1 systems for initial use. It is also essential to distill the core security ideas underlying these core control frameworks to a minimum, and then to restate these ideas in a neutral way, not tied to the particular implementation choices which are intended to evolve over the lifetime of the systems. This is a challenge, as we wish to speak concretely about aspects of real implementations for clarity, but then extrapolate to draw lessons that apply to the architecture, and hence many possible future implementations.

Before continuing, it is worth considering what the term security architecture means, and to acknowledge that there is not necessarily a universal definition of this term. A number of *security services*, such as confidentiality, integrity, availability, etc. may be considered as the realm of a security architecture, without delving into the myriad details as to how these are to be accomplished through various *security mechanisms*. Encryption and cryptographic libraries, access control lists and enforcement functions, policy languages, firewalls, and various operating system protection mechanisms may then be called upon as mechanisms to realize the properties called out in the set of security services. Alternately, security architecture can more broadly include the investigation of tradeoffs between available security mechanisms, and consider such issues as the assurance arguments that may be made in support of an overall system design using a specific set of mechanisms.

As we consider the system over a wider range of its lifecycle, security architecture might be widened as well to include the roles of individuals who will interact with the system; the development tools and methodology used to construct the system, including pedigree of software libraries, language tools and operating systems; the assurance process used to validate that claims about security at design time are reflected in the final software versions; and the on-going operational issues surrounding how security is addressed during routine nominal processing and how security incidents are handled and resolved.

All of these issues are in-scope, but ultimately GENI's Security Architecture must focus on providing answers that are consistent with the central questions facing the community of researchers who will use it: *What interfaces am I allowed to access? What operations may I perform? What resources are available, and how are allocations of those resources parceled out?* And lastly, two dual questions: *What data can I collect or access? What access controls are placed on data I collect and share?*

1.1 Purpose of this Document

This document defines the GENI threat and trust models, and secure operations guidelines and mechanisms in support of the overall security requirements. It also outlines the current approach of each control framework involved in development and prototyping, and highlights unique security challenges in each of these five control frameworks. This document furthermore introduces short-term action items relevant to the current spiral's deployment, as well as posing next step candidate mechanisms in support of an evolving security architecture for GENI. It may be used as a guide in the development of the control framework prototypes as they evolve in the current and subsequent spiral cycles, in the sense that informed thinking on the part of designers leads to approaches that address security concerns earlier and more comprehensively. Within the overall GENI effort, this document falls under the OMIS working group but is significantly linked to the Control Framework working group as well.

1.2 Related Documents

Some of 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-	ORCA GENI Control Framework Overview

ORGO-01.2	http://groups.geni.net/geni/attachment/wiki/OrcaGeniControlFrameworkOverview/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.

2. Security 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.

Chip Elliot in his GEC4 talk in Miami envisioned a 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.

This draft discusses security requirements and issues in the GENI network with respect to authentication and authorization in a distributed network. We discuss the leading risks in such an environment and propose a solution to address those issues based on the current control frameworks. We go on to describe a security architecture derived from our experience and how mechanisms supporting such a security architecture may be integrated in support of the larger GENI architecture.

3. GENI 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.

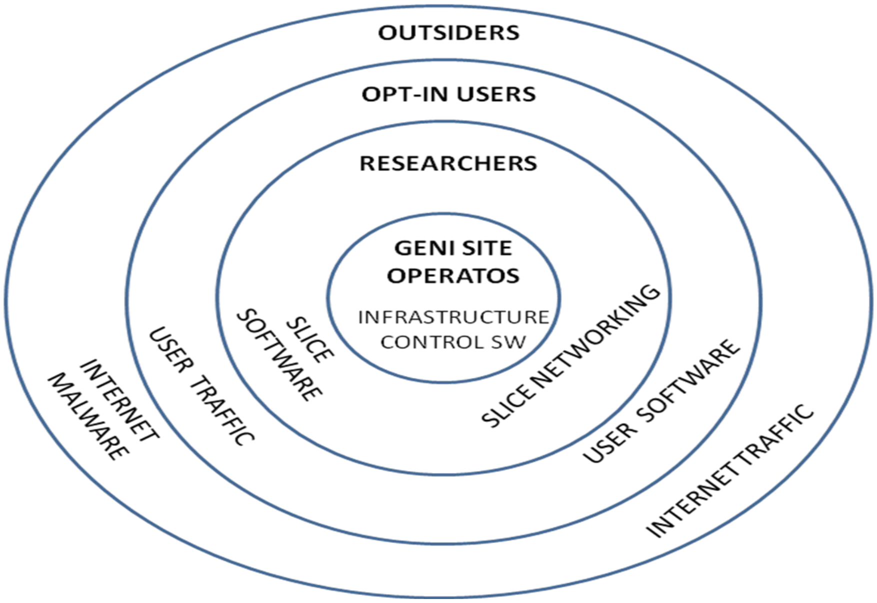


Figure 1. The illustration presents rings of threats. At the center is the infrastructure with the greatest privilege. Working outwards are rings including GENI researchers, opt-in users making use of GENI experimental slices, and finally outsiders.

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.

4. GENI 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 architect 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 concentrated on authorization enforcement to protect GENI networks and the authentication to support authorization enforcement. These ideas are further explored in the subsequent sections.

5. Security Guidelines and Policies

In this section we first summarize the high level guidelines that should be embodied in all GENI software, rules, and policies. They are drawn from GDD 06-10 and GDD 06-23 and discussions at the GENI Engineering Conferences. We then go on to discuss the policy requirements of the GENI ecosystem specifically, how the security architecture and the GMOC teams need to coordinate their efforts to incorporate a practical distributed policy system within GENI.

Security considerations place several guidelines on the GENI architecture:

- **Explicit Trust**: Privileges in a distributed system should be managed explicitly and formally. Enforcing security in GENI will be something of a moving target, as the facility will be used during its construction, and will progress from a single, centralized management entity to a federated, decentralized model. Thus we need a security model that can evolve along with GENI. We need to define access control approaches that provide the required flexibility, rather than hard-coding trust relationships. Without explicit trust, it is likely that trust will be unintentionally misplaced, leading to system-wide vulnerabilities that can be exploited by a determined attacker.
- **Least Privilege**: The principle of least privilege is a tenet of computer security that requires each component of a system be given exactly the authority it needs to perform its tasks and no more. Failures to implement this principle are ubiquitous, and we face the consequences frequently. For example, most web servers do not need to be able to open connections to arbitrary addresses in order to perform their tasks. Yet this is permitted, and exactly this ability has been used numerous times in the epidemic spread of worms. While achieving least privilege in an absolute sense is arguably not feasible, it is our belief that the GENI facility should embrace least privilege as far as is practicable. Least privilege can secure the GENI facility from malicious software, accidental violations, or just simple resource exhaustions—in general, it can mitigate the risks caused by runaway experiments. It is also equally useful in securing the experimenter's environment against attacks from other experiments or faulty system software.
- **Revocation**: Despite our best efforts, it is inevitable that keys, slices, and systems will be compromised in GENI. Thus a critical requirement for GENI is to be able to quickly revoke and replace keys, suspend all permissions (e.g., slices) derived from a compromised key, and reset each node to a known secure state.
- **Auditability**: The possibility of compromise also requires us to be able to trace why a problem occurred so that it can be prevented from recurring. GENI needs to develop mechanisms that identify which slice is responsible for each packet, and also be able to determine the entire chain of responsibility (from central administrator to local

administrator to local user) that gave the user a specific capability that was misused.

- **Scalability**: With large-scale distributed systems such as GENI, simple schemes such as using a small set of authentication servers and/or replicating information required by authentication and authorization tasks are not feasible. We propose a scalable authorization architecture below.
- **Autonomy**: A key requirement for GENI is the ability to federate autonomous facilities. A GENI site should be able to authenticate and authorize requests from users in other sites, support delegation of rights, and it should be able to do so without requiring centralized trust.
- **Usability**: The user must be explicitly modeled as part of the security architecture. Any system that is hard to use will be evaded and ignored. The implication is that GENI needs to develop intuitive and easy interfaces for users to create roles, restrict rights, etc. GENI also needs to make it easy for users to protect their private keys. In essence, secure system and user behavior must happen naturally, in the course of operating or using the system.
- **Performance**: As with usability, the performance overhead of providing security needs to be modest, or users will have an incentive to disable or evade the system. In practice, this means managing security information (such as certificates delegating rights to a specific set of users) locally as far as possible, as cache-coherent, distributed state. Caching means that lookups can be fast in the common case, without compromising system semantics.

In addition, it is very important that the requirements for a policy framework for the security and privacy of measurement data collected by the GMOC project be carefully formulated. 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. 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 and an attribute-based prototype will provide the mechanisms to enforce them. We have already started a discussion on what are the possible implications and requirements to ensure privacy of such data.

6. Security Mechanisms

Over the course of the development and prototyping spirals, we anticipate enumerating a substantial set of security mechanisms that play a role in securing the GENI system. We start with a small set of mechanisms, which well designed secure systems will need to incorporate into their solutions, The four mechanisms listed below include identity, authentication, authorization, and access control.

6.1 Identity

Identity is defined as whom 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 and has not been forged, and to authenticate that the object claiming to correspond to 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. 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.

6.2 Authentication

Authentication verifies the identity of a principal 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 it's users in a users account database and hence granting access to resources across networks is challenging.

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.

6.3 Authorization

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 principal (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 represents 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. Psuedo-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 1 implementations and are discussed in the subsequent sections. The above authorization approach is widely used within the ORCA control framework.

6.4 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.

Table 1 below summarizes the four security mechanisms and possible implementation strategies in GENI.

Terms	Definition [SANS]	GENI Mechanism
Identity	It is who someone or what something is.	GGID
Authentication	It is the process of confirming the correctness of the claimed identity.	GGID along with the key and name authority for mapping the keys
Authorization	It is the approval, permission, or empowerment for someone or something to do something.	Certified tokens, and Credential tickets and Capabilities or attributes
Access Control	It ensures that resources are only granted to those users who are entitled to them.	Slice Authorities control access to the experiment, Aggregate Authorities control access to components.

Table 1. Candidate Security Mechanisms

7. Securing 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 federated together. 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.

7.1 Definition

The GENI control framework is defined in the GENI Control Framework Requirements document at http://geni.bbn.com:8080/docushare/dsweb/Services/Document-1234 .
A control framework has a clearinghouse consisting of principal, component and slice registries, along with the offered services. Principals typically will use tools and act as clients of the control framework. The services offered by the control framework will in most cases be associated with aggregates within the architecture. Each control framework in GENI will unique define interfaces between all entities, planes for transporting messages between all entities, message types, including basic protocols and required functions, message flows necessary to realize key experiment scenarios.

Additionally, there should be mechanisms to federate with other control frameworks in the GENI architecture. The GENI control framework requirements are presented in the GENI Control Framework Requirements document produced by the Control Framework WG at http://geni.bbn.com:8080/docushare/dsweb/Services/Document-1234 .

7.2 ProtoGENI

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 2. 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 entity in GENI. The DN also includes email address.

Authentication of the entity is done 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.

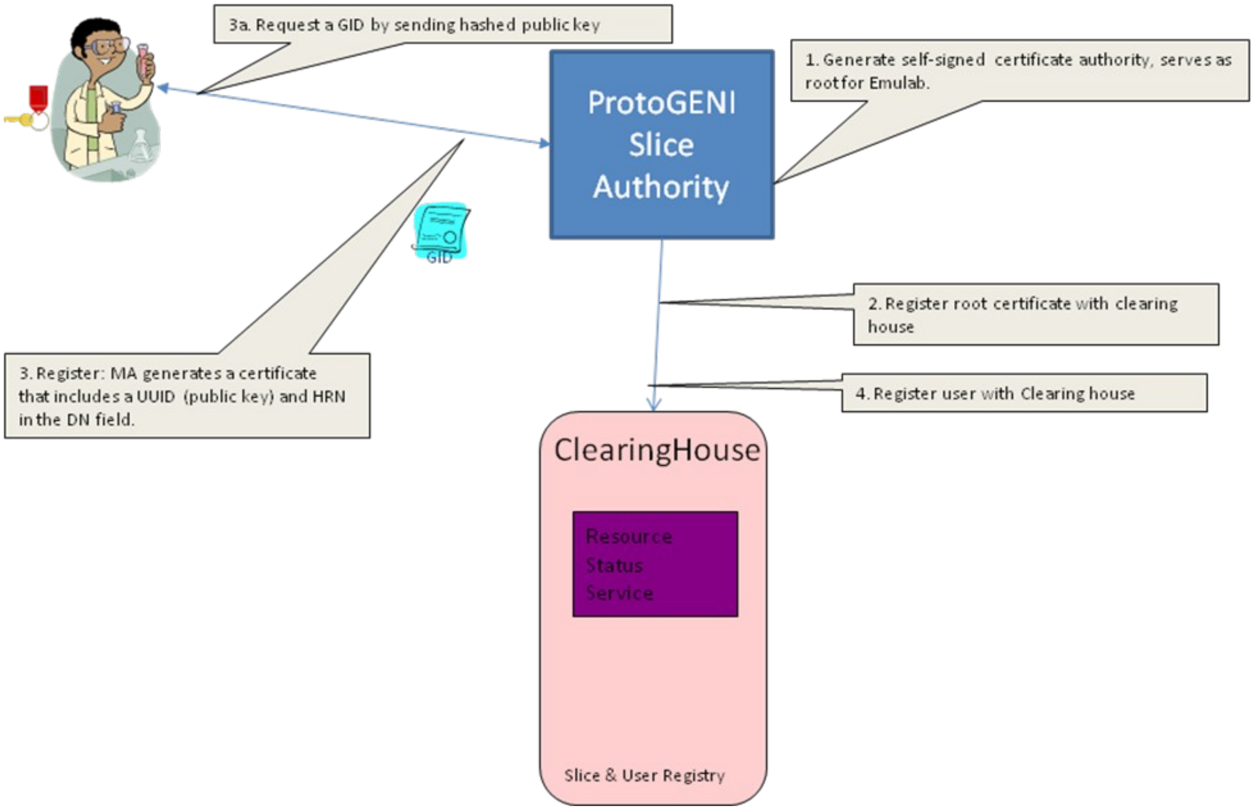
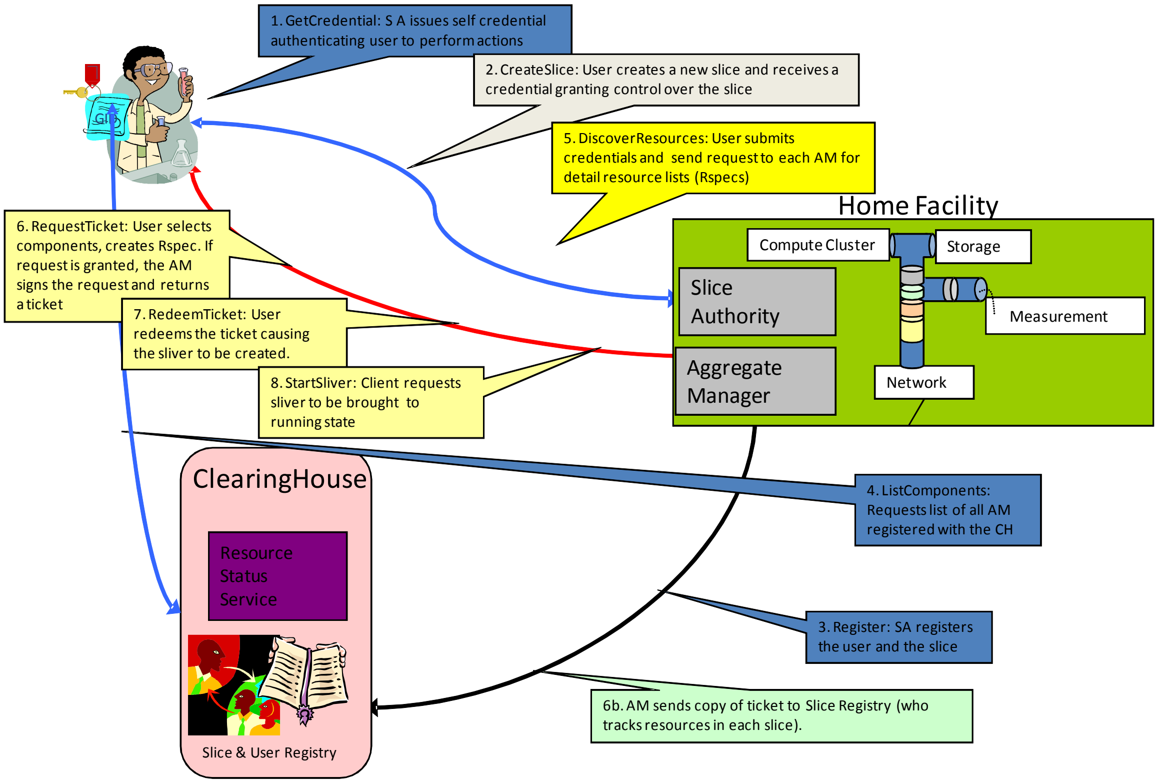

3a. Request a GID by sending hashed public key
ProtoGENI Slice Authority
1. Generate self-signed certificate authority, serves as root for Emulab.
GID
2. Register root certificate with clearing house
4. Register user with Clearing house
3. Register: MA generates a certificate that includes a UUID (public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

Figure 2: 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 objects 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. Figure 3 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)

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

4. ListComponents: Requests list of all AM registered with the CH

3. Register: SA registers the user and the slice

6b. AM sends copy of ticket to Slice Registry (who tracks resources in each slice).

Figure 3: Authorization during Slice Creation in ProtoGENI

The ProtoGENI suite thus uses certificates and credentials to authenticate entities. This approach however combines identity and authentication mechanisms. At this stage, the role of UUID is also not completely defined within the prototype.

7.3 PlanetLab

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 4. 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.

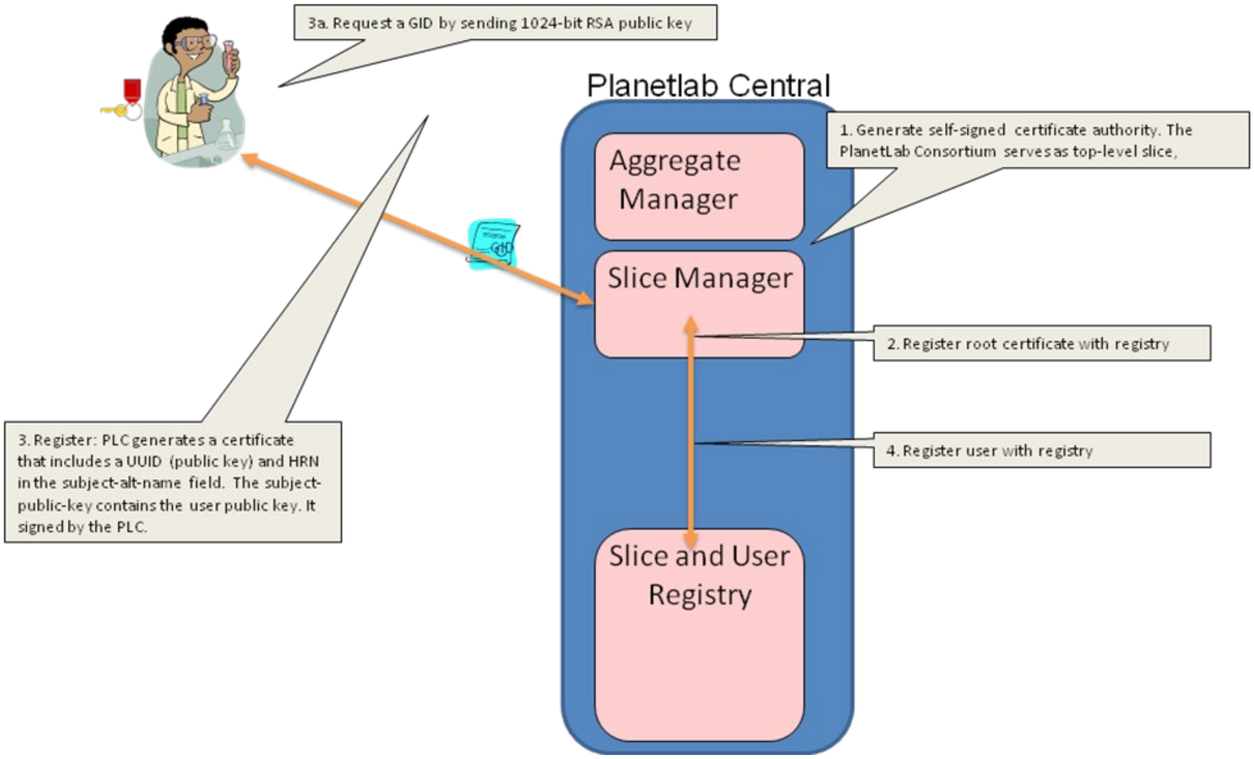
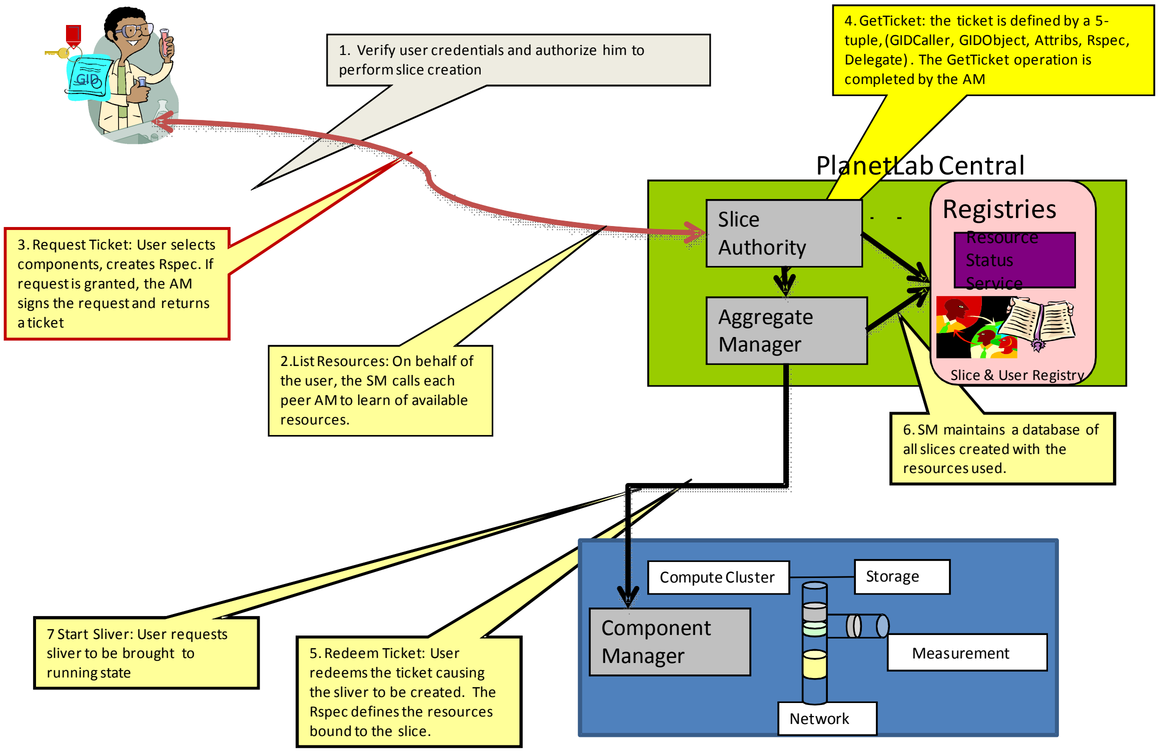

3a. Request a GID by sending 1024-bit RSA public key
Planetlab Central
1. Generate self-signed certificate authority. The PlanetLab Consortium serves as top-level slice,
Aggregate Manager
Slice Manager
2. Register root certificate with registry
3. Register: PLC generates a certificate that includes a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

Figure 4: 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 ProtoGENI system is initiated by the exchange of credentials that facilitate resource authorization and access control by aggregates. Figure 5 shows the slice creation process in PlanetLab.



1. Verify user credentials and authorize him to perform slice creation

2.List Resources: On behalf of the user, the SM calls each peer AM to learn of available resources.

3.Request Ticket: User selects components, creates Rspec. If request is granted, the AM signs the request and returns a ticket

4.GetTicket: the ticket is defined by a 5-tuple, (GIDCaller, GIDObject, Attribs, Rspec, Delegate). The GetTicket operation is completed by the AM

5.Redeem Ticket: User redeems the ticket causing the sliver to be created. The Rspec defines the resources bound to the slice.

6.SM maintains a database of all slices created with the resources used.

7 Start Sliver: User requests sliver to be brought to running state

Figure 5: 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.

7.4 TIED

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.

All users, projects and testbeds have 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.

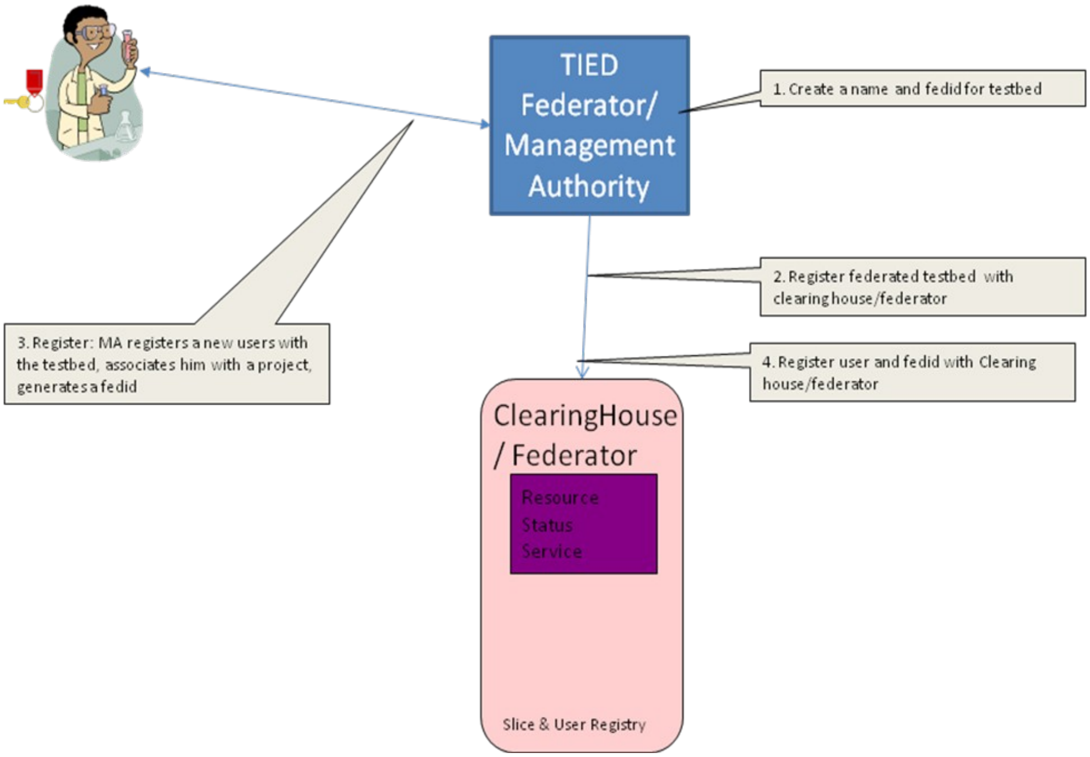

TIED Federator/ Management Authority
1. Create a name and fedid for testbed
2. Register federated testbed with clearinghouse/federator
3. Register: MA registers a new users with the testbed, associates him with a project, generates a fedid
4. Register user and fedid with Clearing house/federator
ClearingHouse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

Figure 6: 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.

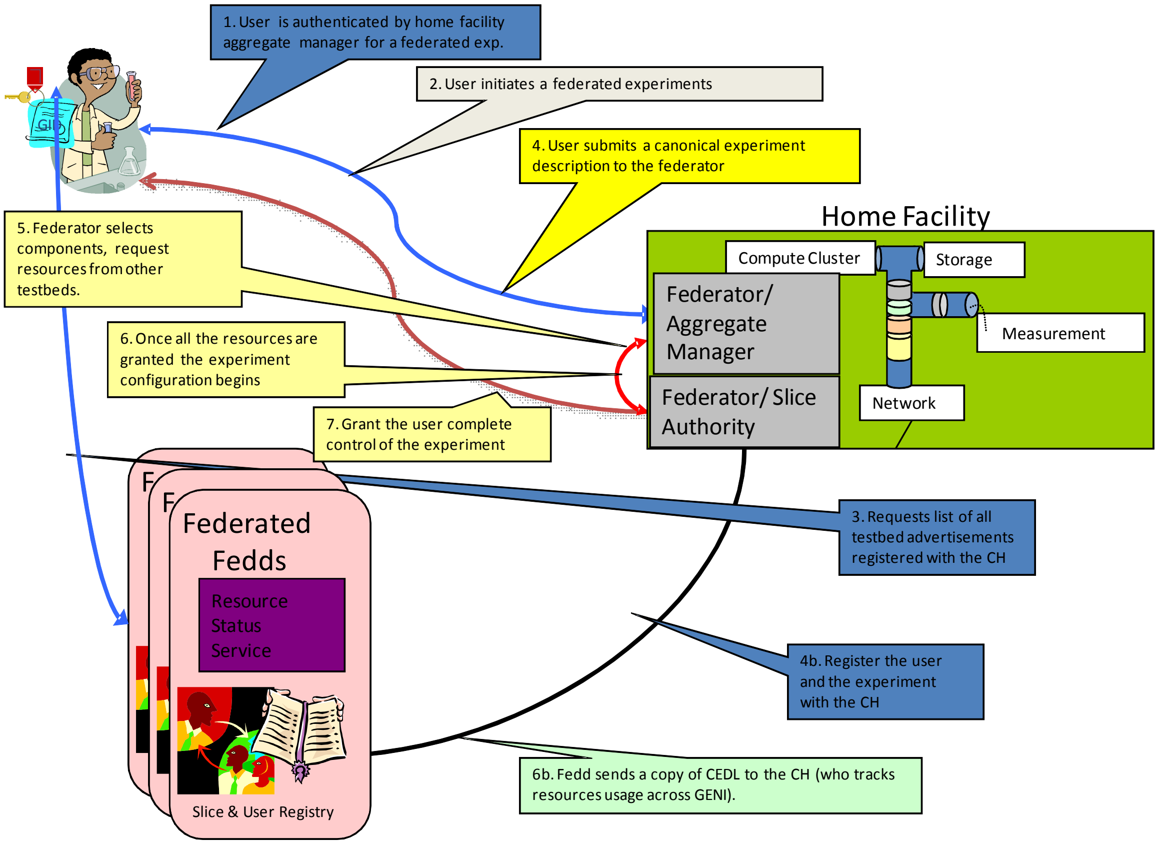

1. User is authenticated by home facility aggregate manager for a federated exp.
2. User initiates a federated experiments
4. User submits a canonical experiment description to the federator
Home Facility
Compute Cluster
Storage
Federator/ Aggregate Manager
Measurement
Federator/ Slice Authority
Network
GID
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
3. Requests list of all testbed advertisements registered with the CH
Federated Fedds
Resource Status Service
Slice & User Registry
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 7: 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.

7.5 ORCA

Open Resource Control Architecture (ORCA) is an extensible architecture for on-demand networked computing infrastructure. 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.

The ORCA GENI control framework consists of four main Shirako-based control servers or actors: brokers, domain authorities, service managers, and identity providers. The broker provides most of the clearinghouse functions. The ticket broker service issues all tickets to experiments. The domain authorities provides the functionality of an aggregate manager and delegate splittable tickets to the broker service, and attempt to honor any tickets issued by the broker. The service managers facilitate principals to setup and manage experiments whereas identity providers vouch for principals. The broker service also maintains a principal registry containing the public keys of identity providers and registered users. The service accepts any user with a registered public key, or bearing an X.509 certificate endorsed by a registered identity provider. The actors (automated control servers) run as SOAP servers exchanging digitally signed messages. Each actor also has a web control portal interface for the user or operator to interact with the system. Every actor maintains a similar basic set of data structures to store local registry information. Currently the registry is implemented as a MySQL database.

Figure 8 summarizes the identity and authentication processes within ORCA. The GENI ORCA control framework includes 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.

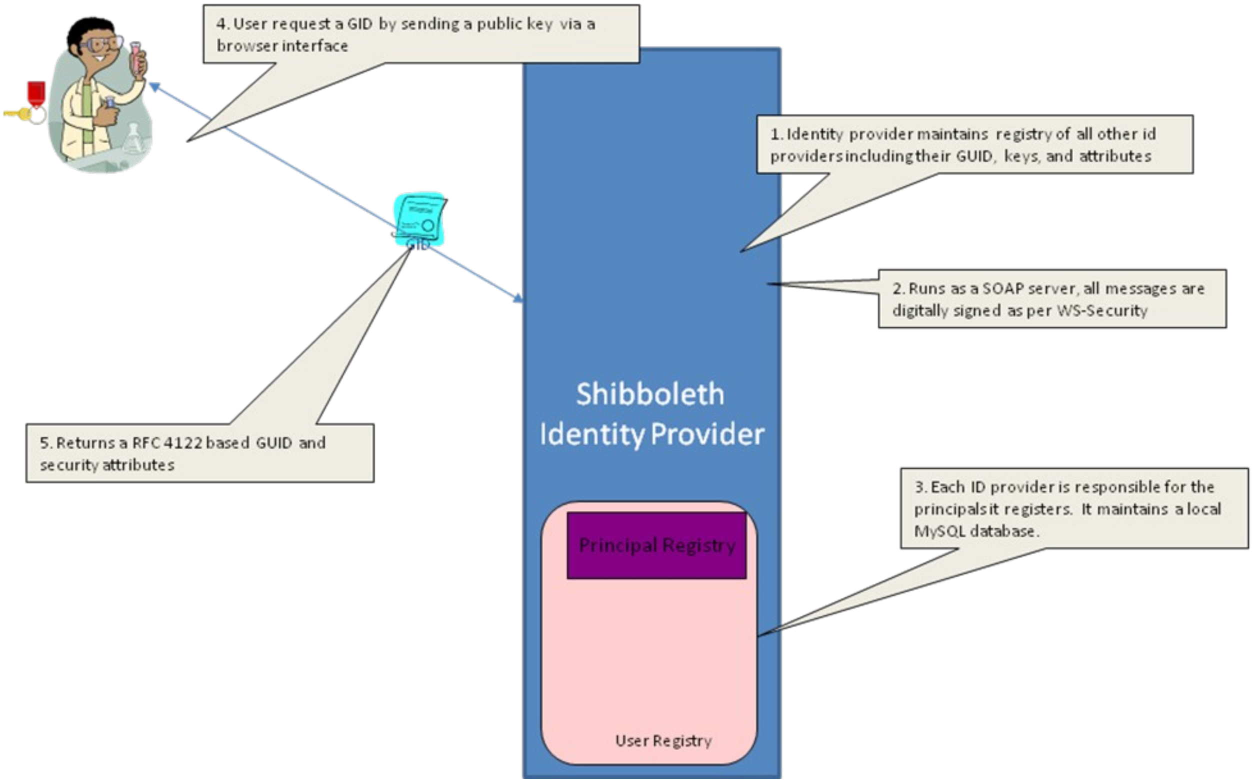
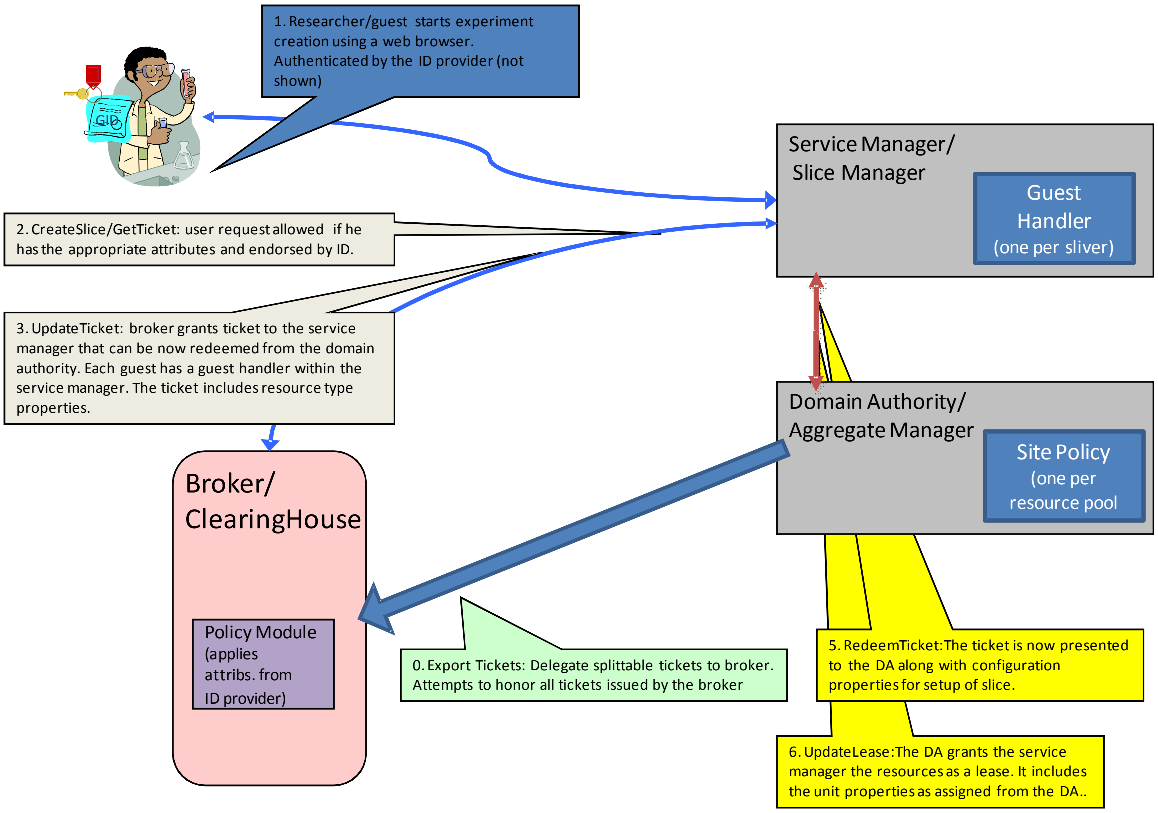

4. User request a GID by sending a public key via a browser interface
1. Identity provider maintains registry of all other id providers including their GUID, keys, and attributes
GID
2. Runs as a SOAP server, all messages are digitally signed as per WS-Security
Shibboleth Identity Provider
5. Returns 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 8: 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 in the real world identity, but it may not be necessary to mandate it. Early binding of identity could complicate the acceptable levels of indirection with GENI. For example, Jeff Chase in his comments on the CF Requirement document says "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 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)

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.

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 9: Slice Creation process in ORCA

7.6 ORBIT

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 access all experiment data files.

7.7 Analysis

Each of the control frameworks is pursuing a different path toward securing the GENI model as they incrementally work toward realizing a fully GENI-compliant implementation. This analysis is an attempt at a high-level characterization of the design space currently being explored by the five cluster control frameworks, and a comparison of how these approaches compare and contrast.

ORBIT has created a secure access mechanism at the perimeter of the system, currently an aggregate manager equivalent. PlanetLab has similarly created a secure access mechanism centered on an aggregate manager, but in this case the implementation provides a functional interface and abstractions that are semantically close to those described in the GENI control framework requirements documents. As the PlanetLab security mechanisms fully incorporate the RSpec definition for describing resources, it may become possible to apply the same structure to solving resource allocation problems within ORBIT. However, the wireless networking environment may pose challenges, especially in describing access to allocations of shared resources.

ORCA and ProtoGENI, in contrast, both are pursuing implementation strategies that incrementally support goals of the GENI control framework but critically provide for multiple manangers. Resource managers in ORCA are close to component managers in the GENI semantics, while each member of a ProtoGENI federation is close to an aggregate manager in terms of resource aggregation and control over allocation. A critical distinction is that ORCA passes authorization rights between the various entities in the system, while ProtoGENI appears to focus on passing authentication information, leaving authorization and fine-grained allocation decisions tightly bound to the aggregate managers (e.g. the Emulab or Emulab-like clusters acting in the role of aggregate managers in the prototype.)

TIED eschews the definition of low-level interfaces at the semantic level of the component manager, and instead inherits the interface and implementation presented by existing testbed software, currently the Emulab implementation underlying the DETER testbed. However, the security mechanism is essentially agnostic as to what specific testbed interface resources are to be allocated from. So long as the user's global identifier can be mapped to a local user identifier on each testbed, authorization decisions can be made locally. However, the TIED approach allows more information than the global or local identifiers to be interpreted on the global (portal or clearinghouse) side of the interface vs. the local (aggregate or component manager) side of the interface. This architecture may sit somewhere between ORCA and ProtoGENI as the implementation matures. Note that the conflation of the global identifiers (clearinghouse) with the local identifiers (Emulab, DETER, PlanetLab, etc.) may make it difficult to analyze where decisions are being made at the semantic level of a clearinghouse vs. an aggregate or component manager.

(N.B. This analysis is incomplete as of this draft.)

Cluster Framework	Current Security Choices
ORBIT	Identification using login information and public-private key pairs, no fine grain access control and authorization.

	authentication tokens encapsulates opaque state for authentication or authorization specific to the monitor for the resource. It has an identity credential associated with it which is the basis for all access control decisions. Per-object access control lists control access of all system resources.
PlanetLab	Entities have Global Identifier (GID) that includes a UUID and the entity's public key. The authentication of a principal is done by the server at a registry, slice or management interface. PKI and X.509 certificates are utilized to cryptographically sign and or verify information.
ProtoGENI	Authentication is based on a GGID represented as an X.509 certificate that binds a Universally Unique Identifier (UUID) to a public key. Authorization is initiated by the exchange of credentials that facilitate resource authorization and access control by aggregates
TIED	Identity established by local testbed authorities, through username plus passwords or certificates. Principals have attributes that are used for access control.

Table 2. Choices of Security Mechanisms in D&P GENI Control Frameworks

8. Spiral I Action Items

Prior to, or shortly after stand-up of GENI control framework or other prototype GENI facilities, a review of the following action items is recommended. These are for guidance only, but may be helpful in evolving a standard way in which the GENI community operates, especially with respect to security issues.

1. Trusted Root Certificates. If the implementation supports, sign all trusted root certificates with a different super-root certificate. Then remove the super-root private key from any on-line system and store off-line (two backups.) While not absolutely necessary, this will provide a way to create new trusted root certificates without self-signing. Also, trusted root certificates should probably have a 13-month lifetime, with the idea being that GENI prototypes plan to test rollover of their trusted root certificates after 12-months. If possible, a revocation format for certificates should be defined.

2. Physical security audit. It would be good practice to identify where the security servers or other testbed supervisor machines are located, and document who (individuals or class of individuals) have access.

3. Super-user audit. Similarly, identify who has root or equivalent access to the testbed supervisor machines, either at the local machine level or via privileges or rights being enabled in their GENI personal certificates. If possible, super-users should have non super-user certificates (or equivalent methods) to support doing work as ordinary GENI researchers vs. super-users managing/administering the GENI site.

4. Review of security relevant source code. A full red-team is not needed at this point for prototypes, but it would be helpful to have two individuals, other than the developers who wrote the security software review the design and source code. This could be as simple as a set of slides describing the software and a meeting to walk through the source code.

5. Operator and Facility POC information. In the future, a GENI operations facility may be able to field problem alerts and remotely kill slices or remotely shut down elements of the GENI substrate as an emergency response of last resort. In the short-term, having good current contact information (name, email, phone, physical location) for the operators of a GENI site and the facility (building or campus manager) where those GENI machines reside would be useful, to assist the GPO in responding to any problems.

6. Written Usage Policy. A short statement describing what can, and what can't, be done with the prototype GENI facility should be created. The PlanetLab AUP or other testbed guidelines might serve as examples. Of immediate concern will be experiments or slices that involve malware (e.g. wild viruses, worms, botnets, etc. captured from the Internet), or services that might be used to store and retrieve user-provided content, or that might cause traffic to be directed from the prototype GENI facility to an arbitrary 3rd party IP address.

7. Research User Management. The approach to assigning identities and credentials to new users, as well as granting access to GENI resources to those users should be defined. While nothing heavy-weight needs to be put in place, it would be a good idea to be able to prepare a list of users, their GENI site identities and authorizations/privileges/access rights, and to maintain this information at the prototype GENI site.

8. Testbed Monitoring. A plan should be in place for monitoring the use of the GENI facility. While no particular requirements have been developed, it would be expected to be able to log slice operations, such as creation, deletion, periods when active, etc. A discussion should take place with the local campus NOC so that they are aware of the GENI facility, and are prepared for the possibility of increases or spikes in traffic that might occur once experimental slices begin using the GENI facility. This is especially important if using shared bandwidth or networks across a campus to support GENI researchers and operations.

9. Attribute Based Access Control

The GENI ecosystem will be a highly decentralized collaborative environment that will contain a diverse set of hardware resources such as wireless nodes and sensors, reconfigurable routers, and optical hardware, and also provide programmability across every layer of the network stack. 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. This is made even more likely as GENI control frameworks interoperate through various forms of federation. For example, one federation may have a central registration authority for facility users that allows only registered (research) users to run experiments on component resources, while another federation may have a fully decentralized model where any user can be authorized locally by the component manager to experiment with resources. Furthermore, even when the authorization mechanism may be the same, the GENI security mechanisms will need to be agile enough to support different implementations because, for example, the authorization semantics for a highly connected optical mesh will be different than the authorization semantics for a sparsely-connected delay tolerant network.

Our goal is to define a GENI security architecture, and also to illustrate the design of feasible GENI security mechanisms that are both safe and usable by the community of experimental network and distributed systems researchers. We propose adopting ABAC [WJ03a] (attribute-based access control) semantics and concepts to authorize access to information and resources in GENI as a starting point. The flexible attribute-based access control mechanism makes access control decisions based on authenticated attributes of entities (i.e., organizations, users, or processes in the system), while simultaneously decentralizing attribute authority. The ABAC approach would permit access to services and information in accordance with security policies to include "limited distribution" within a net-centric environment that promotes discovery and data sharing. Thus while providing the protection and safeguards against malicious users ABAC facilitates automated enforcement of access control policies that allows unanticipated or new users timely access to data and services. Moreover, by focusing discussions on security abstractions with both well-defined and well-explored formal semantics and pre-existing implementations, we aim to avoid intellectual thrashing by providing a sufficiently concrete technology candidate, while focusing attention on the boundary between what is achievable today with minimal development and those aspects of the problem that require research breakthroughs.

In order to provide a flexible, decentralized, and scalable access control for the dynamic GENI network, the ABAC mechanism derives authorization decisions from chains of digitally signed attribute credentials, which is now a standard and well-understood approach. Credential issuers or managers assert their assertion or judgments about the attributes of entities through these credentials. These entities will include both research users and organizations. Since these credentials are digitally signed, they can serve to introduce parties, including strangers, to one another off-line when network connectivity is unavailable, a feature which may be highly useful in a federated network infrastructure, as it may sometimes be required to operate without complete and reliable connectivity. A key to ABAC's scalability is that the issuers of credentials can be strangers whose authority is determined based on their own attributes, as documented in further credentials.

Since many of the GENI sites and organizations will have limited mutual trust between each other, we believe the requestor and the ABAC access mediator will sometimes be unable to agree upon a trusted third-party that might assist them in using any sensitive credentials to establish the mutual trust. (ABAC abstractions admit the possibility that some aspects of an entitity's credentials may be sensitive.) Concern regarding the release of personal or private information or attributes of individuals is one situation which leads to a credential being deemed sensitive. Therefore, the ABAC approach calls for requestor and access mediator to enter into a kind of bilateral credential exchange, which the inventors refer to as a *trust negotiation*. The negotiation consists of a sequence of credential exchanges that begin by disclosing non-sensitive credentials. As credentials flow, more are unlocked, enabling them also to flow. In successful negotiations, credentials eventually flow that satisfy the policy required to access the desired resource. To control transmissions that could disclose whether or not the negotiator has a given attribute, ABAC implements attribute *acknowledgment policies (ACK policies)* and a *trust-target graph protocol*, that supports the ABAC credential language and distributed credential storage as shown in Figure 1.

Acknowledgement policies for example can be of the form SA_k $(1 \leq k \leq K)$, where $RA_m(1 \leq m \leq M)$ and $EA_n(1 \leq n \leq N)$ are the pre-defined attributes resources and the environment attributes. $ATTR(s)$, $ATTR(r)$, and $ATTR(e)$ are attribute assignment relations for subject *s*, resource *r*, and environment *e*, where:

$$\mathrm{ATTR}\ (\ r\ ) \subseteq RA_1 \times RA_2 \times \ldots \times RA_M\ ;$$
$$\mathrm{ATTR}\ (\ e\ ) \subseteq EA_1 \times EA_2 \times \ldots \times EA_N\ ;$$
$$\mathrm{ATTR}\ (\ s\ ) \subseteq SA_1 \times SA_2 \times \ldots \times SA_K\ ;$$

Using ABAC would permit GENI to support strong authenticated identities and authorization policies, while leaving enough flexibility to support organizations and individuals that require some degree of pseudo-anonymity.

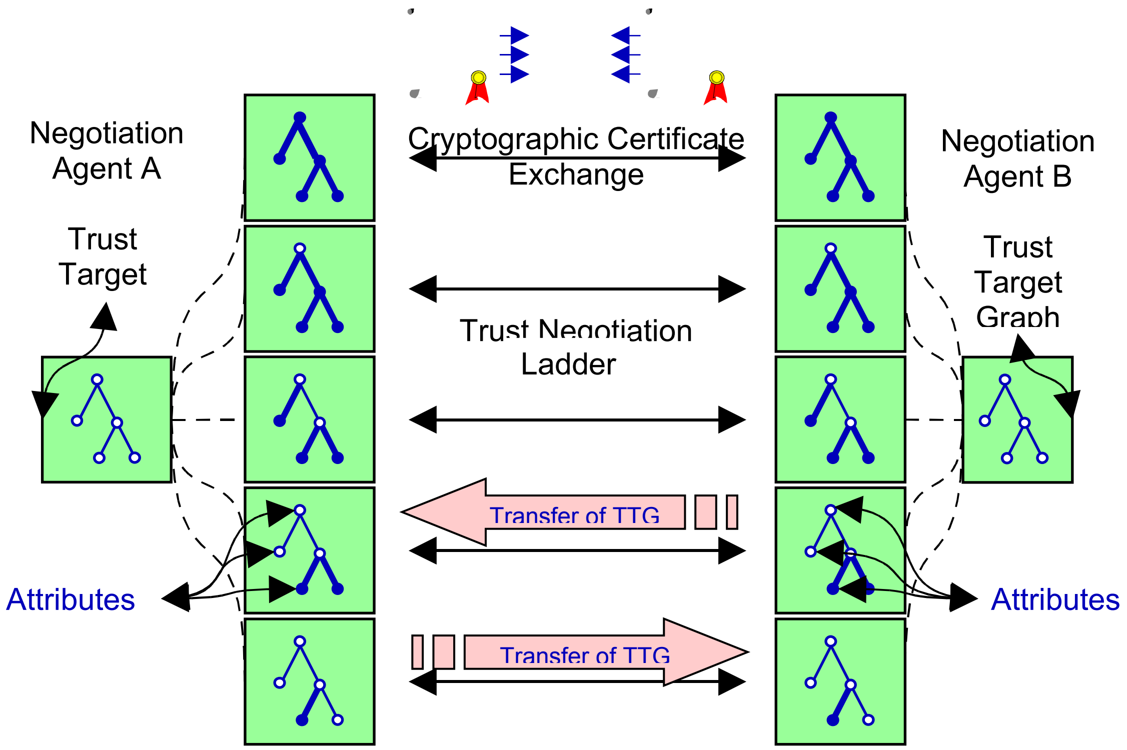


Figure 10: ABAC Trust Negotiation Overview

We also envision abstractions and a supporting mechanism for administration and exchange of attributes across different federates, similar to Shibboleth. While the Shibboleth project has deployed an implementation providing a single sign-on mechanism for universities on Internet2, we are concerned that the large Shibboleth code base is less amenable to rapid, spiral development in support of GENI prototypes. Using the single sign-on and authorization mechanism, Shibboleth, provides a secure framework to transmit attributes to remote authorities. For example when an analysis application attempts to access sensor measurements at a remote domain, the application's own home security domain will send certain information about that application to the remote domain in a trusted exchange. These attributes will then be used by the remote domain to help determine whether to grant the user access to the sensor measurements. Shibboleth mechanisms provide a clean separation of identity and authorization functions and makes use of other attributes to mutually refer to the principal's identity, but are perhaps tied to closely to the web browser/web server model. The GENI Security Architecture should have the ability to interface with, or reuse elements of Shibboleth and the SAML policy language (or other RBAC policy languages) as appropriate, without locking the GENI Security Architecture into being solely dependent on any one technology.