An emulation of GENI access control

Soner Sevinc, Larry Peterson
Princeton University
35 Olden Street, Princeton, NJ, USA
{ssevinc,llp}@cs.princeton.edu

Trevor Jim, Mary Fernández
AT&T Labs Research
180 Park Avenue, Florham Park, NJ, USA
{trevor,mff}@research.att.com

Abstract

This paper describes an emulation of a distributed access control system proposed for use in the GENI network testbed. We use our trust management system, CERTDIST, to realize the system policy, and measure its performance by mapping PlanetLab's centralized access control scheme to GENI's distributed scheme and then replaying logs of PlanetLab access control activity. Our log analysis indicates that any such system must be resilient to both misconfigurations and attacks, and our emulation results show the effect of caching schemes and certificate expiration intervals in reducing load on servers and improving response time.

1 Introduction

Internet-scale distributed systems, like PlanetLab [8] and the planned GENI [3] platform, provide infrastructure for distributed applications to share computational, network, and storage resources. Nodes in such distributed systems publish security policies in the form of cryptographically signed certificates. Before an application can access system resources on behalf of a particular user, the necessary certificates must be collected and security policies checked.

Distributed system platforms may provide a *trust management service* that collects and checks certificates on behalf of an application. Such a service may be implemented by a centralized authority that proactively collects all certificates for all system resource owners and answers all certificate requests. PlanetLab, for example, has provided a centralized trust service in PlanetLab Central (PLC) since its inception. We observe, however, that Internet-scale federated systems will be composed of autonomous administrative domains, each of which manages its own users and security policies and may be reluctant to share sensitive data with any centralized authority, trusted or not. PlanetLab is currently evolving into a federation of independent administrative domains each in control of its own security policies, and GENI is designed similarly. To better support their client applications, such systems should provide *distributed trust management*, where participants cooperate to collect and check security policies.

This paper describes an emulation of a distributed access control system being developed for GENI. The security policies are written in a declarative framework called CERTDIST, which uses certificates to distribute policies within the system using mechanisms including a distributed hash table for load balancing and efficiency. One distinguishing characteristic of our work is our methodology for evaluating our system. To examine how the system functions under a realistic work load, we acquired one month of PlanetLab logs containing users' access control requests from over 400 sites across the world. We translate these requests into the operation request format currently being prototyped for GENI and then re-distribute policies and requests to appropriate geographic locations across PlanetLab. CERTDIST is deployed at about 550 PlanetLab nodes, permitting us to emulate evaluation of security policies as they might transpire in GENI.

GENI is representative of the federated distributed systems that motivate our research, whereas PlanetLab provides us with experimental data and is our experimental platform. We review both and present an example of how security policies are evaluated in Section 2. Section 3 describes the logs that we collected and use in our experiments. In addition to supporting our own systems research, we expect that these logs (http://www.planet-lab.org) and our experimental platform will be of value to others studying distributed access control. Section 4 describes the CERTDIST architecture. The system behavior and the experiment results from our system's certificate caching constructs are explained in Section 5.

2 GENI's security architecture

GENI's security architecture [4] is an evolving proposal, and there have been several competing designs; ours is based on the "geniwrapper" implementation that provides a GENI-like interface to PlanetLab.

In geniwrapper, an experiment runs in a *slice*: a collec-

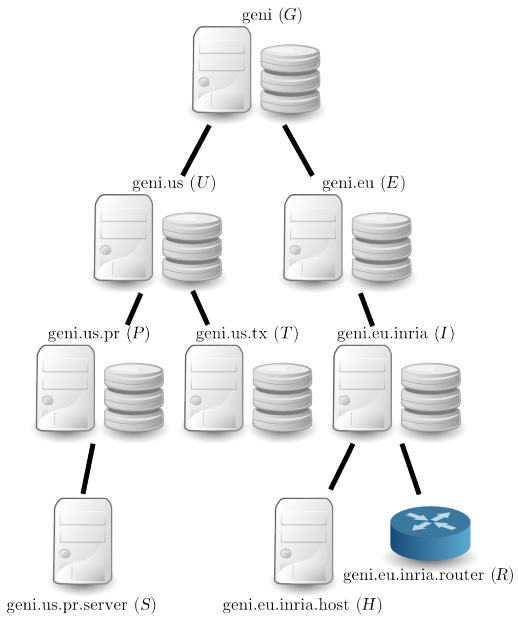

Figure 1: GENI's federated structure.

tion of distributed resources belonging to a set of *components* such as edge computers, customizable routers, etc. An experiment is run by a group of researchers, called the users of the slice. An *authority* is an administrative domain that manages a set of components, users, and authorities, and handles resource allocation by granting slices to its users and other users. Each authority has a *registry* that stores its policy information.

Figure 1 depicts a GENI federation. Each node in the figure has a *human-readable name (HRN)*. The edges in the figure imply a trust relationship; for example, geni.us.pr is under the authoritative purview of geni.us, which can vouch for the behavior of geni.us.pr. The name structure thus reflects the administrative structure of the federation, which in general can be a forest. For our emulation we have used a single global root, geni, so our federation has a tree structure where interior nodes are authorities (servers with associated registries), and leaves are components and users.

Every GENI object (i.e., an authority, slice, user, component, service, etc.) has a *global id (GID)* consisting of a unique identifier and a public key. In Figure 1, the GIDs of objects are in parentheses. An object has a private key corresponding to its GID, which can be used to sign *certificates*. We use the notation "A says ____" for a certificate signed by (the private key of) object A.

Authorization in GENI When a GENI object wants to perform an operation, it must supply certificates that show that it is authorized to do so. Exactly what certificates are required depends on the operation, its parameters, the invoker, etc.; and whether or not a specific certificate will be issued depends on the policy of the signer.

1. G says U is us
2. U says P is pr
3. P says B is bob
4. G says U can Register on geni.us
5. U says T can Register on geni.us.tx
6. T says B can GetSliceCred for geni.us.tx.slice
7. H says B can CreateSlice on geni.eu.inria.host

Figure 2: Certificates.

For example, if a user geni.us.pr.bob of a slice geni.us.tx.slice wants to run an experiment on geni.eu.inria.host, he must first perform the GetTicket() operation at geni.eu.inria.host. If approved, he will receive in return a certificate that represents a set of resources that he can claim when the experiment is launched.

Figure 2 gives the certificates that he must supply when invoking GetTicket(). The GIDs G, U, P, and T are those of the objects geni, geni.us, geni.us.pr, and geni.us.tx (as in Figure 1), and B is the GID of user geni.us.pr.bob.

Every GENI object starts out knowing the root's name (geni) and GID (G), while other GIDs are discovered by chains of certificates starting with one signed by G (i.e., signed by the private key corresponding to the public key of G). GENI uses linked local namespaces [1], so certificate 1 implies that U has name geni.us, certificates 1–2 imply P has name geni.us.pr, and 1–3 imply B has name geni.us.pr.bob. Certificates giving an object's name are required by most operations, for accounting purposes.

Certificates 4–6 say that B is the GID of a user of a slice geni.us.tx.slice. Certificate 6 alone is insufficient for this purpose: Certificates 4–5 are necessary to prove that T has the authority to issue such a certificate [4]. The result of GetTicket() is certificate 7, which states that B can allocate resources on geni.eu.inria.host. The user can later call RedeemTicket() on geni.eu.inria.host, which requires certificates 1–3 and 7 to launch the experiment. The key observation is that a GENI operation call may require gathering several certificates each signed by geographically and organizationally distinct authorities.

3 Access control in PlanetLab

To better understand how access control might perform in GENI, we have logged the access control behavior of PlanetLab, and we have created a projection of how the same behavior would have played out if GENI access control were used instead. This section describes the logging.

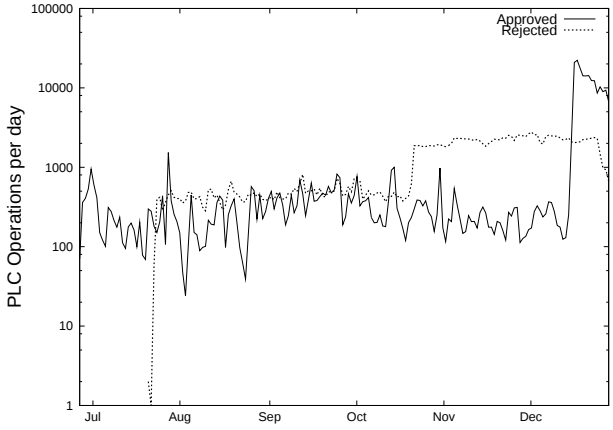

Figure 3: PLC operations per day, 26 June–27 Dec 2008.

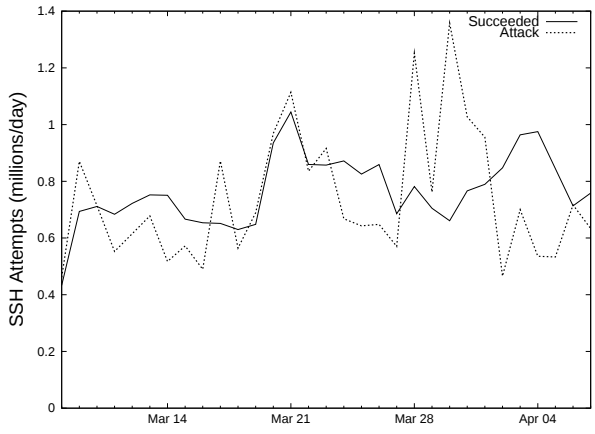

Figure 4: SSH attempts per day, 8 March–7 April 2009.

PlanetLab makes access control decisions at two places: first, at PlanetLab Central (PLC), which maintains records for users, nodes, and slices; and second, at individual nodes, where users run experiments. We have gathered logs of the authorization activity at both places.

Figure 3 shows the access control decisions made at PLC over a six-month period in 2008. One curve shows the number of approved operations per day, while another shows the number of rejected operations per day. The spike in approved operations in December is due to the start of some new experiments. The jump from zero rejected operations in July corresponds to a major upgrade in the PLC software, which improved logging of errors; the PLC administrators think that the number of rejected operations did not in fact increase due to the software upgrade. The large spike in rejected operations in October appears due to a single misconfigured experiment.

Figure 4 shows the number of SSH login attempts per day over a one-month period in 2009 to 573 nodes in PlanetLab. One curve shows successful logins, while another shows unsuccessful logins which we believe are due to brute-force SSH attacks. The successful logins account for 50% of the attempts, and the attacks account for 49%. We also see a small number of failed logins for legitimate slices; these typically occur because of a delay in propagating slice authorization from PLC to nodes. The failed logins are not displayed on the graph, since they account for less than 1% of the total SSH attempts.

We can make some general observations, which we expect are pertinent to any access control system for a network testbed similar to PlanetLab. First, the PLC operations correspond to administrative actions such as adding new users and slices, and they occur much less often than actual accesses (SSH logins). Second, misconfigurations and misuse of the PLC API are common, occurring at roughly the same magnitude as correct uses. Third, brute-force SSH attacks on PlanetLab nodes are also common, occurring at the same frequency as legitimate logins.

4 CERTDIST system overview

CERTDIST is a system for specifying security policies and retrieving and distributing certificates. Like other trust management systems [5, 10], CERTDIST is general (not limited to GENI access control) because policies are written in an application-independent, declarative language. CERTDIST policies are readable and scalable because they do not specify details of certificate signing and distribution; those details are handled automatically by a peer network incorporating both authoritative servers and distributed caching using a DHT.

We apply CERTDIST to our GENI example (cf. Figure 1) as follows. Each GENI object runs a CERTDIST daemon that exposes the object's policy to others, and caches and retrieves certificates from other CERTDIST daemons as needed. Since our experiments have no user machines, user objects are handled by component daemons, keeping policies isolated for each object.

Each daemon loads both a system-wide policy that specifies the set of certificates required to perform each GENI operation, and also a policy specific to the object. The local policy includes the cryptographic key pair of the object and the access control information saying who can use local resources or invoke specific operations. All policies are specified with a declarative language, Prolog, which simplifies policies, increasing the likelihood that the policy writer produces both complete and correct specifications. A GENI policy containing both system-wide and local rules is about 150 lines. CERTDIST uses the SWI-Prolog interpreter [9], which provides a foreign

language interface that allows us to extend the interpreter with functions for cryptographic signing and issuing network requests.

When a GENI object wants to perform an operation, its daemon consults the local policy to determine what certificates are necessary. If the certificates are found in the local cache they can be returned immediately; otherwise the daemon requests them from remote daemons. When a daemon receives a certificate request, it decides whether to grant or reject the request by examining its local policy and any certificates provided by the caller. For example, certificates 1–6 of Figure 2 will be gathered by geni.us.pr.bob's daemon in order to perform the GetTicket() operation, and they can be submitted to geni.eu.inria.host, which will examine its local policy and the certificates before granting certificate 7.

In CERTDIST, a daemon can obtain a missing certificate in two ways. The first way is to make an *authoritative query* to the certificate's signer. For example, if the daemon is missing a certificate signed by geni.us.pr, it can request it directly from geni.us.pr's CERTDIST daemon. In order to do that, however, it must obtain geni.us.pr's IP address, and that is determined by a GENI policy (the GENI policy includes secure name lookup, superseding DNS). An authoritative query therefore requires the daemon to obtain a certificate chain to find the server's address.

The second way to obtain a missing certificate is through a *DHT query*. CERTDIST uses a Kademlia distributed hash table [6] to cache and distribute certificates. Each CERTDIST daemon has a Kademlia key/id that is the cryptographic hash of the object's public key and serves as its routing address. Certificates are stored in the network under the key consisting of the hash of the *query* that retrieved them. For example, if a daemon makes an authoritative query Q and receives a certificate C in return, then it stores C in the network under $\mathrm{hash}(Q)$. Then, when another daemon needs to make query Q, it can look for the answer in the peer network. As usual with Kademlia, whenever a daemon successfully retrieves C from a peer, it tries to store it at a closer peer, to speed subsequent lookups.

Certificates are only stored in the DHT on demand. If a daemon does not find the certificate in the DHT after a certain timeout, it abandons the DHT query and falls back to an authoritative query. When it receives the certificate from the authoritative server, the daemon stores it in the DHT for future use. Certificates contain expiration times, and peers delete certificates as they expire.

5 Preliminary evaluation

We present preliminary experiments that measure the cost of authoritative queries and the effects of certificate caching and expiration times on system load and response time. Our experiments are executed on 550 PlanetLab nodes and are driven by emulated GENI events derived from the PlanetLab event logs described in Section 3. We first describe how the emulation events are derived from the PlanetLab logs and then present the experiments.

A GENI federation and its corresponding HRNs will reflect a complex geographic and administrative structure. To emulate a deeply nested structure from PlanetLab's two-level structure, we essentially reverse the domain names of PlanetLab hosts and root them in the appropriate federation (e.g., host.inria.fr becomes geni.eu.inria.host). We map user and slice names in a similar way (e.g., tx_slice becomes geni.us.tx.slice).

To drive the emulation, we map each access-control event in the PLC API logs and SSH logs to one or more GENI operations. Figure 5 contains an example SSH log event from the log on host.inria.fr and its corresponding representation as a fully-specified GetTicket operation. The SSH event includes a slice name and origin IP address but does not give the user of the slice (in PlanetLab each slice corresponds to an SSH account, and every user of the slice logs into the same SSH account). Therefore, for our emulation we randomly pick a user of the slice to perform all of the GENI operations of the slice originating at a given IP address. The emulated user's public key is obtained from PLC. The destination and slice name are derived from the log entry, and for the origin, we choose a machine near the emulated user in the HRN structure. HRNs are derived using the mapping rules described above. During emulation, evaluation of this GetTicket operation will require gathering the certificates in Figure 2.

Mapping PLC-API events to GENI operations is straightforward, because they contain the user, origin, and detailed operation parameters. The destination of each PLC-API event is PLC itself, so we select a destination authority from one of the 550 available PlanetLab nodes using heuristics that depend on the invoking user and operation type and parameters. For example, an event to update host.tx.edu's records at PLC would likely be mapped to an operation that updates geni.us.tx.host's records at the geni.us.tx authority.

Before emulation, we distribute each GENI operation to the PlanetLab node that corresponds to the operation's origin. Ideally, each operation would be emulated at the true origin, but individual users' machines are not publicly available in PlanetLab. Therefore, we select an origin node from the nodes within the user's authority, or, if necessary, from the parent authority. For example, if no nodes of geni.us.pr were available, we would select one from the geni.us authority. After distributing operations, we launch CERTDIST daemons at every invocation location and begin replay of the local operations.

PL (SSH) event

Slice: tx_slice
Origin: 150.135.68.171
Destination: host.inria.fr
Timestamp: Mar 11 02:39:24

GENI event

Operation: GetTicket(geni.us.tx.slice)
User: geni.us.pr.bob
User's public key: *(in PEM format)*
Origin: geni.us.pr.host
Destination: geni.eu.inria.host
Timestamp: 2008–11–03 02:39:24

Figure 5: A PL event and corresponding GENI event

5.1 Expirations and flash crowds

We use signed certificates for efficiency and scalability. Certificate reuse eliminates repeated signings, which can cost as much as seven times more than returning a signed certificate. Signed certificates can be cached and reused, until their expiration, without the involvement of their authoritative signer. However, the use of expiration times can perversely increase the load on the authoritative server; if many parties are relying on a certificate, they will all want the server to refresh the certificate at its expiration. This can create a periodic flash crowd at the server.

There are several ways we might address flash crowds (e.g., by using both time-to-live fields and expiration times [7]). So far, however, our experiments show that our DHT-based distributed caching scheme sufficiently mitigates the problem. The results of one such experiment are given in Figure 6, which shows the query rate at the most heavily loaded server, the U.S. authority. In the figure, the solid curve corresponds to running the experiment with the DHT disabled, and the dotted curve corresponds to an enabled DHT. For this experiment we used a certificate lifetime of 15 minutes, and the figure shows a rough peak in queries every 15 minutes. The DHT is able to reduce the authoritative query rate by more than 50%.

5.2 Obtaining missing certificates

Section 4 described how a CERTDIST daemon obtains a missing certificate. The process is complicated: performing an authoritative query may require an address lookup which itself may need missing certificates, and a DHT query involves peer routing and may fall back to an authoritative query when the DHT does not have the certificate. We have experimented with several variables that affect the overall cost of obtaining a missing certificate:

Timeout In our query strategy, a daemon first searches the peer network for a missing certificate, and, if that fails, it conducts an authoritative query. Therefore, when the certificate is not in the DHT, a DHT query performs strictly worse than a simple authoritative query. This means that the timeout we select for giving up the peer search is critical for performance. We are currently using 0.7 seconds as our timeout value, as our experiments show that this is sufficient to capture 90% of successful peer searches. Timeout on authoritative queries is important for usability and server load. We use 5 seconds for the authoritative query timeout.

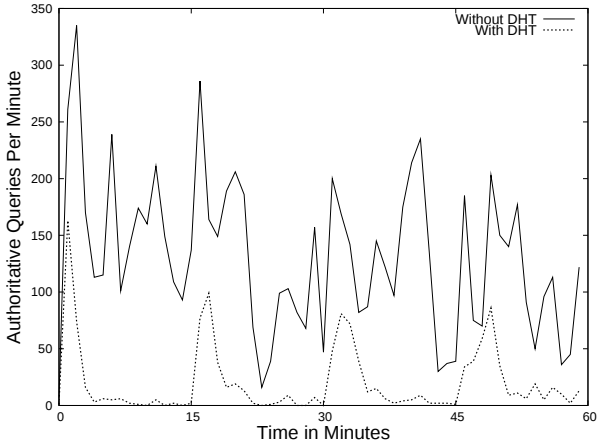

Figure 6: Queries to U.S. authority (15-min. expiration)

Certificate popularity Storing unpopular certificates in the peer network causes load on the peer network but does not reduce the load on authoritative servers or improve response time. In our experiments we are currently only storing certificates signed by the root, U.S., and Europe authorities in the peer network. We plan to automate this selection in future work.

Load The DHT has the most benefit when the system is under high load.

Figure 7 gives the results of several runs of our experiments that examine the effects of load under different configurations. We conducted emulations at the actual load seen on PlanetLab and at ten times that load, and we conducted emulations using the DHT as described and with the DHT disabled (all queries were authoritative queries).

The graph shows the cumulative time to retrieve missing certificates, including successful and failed attempts. For example, 50% of missing certificates are retrieved in one second when the system is configured for authoritative queries only at normal load (AUTH $1\times$ SUCC). The DHT does not improve the retrieval times under normal load, although we observed earlier that, as expected, it reduces load on the authoritative server. Deployment problems, unexpected node failures, etc are challenging with

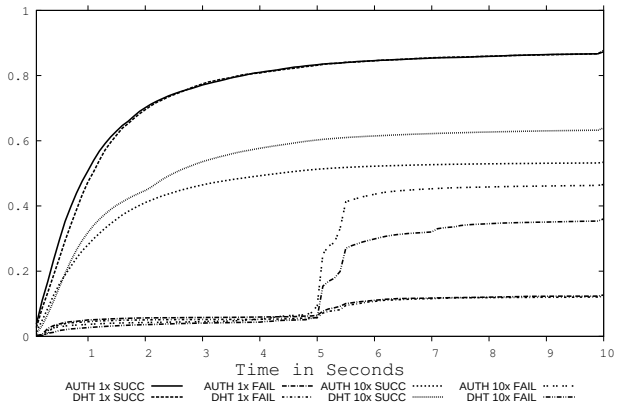

Figure 7: Cumulative probability of certificate retrieval times.

550 machines, and constitute the major reason of 10% failure under normal load. These will be solved by per-site dedicated management teams in a real GENI deployment. In the lower part of the graph, the failed attempts show a sharp jump at five seconds, which is the timeout for authoritative queries when CERTDIST daemons give up on obtaining a missing certificate. Experimenting with $10\times$ load allows us to observe system behavior with a larger user base. Because the PlanetLab nodes are not dedicated servers, under high load many queries time out; however, the DHT improves the success rate by balancing the load. Higher timeouts could improve the success rate but would mean longer response times, less QoS for the user. Higher dedicated resources from PlanetLab is required for better QoS for the user in 10x case, and we see DHT can be a way to achieve same QoS with less dedicated resources. Lastly, we expect running user activity on PlanetLab machines rather than user machines will have little effect on query times since there is similar network latency in queries and little node overhead to affect response times.

6 Discussion

A major focus of our future work will be to explore caching and retrieval strategies for certificates. The CERTDIST implementation currently retrieves a policy's missing certificates sequentially. We plan to study the impact of parallel retrieval of independent certificates on total policy evaluation time. For example, the certificates 1–6 of Figure 2 consist of two independent certificate chains, 1–3 and 4–6, which could be retrieved in parallel.

Currently, CERTDIST makes static, system-wide decisions with respect to cache parameters, such as timeout, certificate popularity, and maximum server load. Dynamic optimization of cache parameters to minimize response time is another area of future work. We also plan to investigate whether cache implementations like one-hop-DHTs and CDNs can improve certificate retrieval time.

Effectiveness of certificate caching in other systems like GENI is an important area for future work. The request patterns for certificates depend on operation request patterns, certificate expiration values, and delegation structure, all of which impact certificate caching. These interactions are the subject of further evaluation.

Evaluating a policy may require contacting many principals in different domains, so the reliability and efficiency of a domain can affect larger portions in a distributed trust system. This is a real concern since in PlanetLab it is not unusual for a significant fraction of nodes to be offline. Our CERTDIST implementation will let us study how uptime, denial-of-service attacks, and variations in the security policies affect reliability.

Finally, we have found PlanetLab's access control logs invaluable for studying attacks, misconfigurations, temporal user behavior across different domains, query distribution, and so on. We plan to make the PlanetLab logs accessible to others to foster further research.

References

[1] M. Abadi. On SDSI's linked local name spaces. In *10th IEEE workshop on Computer Security Foundations*, page 98. IEEE Computer Society, 1997.

[2] P. Druschel, M. F. Kaashoek, and A. I. T. Rowstron, editors. *Peer-to-Peer Systems, First International Workshop, IPTPS 2002, Cambridge, MA, USA, March 7–8, 2002, Revised Papers*, volume 2429 of *Lecture Notes in Computer Science*. Springer, 2002.

[3] GENI. `http://www.geni.net/`.

[4] GENI security architecture spiral 1 draft 0.4. `http://groups.geni.net/geni/attachment/wiki/GENISecurity/GENI-SEC-ARCH-0.4.pdf`, Feb. 2008.

[5] T. Jim. SD3: A trust management system with certified evaluation. In *IEEE Symposium on Security and Privacy*, pages 106–115, 2001.

[6] P. Maymounkov and D. Mazières. Kademlia: A peer-to-peer information system based on the XOR metric. In Druschel et al. [2], pages 53–65.

[7] P. McDaniel and S. Jamin. Windowed certificate revocation. In *IEEE INFOCOM 2000*, volume 2000, pages 1406–1414, 2000.

[8] PlanetLab. `http://www.planet-lab.org`.

[9] SWI-Prolog. `http://www.swi-prolog.org/`.

[10] W. Zhou, Y. Mao, B. T. Loo, and M. Abadi. Unified declarative platform for secure networked information systems. Technical Report 872, Dept of CIS, Univ. of Pennsylvania, Philadelphia, PA, USA, March 2008.