ProtoGENI, A Prototype GENI Under Security Vulnerabilities: An Experiment-Based Security Study

Dawei Li, Xiaoyan Hong, *Member, IEEE,* and Darwin Witt, *Student Member, IEEE*

***Abstract*—ProtoGENI is one of the prototype implementations of global environment for network innovations (GENI). ProtoGENI proposes and executes the GENI control framework, including resource management and allocation for authenticated and authorized experimenters. Security and inevitably are the most important concerns in the whole development process. In this paper, we study and evaluate its security vulnerabilities according to GENI's security goals. We analyze the threat model of ProtoGENI and categorize four broad classes of attacks. Based on the role of an active experimenter, we demonstrate experiments as proof of the concept that each class of attacks can be successfully launched using common open source network tools. We also present analysis and experiments that show perspectives on the potential risks from an external user. Furthermore, we discuss the feasibility and possible defense strategies on ProtoGENI security with respect to our preliminary experiments and potential future directions. Our contribution lies in examining known vulnerabilities without requiring sophisticated experiments while remaining effective. We have reported our findings to the ProtoGENI Team. Our work indicates that the solutions have been deployed. This paper validates that experiment-based vulnerability exploration is necessary.**

***Index Terms*—GENI security, global environment for network innovations (GENI) experiments, ProtoGENI, vulnerability.**

I. Introduction

AN INFRASTRUCTURE like the global environment for network innovations (GENI), a virtual laboratory for at-scale networking experimentation [2], [7], has distinguishable features that bear significant complexities and challenge its development and operation. This global network research testbed consists of a large amount of computing and networking resources of various types. Some examples include server racks, switches, open-flow routers, VLAN backbones, wireless devices, sensor devices, and mobile devices. These devices are geographically distributed and federated through control frameworks. GENI allows networking experimentation to be conducted at an Internet scale using a large variety of network devices and technologies, both current and forthcoming.

One critical issue for such an at-scale infrastructure is security. The primary goals of GENI security are to avoid being abused to conduct illegal activities or being abused as a launch pad for attacks, and to ensure that the availability of services is not compromised by attacks [3]. It is extremely challenging to achieve these security goals due to many system features, including distributed ownership of the resources, distributed users groups, deep programmability of the infrastructure resources, super flexibility of configurability of the virtualized resources, large-scale connectivity to the Internet, and large geographic span. In addition, the vast variety of research experiments that can be conducted at GENI could produce network behaviors in an unexpected pattern, thus making the detection of an anomaly difficult.

To understand the necessity of the rigid vulnerability analysis of GENI (in our case, ProtoGENI), we consider an experimenter's approach to explore the current implementation to identify potential system vulnerabilities. Our results will help understand security requirements, and will provide the development team with the possible suggestions and solutions for those vulnerabilities [5]. We hope that the experiments will assist in building a more secure research infrastructure. Our experimentation was performed in conjunction with related personnel and under the supervision of the applicable authorities. Potentially harmful experiments were limited to our own testbed slices and machines.

Our current work focuses on the ProtoGENI control framework [4]. ProtoGENI is built based on the network research infrastructure of Emulab, UT [1]. It provides researchers with a wide range of networking environments under which they can control conditions and securely conduct repeatable research experiments. ProtoGENI can be regarded both as a hardware facility providing computing and networking resources and as a software defining control framework that can control, manage, and dictate policies for user authentication, resource allocation, and communication between different parties.

In this paper, we introduce the threat model of ProtoGENI (and GENI) and propose four broad classes of attacks that can violate the security goals of GENI. For each class, we describe the network experiments and the exposed vulnerabilities. Our

Manuscript received October 9, 2011; revised March 14, 2012; accepted July 26, 2012. This work was supported in part by the BBN/NSF Contract Project 1783.

D. Li was with the University of Alabama, Tuscaloosa, AL 35487 USA. He is now with the Department of Computer Science and Engineering, Lehigh University, Bethlehem, PA 18015 USA (e-mail: dal312@cse.lehigh.edu).

X. Hong is with the Department of Computer Science, University of Alabama, Tuscaloosa, AL 35487 USA (e-mail: xhong@ua.edu).

D. Witt was with the University of Alabama, Tuscaloosa, AL 35487 USA. He is now with the Human Centered Design and Engineering Program, University of Washington, Seattle, WA 98195 USA (e-mail: dcwitt@crimson.ua.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JSYST.2012.2221959

experiments mimic the attacking behaviors of data plane to control plane, data plane to data plane, data plane to Internet, and Internet to ProtoGENI. We analyze and explain the security limitations in the current ProtoGENI implementations associated with these experiments. The results of our experiments show that the harm can be in two large categories, namely, the possibility of acting as a launch pad and the possibility of being compromised for availability. In this paper, a few necessary conditions for performing these attacks are presented to understand the feasibility of those threats along with the suggestions on the possible defense strategies. The outcomes after reporting our findings to the development teams are introduced with the associated experiments.

Our experiments are proof of concept because no sophisticated experiments are required and can be used to show the potential harmful consequence, when an insider or an outsider is present. As such, GENI's security goals can be compromised. The contribution of this paper is to show that the known vulnerabilities, which affected other systems and networks, can be effective against ProtoGENI. In fact, we have performed more experiments as attacks. In this paper, we summarize and present experiments that can indeed raise cautions in the development of ProtoGENI. We have reported these findings to the corresponding GENI development teams and suggested them possible defense strategies. This paper validates that the experiment-based vulnerability exploration is necessary.

The remainder of this paper will start with a brief introduction on ProtoGENI in Section II, and then the threat model and our methodology are given in Section III. Our attacking experiments of different classes are presented in Sections IV–VII. We discuss and summarize our findings and defense strategies in Section VIII. Related work is presented in Section IX. Conclusions are given in Section X.

II. Background

A. ProtoGENI Control Framework

ProtoGENI is a prototype implementation and deployment of GENI functions. We introduce the key components of ProtoGENI facility and functioning software entities. They describe the management and the usage of ProtoGENI.

1) Clearing house: center for registration and authorization.
2) Component manager: resource provider, managing resources at a particular location.
3) Slice: container for resources, providing a piece of live virtualized network testbed for an experiment; it can cross many resource providers.
4) Slice authority: managing the slices, authenticating users to slices.
5) Sliver: computing resources granted to a slice.
6) RSpec: resource specification, the mechanism used for advertising, requesting, and describing the resources.
7) Vnode: virtual node, sharing with other slices in the current sliver through splitting a physical node using virtualization. Current ProtoGENI realizes Vnode through OpenVZ.
8) VLAN: VLAN links to connect otherwise separated experimental nodes.

B. ProtoGENI Security Architecture

GENI relies heavily on the existing authentication, authorization, and security protocols to allow secure Internet-scale management, operation, and communication [9]. One mechanism that GENI uses is to implement a hierarchy of authorization of individuals and experiment through public key infrastructure (PKI). Such a mechanism is well developed through clearing house and interactions between component manager and slice authority. The credentials will be used and passed around, when an experimenter requests and uses slices and slivers. GENI grants a component manager the authority to start and manage slices locally. The nature of the GENI security architecture will assume that common security practices, such as updating mission-critical software on hardware components, will be in place. In addition, the researcher should not have to assume trust of the nodes, network environments, and other end users of the GENI network, nor should it be necessary for the components or component managers to trust the rest of the GENI control framework to which it is connected. Furthermore, the GENI control framework itself uses the existing Internet protocols for managements and operations with mechanisms to ensure that this control framework is securely constructed.

Due to GENI's proposed size and method for growth via federation, the pre-existing secure protocols, such as corporate and government PKI and authenticated identities, would be far too difficult to maintain. This complexity is due to the fact that different authorities have different authorization schema to manage such separate systems. Currently, it is also under investigation and implementation for the GENI to use attribute-based identities and access control [12]. With this latter scheme, the aspects of the principal's attributes may change as the principal interacts with a federated system.

The early GENI developers' approach toward the security goal catches the following essences: 1) identity management including credential generation and delegation; 2) emergency stop procedures being ready to shut down some experiments or the entire GENI infrastructure; and 3) security best practices at different aggregates (operating organizations with physical resources to offer).

C. ProtoGENI Authorization

Authorization in the ProtoGENI system is initiated by the exchange of credentials that facilitate resource authorization and access control by aggregates. The PKI that is used to authenticate principals provides all of the keys and other structures to sign and verify credentials during this process. The steps include: 1) a user obtains an SSL certificate and passphrase for connecting to the ProtoGENI portal, and 2) then obtains an SSH key for remote access to the experimental nodes in the sliver. After this initial getting ready phase, a user can repeatedly use the established credentials to perform network experiments (the execution phase). For each experiment, a user interacts through scripts with the ProtoGENI control framework (clearing house, slice authority, and component manager) using the credentials for resource allocation as well [4].

III. Threat Model

GENI security architecture and policy are defined in a few documents [3], [9]. These documents describe a general threat model for GENI, which is independent of any single control framework. The design and development of ProtoGENI's security architecture closely follow the general model. Here, we describe the threat model based on the ProtoGENI control framework. The threat model is illustrated as rings in Fig. 1. The inner ring is comprised of threats from ProtoGENI control framework (including control manager, slice authority, clearing house, etc.) and system administrators. Threats from this ring can damage the whole system severely. However, the control framework and system administrators are the most trusted entities in a system. The middle ring includes threats from the experimenters and their authorized slices as well as from the software running within the slices. The outermost ring depicts threats from the outsiders (of GENI) residing in the Internet. This is because a ProtoGENI slice is connected to the Internet through the experimenter's local machine, when it is active.

A. Classes of Attacks

An experimenter processes all the credentials and related access rights. It is attractive for an attacker to seize the control of an experimenter or his slices by stealing the credentials from a lab machine or through Trojan Horse. It is also possible that a legitimate experiment may go wrong without being noticed by the owner. When such situations occur, the experimenter can act as an inside attacker. Our analysis of the ProtoGENI control framework suggests that three classes of attacks can be performed as an insider. First, a ProtoGENI experimenter can use his authorized slice to attack the control framework and ProtoGENI infrastructure. Second, if the isolation between experimental slices is not guaranteed, an experimenter can disrupt another user slice intentionally or by chance. Third, GENI experiments may attack the outside world, i.e., the Internet either accidentally or maliciously. Moreover, as an experimenter, one is subject to the external attacks launched from the Internet.

In this paper, we consider ProtoGENI administrators and control framework software trustworthy. Using experiments, we follow the above four broad classes of attacks to address the security issue.

When the implementation of the GENI security architecture is fully in place, it will be empowered by the strong anomaly detection capability. Although it is too early to conclude that these approaches would be sufficient, let alone that the development procedure itself needs additional safeguards (see TUF project securing software update [14]). In addition, large-scale experimental testbeds need additional security mechanisms, for example, the DETER testbed [8]. Similarly, it is vital to investigate the ProtoGENI implementation and explore its vulnerabilities to deploy more defense solutions.

B. Methodology

We study the feasibility of each class of attacks by analyzing the implementation of the control framework and then by performing experiments to verify the feasibility analysis.

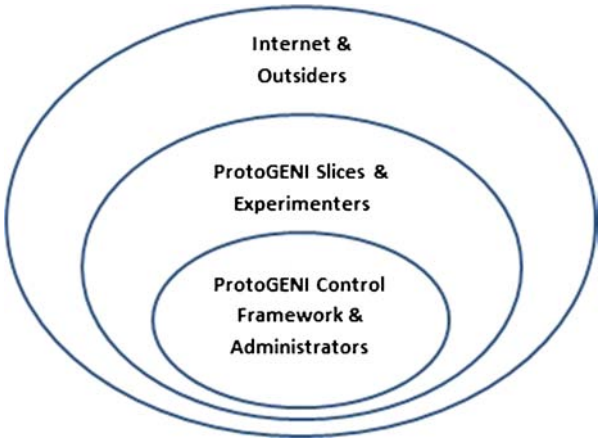

Fig. 1. ProtoGENI ring threat model revised from [3].

When conducting these experiments, we always play two roles, both as an attacker and as a victim experimenter. We use two experiment slices. One slice acts as a launch pad for the attacking experiments, while the other slice acts as a normal experiment slice. We observe the normal experiment slice to check results when the attacking experiments are performed in the other slice.

We use common network testing tools for our experiments, such as *ping* and *iperf*. *Ping* helps us to test the availability of ProtoGENI resources, while *iperf* acts as a network traffic generator. Both tools also report measurements on delay, hop count, and throughput. We also use the software *netwox* [11], an open source network tool set, to perform network attacks, such as packet sniffing and spoofing. *stress* is another system workload generator that can impose a configurable amount of CPU, memory I/O, and disk stress on the system, which is useful when we investigate the implementation of network virtualization.

In the next four sections, we describe our experiments according to the four attacking classes and analyze the associated security issues.

IV. From User Slices to Control Framework

An experimenter has the privilege and opportunity of using authorized ProtoGENI slices. If compromised, the experimenter can act as an inside attacker and generate great damages. One direct hit could be the control framework. Here, we describe our experiments to show that damage can cause the denial of ProtoGENI's services to the users.

A. Feasibility Study

ProtoGENI facility can be viewed as two separate planes: control plane and data (experimental) plane. The data plane contains user slices and this is where the experiments exist. The experiments are configured according to the network topology defined in RSpecs. The control plane is a separate network that configures and interacts with the data plane to support the experiments. It also allows users to access the slices from outside of ProtoGENI, usually through an Internet SSH connection. The architecture is shown in Fig. 2. In this

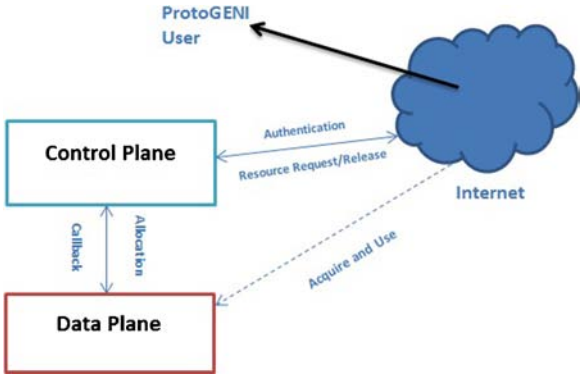

Fig. 2. ProtoGENI control plane and data plane.

architecture, all the experimental nodes in the data plane are connected to the control router in the control LAN. All the nodes have publicly accessible IP addresses. Thus, each experimental node has multiple network interfaces, including one interface connecting to the same LAN with the control router; one to Internet, while the other interface is connected to other experimental nodes in the data plan according to the topology of experiment.

Connecting a large number of available nodes within a single control LAN has potential problems. Once a malicious user obtains the control of one experiment node, he can easily launch attacks disturbing connections that the other experiment uses to connect to the same control router and the Internet. Those victim experimenters may find their experimental nodes unavailable or unaccessible. Attacking experiments described here are LAN specific, such as ARP cache poisoning attack [10]. The attack target is the LAN of the control network. Each node in the LAN has an ARP cache storing entries that map the IP addresses to the corresponding MAC addresses of other nodes interfaces. The entries of the IP-to-MAC address mapping can be altered by spoofed ARP packets sent from an attacker, known as the ARP cache poisoning attack. An attacker can send spoofed ARP packets either to control router or to the experiment nodes, as illustrated in Fig. 3.

In our experiments, we use *netwox* to launch ARP cache poisoning attacks to terminate connection between the control router and an experiment node. The victim experiment node can be a node in use or an available resource for allocation. As a result, the physical experiment node will be disconnected or not be available for allocation. One challenge is a soft state of ARP cache. Thus, an attack has a way to sustain a damage over a longer time. In order to deal with the expiration of the spoofed ARP entry, we use a shell script to repeatedly send spoofed packets using the *netwox* command (Fig. 4).

B. DoS of the Connection to Control Router

Here, the victim experiment node is a node available for resource request and allocation. We create two slices with names experiment1 and experiment2, respectively. The slice experiment1 is alive with a single node node1 (physical node pcwf146) that has *netwox* installed for conducting attacks; experiment2 is a pending slice for acquiring the specified node pcwf142 as its experimental resource. There are two ARP cache poisoning methods to achieve the DOS attack.

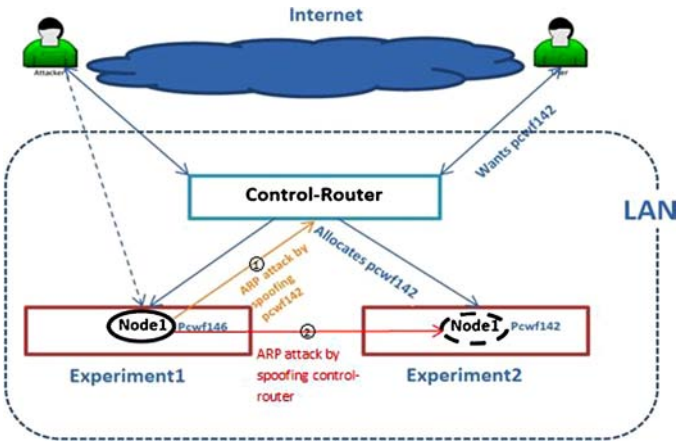

Fig. 3. ARP cache poisoning.

```
#!/bin/bash
i=0
while [ $i != 1 ]
do
 sudo /usr/local/bin/netwox 33 -d eth2 -a 0C:0C:0C:0C:0C:0C -b 00:19:B9:23:AD:69 -c 2054 -e 2
-f 0C:0C:0C:0C:0C:0C -g "155.98.36.1" -h 00:19:B9:23:AD:69 -i 155.98.37.142
done
```

Fig. 4. Repeated ARP cache poisoning.

1) *Poisoning the ARP Cache of the Victim Node:* We use the *netwox* tool no. 33 to poison *pcwf142*'s ARP cache about the control router, i.e., the tool will send a fake MAC address of the control router in an ARP message to *pcwf142*. This method is illustrated as the #2 action in Fig. 3. We would like to modify the MAC address of the control router to $0C:0C:0C:0C:0C:0C$ in *pcwf142*'s ARP cache. In order to verify the attacking result, we try to acquire *pcwf142* for experiment2 to see whether the resource is still available without any exception. If the attack was successful, the victim experiment node *pcwf142* would lose its connection to the control router and hence this resource will no longer be available.

We observed that the experiment2 still acquired the *pcwf142* as its experiment node. We checked the ARP cache of *pcwf142* and found that the ARP entry for the control router was not changed. This is because the ProtoGENI facility sets the ARP entry of the control router to be a static entry at the experimental nodes. This protects ProtoGENI from being attacked by this specific DOS attack. The result suggests that modifying an experimental node does not effectively launch a DOS attack.

2) *Poisoning the ARP Cache of the Control Router:* In this experiment, we exploit the control router directly by poisoning its ARP entry about the experiment node *pcwf142*. If this experiment is successful, then the attack achieves the same goal for cutting connection between the control router and experiment node. The users would still not be able to acquire desired resources. The experiment scenario is shown in Fig. 3 as the #1 action. We use the same two slices experiment1 and experiment2, respectively.

We use the *netwox* to poison the control router's ARP cache by spoofing a faked MAC address for *pcwf142*. We would like to modify the MAC address of *pcwf142* to $0C:0C:0C:0C:0C:0C$ in the control router's ARP cache. Then, we verify the

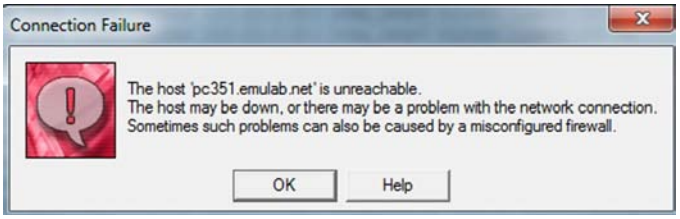

Fig. 5. Connection failure.

result by trying to acquire *pcwf142* for experiment2 to see if the resource is still available without any exception. This time, the result is inclined to the attacker and *pcwf142* is no longer available. At this stage, we cannot check the ARP cache in the control router to directly verify results. But, we observed a series of warning and error messages in the users local interface indicating that *pcwf142* cannot be used. Examples include "pcwf142 appears wedged; it has been 6 min since it was rebooted," "$node\_reboot-reboot\_node$: pcwf142 appears dead; will power cycle," "$ERROR$: $node\_reboot - reboot$: Powercyle failed for *pcwf142*," and "*pcwf142* may be down." This experiment shows that the denial-of-service attack is possible.

In summary, we have shown that denial-of-service attacks on a ProtoGENI node through ARP cache poisoning can be performed by modifying the control router's entry about the victim node. If such an attack was successful, the victim experiment node would lose its connection to the control router and hence would no longer be available. In our experiment setting, users' request for experiment2 would not be satisfied. Furthermore, without knowing a specific physical node, an attacker can poison the ARP table at the control router with a brute force method of trying all the IP addresses in the address space.

C. Links in Virtual Topology

Here, we examine whether or not the virtual links in the data plane are isolated from the physical links in the control network. In other words, will a failure of a physical link affect the virtual links? Again, two slices are created, the slice *vlan* is a regular slice with two experiment nodes, *left* and *right*, and a link connecting them; and the slice *attack* is the attacking slice with a single node with *netwox* installed.

The attacker launches an ARP cache poisoning attack to disconnect the normal node *left* in the active slice *vlan* from the control router, using the method described earlier. As a result, the local machine of the normal user cannot access the *left* via SSH (Fig. 5). Thus, the DOS attack to the connection of the slice in use is successful.

We then use the normal node *right* to *ping* the node *left* through the experimental virtual topology. The result shows that the *left* is still reachable by *right* (Fig. 6). The results of the experiment suggest that the experimental topology in ProtoGENI is actually an LAN composed of experiment nodes with separate real network interfaces through dedicated experimental switches. The failure of the control network connection will not affect the experimental topology.

```
[lidawei@right ~]$ ping left
PING left-center (10.10.1.1) 56(84) bytes of data.
64 bytes from left-center (10.10.1.1): icmp_seq=1 ttl=64 time=0.242 ms
64 bytes from left-center (10.10.1.1): icmp_seq=2 ttl=64 time=0.155 ms
64 bytes from left-center (10.10.1.1): icmp_seq=3 ttl=64 time=0.192 ms
64 bytes from left-center (10.10.1.1): icmp_seq=4 ttl=64 time=0.228 ms
64 bytes from left-center (10.10.1.1): icmp_seq=5 ttl=64 time=0.265 ms
```

Fig. 6. VLAN is still connected after attack.

TABLE I
ALLOCATED RESOURCES

Slice Name	Exp. Node Name	Host Name
test1	shared1	pc175.emulab.net
test1	shared2	pc172.emulab.net
test2	shared1	pc172.emulab.net
test2	shared2	pc175.emulab.net
test3	shared1	pc263.emulab.net
test3	shared2	pc102.emulab.net

V. BREAK THE ISOLATION BETWEEN SLICES

Slices share resources in the data plane. Virtualization keeps them separated and isolated. Here, we show experiments that explore issues related to the resource isolation.

A. Feasibility Study

Virtualization in GENI makes it possible to share resources among many users while being able to meet their computing and networking requirements. However, the virtualized environment should be developed carefully. Bugs within the system could lead to violations in isolation. The first experiment here shows an instance, where an exposed entry exists for another user to send traffic across the slices.

Another key consideration of virtualization is the appropriate distribution of computing and network resources such as CPU, memory, and bandwidth, when allocated to different slices. Potential defects in the virtualization technology could permit a user to interrupt the experiments of others with whom he shares the resources, be it on purpose or not.

B. Explore Bugs in Virtualization

This experiment shows that a defect in implementation can lead to traffic being sent cross slices. We create three slices with names test1, test2, and test3, respectively. Each slice has the same topology of two Vnodes named shared1 and shared2, and a link of bandwidth 100 Mb/s connecting them. The two slices test1 and test2 also share the same physical nodes. The tool *iperf* is installed on every node. All the acquired resources are summarized in Table I.

In our first attempt, only one *iperf* server is running on slice test1 at the node shared1. From the nodes shared2 of both slices test1 and test2, we connect to the server shared1 with the following command: $iperf\ -c\ shared1$. We expect each shared2 to connect to shared1 in its own slice. However, what we observed from the screen of the *iperf* server (shared1 at test1) is that both the clients are connected to the same server even though they are not from the same slice, i.e., the nodes can communicate across slices.

In Fig. 7, we illustrate our experiment with the screen captures of four nodes. The left side shows the *iperf* server

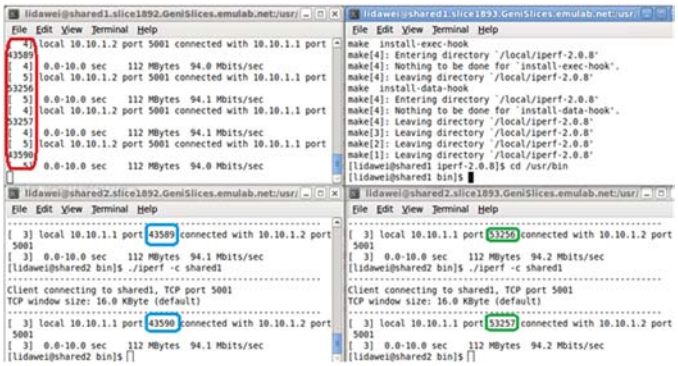

Fig. 7. Cross-slice experiment (I).

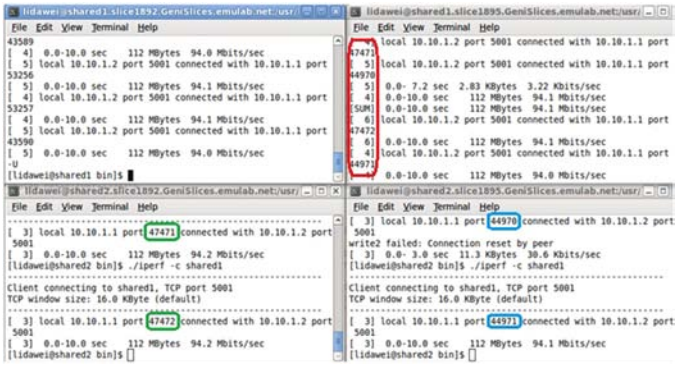

Fig. 8. Cross-slice experiment (II).

and client in test1, and the right side shows those in test2. Fig. 7 shows that the server shared1 in test1 (*slice1892*) (the upper left terminal) is connected by the clients with ports in the sequence of (numbers in the red circle) 43589, 53256, 53257, and 43590, respectively. On the other hand, the first client shared2 in tests1 (*slice1892*, the left lower terminal) connects to the server with the ports (in the blue circle): 43589, 43590, and so on. The second client shared2 in test2 (*slice1893*, the right lower terminal) connects to the server with the ports (in the green circle): 53256, 53257, and so on. These results suggest that the problem could be due to the fact that both the slices (test1 and test2) share the same physical resources (*pc175* and *pc172*).

We then try to see whether the issue occurs to the nonsharing slices. We perform the same experiment with the slices test1 and test3, which do not share physical resources. We obtained the same result, as shown in Fig. 8.

To understand the extent of this issue, we examined many resource allocation combinations, including changing Vnode to a normal node, connecting to the *iperf* server with the IP address, and using different node names for different slices (no matter whether it is a Vnode or a normal node). These experiments do not show this problem. Thus, we can say that the cross-slice communications would happen only when the nodes are Vnodes with the same experiment name defined in the RSpec, and additionally the same names are used in *iperf* as a server. The instance seems difficult to encounter due to the rare conditions that lead to it. However, the chances are much higher, if ProtoGENI users opt to use default names (very likely) that are usually provided in tutorials and the Flack web interface.

This experiment showcased that bugs in implementation of the control framework could result in interference and interruption among experiments, and could also generate abnormal traffic. For the latter, it is also difficult to detect. If the bug was exploited by an attacker, he could send malicious traffic to a normal ProtoGENI experiment and can harm the experiment result. Upon this finding, we have reported findings to the developer team at the University of Utah. The problem has been solved and our follow-up experiments have validated that this isolation problem does no longer exist.

C. Explore Computing Resource Isolation

The virtualization technology adopted by ProtoGENI is able to allocate dedicated computing resources for many slices. Each slice should not be affected or exhausted by another slice that acquires a virtual machine in the same physical machine. We test this function through the following experiment.

We create two slices with names vnode1 for a normal slice, and vnode2 for the attacker's slice. Each slice has a single Vnode. The two slices are located at the same physical node *PC263.emulab.net*. The attacker installs the tool *stress* and uses it to generate excessive usage of CPU and memory in his virtual machine to exhaust the computing resources. If the attack is successful, the computing resources such as CPU and memory of the normal slice vnode1 will be reduced.

In the normal slice vnode1, we use the linux command *top* to show the CPU and memory usage before the attack . The result indicates that CPU is free with 0.0%us and 1087232 kB free memory. The attacker runs the *stress* software in his Vnode with the command

$$stress\ -cpu\ 2\ -vm\ 1\ -vm-bytes\ 256M\ -timeout\ 30s.$$

This command imposes a load by specifying two CPU-bound processes and one memory allocator process requiring 256 MB memory size. With the command running, we observed changes in the resource usage by the normal slice's Vnode. It shows that CPU is still free without being interfered, but the free memory decreases to 833 900 kB. The result suggests that the processor is isolated and dedicated for a Vnode, but the memory is interfered by the shared slice.

The above experiment shows that the virtualization technology used in ProtoGENI will allocate a dedicated CPU resource to a particular Vnode. However, the physical memory resource is not isolated among different Vnodes residing in the same physical machine. Attackers can use this drawback to exhaust the memory resource so that the performance of normal ProtoGENI experiments will degrade, or the experiments will be disabled.

VI. From Slices to Internet

ProtoGENI slices are designed with features that are convenient for experimentation. Some of the examples include easy access though the Internet and file uploading and downloading from local machines. These features, if exploited by a compromised slice, could help launch attacks from the slices to hosts or servers residing in the Internet.

TABLE II
BANDWIDTH TEST RESULT

Traffic Direction and Throughput	
ProtoGENI Node to Lab PC	91.7 Mb/s
Lab PC to ProtoGENI Node	54.7 Mb/s

A. Feasibility Study

An experiment node in ProtoGENI has a public IP address for convenient access through the Internet. The potential vulnerability is quite obvious. If ProtoGENI does not provide any precaution (in fact, it does), a malicious user can use ProtoGENI slices to launch attacks to the Internet. Such attacks can also be launched from multiple nodes or aggregates from different geographic locations within GENI. Here, we perform a proof-of-concept experiment to show the possibility of such attacks.

In addition, an experiment node in ProtoGENI can upload files to a file server or host residing in the Internet (or download) through a typical SSH connection using the public IP address. Such a file transfer will not alter the file itself. This feature helps upload users' experiment data to local machines for analysis. However, as an inside attacker, this could be abused to send unknown viruses or worms to the Internet. Thus, a malware uploaded or downloaded from a ProtoGENI node can remain undetected, and can be invoked with a simple click on an executable code. To prevent the attack, one approach is to separate the click and execution. A proposed method is to encrypt data retrieved from ProtoGENI so that it cannot be executed automatically by a simple click [8].

B. Throughput Test

In this experiment, we verify the achievable throughput on the connection between a ProtoGENI node and a local host. This verification will help create an understanding of the feasibility of launching a flooding attack. We use a single ProtoGENI node and a PC from our laboratory. Both hosts have *iperf* installed to test the connectivity and bandwidth. The ProtoGENI node has an ethernet card with 1000 Mb/s, and our laboratory PC has an ethernet card of 100 Mb/s. Since the laboratory PC has only 100 Mb/s, the traffic is limited by this bottle-neck link. In the client host, we connect the *iperf* server by running

$$iperf \ -c \ ServerIP \ -t \ 30.$$

The *t* flag is to set test time at 30 s. The collected achievable throughput data are listed in Table II. (Throughput is the average over 40 experiments.)

Table II shows that the traffic sent from a ProtoGENI node to the laboratory node uses most of the laboratory node's bandwidth, which makes it possible for the laboratory node to be a victim (with lower bandwidth) of a flooding attack. In contrast, the throughput from the laboratory node to the ProtoGENI node is far less than the ProtoGENI node's bandwidth, and is even less than its own bandwidth of 100 Mb/s. The result suggests that sending data from a ProtoGENI node with a larger bandwidth has potential to overwhelm a laboratory node with a low bandwidth in the Internet side.

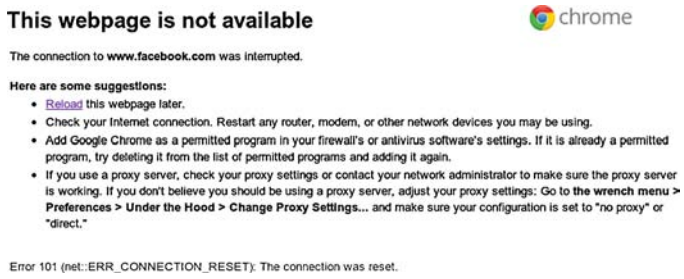

Fig. 9. Ping flood result.

C. Flooding Attack

Most of the ProtoGENI nodes enjoy high bandwidth connections with the Internet. This benefit could potentially help launch a flood attack on a low bandwidth node outside ProtoGENI. Here, we showcase a very simple flooding attack using *ping flood*, i.e., the attacker overwhelms the victim with *ping* packets as the attacker sends more traffic than the bandwidth of the victim. In this experiment, we use a laptop connected through Wi-Fi 802.11 g to further reduce the bandwidth of the laboratory node. The bit rate for 802.11 g connection is at most 54 Mb/s. The experiment node in ProtoGENI (still 1000 Mb/s) launches the ping flood attack with the command

$$ping \ -f \ -s \ 65000 \ VictimIP.$$

The *f* flag is the flood mode of ping with zero intervals and the *s* flag is to set the ping packet size (maximum 65 507 bytes).

We monitor the consequences of the experiment through Internet applications running on the victim laptop. As a result, we observed that several network applications are disconnected (e.g., Skype, IM). We were unable to open a web page (Fig. 9) either. This result proves that attackers can use ProtoGENI resource to attack the Internet.

This simple case implies more devastating extensions. For example, the attacker can launch a DDoS attack from many ProtoGENI nodes and, maybe, from different component managers using ping flood to the victim node even though it may have a large bandwidth (e.g., 1 Gb/s Ethernet connect), or a DDoS attack can be launched to attack many nodes in the Internet.

The flooding attack can easily be monitored at both ends through fire-walls. When detected it should be stopped. An automatic intrusion prevention system can react to such events quickly and can potentially control the damage. At the time of experiments, such a mechanism was not installed. It should be easy for ProtoGENI administrators to track the vast volume of traffic to identify the attacker. In our case, the post warning was given by the ProtoGENI team. Currently, a fire-wall is installed.

VII. ATTEMPTS FROM OUTSIDE OF PROTOGENI

Nevertheless, ProtoGENI (and GENI) could be an attractive target for attackers from outside of the GENI infrastructure.

```
root@wl2:~# nmap pc57.emulab.net

Starting Nmap 5.51 ( http://nmap.org ) at 2011-09-16 13:08 CDT
Nmap scan report for pc57.emulab.net (155.98.36.57)
Host is up (0.070s latency).
Not shown: 843 closed ports, 154 filtered ports
PORT      STATE SERVICE
22/tcp    open  ssh
5001/tcp  open  commplex-link
32769/tcp open  filenet-rpc
```

Fig. 10. Scanned open port when iperf server is running.

```
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0- 2.4 sec  4.12 MBytes  14.7 Mbits/sec
[  5] local 155.98.36.71 port 1024 connected with 130.160.198.86 port 55714
[  5]  0.0-12.4 sec   640 KBytes   422 Kbits/sec
[  4] local 155.98.36.71 port 1024 connected with 155.98.36.75 port 37821
[  4]  0.0-10.1 sec   113 MBytes  94.1 Mbits/sec
[  5] local 10.10.1.1 port 1024 connected with 10.10.1.2 port 49324
[  5]  0.0-10.1 sec   113 MBytes  94.1 Mbits/sec
```

Fig. 11. Iperf connection summary.

Many ways could be used to explore the vulnerabilities of ProtoGENI. Here, our experiments show how common methods might succeed.

A. Feasibility Study

With publicly available IP addresses, attackers residing in the Internet can easily take these ProtoGENI nodes as targets for attack. Open ports, for example, can be the first thing to attempt. In addition, an open port identifies a running service. This can be used by attackers to identify and compromise certain kinds of services. In ProtoGENI (and GENI), many services are offered to facilitate experimentation, for example, measurement software. This kind of software will open ports for the measurement traffic and data collection. These ports can be scanned by an attacker from the Internet if not filtered by a fire-wall.

B. Exposed Port Detection

In this experiment, we examine whether open ports can be detected from outside. We start with *iperf* and then examine a real ProtoGENI instrumentation tool *Instools*. For both the experiments, we run the software on the ProtoGENI nodes and scan the nodes from a laboratory machine through the Internet. The slice has two nodes *exclusive-0* (*pc57.emulab.net*) and *exclusive-1* (*pc75.emulab.net*). The node *exclusive-0* runs an *iperf* server with default port 5001, and *exclusive-1* acts as the *iperf* client that sends packets to the *iperf* server.

In the laboratory PC, we use software network mapper (Nmap) to scan the *iperf* server running on the ProtoGENI node *pc57.emulab.net*. Fig. 10 shows that there are three open ports detected: port 22 for SSH connection, port 5001 for *iperf* connection, and port 32769 for filenet-rpc. Knowing that port 5001 is open, we send packets to the *iperf* server from our laboratory and succeed (Fig. 11). Another note is about port 32 769 for filenet-rpc. It is known to be vulnerable by malformed requests to cause denial of service. Port 32769 also has vulnerabilities caused by trojans and remote code execution.

Then, we installed the software *Instools* on the two ProtoGENI nodes. As required by the *Instools*, an additional node (pc243.emulab.net) is added to the slice to monitor the traffic between the two ProtoGENI nodes and collect data. Note that the added node has a public IP address. When the service is running, we use Nmap to scan one of the normal experiment nodes and the monitoring node from a laboratory PC. Our observation is summarized in Table III. It is clear that *Instool* uses more open ports at its service monitoring node. Those ports are common well-known ports.

TABLE III
OPEN PORT SCAN RESULT

Node	TCP ports	UDP ports
Experiment node	22 (SSH)	none
Measurement node	22 (SSH), 80 (http), 443(https), 3306 (mysql)	none

Through experiments, we discovered four issues.

1) Some ports should only open to the data plane, yet, they have opened to the Internet.
2) Some exposed ports can accept traffic from the Internet. For both cases, countermeasures must be built to verify the legitimate of the traffic.
3) Some well-known vulnerability-prone ports are open to outside. Although they are currently secure with the updated versions, potential cautions must be taken in place to prevent old versions from being loaded into an experimenter node.
4) Our experiment shows that the ProtoGENI fire-wall can filter some important ports (as shown in Fig. 10, 154 filtered ports), but there are still ports exposed and some may have vulnerabilities such as port 32769, and some are default open ports.

In summary, our experiments do not intend to show that these open ports caused or will cause security threats to ProtoGENI, rather they suggest that more ports can be exposed outside of ProtoGENI like these. They offer possibly the first step for attackers to start their attempts by identifying the running services with vulnerability.

VIII. SUMMARY OF FINDINGS AND SUGGESTIONS

The results of our experiments show that we are able to launch attacks, following the threat model with the privileges of an authorized user. Specifically, as an inside attacker, it is possible to compromise the availability of ProtoGENI resources and disturb another running experiment and the Internet. Meanwhile, as an outside attacker, it is also possible to invade the ProtoGENI facility through publicly accessible IP addresses. Although the experiments are not sophisticated, they act as proofs of concept for the threat model.

To summarize, we question to what extent can such an attack cause troubles to ProtoGENI or GENI? First, we discuss the minimum conditions in order to launch the attacks to the control network as an inside attacker. Three conditions are identified: 1) using an active slice; 2) learning the environment; and 3) using a tool. In terms of condition 1, the possibility of compromising an active slice is similar to the case of using a local host such as the safety of SSH credentials, the security

of the communication channel or the operations system. This is because an experimenter will obtain certificates and keys for experiments, and all the experiments are accessed through a local host. In dealing with condition 2, our other experiments have explored several ways in which we are able to learn the names of the physical nodes and also to learn the IP addresses of the entire testbed facilities. This is because the name space and IP address space are maintained with easy-to-follow rules for the simplicity of management, which is desired for ProtoGENI. Thus, an attacker can easily guess or probe (e.g., through *ping*) for the needed information. This helps further learn the MAC address when an IP address is known. A node can then learn all the MAC addresses. Thus, condition 2 is not a barrier to an attacker. Condition 3 is not a barrier as well because open-source tools can be found on the Internet. After all, condition 1 is the minimum condition for launching the attacks. This suggests that the security of authorization and authentication of ProtoGENI is critical.

For an outside attacker, our preliminary port scan experiment has suggested a few ways that could possibly be the first step toward starting the attempts. Cautions must be taken when more services and user contributed software are used in ProtoGENI, if they open ports for communications. Future work can also focus on developing new solutions.

Based on our experiments, we present the following analysis and suggestions.

1) The exposure of the control plane to the data plane leaves attackers (malicious ProtoGENI users) the chance of launching ARP cache poisoning attacks, which could result in failures in experimentation. To prevent and detect malicious ARP packets, several strategies can be considered for the control routers and experimental nodes, such as static ARP tables, firewalls, separate daemon tools like ARPON and ARPwatch, and hardware solutions. Currently, static ARP tables are regarded as an easy implementation by the ProtoGENI team. Our follow-up experiments were not successful, which shows that the defense is in place.
2) Isolation of slices has to deal with many aspects of the implementations in virtualization technology. The GENI component providers and control framework development team could leverage a few selected virtualization technologies in this regard. Our experiments show the case of OpenVZ, where the CPU is well isolated, but not memory sharing. The problem can be mitigated theoretically by using other VM technologies, such as Xen, or by setting the memory isolation of OpenVZ manually. The problem of network traffic isolation caused by bugs in control software would be hard to find. Our experiences are rather sporadic. The occurrence could be rare as well. Systematic maintenance can serve as a precaution.
3) The ProtoGENI architecture uses publicly accessible IP addresses, which is likely to leave chances for attackers from the Internet. It will be important to have a mechanism in place that would allow open ports only for the experiment nodes belonging to the same slice, but not to the rest. Another possible solution is to place the whole system behind a VPN or enforce more strict firewalls. For example, the Emulab testbed of *Testbed@TWIST* [24] requires VPN access before connecting to experimental nodes through SSH. In addition, SSH connections are the major way to operate and control the ProtoGENI experiment nodes. The SSH credentials are stored at users' local hosts. If the credential is stolen from the local machine (the Internet side), damage could be large (satisfying condition 1, so as to act as an insider). Thus, compared to access through a VPN, ProtoGENI's access security is weaker.
4) We have shown that malicious experimenters can attack the Internet servers or hosts from active slices with plenty of handy resources. The feasibility of such attacks is based on: 1) the traffic monitoring capability of ProtoGENI, and 2) the amount of resources that the slice owns if concerning DOS attack or flooding attack. Traffic monitoring at firewalls can help greatly deflect such attacks. Currently, the ProtoGENI facility has the capacities built. On the other hand, for the file transfer function, an existing prevention approach is to encrypt data retrieved from the ProtoGENI before transfer. This will prevent the innocent user from directly invoking an executable file immediately after clicking [8].

IX. Related Work

A similar federated testbed infrastructure is DETER. DETER is designed for medium-scale repeatable experiments in computer security [8]. The testbed is built as an Emulab-based infrastructure. Thus, experiences with DETER can help in ProtoGENI development. DETER includes several mechanisms to deal with threats, such as monitoring the control network, placing fire-walls, and physically separating experimental networks from the control network to prevent packets from being routed outside one's experiment. However, GENI infrastructure is more complex than DETER, with different usage scenarios and a large variety of resources and virtualization techniques.

Data plane to control plane attacks have been reported for the Internet. In [23], the authors used a distributed denial of service attack in the data plane to generate a surge of BGP updates. When BGP sessions are carefully chosen, the surge of updates will surpass the computational capacity of affected routers, crippling the control plane's ability to make routing decisions. However, in ProtoGENI, the data plane and control plane are built differently from the Internet. Typically, both planes in ProtoGENI use the same physical media but the data plane is a virtualized network, while these two planes are separated in terms of their traffic but are over the same physical Internet.

The impact of GENI's uniqueness on its security has raised significant attention. A good reference on GENI security can be found through the Workshop on GENI and Security [16]. Two questions related to this paper are: what is GENI security, and what security support services GENI must provide as a federated network? A wider effort deals with the security of GENI, including resource management, monitoring, privacy,

architecture design, and so on. Security requirements on identification, credencialing, delegation, accountability, privacy, and human and policy aspects are discussed. In addition, different requirements from the stakeholders and the users are discussed. The current GENI security architecture and security related development projects largely reflect the outcomes of the workshop [9]. In Sections II-B and III, we have introduced its approach and threat model. The work presented in this paper also helps achieve one of the goals of the workshop "to ensure that the security issues are considered properly during its development."

There are several security-related projects that contribute to achieve the security goals of GENI. GENI security architecture is described in the Toolkit project [3], [9]. It includes the threat model for GENI. Our work provides a proof-of-concept study for the threat model. Moreover, we perform our experiments with new categories of internal and external attacks. This helps the developing team to prioritize their defense. The overall safeguards are realized through the GENI meta-operations center (GMOC) and several monitoring mechanisms. GMOC's role in GENI security is emergency stop. It manages and coordinates the stop and/or containment of GENI resources among all GENI projects in the case of an urgent request, such as the incidents of interference or resource exhaustion caused either on purpose or accidentally. The HiveMind project [6] uses software agents to observe and collect communications traffic in and out of a host to detect the effect of network-borne attacks.

There are several measurement tools to help users to better understand their experiment performance and results. Some of them also help administrators to detect abnormal experiment traffic. GMOC will be ready to receive such feedback in the decision of emergency stop. Each of these tools offers unique features for the users to select based on their need, for example, MeasurementSystem [18] (database attached), LAMP [19] (extensible storage), ScalableMonitoring [20] (active measurement), Instools [21] (passive measurements), and OnTimeMeasure [22] (a large set of tools). ScalableMonitoring and OnTimeMeasure include anomaly detection. The ShadowNet [17] is another tool that provides meso-scale measurement on a per-slice basis. A user will be allowed to control its measurement infrastructure at backbone routers. However, the purpose of the work presented in this paper is different. Also, it is important to perform the experiment-based vulnerability exploration so that our findings can provide feedback to the development teams during the current development stage. Our future work can consider the monitoring capability of GENI.

X. Conclusion

In this paper, we described the ProtoGENI threat model and identified four broad classes of attacks that may challenge ProtoGENI security. With the opportunity of both being an internal attacker and an external user, we performed several proof-of-concept experiments to validate the hypothesis of a few vulnerabilities. For each of the four classes, we analyzed the feasibility according to the ProtoGENI system architecture and described our findings. The results indicated potential vulnerabilities in ProtoGENI. Our experiments are not sophisticated, but they are effective, partly because this is still the development stage. This paper included potential prevention and improvement approaches. More important, our follow-up experiments showed that solutions were deployed. The presented work suggested that it was necessary to perform further analysis on the current developments and to investigate overall GENI security.

References

[1] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar, "An integrated experimental environment for distributed systems and networks," in *Proc. 5th Symp. OSDI*, Dec. 2002, pp. 255–270.

[2] GENI Project Office and BBN Technologies. (2010, Jun. 3). *GENI Global Environment for Network Innovations Spiral 2 Overview* [Online]. Available: http://groups.geni.net/geni/attachment/wiki/SpiralTwo/GENIS2Ovrvw060310.pdf

[3] GENI Project Office and BBN Technologies. (2010, Mar. 15). *GENI Global Environment for Network Innovations Spiral 2 Security Plan* [Online]. Available: http://groups.geni.net/geni/wiki/SpiralTwoSecurityPlans

[4] ProtoGENI. (2009, Nov.) [Online]. Available: http://www.protogeni.net/trac/protogeni

[5] X. Hong, F. Hu, and Y. Xiao. (2009, Sep.). *GENI Spiral Two project: GENI Experiments for Traffic Capture Capabilities and Security Requirement Analysis* [Online]. Available: http://groups.geni.net/geni/wiki/ExptsSecurityAnalysis.

[6] S. Peisert. (2009, Sep.). *GENI Spiral Two Project: The Hive Mind: Applying a Distributed Security Sensor Network to GENI* [Online]. Available: http://groups.geni.net/geni/wiki/HiveMind

[7] *GENI: Exploring Networks of the Future*. (2009, Sep.) [Online]. Available: http://www.geni.net.

[8] T. Benzel, R. Branden, D. Kim, C. Neuman, A. Joseph, K. Sklower, R. Ostrenga, and S. Schwab, "Design, deployment, and use of the DETER testbed," in *Proc. DETER Community Workshop Cyber Security Experimentation Test*, Aug. 2007, pp. 1–8.

[9] *GENI Security Architecture*. (2009, Oct.) [Online]. Available: http://groups.geni.net/geni/wiki/GENISecurity

[10] D. Li and X. Hong, "Practical exploitation on system vulnerability of protoGENI," in *Proc. 49th ACM Southeast Conf.*, Mar. 2011, pp. 104–114.

[11] Netwox. (2010, Feb.) [Online]. Available: http://ntwox.sourceforge.net/

[12] S. Schwab. (2009, Sep.). *GENI Spiral Two Project: Distributed Identity and Authorization Mechanisms* [Online]. Available: http://groups.geni.net/geni/wiki/ABAC.

[13] D. Li, X. Hong, and J. Bowman, "Evaluation of security vulnerabilities by using ProtoGENI as a launchpad," in *Proc. IEEE Globecom*, Dec. 2011, pp. 5–9.

[14] J. Samuel, N. Mathewson, J. Cappos, and R. Dingledine, "Survivable key compromise in software update systems," in *Proc. ACM Comput. Commun. Security Conf.*, Oct. 2010, pp. 61–72.

[15] *GMOC, GENI Meta-Operations Center*. (2010, May) [Online]. Available: http://groups.geni.net/geni/wiki/GENIMetaOps.

[16] M. Bishop. (2009, Aug. 15). *Report on the Workshop on GENI and Security* [Online]. Available: http://seclab.cs.ucdavis.edu/meetings/genisec/

[17] *ShadowNet: A ShadowBox-Based ProtoGENI Instrumentation and Measurement Infrastructure*. (2011, Mar.) [Online]. Available: http://groups.geni.net/geni/wiki/Shadow

[18] *MeasurementSystem: Instrumentation and Measurement for GENI*. (2011, Mar.) [Online]. Available: http://groups.geni.net/geni/wiki/MeasurementSystem

[19] *Leveraging and Abstracting Measurements With perfSONAR (LAMP)*. (2011, Mar.) [Online]. Available: http://groups.geni.net/geni/wiki/LAMP

[20] *Scalable, Extensible, and Safe Monitoring of GENI*. (2011, Mar.) [Online]. Available: http://groups.geni.net/geni/wiki/ScalableMonitoring

[21] *INSTOOLS: Instrumentation Tools for a GENI Prototype*. (2011, Mar.) [Online]. Available: http://groups.geni.net/geni/wiki/InstrumentationTools

[22] *OnTimeMeasue: Centralized and Distributed Measurement Orchestration Software*. (2011, Mar.) [Online]. Available: https://gmoc-db.grnoc.iu.edu

[23] M. Schuchard, A. Mohaisen, D. Kune, N. Hopper, Y. Kim, and E. Vasserman, "Losing control of the internet: using the data plane to attack the control plane," in *Proc. CCS*, Oct. 2010, pp. 726–728.

[24] *Taiwan Information Security Center (TestbedTWIST)*. (2011, Nov.) [Online]. Available: http://testbed.ncku.edu.tw

Dawei Li received the B.S. degree in software engineering from Beihang University, Beijing, China, in 2009, and the M.S. degree in computer science from the University of Alabama, Tuscaloosa, in 2011. He is currently pursuing the Ph.D. degree in computer science with the Department of Computer Science and Engineering, Lehigh University, Bethlehem, PA.

His current research interests include information systems, cloud computing, and mobile software testing.

Xiaoyan Hong (M'00) received the Ph.D. degree in computer science from the University of California, Los Angeles, in 2003.

She is currently an Associate Professor with the Department of Computer Science, University of Alabama, Tuscaloosa. Her current research interests include mobile and wireless networks, challenged networks, vehicle networks, mobility modeling, and the future Internet.

Darwin Witt (S'09) received the B.S. degree in computer science from the University of Alabama, Tuscaloosa, in 2012. He is currently pursuing the M.S. degree with the Human Centered Design and Engineering Program, University of Washington, Seattle.

His current research interests include the design and engineering of complex socio-technological systems, combining aspects of human–computer interaction, interface design, and systems theory to develop a deeper understanding of how to design technological systems that could meet the needs of their sociological and technological environments.