Explore ProtoGENI Security Problems From Experimentation*

Dawei Li, Xiaoyan Hong
Department of Computer Science
The University of Alabama
Tuscaloosa, Alabama, USA
dli11@crimson.ua.edu, hxy@cs.ua.edu

***Abstract*—ProtoGENI is a prototype implementation and deployment of the Global Environment for Network Innovations (GENI) which is a unique virtual laboratory for at-scale networking experimentation exploring future Internets. The successful development of ProtoGENI has to consider security problems from the design and prototyping stages, such as security communication and user authentication. However, in many cases, security issues cannot be found until the real experimentation. In this paper, we introduce some of our efforts in exploring the security vulnerabilities in ProtoGENI from an experimenter's viewpoint. We also analyze the potential causes of these problems. Some suggestions are given to improve the ProtoGENI development.**

Keywords-future internet; ProtoGENI; GENI; control framework; virtualization; security

I. INTRODUCTION

What will be the Internet like in the future? It has been a prevailing research issue since the current Internet has met inevitable severe problems caused by its design defect for scale expansion. Network Scientists have published many papers on this open topic and some of the papers are rather attractive theoretically. However, researchers are always facing a challenging situation when they want to test their ideas as there is not a currently available experiment environment with an Internet scale which is an obstacle for the future internet development.

The Global Environment for Network Innovations (GENI) [2] is a proposed facility for at-scale networking experimentation as a virtual laboratory. According to expectation, GENI will be a huge network research testbed in which abundant resources (PCs, routers, switchers, wireless devices etc.) are geographically distributed. A GENI user can try his ideas using this testbed with other researchers simultaneously.

The development of GENI itself is a great challenge as it comes up with interesting research areas of network science and engineering. Up to now, four research clusters are involved with different aims. ProtoGENI is the control framework for "Cluster C" of the GENI effort led by Flux research group at the University of Utah [3]. ProtoGENI is mainly derived from their previous network testbed Emulab, which gives researchers a wide range of environments to test their systems. The control framework defines the policies of user authentication, resource allocation and communication among different parties.

Security is an important issue for the development of GENI and ProtoGENI control framework which should be considered at the beginning of the system design. Developers from the University of Utah have already shown a well-designed testbed with security communication channel and user authentication. A working group from SPARTA Inc. [5] is prototyping an Attributed–Based Access Control extensions that allow different security mechanisms to share security information within a single slice-based control framework and eventually within federation of multiple control framework. Another project from University of California, Davis [10] is applying a distributed security sensor network to GENI. Specifically, they will develop prototypes for security monitoring, evaluating and reporting software that could be useful to both GENI experimenters and GENI operations.

However, the experimenters or purposeful users of GENI can experience different security risks or pose security threats when vulnerabilities exist. A purposeful and meaningful system security test is required for a deliverable and powerful network research testbed. Some experiment efforts have been reported in our GENI Spiral Two project [6]. In this paper, we describe more experiments exploring a few possible security weaknesses as an experimenter. We introduce the purpose of our experiment and analyze the results. Possible suggestions are given to improve the system. Despite the vulnerabilities we experimented, the purpose of these experiments is to help build a secure laboratory for network innovations. The suggestions based on the experiments have been informed to the development team. Noticeably, some issues that we study here pertain to the current developing version. With the rapid pace of ProtoGENI development, the security issues mentioned in this paper could be solved.

The rest of the paper is organized as follows. Section II gives brief background information and useful concepts of ProtoGENI. Section III introduces our experiment tools. Section IV presents our work investigating the interactions at runtime with ProtoGENI control framework. In Section V, we

* This work is supported in part by BBN/NSF contract project #1783.

detail our experiments inside the ProtoGENI resources allocation. We explored the special security issues in wireless experiment in Section VI. At last in Section VII, we conclude this paper and discuss our future work.

II. PROTOGENI CONTROL FRAMEWORK

ProtoGENI is a prototype implementation and deployment of GENI under development led by Flux research group at the University of Utah [3]. There are some key concepts help to understand the principles of how ProtoGENI and GENI work.

ClearingHouse (CH): Center for registration;

Component Manager (CM): Resource provider;

Slice: Container for resources;

Slice Authority (SA): Users pass a certificate to authenticate themselves to register a slice;

Sliver: Computing resources granted to you inside of your specified slice;

RSpec: Mechanism used for advertising, requesting and describing the resources;

Vnode: User may wish to create a sliver on a shared node which splits a physical node using virtualization. Current ProtoGENI realize Vnode through OpenVZ.

All ProtoGENI authorities (CH, AM and SA) present an XMLRPC interface [4] over HTTP and SSL. And all the user requests are made via a URL register within the clearinghouse for each of the services. A registered user can interact with these XMLRPC servers using the python code provided by the official ProtoGENI wiki site. The experiment can be created with steps in Figure 1:

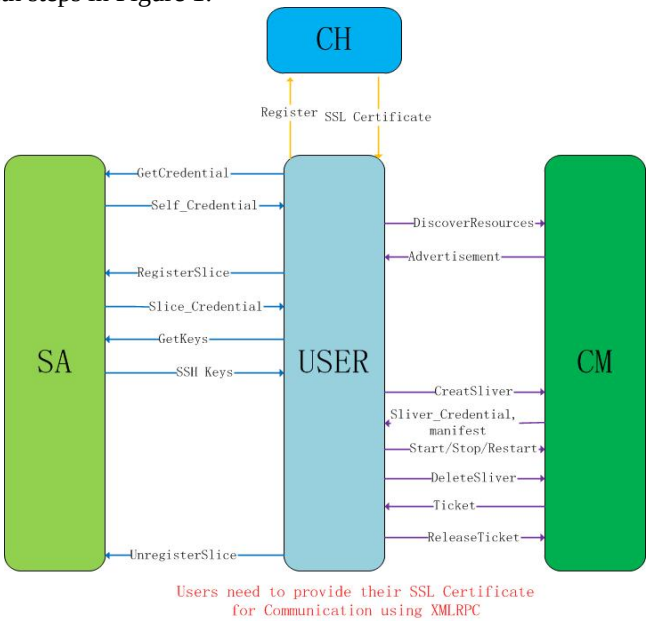


Figure 1. Steps for Experiment in ProtoGENI.

ProtoGENI allows only SSH login to the nodes the user acquires through the above steps. Users have to upload their SSH public keys to the slice authority. When trying to login to a Vnode, user has to SSH to a correct port different from the default port number 22 (the port number can be found in manifest).

III. EXPERIMENT TOOLS

We use common network testing tools for our experiments, for examples, *ping* and *Iperf*. Ping is a computer network tool to test the reachability of a destination host and the round-trip time by sending ICMP echo request packets and waiting for the respond. Iperf is a popular network testing tool that can create TCP and UDP data streams and measure the throughput of a network. Iperf allow user to set various parameters to meet the requirement of network testing. On the other hand, these tools are also included in integrated instrumentation and measurement services of GENI to experimenters [7][8].

For some of the experiments, we adopted "netwox", an open source tool to sniff and spoof network packets of all network layers. The netwox includes a tool set and it uses different numbers to represent different network tools.

In addition, we have developed several automated tools for our experiments with Python. Those tools can be divided into two parts.

A. *RSpec Generating Tool*

We can use this tool to generate RSpec of certain topology types quickly. There are 4 types of topology can be created: line topology, ring topology, arbitrary topology and random topology. Nodes type can also be chosen as Vnodes or normal nodes. User can also choose to install Iperf in the nodes on demand.

B. *Automated Experiment Tools*

We developed two tools for automated experiment creation. The first tool AuSlice.py can help user register multiple slices at the same time. Another tool AuSliver.py can be used to create slivers simultaneously on previously registered slices using the AuSlice.py tool.

In the next two sections, we describe our experiments in two categories: one is experimenter's interaction with the control framework. It usually involves using provided test scripts. The other exploits the virtualization mechanism inside the ProtoGENI where resource allocation is concerned.

IV. EXPERIMENTER'S RUNTIME INTERRACTION WITH PROTOGENI CONTROL FRAMEWORK

When a user of ProtoGENI performs experiments, he needs to use the python scripts provided by the developer to use the APIs with XMLRPC over http and SSL (users could write their own scripts with a language he prefers as long as it supports XMPRPC). As is shown in the ProtoGENI tutorial [4], a user follows a series of steps to do experiments with the provided scripts. We analyzed these steps and also a few test scripts, and performed related experiments to explore potential vulnerability. One of the purposes is to find the weakness in

handshake procedure (like TCP SYN attack). The details are described below with our findings.

A. Getting Ready Phase

SSL Certificate:

The test script will look for the SSL certificate and pass-phrase in $HOME/.ssl/. For most of the users' convenience at this Spiral 2 phase, the code will save the certificate and pass-phrase in $HOME/.ssl/ other than using a command line argument.

Security issues of SSL Certificate: The location makes it easier for being stolen or tampered with. If so, the attacker can obtain all the authorities.

SSH Keys:

This is a step that could incur similar problems as to the SSL certificates. SSH keys are also saved in local machine with a well-known location. Potential problems see the described on SSL certificates.

Security issues of SSH Keys: If stolen, attackers can access to the experimental nodes being used by legal users. From these nodes, more security attacks could be performed.

B. Using the Test Scripts Phase

Code Analysis:

For all the test scripts provided by the developer, they will all execute the test-common.py file first. This file defines the do_method which will be used by all other scripts to call the XMLRPC server over http and SSL. This file also helps to analysis the arguments in user's commands. The file is the core of all the scripts, so the attacker can just easily make change to the test-common.py such as adding a joking print to the code lines of printing "all the resources are busy, please try later". This will be an easy way to confuse the user.

Security issues: For the test-common.py code, once it is changed can affect all the scripts.

Create Sliver:

We use our automated tool to perform possible DoS attack. First we registered several slices with the AuSlice.py tool. Then we create multiple slivers using AuSliver.py. The system works well for experiment creation. However when multiple slivers are created at the same time, the resources acquired for the extra slivers (more than one) cannot be logged into with SSH.

Possible problem: When multiple slivers are created, the SSH public key may not be passed to the all resources properly at the same time.

C. Flash Interface

ProtoGENI [4] now allow user to create slices and slivers with a flash interface. An authenticated user can download his/her SSL certificate from user profile page on the Utah Emulab and save the certificate into the web browser. Then the user can create experiment using the browser.

Security Issues: The flash interface really provides a convenient way for researchers to do experiment with ProtoGENI facilities. However, once the SSL certificate is imported into the web browser, any user can do experiment using this particular browser in the case that the owner of the web browser (authenticated user) leaves the operating system unlocked as there is no further identity check before a user can get a full control as an authenticated ProtoGENI user.

Suggestions: The flash interface should provide a further check of the users' identity before he can create a slice using the interface immediately.

V. EXPERIMENTS WTH RESOURCE ALLOCATION

In experimentation, the security issues can also relate to the ProtoGENI architectural building blocks in the virtualization.

A. Isolation of Slices

A slice provides the networked resources for an experiment. Physically, one slice shares hardware with other slices through virtualization. From control framework point of view, slices are totally separated, isolated from each other. Each one is contained. Control framework should not allow nodes belonging to different slices communicate with each other even though they are created by the same user. Our experiment tries to test the isolation function of the control framework.

Our first set of experiments has shown a case that cross-slice communications is possible. After reporting the issue (see also in our ExptsSec: S2.c document), and obtaining the feedback on a bug fix for this issue, we conducted the same set of experiments again for validation. Here we describe the two sets of experiments.

A.1. Initial set of experiments

Experiment Setup:

In this experiment, three slices with slice names **test1**, **test2** and **test3** are created with the same topology of two Vnodes and a link of bandwidth 100Mb/s as follows:

shared1 ---- shared2

Tool *Iperf* is installed on both *shared1* and *shared 2*. All the resources acquired are summarized in TABEL I.

TABLE I. RESOURES OF THE SLICES

Node Name	Slice Name	Hostname	Port Number
shared1	test1	pc175.emulab.net	32058
shared2	test1	pc172.emulab.net	32058
shared1	test2	pc172.emulab.net	32570
shared2	test2	pc175.emulab.net	32570
shared1	test3	pc263.emulab.net	33850
shared2	test3	pc102.emulab.net	33850

Experiment Steps:

First, only one *Iperf* server is running in slice *test1* at the node named *shared1* with the command:

iperf –s

Second, at nodes *shared2* of both slices **test1** and **test2**, we try to connect to the server *shared1* with the following command:

iperf –c shared1

Then we observed from the screen of the Iperf server (*shared1* at *test1*) that both of the clients connected to the server even though they are not at the same slice, i.e. the nodes can communicate across slices! In Figure 2, we illustrate our experiment with the screen captures of the four nodes. The left sides are the Iperf server and client in *test1* and the right side are those in *test2*. Figure 2 shows that the server *shared1* in *test1* (slice1892) (the upper left terminal) is connected by clients of ports in the sequence (numbers in the red circle):

43589, 53256, 53257, 43590

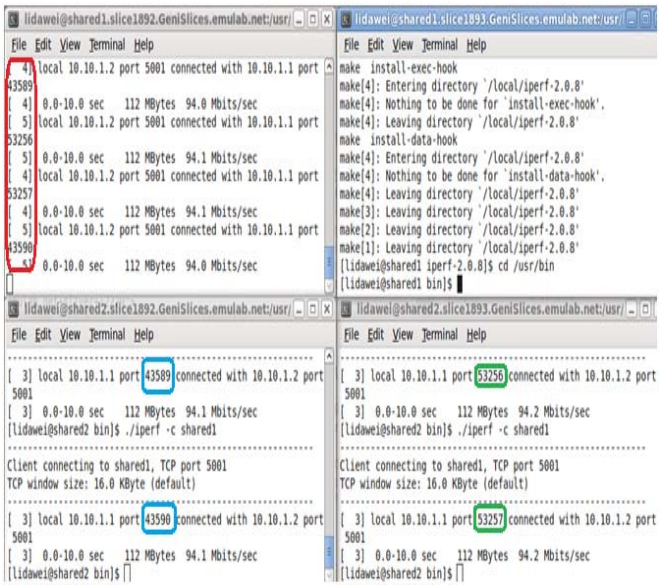


Figure 2. Cross-slice Communication for Test1 and Test2

The first client (*shared2* in *tests1* (slice1892) the left lower terminal) connected to server with sequence (numbers in the blue circle):

… 43589, 43590 …

The second client (*shared2* in *test2* (slice1893) the right lower terminal) connected to server with ports of the sequence (numbers in green circle):

… 53256, 53257…

However, it seems that the problem could due to the fact that the two slices share the same physical resources (pc175 and pc172). So we performed the same experiment with **test1** and **test3**. We obtained the same result as shown in Figure 3.

Further, we tried other possibilities including changing Vnode to a normal node, connecting to the Iperf server with IP address and using different node names for different slices (no matter whether it is a Vnode or a normal node). The results are summarized in TABLE II. It shows that there is only one setting that the cross-slice communication can occur.

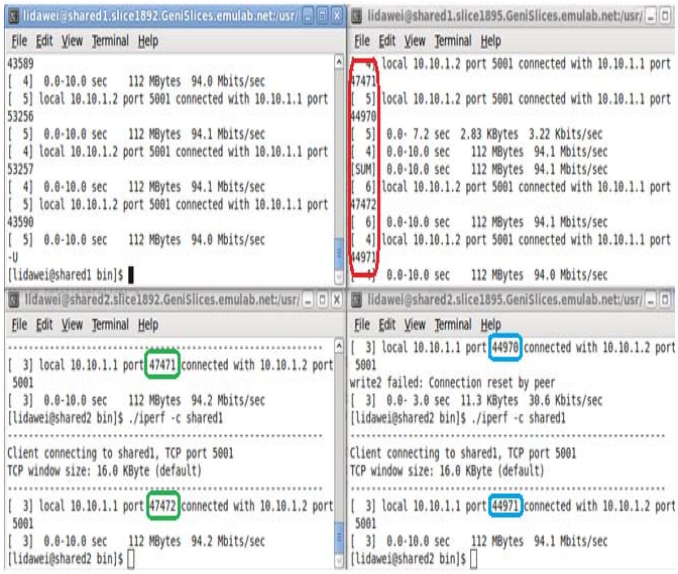


Figure 3. Cross-slice Communication for Test1 and Test3

TABLE II. CROSS-SLICE EXPERIMENTS RESULT

Vnodes or Normal nodes	Iperf to the server with node name or IP address	Same node name or different node name	Result
Vnodes	Node name	Same name	√
Vnodes	IP address	Same name	×
Vnodes	Node name	Different name	×
Normal nodes	Node name	Same name	×
Normal nodes	IP address	Same name	×
Normal nodes	Node name	Different name	×

Experiment Analysis:

The result of this experiment shows that the cross-slice communication can really happen under ProtoGENI control framework when the nodes are Vnodes with the same node name and Iperf to a server through the node name. This may be caused by the control framework implementation of Vnodes and the mapping of the names. And the way for encapsulating the shared Vnode has a potential drawback.

A.2. Reexamination set of experiments

Based on our experiment report, the developer team at The University of Utah has fixed the issue described in previous subsection.

We have recently re-examined this cross-slice traffic issue. We found that the previous problem has been solved. We conducted the same experiment for verification. The experiment setup is summarized in TABLE III.

Reexamine Steps:

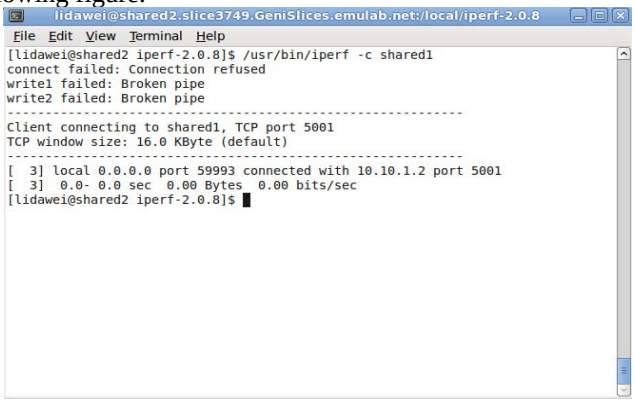
We reexamined both scenarios appeared in the cross-slice experiment.

TABLE III. RESOURCES OF REEXAMINE

Node Name	Slice Name	Hostname	Port Number
shared1	test1	pc459.emulab.net	38714
shared2	test1	pc511.emulab.net	38714
shared1	test2	pc413.emulab.net	38970
shared2	test2	pc510.emulab.net	38970
shared1	test3	pc511.emulab.net	43578
shared2	test3	pc459.emulab.net	43578

1. Slices share different physical resources

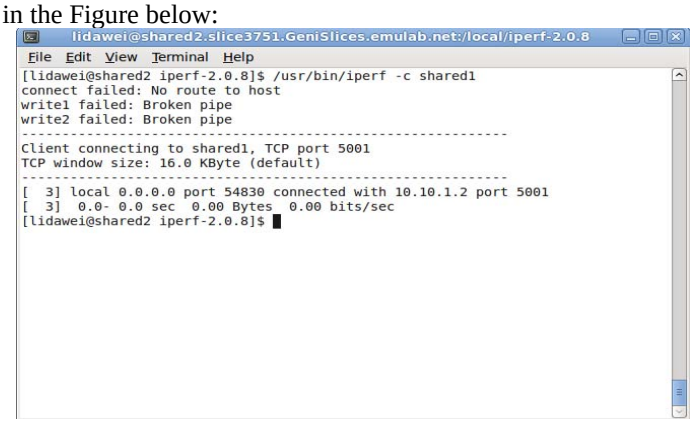
From TABLE III, we can see that the slice **test1** (pc459.emulab.net and pc511.emulab.net) and slice **test2** (pc413.emulab.net and pc510.emulab.net) belonged to different physical nodes. So in *shared1* of slice *test1*, we ran the iperf server; in *shared2* of slice *test2*, we ran the iperf client to connect iperf server *shared1*. The result is in the following figure.



We can see from this figure that there is no traffic from *shared2* in *slice test2* to *shared1* in slice *test1*.

2. Slices share the same physical resources

From TABLE III, it is obviously that nodes in slice **test1** and in slice **test3** belonged to the same physical nodes: pc459.emulab.net and pc511.emulab.net. So in *shared1* of slice *test1*, we ran the iperf server; in *shared2* of slice *test3*, we ran the iperf client to connect iperf server *shared1*. The result is in the Figure below:



Still in this figure, there is no traffic or packet can be send from *shared2* to the iperf server in a different even they are in the same physical node!

Thus, in both cases, be it using shared nodes or not, we validated that the cross-slice traffic does not appear.

B. Nonexclusive use of resources

ProtoGENI user can specify a bandwidth of the link between two nodes. However the link between two Vnodes (sharing the same physical node) is in fact using a loopback (bridged) method as mentioned in [1]. So the link between two Vnodes or link between a Vnode and a normal node may reveal different performance characters.

Experiment Setup:

This experiment has two Vnodes and one normal node with following topology:

shared1 ---- shared2 ---- geni0

The node shared1 (hostname: pc102.emulab.net & port number 31290) and shared2 (hostname: pc263.emulab.net & port number 31290) are Vnodes and geni0 (hostname: pc204.emulab.net & port number 22 as default SSH port number) is a normal node. The link bandwidth between the Vnodes and between shared 2 and geni0 are both 100Mb/s.

Experiment Steps:

First, we try to ping from *shared1* to *shared2* and from *shared2* to *shared1*. We have the following result as shown in Figure 4 and Figure 5. The two results show that the delay variances are obvious.

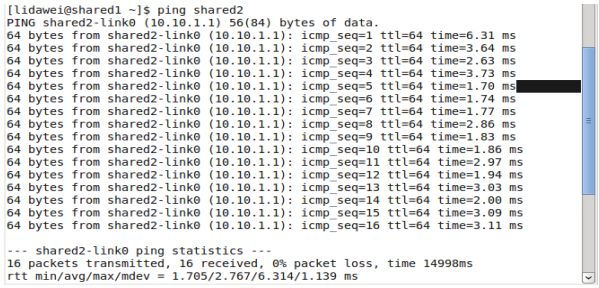


Figure 4. Ping From *shared1* to *shared2*

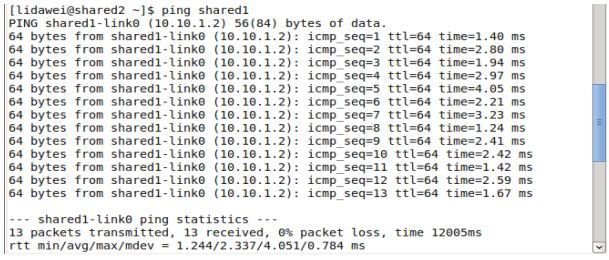


Figure 5. Ping From *shared2* to *shared1*

Then we ping from shared2 to geni0 and from geni0 to shared2. The results are given in Figure 6 and Figure 7. Figure 6 shows that the delay variances from a Vnode to a normal node are mostly small. The initial long delay exists in the many repeated experiments.

Experiment Analysis:

From the results of this experiment we see that when pinging from a normal node to a Vnode or ping between Vnodes, the round-trip time is not stable. This may indicate that the network is not reliable enough for a real network experiment.

Suggestions: the large delay variance at the Vnodes may be because of the current virtualization technology OpenVZ that ProtoGENI is using. Developers may consider further potential defects when applying to a large scale system.

C. Network Stability and Stress Test

This consideration relate to network quality. Unwanted network quality will be a potential problem that affects experiment results which may as severe as security problems. We perform stress tests to see if the recourse usage is confined to its specification, to see if other sliver creations could be affected. The software Iperf (version 2.08) is equipped with some parameters to test network stability and for stress test.

Experiment Setup:

In the experiment, we create a sliver with a topology:

geni1 ---- geni2 ---- geni3

Iperf is installed at geni1 and geni3.

```
[lidawei@shared2 ~]$ ping geni0
PING geni0-link1 (10.10.2.2) 56(84) bytes of data.
64 bytes from geni0-link1 (10.10.2.2): icmp_seq=1 ttl=64 time=3.38 ms
64 bytes from geni0-link1 (10.10.2.2): icmp_seq=2 ttl=64 time=1.13 ms
64 bytes from geni0-link1 (10.10.2.2): icmp_seq=3 ttl=64 time=1.20 ms
64 bytes from geni0-link1 (10.10.2.2): icmp_seq=4 ttl=64 time=1.12 ms
64 bytes from geni0-link1 (10.10.2.2): icmp_seq=5 ttl=64 time=1.19 ms
64 bytes from geni0-link1 (10.10.2.2): icmp_seq=6 ttl=64 time=1.21 ms
64 bytes from geni0-link1 (10.10.2.2): icmp_seq=7 ttl=64 time=1.23 ms
64 bytes from geni0-link1 (10.10.2.2): icmp_seq=8 ttl=64 time=1.18 ms
64 bytes from geni0-link1 (10.10.2.2): icmp_seq=9 ttl=64 time=1.19 ms
64 bytes from geni0-link1 (10.10.2.2): icmp_seq=10 ttl=64 time=1.27 ms
64 bytes from geni0-link1 (10.10.2.2): icmp_seq=11 ttl=64 time=1.19 ms
64 bytes from geni0-link1 (10.10.2.2): icmp_seq=12 ttl=64 time=1.23 ms

--- geni0-link1 ping statistics ---
12 packets transmitted, 12 received, 0% packet loss, time 10999ms
rtt min/avg/max/mdev = 1.123/1.382/3.385/0.606 ms
```

Figure 6. Ping From *shared2* to *geni0*

```
[lidawei@geni0 ~]$ ping shared2
PING shared2-link1 (10.10.2.1) 56(84) bytes of data.
64 bytes from shared2-link1 (10.10.2.1): icmp_seq=1 ttl=64 time=1.15 ms
64 bytes from shared2-link1 (10.10.2.1): icmp_seq=2 ttl=64 time=1.52 ms
64 bytes from shared2-link1 (10.10.2.1): icmp_seq=3 ttl=64 time=1.64 ms
64 bytes from shared2-link1 (10.10.2.1): icmp_seq=4 ttl=64 time=1.64 ms
64 bytes from shared2-link1 (10.10.2.1): icmp_seq=5 ttl=64 time=1.51 ms
64 bytes from shared2-link1 (10.10.2.1): icmp_seq=6 ttl=64 time=0.500 ms
64 bytes from shared2-link1 (10.10.2.1): icmp_seq=7 ttl=64 time=1.48 ms
64 bytes from shared2-link1 (10.10.2.1): icmp_seq=8 ttl=64 time=1.36 ms
64 bytes from shared2-link1 (10.10.2.1): icmp_seq=9 ttl=64 time=1.35 ms
64 bytes from shared2-link1 (10.10.2.1): icmp_seq=10 ttl=64 time=1.35 ms
64 bytes from shared2-link1 (10.10.2.1): icmp_seq=11 ttl=64 time=0.442 ms
64 bytes from shared2-link1 (10.10.2.1): icmp_seq=12 ttl=64 time=1.41 ms
64 bytes from shared2-link1 (10.10.2.1): icmp_seq=13 ttl=64 time=1.41 ms
64 bytes from shared2-link1 (10.10.2.1): icmp_seq=14 ttl=64 time=1.41 ms
64 bytes from shared2-link1 (10.10.2.1): icmp_seq=15 ttl=64 time=0.407 ms

--- shared2-link1 ping statistics ---
15 packets transmitted, 15 received, 0% packet loss, time 14018ms
rtt min/avg/max/mdev = 0.407/1.242/1.649/0.416 ms
```

Figure 7. Ping From *geni0* to *shared2*

Experiment Steps:

First, we ran the command iperf –s in geni1 to start the server.

Then we ran the command iperf -c geni1 -t 120 -i 10 in geni3 to connect to the server geni1. Here the transmission time is set to 120s and interval to 10s. The default window size is 16KB for TCP. Result is given in Figure 8. The result shows that the transmission rate is stable at around 94.0 Mbits/sec.

Further we add the -P * option of Iperf to the above experiment. -P * is used to simulate * multi-threads to connect the server. We used window size 128k. The result shows that the network works well for as many as possible threads connecting the server together. (The default maximum upper bound is 253 threads, and when the * is raised to 254, it will return a thread creation failure).

Experiment Analysis:

In the Iperf client, the Linux terminal will show the transmission rate of each thread and the total rate of all the threads. As the number of threads increases, the transmission rate of each thread decreases, but the total rate keeps stable for a rate of around 94.0 Mbits/sec.

From these results, we can see that the network under ProtoGENI control framework performs correctly in separating the network traffic flows when we use Iperf to test it. So the network quality here will not be an obstacle for researchers to carry out their experiments.

VI. EXPERIMENTS WITH WIRELESS TESTBED

Different from experiments of wired connection from an experimenter in GENI, the open nature wireless media makes it easier for one experimenter to intervene others' experiments. Though security and privacy policies are clearly given to the experimenters, however, to an uninformed user or a purposeful user, the listed policy items are vulnerabilities.

A. Packet Sniff

The attacker who has a wireless node in ProtoGENI can easily sniff a packet in the air from other wireless experiments with netwox. With the sniffed packet, attackers will get enough network information of both the sender and receiver such as IP address and MAC address which can be used to launch network attack.

Experiment Setup:

In this experiment, two slices with slice names **experiment1** and **experiment2** are created with the same topology of two wireless nodes with 802.11g standard:

nodew1 ---- nodew2

For experiment1, we have pc39 for nodew1 and pc28 for nodew2. For experiment2, we have pc35 for nodew1 and pc27 for nodew2.

The four nodes are located in the following physical positions (Figure 8).

Iperf is installed in both nodes of experiment1. "netwox" is installed in nodew1 of experiment2 which is pc35 in the above. Both of the experiments choose channel 14.

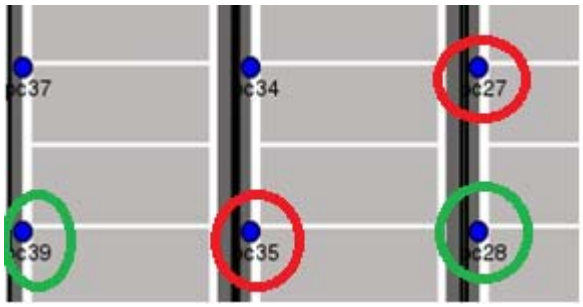


Figure 8. Physical Location

Experiment Steps:

First, we have iperf server running on nodew2 of experiment1 and iperf client running on nodew1 of the same slice to connect the iperf server.

Then, we use the No. 7 tool of netwox to sniff the TCP packet of the communication in experiment1. So in nodew1 of experiment2, the following command is used to sniff the TCP packet of the open air:

netwox 7 –d ath0 –t

The "ath0" is the wireless interface used for sniffing and "-t" is used to sniff TCP packet. The packet sniffed is shown in Figure 9.

```
Ethernet
| 00:17:9A:08:BE:99->00:17:9A:08:C1:CA type:0x0800
IP
|version|  ihl  |      tos      |            totlen
|___4___|___5___|____0x00=0_____|_____0x0034=52_____|
|            id                |r|D|M|     offsetfrag
|_____0xBCAF=48303_____|0|1|0|_____0x0000=0_____|
|      ttl      |   protocol    |           checksum
|____0x40=64____|____0x06=6_____|_____0x680E_____|
|                          source
|_____________________10.1.1.3_____________________|
|                        destination
|_____________________10.1.1.2_____________________|
TCP
|          source port          |       destination port
|_____0x1389=5001_____|_____0x0D1D=3357_____|
|                          seqnum
|_____________0x4EB76E3E=1320644158_____________|
|                          acknum
|_____________0x4E8F2D3A=1318006074_____________|
| doff  |r|r|r|r|C|E|U|A|P|R|S|F|          window
|___8___|0|0|0|0|0|0|0|1|0|0|0|0|_____0x3014=12308_____|
|          checksum            |          urgptr
|_____0x2D6F=11631_____|_____0x0000=0_____|
TCPOPTS
| noop
| noop
| timestamp : val=382329 echoreply=382026
```

Figure 9. TCP Packet Sniffed

Experiment Analysis:

In this sniffed packet, we get all the network information of both server and client which can be used to perform network attack.

Packet sniff can only be performed when both experiments are using the same channel. However, it is easy to find out which channel is being used by other experiments from Emulab website:

Floor	Channels in Use
3	2,14

Figure 10. Channels in Use

B. ARP Cache Poisoning

With the sniffed packet, we can get both mac address and ip address of both nodes. So it is easy for us to launch the arp cache poisoning with netwox tool No. 33.

Experiment Setup:

In this experiment, we have the same experiment scenario with packet sniff just with different physical nodes.

For experiment1, we have pc39 for nodew1 and pc28 for nodew2. For experiment2, we have pc35 for nodew1 and pc27 for nodew2. Netwox is installed in nodew1 of experiment2.

Experiment Steps:

First, we "ping" from nodew2 to nodew1 in experiment1.

Then, we use the No. 7 tool of netwox to sniff the packet of the communication in experiment1. So in nodew1 of experiment2, the following command is used:

netwox 7 –d ath0

The packet sniffed is shown in Figure 11.

```
Ethernet
| 00:17:9A:C3:65:24->00:17:9A:08:C1:79 type:0x0800
IP
|version|  ihl  |      tos      |            totlen
|___4___|___5___|____0x00=0_____|_____0x0054=84_____|
|            id                |r|D|M|     offsetfrag
|_____0x10AC=4268_____|0|0|0|_____0x0000=0_____|
|      ttl      |   protocol    |           checksum
|____0x40=64____|____0x01=1_____|_____0x53F7_____|
|                          source
|_____________________10.1.1.2_____________________|
|                        destination
|_____________________10.1.1.3_____________________|
ICMP4_echo reply
|     type      |     code      |           checksum
|____0x00=0_____|____0x00=0_____|_____0xC3D0=50128_____|
|            id                |          seqnum
|_____0xFB0E=64270_____|_____0x0017=23_____|
| data: 88f4ce4cf7c4070008090a0b0c0d0e0f1011121314151617181 9a1
|       b1c1d1e1f20212223242526272829 2a2b2c2d2e2f30313233343536
|       37
|
```

Figure 11. "Ping" Packet Sniffed

Then we check the ARP table in nodew1 of experiment1:

```
[lidawei@nodew1 ~]$ sudo arp
Address                  HWtype  HWaddress           Flags Mask            Iface
control-router.emulab.n  ether   00:B0:8E:84:69:34   C                     eth4
nodew2-lan0              ether   00:17:9A:08:C1:79   C                     ath0
```

Figure 12. ARP Cache before Attack

We use the following command to launch attack:

```
[lidawei@nodew1 src]$ sudo /usr/local/bin/netwox 33 -d ath0 -a 0C:0C:0C:0C:0C:0C
 -b 00:17:9A:C3:65:24 -c 2054 -e 2 -f 0C:0C:0C:0C:0C:0C -g "10.1.1.3" -h 00:17:9
A:C3:65:24 -i 10.1.1.2
Ethernet_____________________________________________________.
| 0C:0C:0C:0C:0C:0C->00:17:9A:C3:65:24 type:0x0806            |
|_____________________________________________________________|
ARP Reply____________________________________________________.
| this answer : 0C:0C:0C:0C:0C:0C 10.1.1.3                    |
| is for      : 00:17:9A:C3:65:24 10.1.1.2                    |
|_____________________________________________________________|
```

Figure 13. Launch ARP Cache Poisoning

We check the ARP table again:

```
[lidawei@nodew1 ~]$ arp
Address                  HWtype  HWaddress           Flags Mask            Iface
control-router.emulab.n  ether   00:B0:8E:84:69:34   C                     eth4
nodew2-lan0              ether   0C:0C:0C:0C:0C:0C   C                     ath0
```

Figure 14. ARP Cache after Attack

The ARP table is successfully modified.

Experiment Analysis:

In a sniffed packet, attacker can get the IP address and corresponding MAC address. With this information, attacker can easily do an ARP cache poison attack. The victim is just "innocent".

C. SYN Flooding

We can use "netwox" to perform a syn flooding attack. The following netwox command can be used:

netwox 76 –i "10.1.1.3" –p 20

The "-i" is the ip address to attack and "-p" is the port number to attack.

In nodew2 of experiment 1, we use the "netstat –na" to check the current TCP connection status:

```
Proto RefCnt Flags       Type       State         I-Node Path
unix  2      [ ACC ]     STREAM     LISTENING     9810   /dev/gpmctl
unix  2      [ ACC ]     STREAM     LISTENING     9894   /var/run/dbus/system_bu
s_socket
unix  13     [ ]         DGRAM                    9318   /dev/log
unix  2      [ ]         DGRAM                    8422   @udevd
unix  2      [ ACC ]     STREAM     LISTENING     8410   @kudzu_config_socket
unix  2      [ ACC ]     STREAM     LISTENING     9849   /tmp/.font-unix/fs7100
unix  3      [ ]         STREAM     CONNECTED     11348
unix  3      [ ]         STREAM     CONNECTED     11347
unix  2      [ ]         DGRAM                    10050
unix  3      [ ]         STREAM     CONNECTED     9897
unix  3      [ ]         STREAM     CONNECTED     9896
unix  2      [ ]         DGRAM                    9872
unix  2      [ ]         DGRAM                    9815
unix  2      [ ]         DGRAM                    9788
unix  2      [ ]         DGRAM                    9765
unix  2      [ ]         DGRAM                    9749
unix  2      [ ]         DGRAM                    9679
unix  2      [ ]         DGRAM                    9391
unix  2      [ ]         DGRAM                    9352
unix  2      [ ]         DGRAM                    9335
unix  2      [ ]         DGRAM                    9322
[lidawei@nodew2 ~]$
```

We see that some ports are listening and others are established. Then we execute the netwox command and check the tcp status again:

```
Proto RefCnt Flags       Type       State         I-Node Path
unix  2      [ ACC ]     STREAM     LISTENING     9810   /dev/gpmctl
unix  2      [ ACC ]     STREAM     LISTENING     9894   /var/run/dbus/system_bu
s_socket
unix  13     [ ]         DGRAM                    9318   /dev/log
unix  2      [ ]         DGRAM                    8422   @udevd
unix  2      [ ACC ]     STREAM     LISTENING     8410   @kudzu_config_socket
unix  2      [ ACC ]     STREAM     LISTENING     9849   /tmp/.font-unix/fs7100
unix  3      [ ]         STREAM     CONNECTED     11348
unix  3      [ ]         STREAM     CONNECTED     11347
unix  2      [ ]         DGRAM                    10050
unix  3      [ ]         STREAM     CONNECTED     9897
unix  3      [ ]         STREAM     CONNECTED     9896
unix  2      [ ]         DGRAM                    9872
unix  2      [ ]         DGRAM                    9815
unix  2      [ ]         DGRAM                    9788
unix  2      [ ]         DGRAM                    9765
unix  2      [ ]         DGRAM                    9749
unix  2      [ ]         DGRAM                    9679
unix  2      [ ]         DGRAM                    9391
unix  2      [ ]         DGRAM                    9352
unix  2      [ ]         DGRAM                    9335
unix  2      [ ]         DGRAM                    9322
[lidawei@nodew2 ~]$
```

We see that the attack is not successful. This is because most of the Linux operating system has a tcp_mechanism to defense this attack.

However, another interested issue was found here when performing the syn flooding attack. Again, we run iperf server in nodew2 of experiment1 and iperf client in nodew1 to connect the iperf server. Before we run netwox command:

```
[lidawei@nodew2 ~]$ /usr/bin/iperf -s
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  4] local 10.1.1.3 port 5001 connected with 10.1.1.2 port 3827
[  4]  0.0-10.0 sec  30.7 MBytes  25.7 Mbits/sec
[  5] local 10.1.1.3 port 5001 connected with 10.1.1.2 port 3828
[  5]  0.0-10.0 sec  30.8 MBytes  25.7 Mbits/sec
[  4] local 10.1.1.3 port 5001 connected with 10.1.1.2 port 3829
[  4]  0.0-10.0 sec  31.5 MBytes  26.3 Mbits/sec
```

The throughput showed here is around 26Mbits/sec. Then we run the netwox command:

netwox 76 –i "10.1.1.3" –p 20

We get the following result:

```
[lidawei@nodew2 ~]$ /usr/bin/iperf -s
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  4] local 10.1.1.3 port 5001 connected with 10.1.1.2 port 3827
[  4]  0.0-10.0 sec  30.7 MBytes  25.7 Mbits/sec
[  5] local 10.1.1.3 port 5001 connected with 10.1.1.2 port 3828
[  5]  0.0-10.0 sec  30.8 MBytes  25.7 Mbits/sec
[  4] local 10.1.1.3 port 5001 connected with 10.1.1.2 port 3829
[  4]  0.0-10.0 sec  31.5 MBytes  26.3 Mbits/sec
[  5] local 10.1.1.3 port 5001 connected with 10.1.1.2 port 3830
[  5]  0.0-10.0 sec  19.3 MBytes  16.2 Mbits/sec
[  4] local 10.1.1.3 port 5001 connected with 10.1.1.2 port 4328
[  4]  0.0-10.1 sec  19.0 MBytes  15.9 Mbits/sec
[  5] local 10.1.1.3 port 5001 connected with 10.1.1.2 port 4329
[  5]  0.0-10.0 sec  18.6 MBytes  15.6 Mbits/sec
```

The throughput is decreased to around 16Mbits/sec. Here we may think that the throughput is affected because the two experiments are using the same communication channel. Then we run the netwox command:

netwox 76 –i "10.1.1.4" –p 20

The "10.1.1.4" does not exist in the network topology. We get the following result:

```
[lidawei@nodew2 ~]$ /usr/bin/iperf -s
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  4] local 10.1.1.3 port 5001 connected with 10.1.1.2 port 3827
[  4]  0.0-10.0 sec  30.7 MBytes  25.7 Mbits/sec
[  5] local 10.1.1.3 port 5001 connected with 10.1.1.2 port 3828
[  5]  0.0-10.0 sec  30.8 MBytes  25.7 Mbits/sec
[  4] local 10.1.1.3 port 5001 connected with 10.1.1.2 port 3829
[  4]  0.0-10.0 sec  31.5 MBytes  26.3 Mbits/sec
[  5] local 10.1.1.3 port 5001 connected with 10.1.1.2 port 3830
[  5]  0.0-10.0 sec  19.3 MBytes  16.2 Mbits/sec
[  4] local 10.1.1.3 port 5001 connected with 10.1.1.2 port 4328
[  4]  0.0-10.1 sec  19.0 MBytes  15.9 Mbits/sec
[  5] local 10.1.1.3 port 5001 connected with 10.1.1.2 port 4329
[  5]  0.0-10.0 sec  18.6 MBytes  15.6 Mbits/sec
[  4] local 10.1.1.3 port 5001 connected with 10.1.1.2 port 4330
[  4]  0.0-10.0 sec  31.2 MBytes  26.1 Mbits/sec
[  5] local 10.1.1.3 port 5001 connected with 10.1.1.2 port 4331
[  5]  0.0-10.0 sec  30.8 MBytes  25.8 Mbits/sec
[  4] local 10.1.1.3 port 5001 connected with 10.1.1.2 port 4332
[  4]  0.0-10.0 sec  31.2 MBytes  26.1 Mbits/sec
```

The throughput is then increase back to around 26Mbits/sec. We can think the attack is successful not because of the experiments are using the same channel. Further we change the communication channel or experiment 2 to channel 4:

```
Virtual Lan Settings:
ID              Member          Key                 Value
--------------- --------------- ------------------- --------------------
lan0                            channel             4
lan0                            mode                adhoc
```

Then we perform the syn flooding attack again, as we can predict, the attack can still decrease the throughput.

Experiment Analysis:

The syn flooding cannot success in attacking the tcp connection directly because most of the Linux operating system has a tcp_mechanism to defense this attack However, it is still possible to use the sniffed IP address to attack another experiment and decrease the throughput even the attacker is using a different channel. The victim is just "innocent".

D. Resource Scramble

When experimenters want to use some particular resources, the control framework provides a way to discover the resource. In ProtoGENI, user can use XMLRPC to call the resource discover API to find out what resources are available.

```
[lidawei@geni3 iperf-2.0.8]$ iperf -c geni1 -t 120 -i 10
------------------------------------------------------------
Client connecting to geni1, TCP port 5001
TCP window size: 16.0 KByte (default)
------------------------------------------------------------
[  3] local 10.10.2.2 port 32783 connected with 10.10.1.1 port 5001
[  3]  0.0-10.0 sec    112 MBytes  94.2 Mbits/sec
[  3] 10.0-20.0 sec    112 MBytes  93.6 Mbits/sec
[  3] 20.0-30.0 sec    112 MBytes  94.1 Mbits/sec
[  3] 30.0-40.0 sec    112 MBytes  94.0 Mbits/sec
[  3] 40.0-50.0 sec    112 MBytes  94.1 Mbits/sec
[  3] 50.0-60.0 sec    112 MBytes  94.0 Mbits/sec
[  3] 60.0-70.0 sec    112 MBytes  94.0 Mbits/sec
[  3] 70.0-80.0 sec    112 MBytes  94.1 Mbits/sec
[  3] 80.0-90.0 sec    112 MBytes  94.0 Mbits/sec
[  3] 90.0-100.0 sec    112 MBytes  94.1 Mbits/sec
[  3] 100.0-110.0 sec    112 MBytes  94.1 Mbits/sec
[  3] 110.0-120.0 sec    112 MBytes  94.0 Mbits/sec
[  3]  0.0-120.0 sec  1.31 GBytes  94.0 Mbits/sec
[lidawei@geni3 iperf-2.0.8]$
```

Figure 15. ProtoGENI Network Stability Test

For wireless experiments, users can check the floormap of all the wireless nodes to find the free wireless nodes and choose the nodes wanted. However, this mechanism has a significant drawback especially for the wireless that when users find the nodes he wants, he cannot reserve the nodes in time. The user still has to change their RSpec to request the particular nodes. However, at this time the nodes they wanted may already be taken by other users especially that the wireless nodes may be requested by other users (wired topology) with a random resource request even without their "fault".

What is more, when users want to do multiple wireless experiments, they may want to have nodes at particular area. However, user needs to request these resources of different experiments separately, which may give the attacker a chance to perform the "resource scramble".

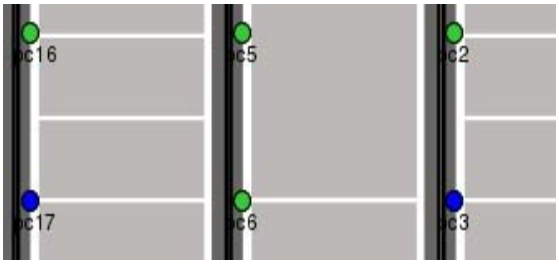


Figure 16. Wireless Nodes Floormap

As shown in the above picture, the user may need pc16 and pc2 for experiment 1; pc5 and pc6 for experiment 2. When the attacker finds someone already reserved pc16 and pc2, he can request pc5 and pc6 on purpose so that the experimenter cannot continue the experiment as the nodes he wanted are no more existing.

Suggestions: When a user wants to reserve some nodes, the ProtoGENI should provide a way to do it on-the-fly in order to avoid later conflicts. This will save the users effort and increase the users' efficiency.

VII. SUMMARY

The work presented in this paper reports our analysis for potential security vulnerability in ProtoGENI from experimentations. We presented our experimentations and analysis in three aspects: runtime interaction with control framework, experiments with ProtoGENI resource allocations using virtualization, and security issues in wireless experiments. We provided a few suggestions according to the results for possible improvements on ProtoGENI security.

Our future work will explore security problems based on investigations through ProtoGENI experiments on at least two aggregates. We will also install our local component manager and analyze security behaviors related to CM. For wireless experiments, we will explore more when security mechanisms such as WPA/WPA2 are applied.

REFERENCES

[1] M. Hibler, R. Ricci, L. Stoller, J. Duerig, S. Guruprasad, T. Stack, K. Webb, and J. Lepreau. "Large-scale Virtualization in the Emulab Network Testbed." In Proc. USENIX Annual Technical Conference, Boston, MA, June 2008.

[2] “GENI Global Environment for Network Innovations Spiral 2 Overview”,http://groups.geni.net/geni/attachment/wiki/SpiralTwo/GENIS2Ovrvw060310.pdf.
[3] “GENI Global Environment for Network Innovations Spiral 2 Security Plan”, http://groups.geni.net/geni/wiki/SpiralTwoSecurityPlans.
[4] “ProtoGENI wiki page”, http://www.protogeni.net/trac/protogeni.
[5] S. Schwab, “GENI Spiral Two Project: Distributed Identity and Authorization Mechanisms”, http://groups.geni.net/geni/wiki/ABAC.
[6] X. Hong, F. Hu, Y. Xiao. “GENI Spiral Two Project: GENI Experiments for Traffic Capture Capabilities and Security Requirement Analysis”, http://groups.geni.net/geni/wiki/ExptsSecurityAnalysis.
[7] INSTOOLS, http://groups.geni.net/geni/wiki/InstrumentationTools.
[8] OnTimeMeasure,http://groups.geni.net/geni/wiki/OnTimeMeasur.
[9] W. Du, T. Daniels, N. Gaubatz, P. Ning, G. Spafford. SEED Project http://www.cis.syr.edu/~wedu/seed/.
[10] S. Peisert. “GENI Spiral Two Project: The Hive Mind: Applying a Distributed Security Sensor Network to GENI “, http://groups.geni.net/geni/wiki/HiveMind