ExptsSec: S2.c. Report on Experimentation Exploiting Vulnerabilities and Validating Vulnerability Hypotheses

Xiaoyan Hong, Fei Hu, Yang Xiao
Jingcheng Gao, Dawei Li, Sneha Rao, Fnu Shalini, Dong Zhang
University of Alabama
July 09, 2010

I. INTRODUCTION

This document reports many experiments conducted following the *security issues and experiments* design document (see the project Milestone#2 deliverables [1]). The experiments show unique ways in approaching the issues listed in [2]. The experiments are not overlapped mostly. A couple experiments are the validations and extensions of the experiments reported in Milestone#2 deliverables [2].

Suggestions are given to improve the development of ProtoGENI in its security from an experimenter's view point. 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 document could be solved.

The rest of the document is organized as follows. Section II presents our work investigating the interactions at runtime with ProtoGENI control framework. Section III gives experiments exploiting ProtoGENI resource allocation. In Section IV, more details on DoS attack to Emulab resources at run-time is described. Section V focuses on attacking authentication and experiments on stolen credentials. Section VI includes more details on port scan attack from inside/outside ProtoGENI nodes. At last in Section VII, we summarize the experiment results and suggestions.

II. EXPERIMENTER'S INTERACTION WITH THE CONTROL FRAMEWORK

A. Run- time security parameters

The experiment can be created with the steps shown in Figure 1. It usually involves using provided test scripts. 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.

ProtoGENI allows only SSH login to the nodes the user acquires through the 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). These ports could open vulnerability to port scan and exploration.

Figure 1 shows the steps and communications for a ProtoGENI experiment. In the Figure, the security related parameters are illustrated. This figure provides an overview for where threats could be introduced. In addition, the storage location of the *SSL Certificate and SSH Keys* makes it easier for being stolen or tampered with. If stolen, attackers can access to the experimental nodes being used by legal users. From these nodes, more security attacks could be performed.

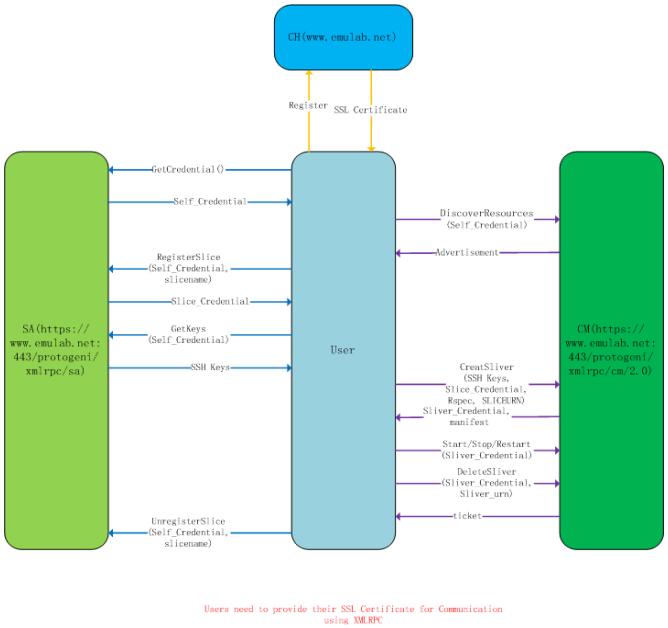

Figure 1. Steps for Experiment with Securiry Parameters

B. Automated exploiraiton with tools

We have developed several automated tools for our experiments. The tools are developed in order to mimic a real attacker, who would be greedy and efficient in

grabbing ProtoGENI network information and resources to make the attacking powerful.

Assume the attacker can intrude a ProtoGENI user, or active slices or himself is a malfunctioning user. The purpose is to perform DoS attack by requesting as many resources as possible. In addition, the attacker is cautious about its action, he may choose to disguise his action with some randomness.

We describe the steps of our experiments with the tools to perform possible DoS attack and security issues below:

1) Use automated experiment tools to register many slices and create cooresponding slivers.

The 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. ProtoGENI Control Framework (CM) works well for multiple experiment creation until the last slice that the resources acquired for the extra slivers (more than one) can not be met. Usually, the user cannot be logged into the sliver with SSH.

2) Use RSpec henerating tool to create various topology

genRspec.py can generate RSpec of certain topology types quickly with the number of nodes as an input. 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.

3) Security issue:

One issue is that when multiple slivers are created, the SSH public key may not be passed to the all resources properly at the same time.

Another issue is to disguise attacking behavior. The attacker can use the tool to generate different Rspec of different topologies at different slices to be combined with the above step. These different topologies help him with more disguises than a fixed topology used by many different slices. He could always use the same Rspec with a large topology, but that raises his chance of being discovered.

The third issue is to maximize its attacking gain. Due to the dynamic usage of the GENI resources, the attacker intends to use as much residual recourse as possible. So he may try different topology sizes to approach the current available resource. A simple adaptation algorithm can serve this purpose, e.g., binary search algorithm.

In addition, our early experiments have shown that ProtoGENI CM does not allow multiple *gettickets* from one slice. It always takes the latest Rspec for the current slice. This is to say that automated tool to generate multiple tickets with in a slice doesn't help the attacker.

C. Flash Interface

ProtoGENI now allows 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.

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

III. Experiments wth Resource Allocation

Here we describe experiments that exploits the virtualization mechanism inside the ProtoGENI where resource allocation is concerned. In experimentation, the security issues are related 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.

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.

Experiment Steps:

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

iperf –s

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

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 eventhough 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

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…

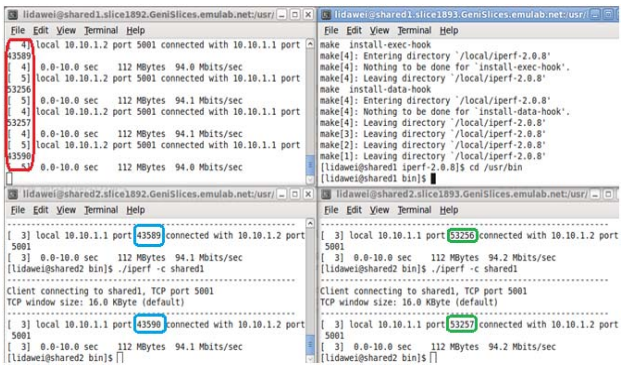

Figure 2. Cross-slice Communication for Test1 and Test2

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.

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.

Suggestions: The developers of ProtoGENI control framework may need a fix at its design and implementation for Vnode according to its different mechanism.

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 [3]. So the link between two Vnodes or link between a Vnode and a normal node may reveal different performance characters.

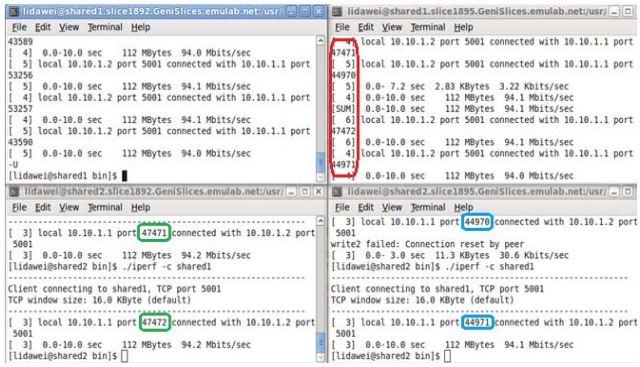

Figure 3. Cross-slice Communication for Test1 and Test3

TABLE II. CROSS-SLICE EXPERIMENTS RESULT

Vnodes or Normal	Iperf to the server with	Same node name or	Result

nodes	node name or IP address	different node name	
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 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.

```
[lidawei@shared1 ~]$ ping shared2
PING shared2-link0 (10.10.1.1) 56(84) bytes of data.
64 bytes from shared2-link0 (10.10.1.1): icmp_seq=1 ttl=64 time=6.31 ms
64 bytes from shared2-link0 (10.10.1.1): icmp_seq=2 ttl=64 time=3.64 ms
64 bytes from shared2-link0 (10.10.1.1): icmp_seq=3 ttl=64 time=2.63 ms
64 bytes from shared2-link0 (10.10.1.1): icmp_seq=4 ttl=64 time=3.73 ms
64 bytes from shared2-link0 (10.10.1.1): icmp_seq=5 ttl=64 time=1.70 ms
64 bytes from shared2-link0 (10.10.1.1): icmp_seq=6 ttl=64 time=1.74 ms
64 bytes from shared2-link0 (10.10.1.1): icmp_seq=7 ttl=64 time=1.77 ms
64 bytes from shared2-link0 (10.10.1.1): icmp_seq=8 ttl=64 time=2.86 ms
64 bytes from shared2-link0 (10.10.1.1): icmp_seq=9 ttl=64 time=1.83 ms
64 bytes from shared2-link0 (10.10.1.1): icmp_seq=10 ttl=64 time=1.86 ms
64 bytes from shared2-link0 (10.10.1.1): icmp_seq=11 ttl=64 time=2.97 ms
64 bytes from shared2-link0 (10.10.1.1): icmp_seq=12 ttl=64 time=1.94 ms
64 bytes from shared2-link0 (10.10.1.1): icmp_seq=13 ttl=64 time=3.03 ms
64 bytes from shared2-link0 (10.10.1.1): icmp_seq=14 ttl=64 time=2.00 ms
64 bytes from shared2-link0 (10.10.1.1): icmp_seq=15 ttl=64 time=3.09 ms
64 bytes from shared2-link0 (10.10.1.1): icmp_seq=16 ttl=64 time=3.11 ms

--- shared2-link0 ping statistics ---
16 packets transmitted, 16 received, 0% packet loss, time 14998ms
rtt min/avg/max/mdev = 1.705/2.767/6.314/1.139 ms
```

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

```
[lidawei@shared2 ~]$ ping shared1
PING shared1-link0 (10.10.1.2) 56(84) bytes of data.
64 bytes from shared1-link0 (10.10.1.2): icmp_seq=1 ttl=64 time=1.40 ms
64 bytes from shared1-link0 (10.10.1.2): icmp_seq=2 ttl=64 time=2.80 ms
64 bytes from shared1-link0 (10.10.1.2): icmp_seq=3 ttl=64 time=1.94 ms
64 bytes from shared1-link0 (10.10.1.2): icmp_seq=4 ttl=64 time=2.97 ms
64 bytes from shared1-link0 (10.10.1.2): icmp_seq=5 ttl=64 time=4.05 ms
64 bytes from shared1-link0 (10.10.1.2): icmp_seq=6 ttl=64 time=2.21 ms
64 bytes from shared1-link0 (10.10.1.2): icmp_seq=7 ttl=64 time=3.23 ms
64 bytes from shared1-link0 (10.10.1.2): icmp_seq=8 ttl=64 time=1.24 ms
64 bytes from shared1-link0 (10.10.1.2): icmp_seq=9 ttl=64 time=2.41 ms
64 bytes from shared1-link0 (10.10.1.2): icmp_seq=10 ttl=64 time=2.42 ms
64 bytes from shared1-link0 (10.10.1.2): icmp_seq=11 ttl=64 time=1.42 ms
64 bytes from shared1-link0 (10.10.1.2): icmp_seq=12 ttl=64 time=2.59 ms
64 bytes from shared1-link0 (10.10.1.2): icmp_seq=13 ttl=64 time=1.67 ms

--- shared1-link0 ping statistics ---
13 packets transmitted, 13 received, 0% packet loss, time 12005ms
rtt min/avg/max/mdev = 1.244/2.337/4.051/0.784 ms
```

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.

```
[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*

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*.

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 the 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.

```
[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 8. ProtoGENI Network Stability Test

IV. DoS Attack to Test ProtoGENI Run-time Vulnerability

We repeatedly requested ProtoGENI resources by running C++ programs, which automatically generated specification of the sliver XML files and created slices and slivers. One program is responsible for creating slices and slivers; the other is responsible for deleting the slices and slivers after testing the results in order to give no real trouble to the Emulab site. Fig. 9 shows the creating slices and slivers program running at the 4th slice and sliver

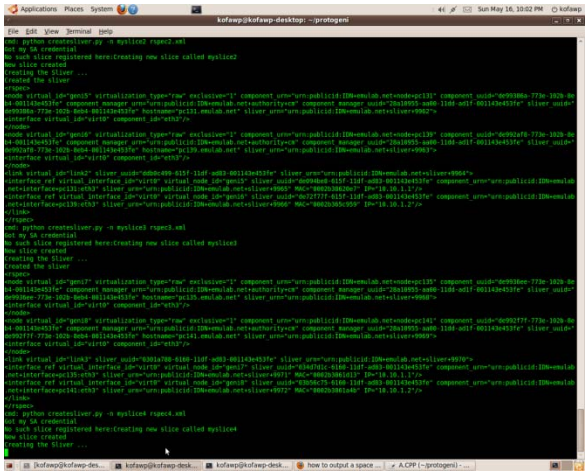

Fig.9 The Creating Slices and Slivers Program Running at the 4th slice and sliver

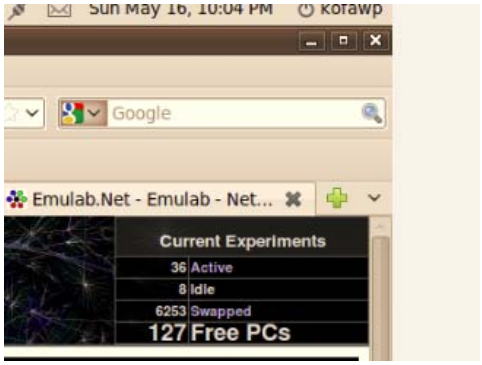

Fig. 10 The Remaining PCs are 127 at the time of our running the 4th slice and sliver

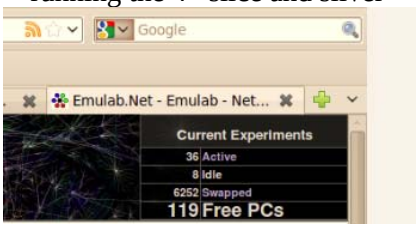

Fig. 11 The Remaining PCs are 119 at the time of our running the 10th slice and sliver

Fig. 12 The Remaining PCs are 135 at the time of deleting all slices and slivers

Our tests show that our programs can easily create slices and livers. In Fig. 10 and Fig. 11, the number of available PCs decreased from 127 (4th slice) to 119 (10th slice) due to the requesting resource procedure. However, from Fig. 11 to Fig. 12, the number of available PCs increased from 119 to 135 due to finishing deleting. We chose the program argument 10, which means that we

only requested 10 slices. However, it is dangerous to the ProtoGENI if we run the program with an argument of 100, 1000, etc. Also, the ProtoGENI tutorial wiki says that the slice is a set of slivers. But our tests showed that one slice can only contain one sliver.

In this way, we tested the vulnerabilities of the Emulab site of ProtoGENI to Denial of Service (DoS) issues.

We conducted another experiment for threats to the availability of resources as follows:

- We created of as many slices as possible to exhaust resources
- We created slices and allocated resources to slivers
- Initial Emulab statuses : 33 free PCs; 16 slices were created as a series of similar names like shailslice1, shailslice2…. shailslice16, each with a request of 2 PCs , then we watched for 17th slice

We could not create all 16 slices: 3 slices were aborted, and only one free PC was left after the creation of the 14th slice. The results are shown in Fig. 5.

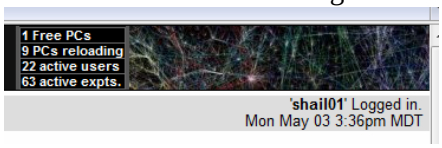

Fig. 13 One PC was left.

V. ATTACKING AUTHENTICATION AND EXPERIMENTS ON STOLEN CREDENTIALS

The authentication process of ProtoGENI involves saving users' credentials on their local Linux machines. The whole attack design process is divided into three levels addressed in the following paragraphs. Fig. 6 shows a ProtoGENI experimental flow chart.

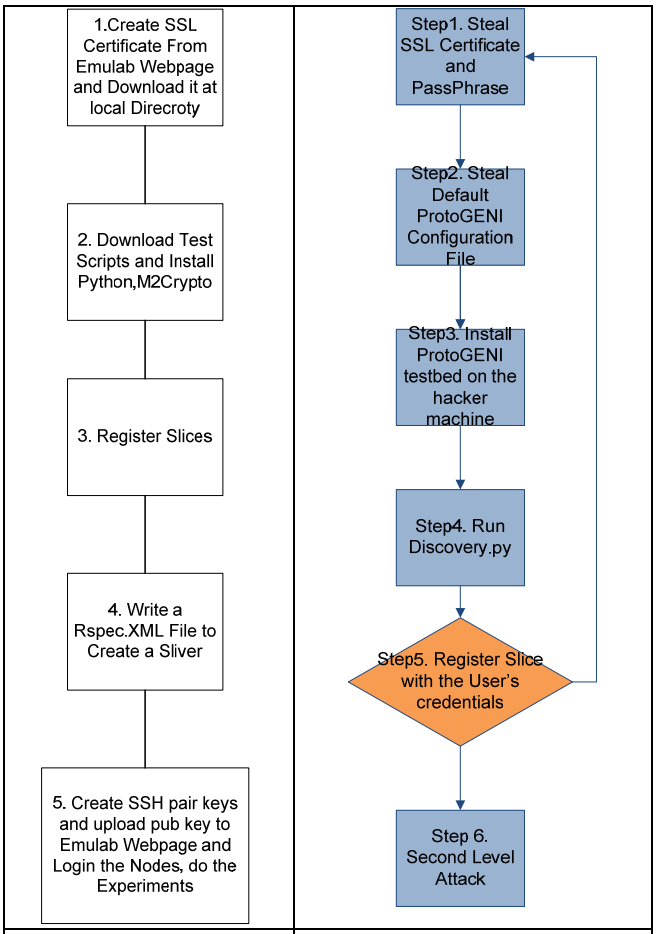

Fig. 14 ProtoGENI Experimental Flow Chart

Fig. 15 First Level Attack Design Diagram

A. *Trojan Horses and How This Works*

Many Linux users believe that they are immune to malware and Trojan threats. However, Netinfinity [1] showed that, if a hacker can combine a victim's shell with a port, the hacker can connect and execute arbitrary commands on the victim's computer without the owner's knowledge. Thus, there is a remote shell available to the attacker. As most users are invariably logged in as a root user, it is highly probable that this would become a remote root shell.

Once the shell binds to the port, the attacker could have the victim's IP address sent to a remote FTP server or even an IRC. The attacker has thus converted the victim machine into a Zombie (orbot).

We created a Malware described as follows:

1. *Trojan*: (1) Make a directory .gnome-system; (2) Startup the Gnome-system script so that the victim's malware starts;
2. *shellbind:* A netcat command that binds a port of the victim's shell to the port 5555;

3. *ftp2ftp:* Sends the victim's IP address and username to the Hacker's ftp.

The Trojan Horses source codes and further explanations will be provided in the later subsections.

B. First Level Attack

First, Step 1 for ProtoGENI is to acquire an SSL certificate as shown in Fig. 14. The users use the Emulab webpage to generate a certificate from their Emulab passwords and a Pass Phrase, and they can then download their certificates to the local machine before using ProtoGENI. The potential vulnerability could be that attackers can steal the certificates of the authentication if they can inject malicious code, such as a Trojan horse to the users' local machines. This operation potentially compromise the ProtoGENI nodes ; because hackers could copy the saved SSL and passwords files from the users' machines to their own, and then get the right to setup and interact with the ProtoGENI. After the hackers have access to the ProtoGENI, they will be able to act as the real users and register the slices from the ProtoGENI. We will show you how this works later in the paper.

Fig. 15 shows the first level attack design diagram. In the first level attack, we used the Trojan horse to open a back door to be used to steal the SSL certificate, as shown in Fig. 16. After we bounded one of the victim's shells (the victim is another experimental account) to the port 5555, we used the remote shell to connect to the victim's computer and steal the SSL and Pass Phrase from the machine, as shown in Fig. 17. As shown in Fig. 18 and Fig. 19, the hacker successfully used the user's SSL certificate and configuration file to obtain the ProtoGENI resources and to register a "hackerslicer" slice.

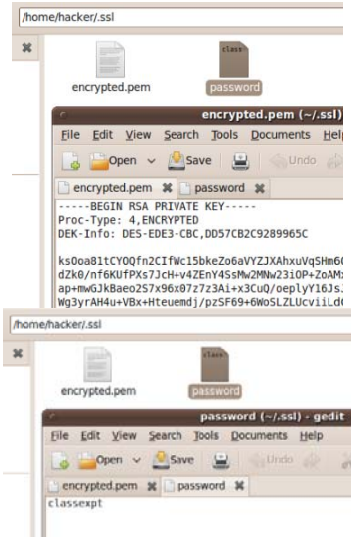

Fig. 16 Setup the Stolen SSL

Fig. 17 Setup the Stolen Passphrase

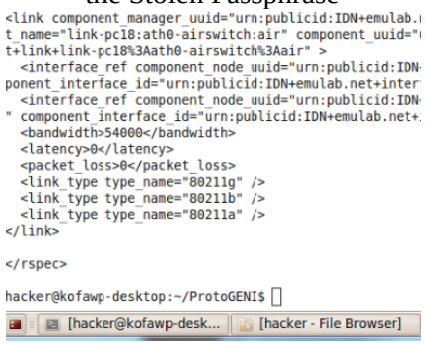

Fig. 18 Hacker Successfully Ran Discovery

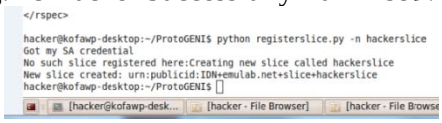

Fig.19 Hacker Successfully Create a Slice

After being hacked, we checked whether the user could still register the slice. Fig. 20 shows that the user can still register the slice because the user picked up a slice name that was different from the hacker. In future experiments, we need to test with both the hacker and the user using default slice names and then check the DoS attack. Ultimately, the hacker did compromise the user's machine.

Fig. 20 User still can create a slice after been hacked

C. Second Level Attack Design

In the second level, Step 4 in Fig. 14 shows that the users can create slivers under slices by creating their own resource specification XML Files (RSpec.XML). This means that hackers could also steal the resource specification XML files from the users. This would give the hackers the ability to pretend to be the users and create their experiments without the acknowledgement of the actual users.

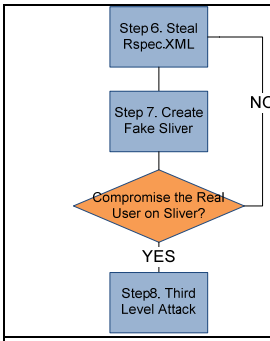

Fig. 21 Second Level

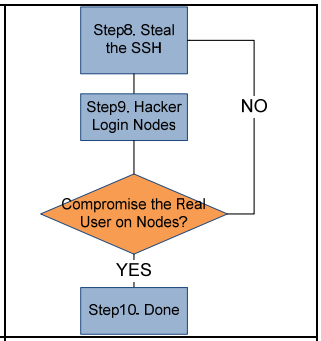

Fig. 22 Third Level

Attack Design Diagram	Attack Design Diagram

Fig. 21 shows the second level attack design diagram. After stealing a resource specification XML file, shown in Fig. 23, the hacker successfully created a sliver using user's specification, as shown in Fig. 24. But, the user also created a sliver in Fig. 25 because the two slivers were created under two different slice names. The hacker compromised the user's machine in the second level attack. We now study go to the third level attack.

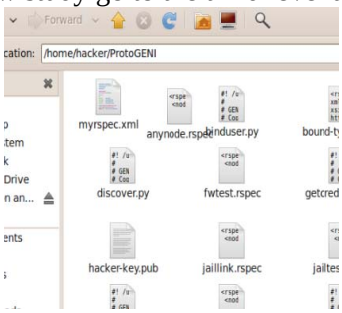

Fig. 23 The Hacker Stole the User's Resource Specification XML

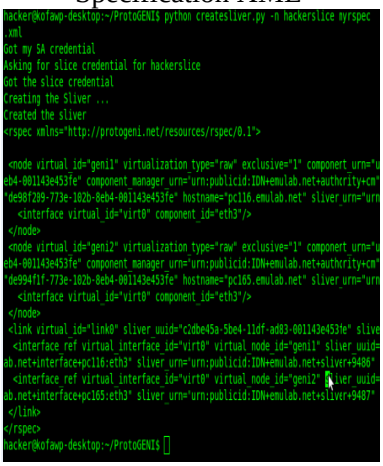

Fig.24 The Hacker Created a Sliver

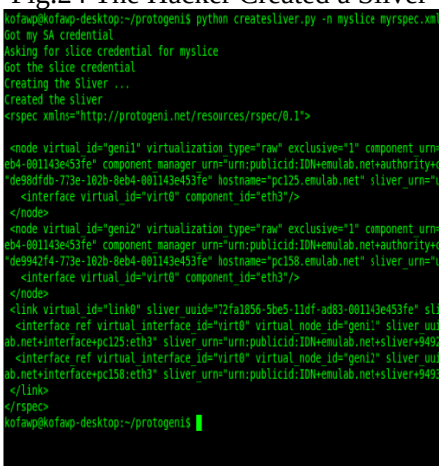

Fig. 25 The User Created a Sliver after Being Hacked

D. Third Level Attack Design

In the third level, Step 5 in Fig. 14 shows that, after the users acquire the nodes and links from the ProtoGENI, they need to create SSH pair keys and to upload the public key to their Emulab webpage. Unfortunately, they also have to store the paired key on their own local machine. This means that the same Trojan hacker's attack strategy could be used to steal the SSH paired keys from the user machine and to let the hacker login the nodes created by the real user; and then the real user could not log into their own nodes. This behavior is characterized as compromising the user's local machine to create a DoS attack.

Fig. 22 shows the third level attack design diagram. After stealing the SSH keys from the user, shown in Fig. 26, the hacker tried all the paired keys that the user used. But this hacking process was not successful, as shown in Fig. 27 and Fig. 28. The reason could be either 1) the unreliability of the ProtoGENI testbed because even the normal users often have problems logging into their designated nodes or 2) SSH key infrastructure is not secure enough. The answer to this question is yet to be determined. We need further experiments to investigate it.

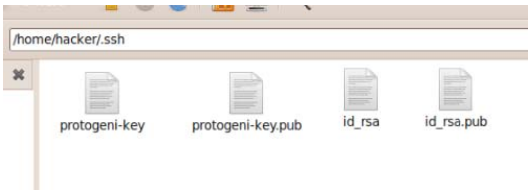

Fig. 26 Stolen SSH Key

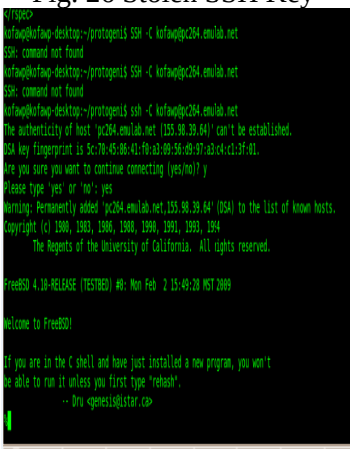

Fig. 27 User Login the Nodes

Fig. 28 Hack Could not Login the Nodes

VI. PORT SCAN ATTACK FROM INSIDE/OUTSIDE NODES

Port scanning is a common method used by attackers to find out which ports are open and can be attacked. This experiment scans the ProtoGENI nodes both from outside ProtoGENI (i.e., from one of our non-ProtoGENI desktops) and from within the nodes to check for open ports.

The experiment was conducted on two ProtoGENI nodes. We used NMap, a port scanner, to analyze the ports that were open and vulnerable to attacks. Steps to initiate the experiment included the following:

1. Download python and M2Crypto.
2. Assuming that we already have an Emulab account, login to Emulab with your id and password and generate a certificate. Download the certificate and save it in $HOME/.ssl/encrypted.pem.
3. Generate the ssh key using the command ssh-keygen -f protogeni-key
4. The key that is generated is saved as protogeni-key.pub and protogeni-key.
5. Upload the public key into protogeni.
6. Download the test script from the link below: http://www.emulab.net/downloads/protogeni-tests.tar.gz.
7. Unpack the tarball somewhere.
8. Make sure everything is working fine and run the python program discover.py.
9. Create a slice using the command registerslice.py
10. Once the slice is created, create a sliver which specifieswhich resources we need. We used two nodes in our experiment: Geni1 and Geni2.
11. Create the slice using createsliver.py and myrpsec.xml. myrspec.xml contains all the resources that we want to request.

Once the sliver is obtained and we have the two nodes that we requested, we are going to conduct two different experiments.

A. Scanning Nodes from Outside

In this part of the experiment, we scan nodes Geni1 and Geni2 from our desktop using a port scanner, as shown in Fig. 21.

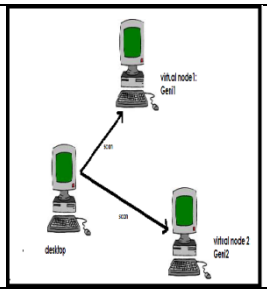

Fig. 29 Desktop scanning the two nodes

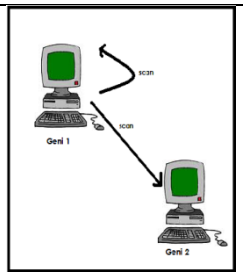

Fig. 30 Geni1 scanning itself and the other node Geni2.

Steps to initiate the experiment included the following:

1. Download a port scanner which is available online. (We used NMap scanner.)
2. Scan the nodes from outside protogeni (i.e., from a desktop).
3. Use the addresses of the two nodes to scan them individually.

Figs. 31-32 show the screenshots of the two nodes being scanned. We observe that port 22 is open. Thus this port is vulnerable to attack.

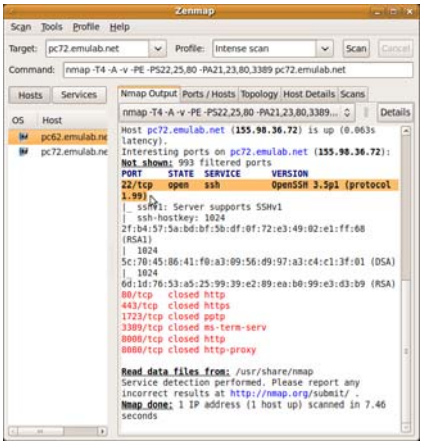

Fig. 31 Geni1 was scanned by NMap

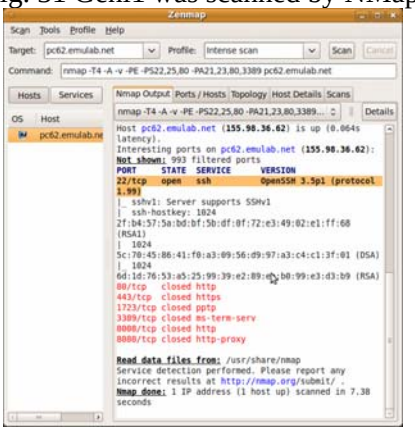

Fig. 32 Geni2 was scanned by Nmap

B. Scanning Nodes from inside ProtoGENI

In this part of the experiment, we login to a node and

let the node scan itself and other ProtoGENI nodes. Steps to initiate the experiment included the following:

1. Login to each node and scan it and the other node.
2. First, using Geni1, scan it using the command: Nmap -sS localhost
3. Scan Geni 2 using the command: Nmap -sS address of geni2
4. Repeat the same with Geni 2 by scanning it and Geni 1.

The results are shown in Fig. 33-36, which show that port 22, which is the ssh port, is open. The scan results are shown in Table 1.

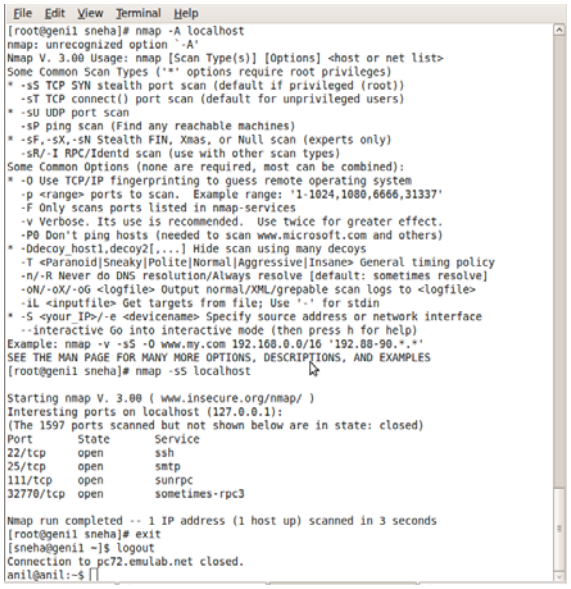

Fig. 33 Geni1 self scan

Fig. 34 Geni1 scanning geni2

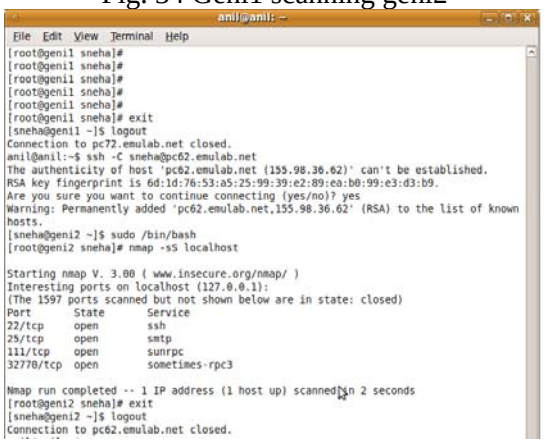

Fig. 35 Geni2 self scan

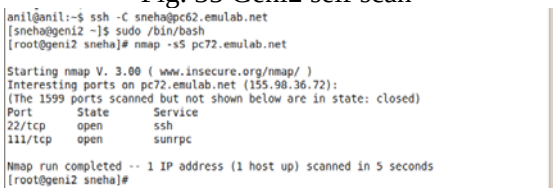

Fig. 36 Geni2 scanning Geni1

TABLE 1 SCAN RESULTS

Node which is scanning	Node being scanned	Open ports	Service
Desktop	Geni 1	22/tcp	ssh
Desktop	Geni 2	22/tcp	ssh
Geni 1	Geni 1	22/tcp 25/tcp 111/tcp	Ssh Smtp Sunrpc
Geni 1	Geni 2	22/tcp 111/tcp	Ssh sunrpc
Geni 2	Geni 1	22/tcp 111/tcp	Ssh sunrpc
Geni 2	Geni 2	22/tcp 25/tcp 111/tcp	Ssh Smtp Sunrpc

VII. SUMMARY AND SUGGESTIONS

First, we performed an analysis of the rum-time communication steps which provide an overview of where the security parameters are used and located, so when intrusion happens, where could be the vulnerabilities. We show that accessing Vnodes opens more ports, adding vulnerability to port scan and exploration. Also, the security parameters such as SSL certificates and SSH keys that are stored in the local machine could be stolen, subjecting to the local machine compromise. If this happens, more attacks could happen.

Second, we introduced a few self-developed Python tools, including register multiple slices, creating slivers and generating various topologies for Rspec. With the help of these automated experimental tools, attackers can be quicker, and efficient to use the residual ProtoGENI resources and remain disguising himself. In addition, we also found that attacker can only use the latest ticket for one sliver. The automated tool to generate multiple tickets within a slice doesn't help the attacker in DoS.

Third, *flash interface* uses local browser's location storing SSL certificate, adding risks of opening experiments to local misuses and intrusion. *Suggestions:* The *flash interface* could provide a further check of the

users' identity before he can create a slice using the interface.

Fourth, we performed experiments using shared nodes (Vnodes). One issue is that the current implementation of using shared nodes has a particular problem, i.e. when the Vnodes have the same name in different slices, we can send traffic across slices. *Suggestions*: This may be caused by the implementation of Vnodes in the mapping of the names.

Fifth, the experiments using shared nodes (Vnodes) show that using Vnodes generates large delay variance in RTTs when using repeated pinging. *Suggestions:* Though this is not a security problem, the current virtualization technology could be related.

Sixth, we performed stress tests that relate to the network isolation and quality, particularly, whether the recourse usage is confined to its specification, to see if other sliver creations could be affected. The results are positive.

Seven, we tested the vulnerability of requesting many slices and slivers of the Emulab site of ProtoGENI by writing C++ programs which repeatedly asked for resources and deleted them. Surely, the experiments show the usage of the resources, such as PCs, till exhaust.

Eight, we performed the experiments to attack the authentication and then the following-up attacks. We started by planting a Trojan Horse Malware. We succeeded in stealing the SSL credentials from the user's setup machine for ProtoGENI. With the stolen credential, the attacker is able to act as the real users and register the slices and further create slivers under slices by creating their own resource specification. Then the same Trojan can steal the SSH paired keys. However, the attacker can not login the experimental nodes created by the real user. This is positive. At this point, we'd still say that there is still a large space for the hacker to use other high-level attacking techniques to do more damage to the user's local machine, the ProtoGENI nodes, and even the whole ProtoGENI testbed.

Ninth, we conducted experiments to scan nodes from inside and outside of the ProtoGENI nodes. We concluded that port 22, which is the SSH port, is open and thus vulnerable to attacks.

REFERENCES

[1] Project Technical Documents, "Revised description of planned security experiments," http://groups.geni.net/geni/wiki/ExptsSecurityAnalysis.

[2] Project Technical Documents, "Report on initial experiments and findings", http://groups.geni.net/geni/wiki/ExptsSecurityAnalysis.

[3] 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.