Practical Exploitation on System Vulnerability of ProtoGENI

Dawei Li, Xiaoyan Hong
Department of Computer Science
University of Alabama, Tuscaloosa, AL 35487

ABSTRACT

Global Environment for Network Innovations (GENI) is a unique virtual laboratory for at-scale networking experimentation exploring future Internets. The successful development of GENI has to consider security problems from the design and prototyping stages. However, in many cases, system vulnerability cannot be found unless through real experimentation bearing purposeful and meaningful designs. In this paper, we introduce some of our efforts in exploring the security vulnerabilities in ProtoGENI, a prototype implementation and deployment of GENI. Our results show potential breach on security of GENI in terms of availability. We make suggestions on potential defense strategies in order to improve the ProtoGENI security and its development. [1]

Keywords

ProtoGENI, GENI security, vulnerability, GENI experiments

1. INTRODUCTION

The Global Environment for Network Innovations (GENI) is a virtual laboratory for at-scale networking experimentation [11]. One critical issue for such an at-scale infrastructure is security. The primary goal of GENI security is "not (to be) used for illegal activities or as launchpad for attacks, (and) GENI availability not compromised by attacks [2]." However it is extremely challenging to achieve the goal due to the many unique features. The early GENI developers' approach towards the security goal catches the following essences: (1) researcher identity management including credential generation and delegation; (2) emergency stop procedures being ready to shut down some experiments or the entire GENI infrastructure; (3) security best practices at different aggregates (operating organizations with physical resources to offer).

[1]This work is supported in part by BBN/NSF contract project 1783.

Yet, there is more to understand the security requirements of GENI (in our case, ProtoGENI). Following the need, our project takes an experimenter's approach to explore the system vulnerabilities [6]. The rational of our experimenters is to act as a purposeful user of ProtoGENI who poses security threats when vulnerabilities exist, and then makes suggestions to ProtoGENI development team. The purpose of our experiments is to help build a secure research infrastructure through the suggestions made based on our experiments. The experiments were performed with careful supervision, and with notifications to related personals and were only performed against own testing PCs and testbed slices.

Our current work uses one of the four prototyping clusters of GENI functions, namely, the *Cluster C*. Each prototyping cluster involves unique network resources and control frameworks. ProtoGENI is the control framework of *Cluster C* [4]. It is mainly derived from the research infrastructure Emulab at Utah [1], which gives researchers a wide range of environments to control the test conditions and allows repeatable research results. The ProtoGENI can be regarded as both a hardware facility providing computing and networking resources and a software, i.e., a control framework, defining the policies of user authentication, resource allocation and communication among different parties.

In this paper, we describe the network experiments that exploit the interface between the experimental plane and control plane. The results of experiments show that the damage can be on two directions when acting as an experimenter. One direction is that we are able to make the ProtoGENI resource not available to other experimenters. And the other direction is that we are able to disconnect an active network node from its experiment. That's to say, our preliminary experiments could breach the GENI security goal of "GENI availability not compromised by attacks". We report this finding to corresponding GENI development teams and to suggest possible defense strategies.

Our paper will be organized to introduce the ProtoGENI, and its control plane and data plane (in Section II), the exploitation method (Section III), and experiment steps and the results we obtained (Section IV). We conclude the paper by outline possible defenses strategies (Section V).

2. BACKGROUND

2.1 ProtoGENI Control Framework

ProtoGENI is a prototype implementation and deployment of GENI functions. We introduce the key components of ProtoGENI facility and functioning software en-

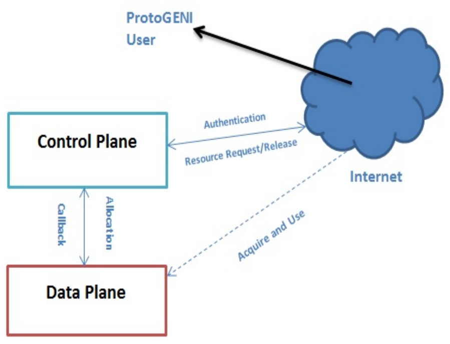


Figure 1: Control plane and data plane

tities. They describe the management and the principles of the usage of GENI and ProtoGENI.

- ClearingHouse (CH): Center for registration, managing the users;
- Component Manager (CM): Resource provider, managing resources at a particular location;
- Slice: Container for resources, providing a piece of live virtualized network testbed for an experiment. It can cross many resource providers;
- Slice Authority (SA): Managing the slices, authenticating users to slices;
- Sliver: Computing resources granted to a user inside of his specified slice;
- RSpec: Resource specification, the mechanism used for advertising, requesting and describing the resources;
- Vnode: virtual node, sharing with other slices in the current sliver through splitting a physical node using virtualization. Current ProtoGENI realizes Vnode through OpenVZ.

2.2 ProtoGENI Control Plane and Data Plane

ProtoGENI facility can be viewed as two separate planes: Control Plane and Data (Experimental) Plane. The data plane is users' slices and is where experiments are, which can be configured for the topology that experiments define. The control plane is a separate network that configures and interacts with the data plane to support the network experiments. It also allows users to access from outside of GENI, usually through Internet connection, to use the slices in their data plane. The architecture is shown in Figure 1.

2.3 Steps to Use ProtoGENI

A user follows a series of steps to do experiments with the provided scripts[4]. Typically, there will be an initial *Getting Ready* phase and repeatedly *Execution* phase. In the *getting ready* phase, a user will need to obtain a SSL certificate and pass-phrase for connecting to the web-based ProtoGENI user portal (currently is the same as Emulab). He will then obtain a SSH key for remotely access to the experimental nodes in the sliver.

After the initial phase, a user can repeatedly use the established credentials to perform network experiments. For each experiment, a user will use scripts to interact with ProtoGENI control framework (clearinghouse, slice authority, component manager) to register and then to create a slice. The user will write an RSpec to express his request of resources in the creation action. Then the user starts the sliver and uses it. When finished, he deletes and deregisters the slice.

3. EXPERIMENTATION METHODS

The experimentation presented in this paper aims at exploiting the potential weaknesses of ProtoGENI through experimental environment exposed to a user of the system internals. One such interface is the connections of the data plane and the control plan. The control plane - data plane architecture provides the attackers a good chance to launch the denial of service attack. Typically, a control node works as the default gateway in the control plane while interacting with experimental nodes. If a denial of service attack being successful, valuable network research performed in GENI will suffer badly and lead to potential inestimable loss. The network protocols running at the control node can be at stake. In this work, we examine the ARP protocol and its potential threats to the availability of ProtoGENI. Our results will show that the ARP cache poisonning attack is particularly harmful in GENI-like network experimentation infrastructure.

3.1 ARP Protocol and ARP Cache Poisoning

ARP (Address Resolution Protocol) is a protocol running on all the TCP/IP based network devices including our daily hosts and routers. When a packet goes cross Internet, it is being forwarded according to the IP addresses of NICs. At lower level, a packet has to go through each physical link which is addressed using MAC addresses. The ARP is utilized to map an IP address to the corresponding MAC address. A device maintains an ARP cache for this mapping. When a sender has a message to send to a destination, the sender obtains the IP address from the routing table of the receiver at the other end of the physical link in the same LAN. It then lookups its ARP cache. If there is an entry for this IP address, the sender forms a frame using the destination's MAC address and send; if not, the sender broadcasts an ARP request for the destination IP address and only the destination responds with its MAC address. The entry in ARP cache is added upon receiving the response. The time stamp of an entry is refreshed every time it is used. An entry expires if it is not used for a certain period of time.

The entries of the IP-to-MAC addresses mapping can be altered by an attacker. A well-known attack is the ARP cache poisoning which is commonly done by sending to a computer's network interface with spoofed ARP packets. Two possible methods can be used to exploit the ARP function in a LAN setting. It starts by flooding a Ethernet LAN with spoofed ARP packets. One method results in hijacking legitimate traffic and also denial of the service of the gateway. The attacker will associate its MAC address with another IP address, for example, the default gateway's IP address. Such attack can open to way for the attacking node to intercept the messages which are originally intended to the default gateway. The other method results in a denial of the service of the gateway. The attacker will send an ARP response to associate a gateway's IP address with a meaningless MAC address. This causes any packets that send to the gateway not being able to reach the gateway network interface card due to the wrong MAC address. This denial of the service attack leads to the factor that any packets go through this gateway will fail.

3.2 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 the connection. *Iperf* allows a user to set various parameters to meet the requirements for network performance testing. Due to the popular functions and measurement metrics they provide, these tools are also included in integrated instrumentation and measurement services of GENI for experimenters [7][8]. In particular, our experiments use a software named *netwox*, an open source software to sniff and spoof network packets of all network layers. Although many other tools like *Wireshark* are available for sniffing and spoofing, *netwox* is easy to install in ProtoGENI and easy to view results through plain texts in Linux terminals. Especially, *netwox* includes a tool set and it uses different numbers to represent different network tools making it an easy mean for attacking experiments.

4. EXPERIMENTS AND RESULTS

In our experiments, we use two slices. One slice acts as the launch pad for the exploitation experiments. And the other slice acts as a normal experiment slice. There are a few challenges for conducting the exploration experiments. The first challenge comes from factor that for the ARP table based attacks to work, an attacker needs to know the system environment, e.g, the hosts and routers within the LAN. The second challenge is that the ARP cache is softstate. It flushes stale entries periodically. Thus an attack shall have a way to sustain the damage over a longer period of time. In addition, ProtoGENI offers hardware facilities as both wired aggregate and wireless aggregate based on Emulab. Our experiments are performed for the two aggregates because they represent the two general types of network research in wired Internet and wireless networks. The two types of aggregates also represent many other available resources in GENI. The challenge is that these two types of aggregates differ in their way for offering experiments.

In this section, we first study the feasibility of learning the experiment environments of both wired and wireless facilities. Then we introduce the experiments with ARP protocol. In our experiment design, we use scripts to sustain the attacking damage.

4.1 Feasibility

For the ARP table based attacks to work, an attacker needs to know the hosts and routers within the LAN. In ProtoGENI, we investigate methods that are feasible to obtain these information in the wired facility and also in wireless testbed.

4.1.1 *Learning in Wired Facility*

When an experiment is created and started at ProtoGENI, it is allocated various resources for the requested network topology and conditions. Figure 2 shows the system and net-

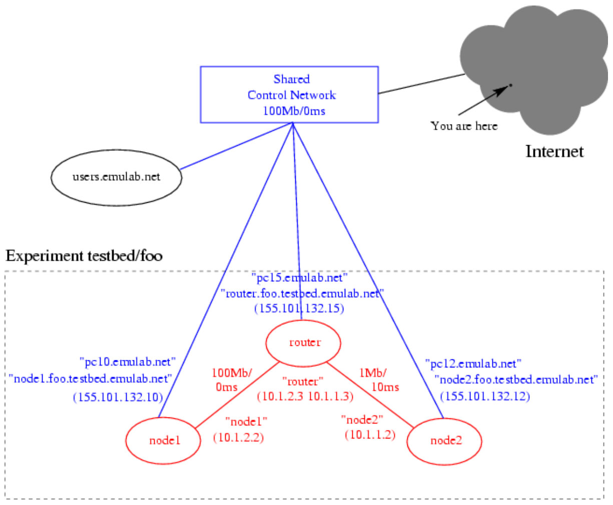
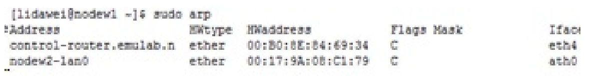


Figure 2: Experiment environment (cited from Emulab Tutorial [1])



```
 [lidawei@nodew1 ~]$ sudo arp
Address                  HWtype  HWaddress           Flags Mask            Ifac
 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 3: ARP Cache before Attack

work environment an user is exposed to in an experiment. It depicts an experimental plane with two hosts and one router (in red) and links in two subnets (in red, $10.1.1.0 and 10.1.2.0$). Each experimental node connects to a shared control network (in blue). Note that the backend control plane's interfaces' IP addresses belong to the same subnet 155.101.132.0. Thus, an experimental node is in the same LAN with the control network and other experimental nodes through the control plan, while each has interfaces in data plan according to experiment topology. The ARP cache at each node for the control network will then store entries that map the IP addresses to corresponding MAC addresses of the control network gateway and the experimental nodes. The nodes also open to Internet with external IP addresses.

We tried with a real experiment which has a two-node configuration. One node's ARP cache is shown in Figure 3, we see there is a "$control - router.emulab.net$" entry in the ARP cache. Checking the routing table, we found the IP address for this interface is 155.98.36.1. Meanwhile, the routing table also indicates that this node can be accessed by SSH with an IP address of 155.98.36.39 from external.

4.1.2 *Packet Sniff in Wireless Testbed*

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 about both the sender and receiver, such as IP addresses and MAC addresses, which can be used to launch attacks.

We perform the following experiment to test the feasibility of sniffing a packet from the air. In this experiment, two slices are created. Each slice has two wireless nodes named *nodew1* and *nodew2*. They connect using 802.11g standard in the topology of $nodew1 - -nodew2$. One slice named as *experiment1* is a normal experiment slice, the two nodes are installed with *iperf*. While the other slice *experiment2* acts a launch pad for an attacker, *netwox* is installed on *nodew1*. The four nodes are located in the following physical positions (Figure 4). The mapping of the physical nodes to the slices are shown in Table 1. Both experiments choose channel 14.

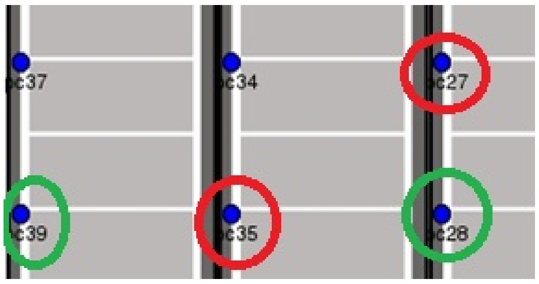


Figure 4: Wireless node locations

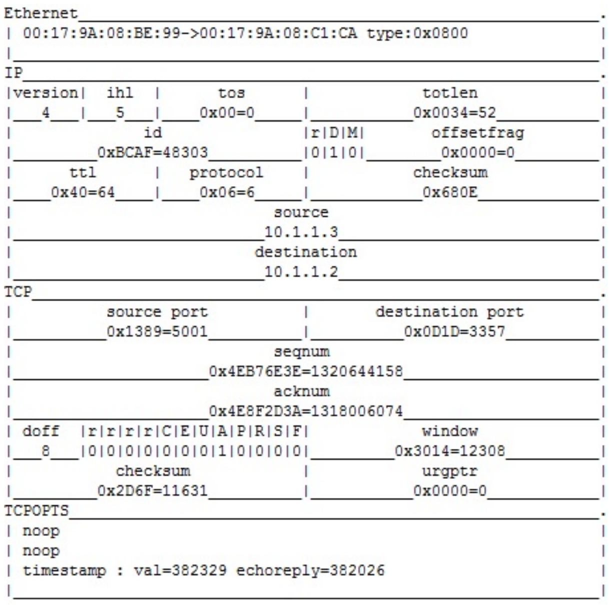
Table 1: Resource Allocation

Slice Names	Nodes	Physical Nodes	Acting
experiment1	nodew1	pc39	iperf
	nodew2	pc28	iperf
experiment2	nodew1	pc35	netwox
	nodew2	pc27	

Experiment Steps: First, in the slice *experiment1*, *iperf* server runs on *nodew2* and *iperf* client runs on *nodew1* to connect to the iperf server. Then, *nodew1* of *experiment2* uses the No. 7 tool of netwox to sniff the TCP packets in the *experiment1*. The following command is used to sniff the TCP packet in the air:

$$netwox\ 7\ \ -d\ ath0\ \ -t\ ,$$

where *ath0* is the wireless interface and *-t* is used to sniff TCP packet. The packet sniffed is shown in Figure 5.



```
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 5: TCP Packet Sniffed

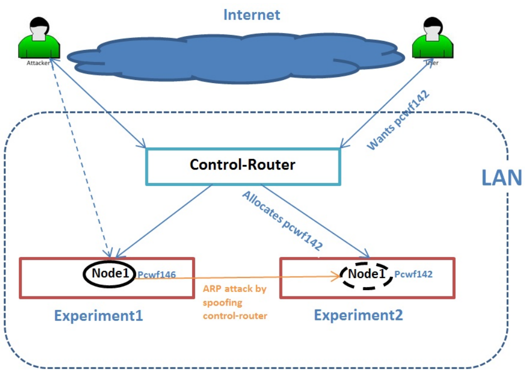


Figure 6: Target at experiment node

In this sniffed packet, we obtain information about both server and client, including IP address, MAC address and time stamp of the transmission. The information can be used to further exploration. Packet sniff in wireless can only be performed when both experiments are using the same channel. However, it is easy to learn which channel is being used by other experiments from Emulab website. Typically, information on which channels are in use at each floor is provided.

4.2 Experiments in the Wired Facility of ProtoGENI

Two slices with slice named *experiment1* and *experiment2* are created. *Experiment1* is alive with a single node *node1* (physical node *pcwf146*) installed *netwox* for conducting attacking experiments. *Experiment2* is a pending slice for acquiring a specified node pcwf142 as its experimental resource (Table 2).

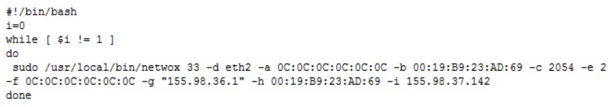
Table 2: Resource Allocation 2

Slice Names	Nodes	Physical Nodes	Acting
experiment1	node1	pcwf146	netwox
experiment2	pending	pcwf142	

The experiments will perform denial-of-service attack through ARP cache poisoning. That will cause failures of using GENI for experimentation. There are two ways to achieve the DOS attack. For method 1, the attacker tries to poison the ARP entry of the control router in an experiment node; For method 2, the attacker tries to poison the ARP entry of an experiment node in the control router. For each of the method, if the attack was successful, the victim experiment node would lose its connection to the control-router node and hence this resource would no longer be available. In our experiment setting, the user's request to acquire this "dismissed" experiment node for *experiment2* would not be satisfied.

4.2.1 Attack an experiment node

The experiment scenario is shown in Figure 6. From *node1* in *experiment1*, the attacker can get pcwf142's MAC address by *ping* pcwf142. A new ARP entry for pcwf142 will be added to *node1*'s ARP cache, and we get the MAC address $00:19:B9:23:AD:69$ of pcwf142. $control-router.emulab.net$ with 155.98.36.1 is still the router interface in *node1*.



```
#!/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
```

Figure 7: Endless ARP Cache Poisoning

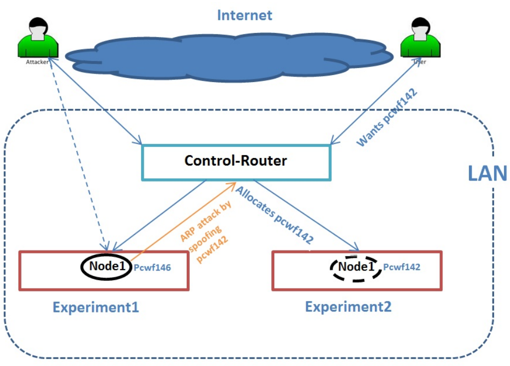


Figure 8: Target at control-router

Then we use the *netwox* tool No. 33 to poison pcwf142's ARP cache by spoofing a faked MAC address for control-router. The tool will send an ARP message to pcwf142. In order to prevent the expiration of the spoofed ARP entry, we use a shell script to run the *netwox* command endlessly. See Figure 7 for details. We would like to modify the MAC address of the control router to $0C : 0C : 0C : 0C : 0C : 0C$ in pcwf142's ARP cache. Then we try to acquire the *pcwf142* for *experiment2* to see if the resource is still available without any exception.

Experiment Result and Analysis: We observed that the *experiment2* still acquired the *pcwf142* as its experiment node. We checked the ARP cache of pcwf142 and found the ARP entry for the control-router is not changed. The is because that the ProtoGENI facility sets the ARP entry of the control router to be a static entry at the experimental nodes for virtualization. It 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. Through this experiment, we validate that ProtoGENI is safe in terms of this specific attack.

4.2.2 Attack the control-router

In this experiment, we exploit the control router directly by poisoning its ARP entry about an experiment node. If this experiment could be successful, we realize the same goal for cutting the connection between the control-router and the experiment nodes. The users could not acquire the desired resources. The experiment scenario is shown in Figure 8 with the same two slices *experiment1* and *experiment2*.

We created another shell script in *node1* of *experiment1* to launch the endless ARP cache poisoning to attack to the control router to modify the ARP entry of pcwf142. Then, *experiment2* tries to get *pcwf142* as its resource node.

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 verify the result directly. But we observed a series of warning and error messages returned in the user's local interface which indicate that pcwf142 can not be used. The messages read like: "*pcwf142 appears wedged; it has been 6 minutes 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.*"

The experiment shows that the denial-of-service attack is possible.

4.2.3 Generalization and security analysis

The above results show that the specific attack to the control plan (namely, poisoning ARP table at the control router) is possible and can result in failure of using the experimentation resources. However, to what extend such an attack can cause troubles to ProtoGENI or GENI? Here we discuss the minimum conditions to launch such an attack.

Three conditions are used in our experiment (acting as attacker): (1) using an active slice, (2) learning the environment, (3) a tool. The condition (3) is not a constraint because open-source tools exist in Internet. For condition (2), our other experiments have explored a couple of ways to learning the names of the physical nodes and also to learn the IP addresses of the entire testbed facilities. The reason is that the name space and IP address space are maintained with easy-to-follow rules for the simplicity of management, which is desired for GENI. Thus, an attacker can easily guess or probe (e.g., through ping) the name space and IP address space of the entire facility. The ping method we used in our experiments helps to learn the MAC address when an IP address is known. A node can then learn all the MAC addresses. Further, without knowing a specific physical node, an attacker can poison the ARP table at control router with a brute force of trying all the IP addresses in the address space. Thus, the condition (2) of learning the environment is not a limitation to an attacker.

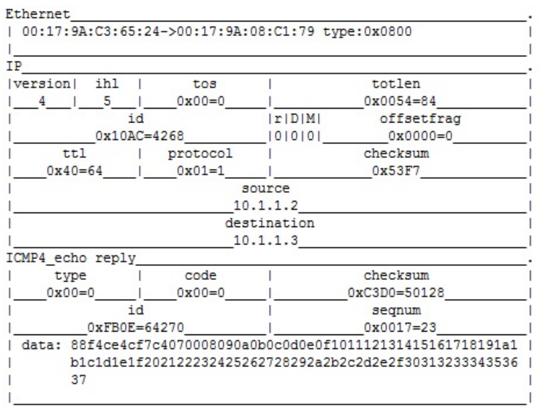
To fulfill the condition (1) of using an active slice, an experimenter uses the authorization and authentication steps of ProtoGENI. He will obtain certificates and keys for experiments. Afterwards, he would keep them safe. The vulnerability here is similar to our daily use of a PC. For the attacker, this is the minimum condition for launching the attack.

4.3 Experiments on wireless nodes

ProtoGENI supports resources for experimenters to design and implement wireless network experiments. Different from experiments of wired connection, the open nature of 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. By sniffing packets of a wireless communication, we can get both MAC address and IP address of the communication pairs. So it will be a potential vulnerability for attackers to launch the ARP cache poisoning. Here we design a demo experiment showing how an ProtoGENI user can use ARP cache poisoning to attack another experiment slice with the netwox tool sets.

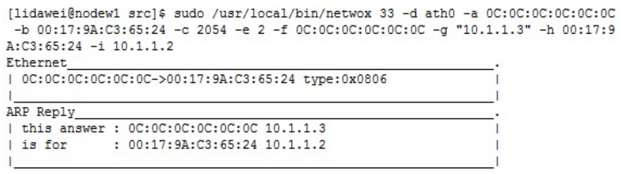
In this experiment, we use the same two slice experiment as previously used for performing packet sniffing in wireless tested (see Table 1 and Figure 4).

In this experiment, we first *ping* from *nodew2* to *nodew1* in slice *experiment1*. This will generate a series of ICMP packets in the wireless channel between the two nodes. Then,



```
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: 88f4ce4cf7c4070008090a0b0c0d0e0f101112131415161718191a1 |
|       b1c1d1e1f202122232425262728292a2b2c2d2e2f30313233343536 |
|       37                                                      |
|_______________________________________________________________|
```

Figure 9: Ping Packet Sniffed



```
[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 10: Launch ARP Cache Poisoning

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 *ath0* is the wireless interface used for sniffing. The packet sniffed is shown in Figure 9.

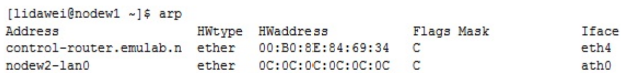
From the sniffed packet, we know the source's IP address is 10.1.1.2 and MAC address is $00 : 17 : 9A : C3 : 65 : 24$; the destination's IP address is 10.1.1.3 and MAC address is $00 : 17 : 9A : 08 : C1 : 79$. We validate this information by checking the ARP table in *nodew1* of *experiment1*, we observe that the MAC address for *nodew2* (10.1.1.3) is $00 : 17 : 9A : 08 : C1 : 79$, which is the same as shown in the sniffed packet.

In *nodew1* of *experiment2*, we use No. 33 tool of *netwox* to launch ARP cache poisoning to attack *nodew1* in *experiment1* as shown in Figure 10 which modifies the MAC address of *nodew2* to $0C : 0C : 0C : 0C : 0C : 0C$. Figure 10 shows the table after the attack. In this case, any traffic sending to *nodew2* will not be received by it. This results in DOS at *nodew2*.

Again, in *nodew1* of *experiment1*, we validate by checking the ARP table again as seen in Figure 11. The ARP table is successfully modified.

4.3.1 Generalization and security analysis

It is clear that the MAC address of *nodew2* at *nodew1*'s ARP table is changed to $0C : 0C : 0C : 0C : 0C : 0C$. This results in DOS at *nodew2*. This experiment is different from the previous experiment targeting at the wired facility, in that, the victim slice is active when performing attacking. This gives convenience to the attacker for sniffing the IP addresses and corresponding MAC addresses from the air. With this information, attacker can easily do an ARP cache poisoning attack. In real case, the *experient2* would



```
[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 11: ARP Cache after Attack

be owned by another ProtoGENI user. The experiment will experience failure.

Again, for an attacker to perform the ARP cache poisoning attack in the wireless testbed, he must use an active slice. As we analyzed previously, the vulnerability for ProtoGENI in this case, will be subject to the security of the experimenter's authentication materials.

Some mechanisms can be adopted to prevent the ARP cache poisoning in ProtoGENI. One effective way is to use a static ARP table for control plane interaction which is quite reasonable in GENI where resources need to be available at all the time. Another approach can be to instal an inspection software that can monitor Ethernet activity and report suspicious messages via email immediately. One such software can be *Arpwatch*.

5. CONCLUSIONS

In this paper, we introduce one security aspect of ProtoGENI, namely the potential security vulnerability in the control plan. Security vulnerability is analyzed using a practical experimental way through our work. We have shown the possibility of an attack through ARP cache poisoning. We have reported our detailed analysis and experiments that exploit both the wired facility and wireless facility of ProtoGENI. We discussed conditions for such an attack according to the results and suggested possible improvements on ProtoGENI security.

6. REFERENCES

[1] Emulab Tutorial, http://www.protogeni.net/trac/emulab/wiki/Tutorial.

[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.

[11] GENI: Exploring Networks of the Future, http://www.geni.net/.