Computer Science 530 - Lab Assignment #8 - Intrusion Detection -- Fall 2022

Due: Friday November 18th, 2022, 4:30 p.m.

(Lab will be accepted up to one week late with no penalty)

Overview

Infrastructure for Lab

Location of files

There is only one appliance (fedora30 Fall 20) that you will use in this lab. You will create 4 instances (node1 node2 node3 and node4) using the populate script. You have already downloaded and loaded the ova file for this appliance, but I have also made it available in the CSci530 google drive in the folder for Lab 8 here.

Please note that you may need to login to google drive with your USC account in order to access these files.

You will note that there is a directoy with scripts (or BATCH files) for this lab. There is a directory for windows machines, and another for Linux and apple systems. Download the scripts from the directory that is relevant to your machine. The scripts in these directories are used to clone the virtual machines (populate), start them (poweron), configure the network between them (construct network, set internal settings for the guest machine (guestOS-internal-setting), power them off, and get rid of them when you are done with the lab (destroy).

You will run these scripts at the appropriate time for the experment nftables below.

Some notes on this instance of fedora Linux

We have already loaded most of the programs you will need for this lab into the virtual appliance. When you start the virtual machine you will be asked to login. The password for both the root and students accounts is "c$l@bLinuX". The third character is the letter "l" as in lab.

Crafting man-in-the-middle founded on arp

Setting up this experiment

Scripts are used. A set of them is provided for each experiment. Please see the "Virtual Machine Usage" section of this document.
A summary of the scripts' use and intended execution order follows.

"vmconfigure-populate" is first, creates machines
"vmconfigure-construct-network" is next, if it exists
"vmconfigure-guestOS-internal-settings" is next, it makes the internal settings after powering the machines on;
so if this one exists "vmconfigure-poweron" is not needed
"vmconfigure-poweron" is next, if there is no "vmconfigure-guestOS-internal-settings"

Now you can use your VM(s). When you are finished:

"vmconfigure-poweroff"
"vmconfigure-destroy" if you want to delete the machines, but not if you plan to come back to them later

If you do come back to them later,

"vmconfigure-guestOS-internal-settings" should be repeated, if present
"vmconfigure-poweron" if there is not "vmconfigure-guestOS-internal-settings" present

Synopsis

This exercise crafts a man-in-the-middle attack by using-- without abusing-- arp. arp is the address resolution protocol. Its design lends itself to the attack. A substantial category of network attacks operate by using protocols per design, but for different purposes than motivated the design. To make the attack, no attempt to *change* the protocol behavior is needed, only to *use* it as-is. This attack, arp spoofing, is in that category. arp is a mechanism designed to let one computer tell another how to reach it. The mechanism's workings are inherently suitable for one computer to wrongly tell another to reach *it*, when the other really wants to reach some 3rd machine.

arp's operation is usually implicit in other network activities, transparent to users. Among other things the arp protocol maintains an in-memory table of IP-to-ethernet address mappings derived from its operation. There is a related command that's also named arp. Its focus is the table, and it is a tool whereby maintenance of the arp table can be done manually. Another command of interest is arping. It is for explicitly, manually triggering the arp protocol to action, emitting arp packets. ettercap is a utility that can use arp to set up a man-in-the-middle attack.

This exercise demonstrates arp spoofing by surfacing normal arp mechanics to view, then using arp as the central component of a man-in-the-middle attack. Students will observe both the normal, implicit operation of arp with tshark watching the ping program, and the explicit operation by using the arping utility. Then they will manipulate arp with ettercap specifically to jockey one computer into position between two others. The others' conversation then flows through that computer as a man-in-the-middle. This is without damaging, debasing, or deforming arp in any way. Arp itself, turned to deliberate usage, is the attack tool.

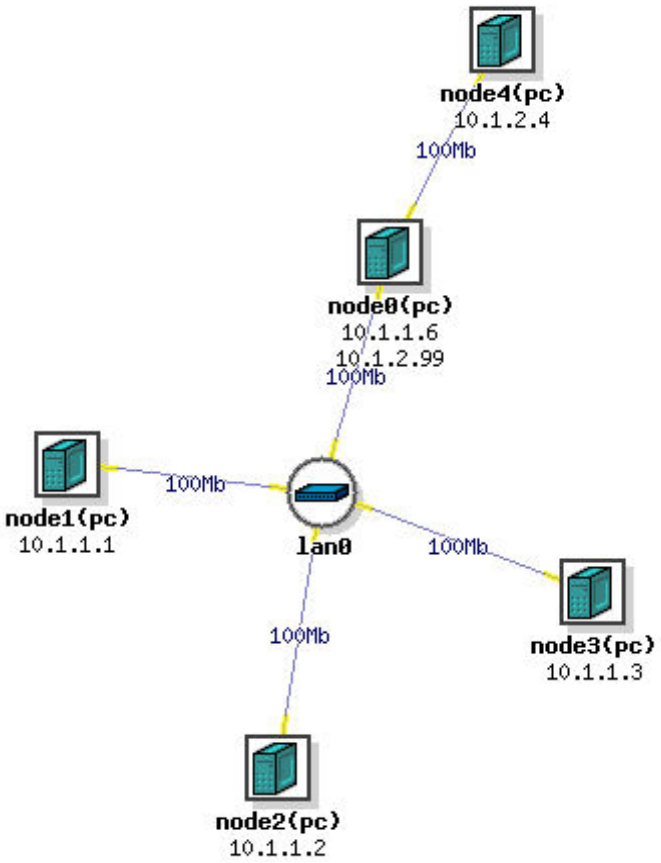
Background and recommended reading materials

- An explanation of arp spoofing
- RFC defining the ARP Protocol
- Home page, ettercap project
- man pages for arp, arping, tshark

Project specification

1. Setting up the topology and tools

This exercise is conducted in this internetwork:



It is a 2-subnet internetwork with node0 serving as a router that joins the subnets. The main, lower network is:

10.1.1.0 / 255.255.255.248 or alternatively **10.1.1.0 / 29**

Activate this network. Scripts to create it are provided. You run them in your host computer. To create the vm's:

vmconfigure-populate (may have filename extension .sh or .bat depending on your host platform)

It instantiates the virtual machines. Typically you will want to do this only once, when you begin the exercise for the first time. If you step away from the exercise and return to it later you need not re-popluate. Then:

vmconfigure-construct-network
vmconfigure-guestOS-internal-settings

The first script does the equivalent of provisioning interfaces (inserting network cards) and plugging cables into them per the exercise's topology. The second causes execution of guest OS commands, typically setting hostnames and applying IP addresses to the interfaces. Because running commands in the guestOS requires the guestOS to be booted and booting takes time, the script boots any of the machines that are not already running then pauses a decent interval to let the guest boot process finish before trying to run guest commands. Be patient.

At this point the virtual machines are running, each in a VirtualBox window on your desktop. You may wish to cascade or otherwise arrange them for accessibility. There are a few adjustments to be made. (You might find that some of these adjustments are already made. Check, and if not, make them. You can check the presence of executables with the "which" command, of files with the "ls" command, of user accounts with the "id" command.)

On node1

This will be the ettercap attack platform. It's the "man" who will position himself in the middle between other nodes. On it you will use ettercap, tshark, and nmap. Log in as root. You need an ettercap filter. Put it in your home directory:

```
cp  /home/public/laughingskull.filter  ~
```

On node2

node2 will operate as an ftp client, and a web server. We want node2 to offer a certain image file to web browsers. Log in as root. Then:

```
systemctl start  httpd    (httpd is the apache web server)
cp  /home/public/laughingskull.gif   /var/www/html/   (apache serves content out of /var/www/html)
chmod  644   /var/www/html/*
```

On node4

node4 must provide ftp and http servers. It also requires a user account whose password we will sniff. Log in as root. Then:

```
systemctl  start  vsftpd

systemctl  start  httpd
cp  /home/public/index.html   /var/www/html
cp  /home/public/congratulations.jpg   /var/www/html
chmod  644   /var/www/html/*

useradd joe
passwd joe    (establish a password of your choosing and remember it)
```

2. Manipulating the arp table

Operate on node1. Examine, populate, and depopulate the arp table. To print the arp table:

```
arp  -n
```

The table might be empty, especially if you just started your machine or haven't been using it for a little while. It gets populated with machines' address pairs in the course of machine interaction. So populating the table calls for a little interaction. Pinging nearby machines will do. But you can't ping a machine if you don't already know its address. The nmap utility could automatically ping the entire range of addresses in your subnet for you. node1's subnet is supposed to be 10.1.1.0/255.255.255.248. Verify:

```
ifconfig  eth0  |  grep  "inet 10.1.1"
```

("eth0" might not be the right interface name. Ascertain. If you want to do so by examination, run "ifconfig -a" and observe the one that possesses the address. If you want to do it programmatically:

```
ifconfig  -a  |  sed  -n  '/10.1.1/{x;p;d;};x'  |  cut  -d  ":"  -f  1
```

will reveal it. Use the name of *that* NIC interface in these instructions wherever I have used eth0, as in the above command for example.)

The netmask value shown by the command above half-identifies your subnet. Convert it from a dotted quad (e.g., 255.255.255.0) into a corresponding CIDR bit count (e.g., 24) using the rules of CIDR. To fully

identify your subnet, it remains to determine its network address. Note the IP address for your machine, which was also shown. Determine your subnet's network address:

```
ipcalc  -n  <your machine's IP address>  <the netmask>
```

Now have nmap ping all the other machines on your subnet (by trying all the address in the subnet):

```
nmap  -sP  -n  --send-ip  <subnet network address>/<CIDR bit count>
```

The reported addresses should accord with the ones shown in the lower subnet of the above diagram. When this is done, again view the arp table. It should contain some fresh, further enries:

```
arp  -n
```

Now that the table has entries, choose one to manually delete (not an "incomplete"). Note its IP address. Delete it:

```
arp  -d  <IP address>
```

Re-examine the arp table:

```
arp  -n
```

If you don't delete an entry manually (you hardly ever do), it will disappear after a certain timeout period (seconds or minutes by default).

3. Recording actual address mappings

These address mappings are hard to remember. Spoofing puts false mappings in arp tables. To grasp what's going on you'll need to be able to differentiate right from wrong mappings on sight. To record the right ones for future reference, still on node1, run the "make-realmappings2" srcript:

```
cp  /home/public/make-realmappings2  ~
chmod  +x  ~/make-realmappings2
~/make-realmappings
```

It deposits the information in file "realmappings." At any time during this exercise, view it as a baseline:

```
cat  ~/realmappings
```

The output should look something like:

```
[dbmnorth@node1 ~]$ cat realmappings
10.1.1.1    00:15:17:57:BF:CE
10.1.1.2    00:15:17:57:C6:BE
10.1.1.3    00:15:17:57:C7:89
10.1.1.6    00:15:17:57:C4:38
[dbmnorth@node1 ~]$
```

Your ethernet addresses will have different values. They may also change if you destroy and later recreate the exercise, depending how VirtualBox assigns them.

4. Triggering arp activity both implicitly and explicitly

At this point we want to see two terminal windows simultaneously. We'll want a graphical interface for this. You are currently on node1 logged in as root in character mode. Log out:

```
exit
```

Then, log in as student and thereafter launch the graphical user interface:

```
startx
```

(If this is sluggish, your virtual machine may need more memory. power it off, edit its memory setting in the Oracle VM VirtualBox Manager, and Start it afresh. Reassign its IP address, which was lost upon poweroff-- ifconfig enp0s3 10.1.1.1/24)

A terminal emulator with good screen splitting features is "terminator". Click "Activities" then enter "terminator" in the resulting search field. You get a terminal window. Maximize it (double-click its task bar). Then split the screen horizontally either from the right-click context menu, or the keyboard shortcut ctrl-shift-O. You have two shells, each in its own half of the window. Shift back and forth between them with the keyboard shortcut ctrl-tab.

In the lower window become root and launch the tshark packet sniffer:

```
sudo  su
tshark -n -i eth0 arp or icmp
```

arp packets are usually issued by the network stack during operations, when needed. That's automatic. They could also be issued by explicit use of the arping command. That's manual. Identify the IP address of a computer on your subnet (e.g., 10.1.1.3). Check your arp table and if that computer appears, delete it, for example (in the upper window):

```
sudo  arp  -d  10.1.1.3
```

In the upper window:

```
ping  -c 1  10.1.1.3
```

Note the arp actvity in the dump. Did you ask for it? Why was it performed? Now again, for the arp table entry of target computer 10.1.1.3, delete it. Now in the "command" window:

```
arping  -c 1  -I  eth0  10.1.1.3
```

Note the arp actvity in the dump. Why was it performed? For which of the 2 commands, ping and arping, was the arp activity incidental and for which was it explicit to the command? Leave tshark running in the lower window.

5. Manual IP spoofing with arping

arping can compose and send an arp request that represents the machine to have an arbitrary (e.g., false) IP address. To allow that in the kernel, in the upper window on node1:

```
sudo  su  -
echo  1  >  /proc/sys/net/ipv4/ip_nonlocal_bind
```

then choose both an IP address within your subnet that is not in actual use by any computer (e.g., 10.1.1.5), and one that some other computer uses (e.g., 10.1.1.3), while tshark continues running in the other, lower window:

```
arping  -c 1  -U  -s 10.1.1.5  -I eth0  10.1.1.3
```

Observe the outgoing arp request shown by tshark, and in particular the value in its sender IP field (which IP it claims to come from). It would be interesting to view the arp table on target machine 10.1.1.3, and what impact this command may have had on it. Remotely (and without great delay) run the command to do so:

```
ssh  student@10.1.1.3  "sudo arp -n"  (give password when requested)
cat  realmappings
```

Note you are using ssh here as a remote command executor. (That's actually ssh's essential but overlooked purpose. Running it the usual way, for a remote command session, is really just the special case where the to-be-executed remote command happens to be a shell.) The arp table output by the above command *is not yours*. It is that of the other machine. This way, you can view the arp tables of linux machines in your subnet on demand without having to log in to them. Note 10.1.1.3's arp table contains a mapping for 10.1.1.5. Is that mapping valid or invalid? (Big hint: there is no 10.1.1.5.) Huh? What ethernet address is non-existent 10.1.1.5 supposed to have then? Is it too non-existent? Check the realmappings file. Whose ethernet address does the poisoned arp table on 10.1.1.3 claim that 10.1.1.5 has?

To our deceived machine (10.1.1.3), arping is able to (mis)represent that the ethernet address in the machine where arping runs (10.1.1.1) belongs with some *other* machine's IP (10.1.1.5). It does not. Once that falsehood is lodged in the deceived machine's arp table, that machine will send frames wrongly to arping's machine (10.1.1.1) whenever it wishes to reach the other IP (10.1.1.5). *Instead of* sending to the machine that actually holds that IP (if any). That's what "spoofing" means. I tell you that I'm Joe. Every time you have something to say to Joe you turn to me and say it, unaware that I'm not him.

How does arping get away with this injustice? By following the design operation of arp. arp intentionally provides that whenever machines get requests for their *own* ethernet-to-IP mapping (i.e., when somebody else arps them), they should then and there record in their table the *sender's* mapping. That's always embedded in the incoming arp request. So there are 2 ways a machine can get another machine's mapping into its table-- actively by asking for it, or passively as side-effect of being asked. When being asked, the embedded incoming sender mapping is taken at face value, no questions asked. Straight into the arp table it goes. Why?? Because arp is smart. It reasons that the requestor is not requesting idly, but because it is about to initiate communication which will require responses and so, also, the sender's mapping. It's a sure bet his mapping info will be needed so, now that it's in hand, don't discard it but put it in the table immediately for efficiency. So, I can manipulate your arp table by composing any mapping I want and embedding it as the sender mapping into an arp request to you. The arping utility is one tool that can do this for you.

In the lower "dump" window, terminate tshark (ctrl-C).

6. Arp-spoofing your way into the middle of a traffic stream with ettercap

Ettercap can do the same thing you just did with arping among *multiple* nodes in a net. Just as I could do it among multiple people. I could tell you I'm Joe. And I tell Sally I'm you. Now the both of you will turn to me whenever you want to talk to each other and say it to me, unaware I'm not the other. If I parrot your stuff on to her, and hers back to you, neither of you will notice anything out of the ordinary. I would become a man in the middle. Optionally, I could record all your secrets or insert lies. Ettercap does exactly the same thing for ethernet frame conversations.

Let's insert node 1 in the middle between node2 and node0 then watch 2 and 0 ping each other. In node1's upper window where you are running as root:

```
ettercap -i eth0 -T -M arp /10.1.1.2// /10.1.1.6//
```

From node1's lower terminal window look at the arp tables on both ettercap target nodes:

```
please execute "arp -n" on the machines of interest as in the red comments
ssh  student@10.1.1.2  "sudo arp -n"  [ or, as above if necessary, go operate directly on 10.1.1.2/node2 in its own terminal window ]
ssh  student@10.1.1.6  "sudo arp -n"  [ or, as above if necessary, go operate directly on 10.1.1.6/node0 in its own terminal window ]
sudo cat /root/realmappings
```

and compare appearance (in those two nodes' eyes) with reality (in realmappings). Nodes 2 (IP 10.1.1.2) and 0 (IP 10.1.1.6) appear to have a certain hardware address. Whose? On node1 (from 2nd terminal window)

run tshark:

```
tshark -c 4 -i eth0 icmp
```

Then concurrently on node2:

```
ping  -c 1  10.1.1.6
```

The above 1-count ping should evoke a single request and corresponding, single reply between nodes 2 and 0. The tshark trace (on node1!) shows *2 of each* in terms of their IP addresses. What's going on? IP addresses are *unreliable*; ethernet addresses are *fundamental*. Find out the *real* comings and goings of these 4 frames by rerunning node1's tshark command as follows:

```
tshark -c 4 -i enp0s3 -T fields -e eth.src -e eth.dst -E header=yes icmp
```

Again from node2:

```
ping  -c 1  10.1.1.6
```

and compare with the real-world mappings; on node1:

```
cat  /root/realmappings
```

The trace again shows 4 frames, but in terms of their ethernet addresses this time. Study the trace with meticulous attention to the addresses and the nodes they belong to. Follow which node each frame comes from and which it goes to. How does traffic between node2 and node0 get from node2 to node0?

7. Manipulating the traffic stream once you have it - sniffing it

When something passes through your hands you have it in your hands. So you can handle it. Or not. By default ettercap does not. It just plays innocent relay-man much as any router would. But if it decides to intercede, there are several ways to do it. One possibility is to dynamically read payloads and sniff out targets such as passwords. Let's have node2 log into node4's ftp server and have ettercap sniff out the login password. Log into node4.

In the node1 terminal window running ettercap, terminate it with the "q" key (note the screen messages). Then run:

```
ettercap -i eth0 -Tq -M arp:remote /10.1.1.2// /10.1.1.6//
```

In the node2 terminal window, while leaving the above node1 window visible on the display:

```
ftp 10.1.2.4
```

and log in as joe with joe's password when prompted. What appears in the node1 terminal window where ettercap runs? How? (Terminate the ftp client session-- "quit".)

8. Manipulating the traffic stream once you have it - altering it

While the datastream is passing through, another way ettercap can intercede is to actually change it.ettercap can apply a dynamic search-and-replace. Let node3 browse the node4 web server's default page, but modify it on the way from node4 to node3, in node1.

Run a web browser on node3. A character mode web browser available on linux for such diagnostic purposes in lynx. Use it to browse node4. The default web page on node4 is:

<html><body>

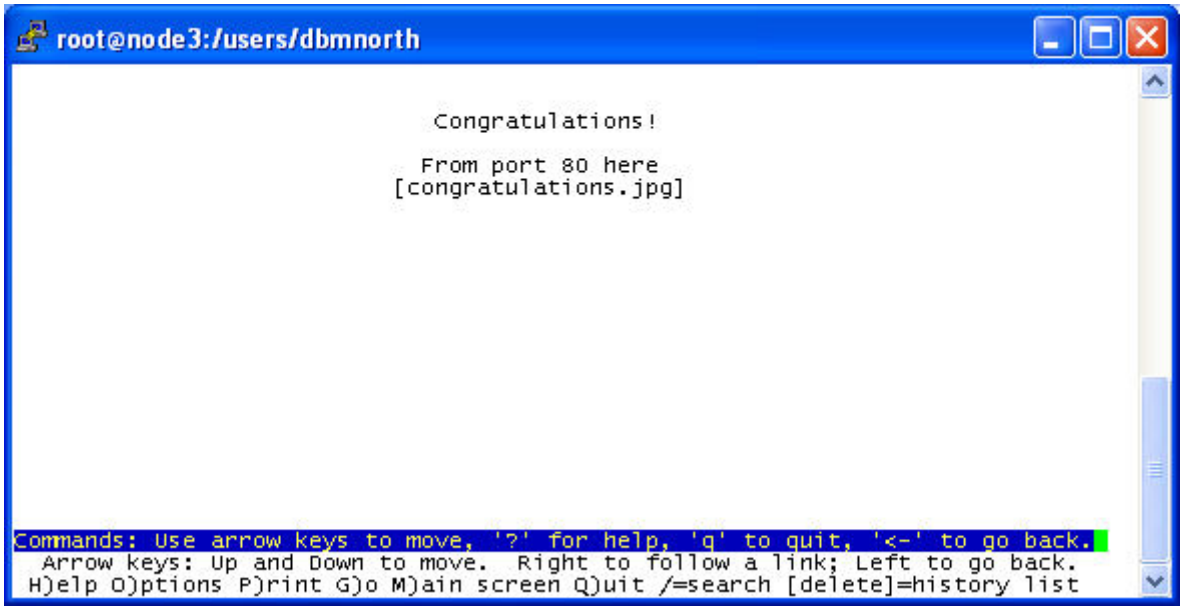
```
<h1>Congratulations!</h1>
<p>From port 80 here</p>
<img src=congratulations.jpg>

</center></body></html>
```

On node3:

lynx 10.1.2.4

It gets the default web page containing its photo of a happy computer "congratulations.jpg." In lynx it looks like this:



while in a graphical browser it would have this appearance:

(If you want to see the result graphically, you could start the graphical interface on node3 and use firefox instead of lynx. But memory constraint is the reason for sticking with the character mode interface. It may be very sluggish. If node 3 were provisioned in VirtualBox settings with enough memory it would work fine but here we are trying to be memory-economical, and the character mode result fully makes the point.)

node2 also runs a web server, and makes available a different image. In the node1 terminal window running ettercap, terminate it with the "q" key. Then run:

```
sudo su
etterfilter  ~/laughingskull.filter   -o ~/laughingskull
ettercap -i eth0 -T -M arp:remote /10.1.1.3// /10.1.1.6// -F ~/laughingskull
```

Ettercap has a filter capability, with a filter language and the "etterfilter" compiler for it. Glance at the filter we will use:

```
cat  ~/laughingskull.filter
```

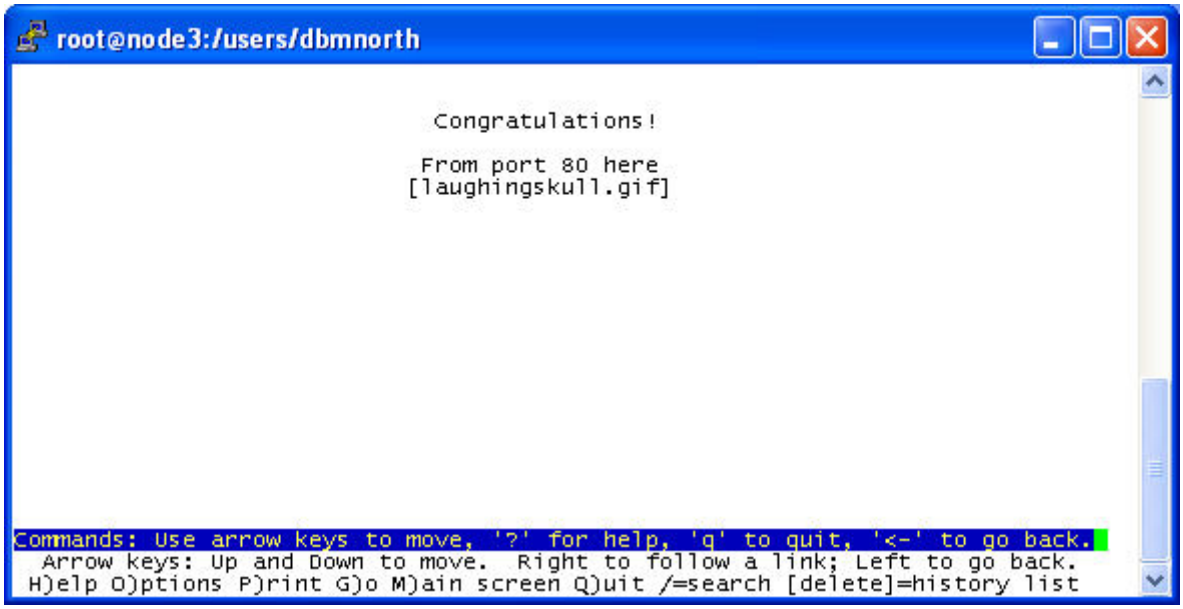
It expresses:

search for: img src=
replace with: img src=http://10.1.1.2/laughingskull.gif

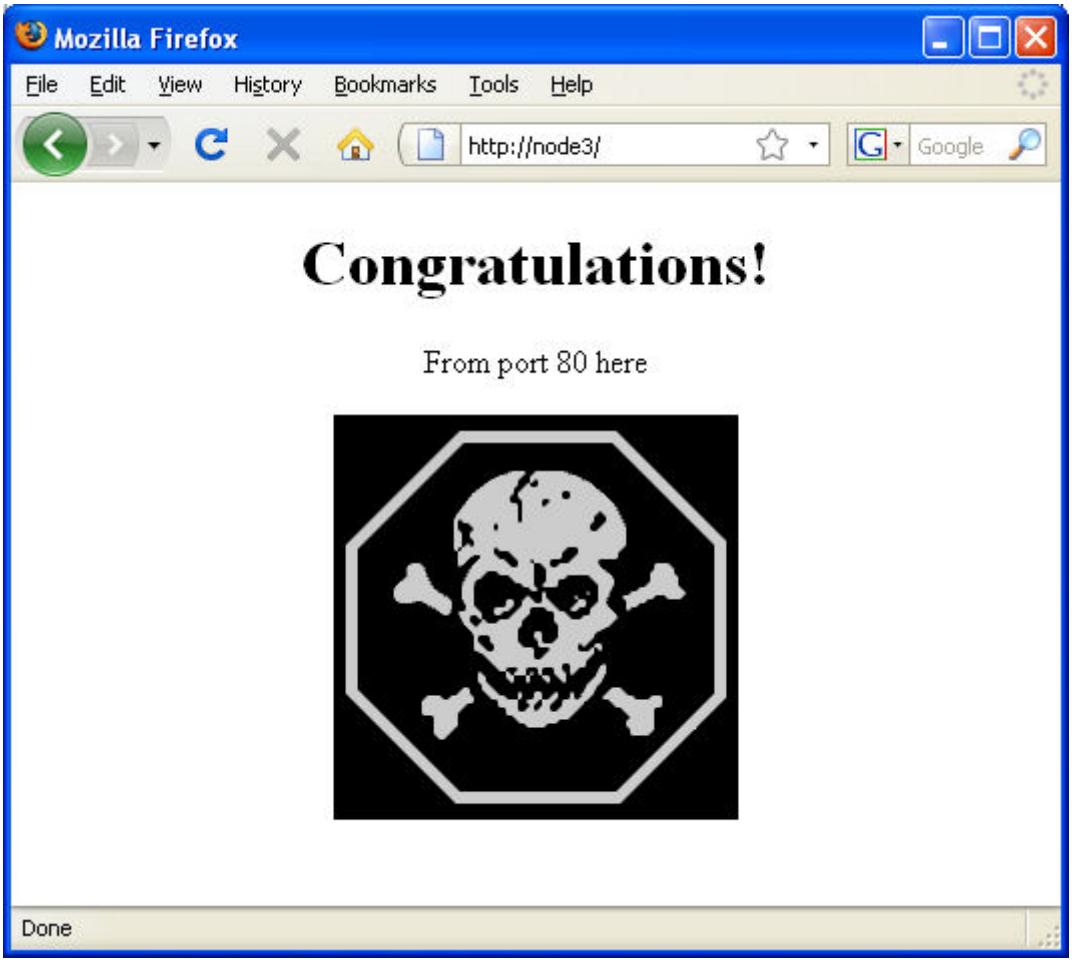
This inserts the appropriate html to make the page fetch our image, foreign to the one it's written to fetch. Run the web browser on node3 to browse node4 again. On node3:

```
lynx  10.1.2.4
```

It gets the default web page containing its photo of a laughing skull "laughingskull.gif." In lynx it looks like this:



while in a graphical browser it would have this appearance:



Ettercap has caused the client browser's received page to differ from the server's sent one. It did it by 1) datastream diversion to/through itself, and 2) dynamic search and replace on the datastream's in-transit html.

When you are finished you can poweroff all the virtual machines, from the host machine's prompt (linux or Apple "terminal," Windows "command box"):

vmconfigure-poweroff

and further, if you don't plan to return to the experiment, clean away the VMs:

vmconfigure-destroy

But further, if you don't want to destroy the exercise and might want to return to it, you will need at that time to re-run vmconfigure-guestOS-internal-settings to restore transient settings (like IP addresses and route tables) that are lost upon poweroff.

What can go wrong

screen gets messed up or unresponsive following a command - issue the "reset" command (blindly type it even if the screen won't show the echo).

Questions for you to answer

1. If node1 is a "man in the middle" then node4 is an "odd man out." In particular, node 4 was unaccounted for in section 3 "Recording actual address mappings." Later you arp poisoned node2 and node0 from node1; how about arp poisoning node 4 from node1? You accomplish poisoning by sending a crafted arp message to a node. Comment on the ways and means of poisoning node4 from node1.

2. Answer the question at the end of section 6. Under the circumstances of that section, "How does traffic between node2 and node0 get from node2 to node0?"

3. Answer the question at the end of section 7, "How?" Recall that node2 logged into ftp on node4 and somehow node1 figured out the user password given by node2. How?

4. Imagine you run a web hosting company. The manager at one of your clients, a medium sized business, calls you in alarm and reports the apparent defacement of his website running on your host machine. Images on the site have all been replaced with various hacker images like the laughing skull. He heard about it from several of his employees, then saw it with his own eyes on their terminals. His website has fallen victim to the same mischief as the one on our node4. What is your course of action?