Evaluation network lab

Local DNS Attack

Overview

DNS (Domain Name System) is the Internet's phone book; it translates hostnames to IP addresses (and vice versa). This translation is through DNS resolution, which happens behind the scene. DNS attacks manipulate this resolution process in various ways, with an intent to misdirect users to alternative destinations, which are often malicious. The objective of this first part is to understand how such attacks work. Students will first set up and configure a DNS server, and then they will try various DNS attacks on the target that is also within the lab environment.

The difficulties of attacking local victims versus remote DNS servers are quite different.

This first part covers the following topics:

- DNS and how it works
- DNS server setup
- DNS cache poisoning attack
- Spoofing DNS responses
- Packet sniffing and spoofing
- The Scapy tool

(Part I): Setting Up a Local DNS Server

The main purpose of this part is on DNS attacks, and our attacking target is a local DNS server. Obviously, it is illegal to attack a real machine, so we need to set up our own DNS server to conduct the attack experiments.

The lab environment needs three separate machines:

- one for the victim,
- one for the DNS server,
- and the other for the attacker.

We will run these three virtual machines on one physical machine.

For the VM network setting, if you are using VirtualBox, please use "NAT Network" as the network adapter for each VM. If you are using Vmware, the default "NAT" setting is good enough.

For the sake of simplicity, we put all these VMs on the same network. In the following sections, we assume that the user machine's IP address is 10.0.2.18, the DNS Server's IP is 10.0.2.16 and the attacker machine's IP is 10.0.2.17. We need to configure the user machine and the local DNS server; for the attacker machine, the default setup in the VM should be sufficient.

Task 1: Configure the User Machine

On the user machine 10.0.2.18, we need to use 10.0.2.16 as the local DNS server.

This is achieved by changing the resolver configuration file *(/etc/resolv.conf)* of the user machine, so the server 10.0.2.16 is added as the first nameserver entry in the file, i.e., this server will be used as the primary DNS server. Unfortunately, our provided VM uses the Dynamic Host Configuration Protocol (DHCP) to obtain network configuration parameters, such as IP address, local DNS server, etc. DHCP clients will overwrite the */etc/resolv.conf* file with the information provided by the DHCP server.

One way to get our information into */etc/resolv.conf* without worrying about the DHCP is to add the following entry to the */etc/resolvconf/resolv.conf.d/*head file:

```
Add the following entry to /etc/resolvconf/resolv.conf.d/head
  nameserver 10.0.2.16
Run the following command for the change to take effect
$ sudo resolvconf -u
```

The content of the head file will be prepended to the dynamically generated resolver configuration file. Normally, this is just a comment line (the comment in /etc/resolv.conf comes from this head file).

After you finish configuring the user machine, use the dig command to get an IP address from a hostname of your choice. From the response, please provide evidences to show that the response is indeed from your server. If you cannot find the evidence, your setup is not successful.

Task 2: Set up a Local DNS Server

For the local DNS server, we need to run a DNS server program. The most widely used DNS server software is called BIND (Berkeley Internet Name Domain), which, as the name suggests, was originally designed at the University of California Berkeley in the early 1980s.

The latest version of BIND is BIND 9, which was first released in 2000. We will show how to configure BIND 9 for our lab environment.

Step 1: Configure the BIND 9 server. BIND 9 gets its configuration from a file called */etc/bind/ named.conf.* This file is the primary configuration file, and it usually contains several "include" entries, i.e., the actual configurations are stored in those included files. One of the included files is called */etc/bind/named.conf.options.*

This is where we typically set up the configuration options. Let us first set up an option related to DNS cache by adding a dump-file entry to the options block:

```
options {
   dump-file "/var/cache/bind/dump.db";

};
```

The above option specifies where the cache content should be dumped to if BIND is asked to dump its cache. If this option is not specified, BIND dumps the cache to a default file called */var/cache/bind/ named_dump.db*. The two commands shown below are related to DNS cache.

The first command dumps the content of the cache to the file specified above, and the second command clears the cache.

$ sudo rndc dumpdb -cache // Dump the cache to the sepcified file
$ sudo rndc flush // Flush the DNS cache

Step 2: Turn off DNSSEC. DNSSEC is introduced to protect against spoofing attacks on DNS servers. To show how attacks work without this protection mechanism, we need to turn the protection off. This is done by modifying the named.conf.options file: comment out the dnssec-validation entry, and add a dnssec-enable entry.

```
options {
   # dnssec-validation auto;
   dnssec-enable no;

};
```

Step 3: Start DNS server. We can now start the DNS server using the following command. Every time a modification is made to the DNS configuration, the DNS server needs to be restarted. The following command will start or restart the BIND 9DNSserver.

$ sudo service bind9 restart

Step 4: Use the DNS server. Now, go back to your user machine, and ping a computer such as www. google.com and www.facebook.com, and describe your observation. Please use Wireshark to show the DNS query triggered by your ping command. Please also indicate when the DNS cache is used.

Task 3: Host a Zone in the Local DNS Server

Assume that we own a domain, we will be responsible for providing the definitive answer regarding this domain. We will use our local DNS server as the authoritative nameserver for the domain.

In this lab, we will set up an authoritative server for the example.com domain. This domain name is reserved for use in documentation, and is not owned by anybody, so it is safe to use it.

Step 1: Create zones. We need to create two zone entries in the DNS server by adding the following contents to */etc/bind/named.conf.* The first zone is for forward lookup (from hostname to IP), and the second zone is for reverse lookup (from IP to hostname). It should be noted that the example.com domain name is reserved for use in documentation, and is not owned by anybody, so it is safe to use it.

```
zone "example.com" {
      type master;
      file "/etc/bind/example.com.db";
    };
```

```
zone "0.168.192.in-addr.arpa" {
      type master;
      file "/etc/bind/192.168.0.db";
    };
```

Step 2: Setup the forward lookup zone file. The file name after the file keyword in the above zone definition is called the zone file, and this is where the actual DNS resolution is stored. In the

/etc/bind/ directory, create the following example.com.db zone file. Readers who are interested in the syntax of the zone file, can refer to RFC 1035 for details.

```
$TTL 3D ; default expiration time of all resource records without
@ IN 1

8H 2H 4W

SOA

ns.example.com. admin.example.com. (
; Serial
; Refresh
; Retry

; Expire

;   their own TTL
1D )

    • @  IN NS
    • @  IN MX

www IN A

; Minimum

ns.example.com.      ;Address of nameserver
10 mail.example.com.  ;Primary Mail Exchanger
192.168.0.101   ;Address of www.example.com
```

```
mail   IN     A
ns     IN     A
*.example.com. IN A
192.168.0.102
192.168.0.10
192.168.0.100
;Address of mail.example.com
;Address of ns.example.com
;Address for other URL in
;  the example.com domain
```

The symbol '@' is a special notation representing the origin specified in named.conf (the string after "zone"). Therefore, '@' here stands for example.com. This zone file contains 7 resource records (RRs), including a SOA (Start Of Authority) RR, a NS (Name Server) RR, a MX (Mail eXchanger) RR, and 4 A (host Address) RRs.

Step 3: Set up the reverse lookup zone file. To support DNS reverse lookup, i.e., from IP address to hostname, we also need to set up the DNS reverse lookup file. In the /etc/bind/ directory, create the following reverse DNS lookup file called 192.168.0.db for the example.net domain:

```
$TTL 3D
@      IN     SOA     ns.example.com. admin.example.com. (
```

```
$TTL 3D
@      IN     SOA     ns.example.com. admin.example.com. (
```

```
8H 2H 4W 1D)
@ IN NS

    101.        101  IN PTR
    102.        102  IN PTR

10 IN PTR

ns.example.com.
www.example.com.
mail.example.com.
ns.example.com.
```

Step 4: Restart the BIND server and test. When all the changes are made, remember to restart the BIND server. Now, go back to the user machine, and ask the local DNS server for the IP address of www. example.com using the dig command. Please describe and explain your observations.

Attacks on DNS

The main objective of DNS attacks on a user is to redirect the user to another machine B when the user tries to get to machine A using A's host name. For example, when the user tries to access the online banking, if the adversaries can redirect the user to a malicious web site that looks very much like the main web site of bank, the user might be fooled and give away password of his/her online banking account.

When a user types in http://www.example.net in his/her browsers, the user's machine will issue a DNS query to find out the IP address of this web site. Attackers' goal is to fool the user's machine with a faked DNS reply, which resolves the hostname to a malicious IP address. There are several ways to launch such a DNS attack. See Figure 2 for the illustration of the attack surface and read Chapter 15 of the SEED book for detailed analysis of the attack surface.

We will launch a series DNS attacks on the example.net domain. It should be noted that we are using example.net as our target domain, not the example.com. The latter one is already hosted by our own local DNS server in the setup, so no DNS query will be sent out for hostnames in that domain.

Task 4: Modifying the Host File

The host name and IP address pairs in the HOSTS file (/etc/hosts) are used for local lookup; they take the preference over remote DNS lookups. For example, if there is a following entry in the HOSTS file in the user's computer, the www.example.com will be resolved as 1.2.3.4 in user's computer without asking any DNS server:

1.2.3.4 www.example.net

If attackers have compromised a user's machine, they can modify the HOSTS file to redirect the user to a malicious site whenever the user tries to access www.example.com. Assume that you have already compromised a machine, please try this technique to redirect www.bank32.com to any IP address that you choose.

It should be noted that /etc/hosts is ignored by the dig command, but will take effect on the ping command and web browser etc. Compare the results obtained before and after the attack.

Task 5: Directly Spoofing Response to User

In this attack, the victim's machine has not been compromised, so attackers cannot directly change the DNS query process on the victim's machine. However, if attackers are on the same local area network as the victim, they can still achieve a great damage.

When a user types the name of a web site (a host name, such as www.example.net) in a web browser, the user's computer will issue a DNS request to the DNS server to resolve the IP address of the host name. After hearing this DNS request, the attackers can spoof a fake DNS response (see Figure 3). The fake DNS response will be accepted by the user's computer if it meets the following criteria:

1. The source IP address must match the IP address of the DNS server.
2. The destination IP address must match the IP address of the user's machine.
3. The source port number (UDP port) must match the port number that the DNS request was sent to (usually port 53).
4. The destination port number must match the port number that the DNS request was sent from.
5. The UDP checksum must be correctly calculated.
6. The transaction ID must match the transaction ID in the DNS request.
7. The domain name in the question section of the reply must match the domain name in the question section of the request.
8. The domain name in the answer section must match the domain name in the question section of the DNS request.
9. The User's computer must receive the attacker's DNS reply before it receives the legitimate DNS response.

To satisfy the criteria 1 to 8, the attackers can sniff the DNS request message sent by the victim; they can then create a fake DNS response, and send back to the victim, before the real DNS server does. Netwox tool 105 provide a utility to conduct such sniffing and responding. We can make up any arbitrary DNS answer in the reply packets. Moreover, we can use the "filter" field to specify what kind of packets to sniff. For example, by using"src host 10.0.2.18", we can limit the scope of our sniffing to packets only from host 10.0.2.18. The manual of the tool is described in the following:

```
Title: Sniff and send DNS answers
   Usage: netwox 105 -h data -H ip -a data -A ip [-d device]

Parameters:

[-T uint32] [-f filter] [-s spoofip]
```

-h|--hostname data -H|--hostnameip ip -a|--authns data -A|--authnsip ip -d|--device device -T|--ttl uint32 hostname IP address authoritative nameserver authns IP device name ttl in seconds
-f|--filter filter pcap filter -s|--spoofip spoofip IP spoof initialization type

While the attack program is running, on the user machine, you can run dig command on behalf of the user. This command triggers the user machine to send out a DNS query to the local DNS server, which will eventually send out a DNS query to the authoritative nameserver of the example.net domain (if the cache does not contain the answer). If your attack is successful, you should be able to see your spoofed information in the reply. Compare your results obtained before and after the attack.

Task 6: DNS Cache Poisoning Attack

The above attack targets the user's machine. In order to achieve long-lasting effect, every time the user's machine sends out a DNS query for www.example.net the attacker's machine must send out a spoofed DNS response. This might not be so efficient; there is a much better way to conduct attacks by targeting the DNS server, instead of the user's machine.

When a DNS server Apollo receives a query, if the host name is not within the Apollo's domain, it will ask other DNS servers to get the host name resolved. Note that in our lab setup, the domain of our DNS server is example.com; therefore, for the DNS queries of other domains (e.g. example.net), the DNS server Apollo will ask other DNS servers. However, before Apollo asks other DNS servers, it first looks for the answer from its own cache; if the answer is there, the DNS server Apollo will simply reply with the information from its cache. If the answer is not in the cache, the DNS server will try to get the answer from other DNS servers. When Apollo gets the answer, it will store the answer in the cache, so next time, there is no need to ask other DNS servers. See Figure 3.

Therefore, if attackers can spoof the response from other DNS servers, Apollo will keep the spoofed response in its cache for certain period of time. Next time, when a user's machine wants to resolve the same host name, Apollo will use the spoofed response in the cache to reply. This way, attackers only need to spoof once, and the impact will last until the cached information expires. This attack is called *DNS cache poisoning*.

We can use the same tool (Netwox 105) for this attack. Before attacking, make sure that the DNS Server's cache is empty. You can flush the cache using the following command:

$ sudo rndc flush

The difference between this attack and the previous attack is that we are spoofing the response to DNS server now, so we set the filter field to "src host 192.168.0.10",which is the IP address of the DNS server. We also use the ttl field (time-to-live) to indicate how long we want the fake answer to stay in the DNS server's cache. After the DNS server is poisoned, we can stop the Netwox 105 program. If we set ttl to 600 (seconds), then DNS server will keep giving out the fake answer for the next 10 minutes.

Note: Please select the raw in the spoofip field; otherwise, Netwox 105 will try to also spoof the MAC address for the spoofed IP address. To get the MAC address, the tool sends out an ARP request, asking for the MAC address of the spoofed IP. This spoofed IP address is usually the IP address of an external DNS server, which is not on the same LAN. Therefore, nobody will reply the ARP request. The tool will wait for the ARP reply for a while before going ahead without the MAC address. The waiting will delay the tool from sending out the spoofed response. If the actual DNS response comes earlier than the spoofed response, the attack will fail. That's why you need to ask the tool not to spoof the MAC address.

You can tell whether the DNS server is poisoned or not by observing the DNS traffic using Wireshark when you run the dig command on the target hostname. You should also dumping the local DNS server's cache to check whether the spoofed reply is cached or not. To dump and view the DNS server's cache, issue the following command:

Task 7: DNS Cache Poisoning: Targeting the Authority Section

In the previous task, our DNS cache poisoning attack only affects one hostname, i.e., www.example.net. If users try to get the IP address of another hostname, such as mail.example.net, we need to launch the attack again. It will be more efficient if we launch one attack that can affect the entire example.net domain.

The idea is to use the Authority section in DNS replies. Basically, when we spoofed a reply, in addition to spoofing the answer (in the Answer section), we add the following in the Authority section. When this entry is cached by the local DNS server, ns.attacker32.com will be used as the nameserver for future queries of any hostname in the example.net domain. Since attacker32.com is a machine controlled by attackers, it can provide a forged answer for any query.

$ sudo rndc dumpdb -cache
$ sudo cat /var/cache/bind/dump.db

```
;; AUTHORITY SECTION:
example.net.        259200  IN      NS      attacker32.com.
```

The purpose of this task is to conduct such as an attack. You need to demonstrate that you can get the above entry cached by the local DNS server. After the cache is poisoned, run a dig command on any hostname in the example.net domain, and use Wireshark to observe where the DNS query goes. It should be noted that the attacker32.com is owned by Wenliang Du, the author of the SEED labs, but this machine is not set up to serve as a DNS server. Therefore, you will not be able to get a answer from it, but your Wireshark traffic should be able to tell you whether your attack is successful or not.

You need to use Scapy for this task. See the Guideline section for a sample code.

Task 8: Targeting Another Domain

In the previous attack, we successfully poison the cache of the local DNS server, so attacker32.com becomes the nameserver for the example.com domain. Inspired by this success, we would like to extend its impact to other domain. Namely, in the spoofed response triggered by a query for www.example.net, we would like to add additional entry in the Authority section (see the following), so attacker32.com is also used as the nameserver for google.com.

Please use Scapy to launch such an attack on your local DNS server; describe and explain your observa- tion. It should be noted that the query that we are attacking is still the query to example.net, not one to google.com.

Task 9: Targeting the Additional Section

In DNS replies, there is section called Additional Section, which is used to provide additional information. In practice, it is mainly used to provide IP addresses for some hostnames, especially for those appearing in the Authority section. The goal of this task is to spoof some entries in this section and see whether they will be successfully cached by the target local DNS server. In particular, when responding to the query for www.example.net, we add the following entries in the spoofed reply, in addition to the entries in the Answer section:

Entries and are related to the hostnames in the Authority section. Entry is completely irrelevant to any entry in the reply, but it provides a “gracious” help to users, so they do not need to look up for the IP address of Facebook. Please use Scapy to spoof such a DNS reply. Your job is to report what entries will be successfully cached, and what entries will not be cached; please explain why.

What’s Next

In the DNS cache poisoning attack of this lab, we assume that the attacker and the DNS server are on the same LAN, i.e., the attacker can observe the DNS query message. When the attacker and the DNS server

```
;; AUTHORITY SECTION:
example.net.        259200  IN    NS    attacker32.com.
google.com.         259200  IN    NS    attacker32.com.
```

re not on the same LAN, the cache poisoning attack becomes much more challenging. If you are interested in taking on such a challenge, you can try our “Remote DNS Attack Lab”.

Guideline

You need to use Scapy for several tasks in this lab. The following sample code shows how to sniff a DNS query and then spoof a DNS reply, which contains a record in the Answer section, two records in the Authority section and two records in the Additional section.

```
#!/usr/bin/python
from scapy.all import *
```

```
def spoof_dns(pkt):
  if (DNS in pkt and 'www.example.net' in pkt[DNS].qd.qname):
    # Swap the source and destination IP address
    IPpkt = IP(dst=pkt[IP].src, src=pkt[IP].dst)
```

```
# Swap the source and destination port number
UDPpkt = UDP(dport=pkt[UDP].sport, sport=53)
# The Answer Section
Anssec = DNSRR(rrname=pkt[DNS].qd.qname, type='A',
ttl=259200, rdata='10.0.2.5')
# The Authority Section
NSsec1 = DNSRR(rrname='example.net', type='NS',
          ttl=259200, rdata='ns1.example.net')
NSsec2 = DNSRR(rrname='example.net', type='NS',
ttl=259200, rdata='ns2.example.net')
# The Additional Section
Addsec1 = DNSRR(rrname='ns1.example.net', type='A',
           ttl=259200, rdata='1.2.3.4')
Addsec2 = DNSRR(rrname='ns2.example.net', type='A',
ttl=259200, rdata='5.6.7.8')

# Construct the DNS packet
DNSpkt = DNS(id=pkt[DNS].id, qd=pkt[DNS].qd, aa=1, rd=0, qr=1, 

         qdcount=1, ancount=1, nscount=2, arcount=2,
         an=Anssec, ns=NSsec1/NSsec2, ar=Addsec1/Addsec2)
# Construct the entire IP packet and send it out
  spoofpkt = IPpkt/UDPpkt/DNSpkt
  send(spoofpkt)
# Sniff UDP query packets and invoke spoof_dns().
pkt = sniff(filter='udp and dst port 53', prn=spoof_dns)
```

Line constructs the DNS payload, including DNS header and data. Each field of the DNS payload is explained in the following:

- id: Transaction ID; should be the same as that in the request.
- qd: Query Domain; should be the same as that in the Request.
- aa: Authoritative answer (1 means that the answer contains Authoritative answer).
- rd: Recursion Desired (0 means to disable Recursive queries).
- qr: Query Response bit (1 means Response).
- qdcount: number of query domains.
- ancount: number of records in the Answer section.
- nscount: number of records in the Authority section.
- arcount: number of records in the Additional section.
- an: Answer section
- ns: Authority section
- ar: Additional section

Submission

You need to submit a detailed lab report, with screenshots, to describe what you have done and what you have observed. You also need to provide explanation to the observations that are interesting or surprising. Please also list the important code snippets followed by explanation. Simply attaching code without any explanation will not receive any scoring.

Second part of the TP

The learning objective of this lab is for students to gain the insights on how firewalls work by playing with firewall software and implement a simplified packet filtering firewall. Firewalls have several types; in this lab, we focus on *packet filter*. Packet filters inspect packets, and decide whether to drop or forward a packet based on firewall rules. Packet filters are usually *stateless*; they filter each packet based only on the information contained in that packet, without paying attention to whether a packet is part of an existing stream of traffic. Packet filters often use a combination of a packet's source and destination address, its protocol, and, for TCP and UDP traffic, port numbers. In this lab, students will play with this type of firewall, and also through the implementation of some of the key functionalities, they can understand how firewalls work. Moreover, students will learn how to use SSH tunnels to bypass firewalls. This lab covers the following topics:

- Firewall
- Netfilter
- Loadable kernel module
- Bypassing firewalls using SSH tunnel
- How to bypassing firewalls using VPN is covered in a separate lab.

Lab Tasks

Task 1: Using Firewall

Linux has a tool called iptables, which is essentially a firewall. In this task, the objective is to use iptables to set up some firewall policies, and observe the behaviors of your system after the policies become effective. You need to set up at least two VMs, one called Machine A, and other called Machine B. You run the firewall on your Machine A. Basically, we use iptables as a personal firewall. Optionally, if you have more VMs, you can set up a firewall at your router, so it can protect a network, instead of just one single computer. After you set up the two VMs, you should perform the following tasks:

- Prevent A from doing telnet to Machine B.

- Prevent B from doing telnet to Machine A.

- Prevent A from visiting an external web site. You can choose any web site that you like to block, but keep in mind, some web servers have multiple IP addresses.

You can find the manual of iptables by typing "man iptables" or search it online. We list some commonly used commands in the following:

// List all the rules in the filter table

$ sudo iptables -L
$ sudo iptables -L --line-numbers
// Delete all the rules in the filter table
$ sudo iptables -F

// Delete the 2nd rule in the INPUT chain of the filter table
$ sudo iptables -D INPUT 2
// Drop all the incoming packets that satisfy the <rule>

```
$ sudo iptables -A INPUT <rule>  -j DROP
```

Task 2: Implementing a Simple Firewall

The firewall you used in the previous task is a packet filtering type of firewall. The main part of this type of firewall is the filtering part, which inspects each incoming and outgoing packets, and enforces the firewall policies set by the administrator. Since the packet processing is done within the kernel, the filtering must also be done within the kernel. Therefore, it seems that implementing such a firewall requires us to modify the Linux kernel. In the past, this had to be done by modifying and rebuilding the kernel. The modern Linux operating systems provide several new mechanisms to facilitate the manipulation of packets without rebuilding the kernel image. These two mechanisms are *Loadable Kernel Module* (LKM) and Netfilter.

LKM allows us to add a new module to the kernel at the runtime. This new module enables us to extend the functionalities of the kernel, without rebuilding the kernel or even rebooting the computer. The packet filtering part of a firewall can be implemented as an LKM. However, this is not enough. In order for the filtering module to block incoming/outgoing packets, the module must be inserted into the packet processing path. This cannot be easily done in the past before the Netfilter was introduced into the Linux.

Netfilter is designed to facilitate the manipulation of packets by authorized users. Netfilter achieves this goal by implementing a number of *hooks* in the Linux kernel. These hooks are inserted into various places, including the packet incoming and outgoing paths. If we want to manipulate the incoming packets, we simply need to connect our own programs (within LKM) to the corresponding hooks. Once an incoming packet arrives, our program will be invoked. Our program can decide whether this packet should be blocked or not; moreover, we can also modify the packets in the program.

In this task, you need to use LKM and Netfilter to implement the packet filtering module. This module will fetch the firewall policies from a data structure, and use the policies to decide whether packets should be blocked or not. To make your life easier, so you can focus on the filtering part, the core of firewalls, we allow you to hardcode your firewall policies in the program. You should support at least five different rules, including the ones specified in the previous task.

Task 3: Evading Egress Filtering

Many companies and schools enforce egress filtering, which blocks users inside of their networks from reaching out to certain web sites or Internet services. They do allow users to access other web sites. In many cases, this type of firewalls inspect the destination IP address and port number in the outgoing packets. If a packet matches the restrictions, it will be dropped. They usually do not conduct deep packet inspections (i.e., looking into the data part of packets) due to the performance reason. In this task, we show how such egress filtering can be bypassed using the tunnel mechanism. There are many ways to establish tunnels; in this task, we only focus on SSH tunnels.

You need two VMs A and B for this task (three will be better). Machine A is running behind a firewall (i.e., inside the company or school's network), and Machine B is outside of the firewall. Typically, there is a dedicated machine that runs the firewall, but in this task, for the sake of convenience, you can run the firewall on Machine A. You can use iptables to set up the following two firewall rules:

- Block all the outgoing traffic to external telnet servers. In reality, the servers that are blocked are usu- ally game servers or other types of servers that may affect the productivity of employees. In this task, we use the telnet server for demonstration purposes. You can run the telnet server on Machine B. If you have a third VM, Machine C, you can run the telnet server on Machine C.
- Block all the outgoing traffic to www.facebook.com, so employees (or school kids) are not distracted during their work/school hours. Social network sites are commonly blocked by companies and schools. After you set up the firewall, launch your Firefox browser, and try to connect to Face- book, and report what happens. If you have already visited Facebook before using this browser, you need to clear all the caches using Firefox's menu: Edit -> Preferences -> Privacy & Security (left pane) -> Clear History (Button on right); otherwise, the cached pages may be displayed. If everything is set up properly, you should not be able to see Facebook pages. It should be noted that Facebook has many IP addresses, it can change over the time. Remember to check whether the address is still the same by using ping or dig command. If the address has changed, you need to update your firewall rules. You can also choose web sites with static IP addresses, instead of using Facebook. For example, most universities' web servers use static IP addresses (e.g. www.syr.edu); for demonstration purposes, you can try block these IPs, instead of Facebook.

Task 3.a: Telnet to Machine B through the firewall To bypass the firewall, we can establish an SSH tunnel between Machine A and B, so all the telnet traffic will go through this tunnel (encrypted), evading the inspection. Figure 1 illustrates how the tunnel works. The following command establishes an SSH tunnel between the localhost (port 8000) and machine B (using the default port 22); when packets come out of B's end, it will be forwarded to Machine C's port 23 (telnet port). If you only have two VMs, you can use one VM for both Machine B and Machine C.

```
$ ssh -L 8000:Machine_C_IP:23  seed@Machine_B_IP
// We can now telnet to Machine C via the tunnel:
$ telnet localhost 8000
```

When we telnet to localhost's port 8000, SSH will transfer all our TCP packets from one end of the tunnel on localhost:8000 to the other end of the tunnel on Machine B; from there, the packets will be forwarded to Machine C:23. Replies from Machine C will take a reverse path, and eventually reach our telnet client. Essentially, we are able to telnet to Machine C. Please describe your observation and explain how you are able to bypass the egress filtering. You should use Wireshark to see what exactly is happening on the wire.

Task 3.b: Connect to Facebook using SSH Tunnel. To achieve this goal, we can use the approach similar to that in Task 3.a, i.e., establishing a tunnel between your localhost:port and Machine B, and ask B to forward packets to Facebook. To do that, you can use the following command to set up the tunnel: "ssh -L 8000:FacebookIP:80 ...". We will not use this approach, and instead, we use a more generic approach, called dynamic port forwarding, instead of a static one like that in Task 3.a. To do that, we only specify the local port number, not the final destination. When Machine B receives a packet from the tunnel, it will dynamically decide where it should forward the packet to based on the destination information of the packet.

$ ssh -D 9000 -C seed@machine_B

Similar to the telnet program, which connects localhost:9000, we need to ask Firefox to connect to localhost:9000 every time it needs to connect to a web server, so the traffic can go through our SSH

tunnel. To achieve that, we can tell Firefox to use localhost:9000 as its proxy. To support dynamic port forwarding, we need a special type of proxy called *SOCKS proxy*, which is supported by most browsers. To setup the proxy in Firefox, go to the menubar, clickEdit -> Preferences, scroll down to Network Proxy, and click the Settings button. Then follow Figure 2. After the setup is done, please do the following:

1. Run Firefox and go visit the Facebook page. Can you see the Facebook page? Please describe your observation.
2. Afteryougetthefacebookpage,breaktheSSHtunnel,cleartheFirefoxcache,andtrytheconnection again. Please describe your observation.
3. Establish the SSH tunnel again and connect to Facebook. Describe your observation.
4. Please explain what you have observed, especially on why the SSH tunnel can help bypass the egress filtering. You should use Wireshark to see what exactly is happening on the wire. Please describe your observations and explain them using the packets that you have captured.

Task 4: Evading Ingress Filtering

Machine A runs a web server behind a firewall; so only the machines in the internal network can access this web server. You are working from home and needs to access this internal web server. You do not have VPN, but you have SSH, which is considered as a poor man's VPN. You do have an account on Machine A (or another internal machine behind the firewall), but the firewall also blocks incoming SSH connection, so you cannot SSH into any machine on the internal network. Otherwise, you can use the same technique from Task 3 to access the web server. The firewall, however, does not block outgoing SSH connection, i.e., if you want to connect to an outside SSH server, you can still do that.

The objective of this task is to be able to access the web server on Machine A from outside. We will use two machines to emulate the setup. Machine A is the internal computer, running the protected web server; Machine B is the outside machine at home. On Machine A, we block Machine B from accessing its port 80 (web server) and 22 (SSH server). You need to set up a reverse SSH tunnel on Machine A, so once you get home, you can still access the protected web server on A from home.

Guidelines

Loadable Kernel Module

The following is a simple loadable kernel module. It prints out "Hello World!" when the module is loaded; when the module is removed from the kernel, it prints out "Bye-bye World!". The messages are not printed out on the screen; they are actually printed into the /var/log/syslog file. You can use dmesg | tail -10toreadthelast10linesofmessage.

```
#include <linux/module.h>
#include <linux/kernel.h>
int init_module(void)
{
printk(KERN_INFO "Hello World!\n");

return 0; }

void cleanup_module(void)
{
printk(KERN_INFO "Bye-bye World!.\n");
```

```
}
```

We now need to create Makefile, which includes the following contents (the above program is named hello.c). Then just type make, and the above program will be compiled into a loadable kernel module.

```
obj-m += hello.o
all:
clean:

make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

Once the module is built by typing make, you can use the following commands to load the module, list all modules, and remove the module:

$ sudo insmod mymod.ko
$ lsmod
$ sudo rmmod mymod.ko
(inserting a module)
(list all modules)
(remove the module)

Also, you can use modinfo my mod.ko to show information about a Linux Kernel module.

A Simple Program that Uses Netfilter

Using Netfilter is quite straightforward. All we need to do is to hook our functions (in the kernel module) to the corresponding Netfilter hooks. Here we show an example:

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/netfilter.h>
#include <linux/netfilter_ipv4.h>
/* This is the structure we shall use to register our function */
static struct nf_hook_ops nfho;
/* This is the hook function itself */
unsigned int hook_func(void *priv, struct sk_buff *skb,
                       const struct nf_hook_state *state)
{
   /* This is where you can inspect the packet contained in
      the structure pointed by skb, and decide whether to accept
      or drop it. You can even modify the packet */
// In this example, we simply drop all packets
   return NF_DROP;          /* Drop ALL packets */
}
/* Initialization routine */
int init_module()
{   /* Fill in our hook structure */
   nfho.hook = hook_func;        /* Handler function */
   nfho.hooknum  = NF_INET_PRE_ROUTING; /* First hook for IPv4 */
   nfho.pf       = PF_INET;
   nfho.priority = NF_IP_PRI_FIRST;   /* Make our function first */
```

```
nf_register_hook(&nfho);

return 0; }

/* Cleanup routine */
void cleanup_module()
{
   nf_unregister_hook(&nfho);

}
```

Submission

You need to submit a detailed lab report, with screenshots, to describe what you have done and what you have observed. You also need to provide explanation to the observations that are interesting or surprising. Please also list the important code snippets followed by explanation. Simply attaching code without any explanation will not receive any scoring.