CSci 4271W Developing Secure Software Systems (section 010)

Homework 4 **Due: March 25th, 2025**

Ground Rules. You may choose to complete these exercises in a group of up to three students. Each group should turn in **one** copy with the names of all group members on it. You may use any source you can find to help with this assignment but you **must** explicitly reference any source you use besides the lecture notes or assigned readings. No answers should come from people outside your group, or from AI tools like ChatGPT. If you use an AI tool to revise your writing, save a copy of the first draft you wrote yourself to provide it was originally your work. Electronically typeset copies of your solution should be submitted via Gradescope by 11:59pm on March 25th, 2025.

1. **Isolation.** In the last homework, we learned about confused deputies: A "Confused Deputy" bug happens when a program that needs access to a resource or file for one reason can be "manipulated" into accessing a resource or file inappropriately. For example, we saw the `set-motd` program that ran as `root` in order to have permission to write to the login message (Message of the day) file `/etc/motd`; this program could lead to vulnerabilities due to the overly broad access that comes from having UID 0. Let's think about how some of the isolation techniques we discussed in Lecture 12 might or might not help with this problem:

   (a) Mandatory Access Controls (MAC): Suppose we were to write an AppArmor profile for `set-motd`. Could this effectively mitigate the confused deputy problem? What would you put in the profile?

   (b) System Call Sandboxing: What about a seccomp filter?

   (c) Containers and virtualization: would a container or VM be an effective or useful tool in this context? Why or why not?

2. **Networking Exercise**. Login to one of the Keller 1-250 lab machines via ssh – for example, in a Terminal window run the command

   ```
   $ ssh (yourid)@csel-kh1250-07.cselabs.umn.edu
   ```

   From this terminal, let's see some tools that let us interact with the link, network, and domain-layer routing infrastructures in TCP/IP.

   (a) At the network layer, it can be useful to know what IP links a packet traverses on its way to a given destination. `traceroute` is a command line tool that will allow us to do that. Try finding out how packets get from Keller 1-250 to the University of Tokyo by running the command `traceroute www.u-tokyo.ac.jp`. The Nth line of the output of this command represents an attempt to send a packet N hops along the way to its destination; an entry of the form `domain.name (ip.ad.dr.ess) XX ms` means that a router with the given IP address sent us a notification that our packet had reached N steps, and the message had arrived XX ms after sending it. A `*` means that the router didn't notify us or took too long to do so. List the output in your homework submission, and for each router, try to guess from its domain name where it is located.

   Now do the same thing for a traceroute to `www.gov.za`.

   (b) Humans most commonly do not use IP or MAC addresses to refer to hosts, but domain names. The command-line tool `dig` can be used to look at how DNS translates a domain name to an IP address, and the reverse lookup process as well. For example, if we wanted to find the IP address of `www.u-tokyo.ac.jp`, we would run:

```
$ dig www.u-tokyo.ac.jp

; <<>> DiG 9.18.28-0ubuntu0.24.04.1-Ubuntu <<>> www.u-tokyo.ac.jp
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 54796
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 65494
;; QUESTION SECTION:
;www.u-tokyo.ac.jp.     IN  A

;; ANSWER SECTION:
www.u-tokyo.ac.jp.  7004    IN  A   210.152.243.234

;; Query time: 0 msec
;; SERVER: 127.0.0.53#53(127.0.0.53) (UDP)
;; WHEN: Sat Sep 07 13:19:22 CDT 2024
;; MSG SIZE  rcvd: 62
```

The output of `dig` is split into several interesting sections. For example, the "`;; ANSWER SECTION:`" tells us that one "record" for the domain name was found; it is of type (`A`)ddress (and not IPv6 (`AAAA`)dress records). The `;; ->>HEADER<<-` line tells us about the packet that was sent in response to our DNS query. It includes the `id` field and the DNS flags that were set in the response. If you re-run the query, you'll see that the `id` field changes each time. Try doing it ten times. What are the query ids that you see? How many bits do you think this field uses?

A useful option in `dig` is `+trace`, which displays the results obtained at each step of the recursive resolution process; it also displays the cryptographic signatures that come with these results, but we can turn those off with the `+nocrypto` option. Try using this to see the recursive results of looking up `www.cl.cam.ac.uk`: `dig +trace +nocrypto @8.8.8.8 www.cl.cam.ac.uk`.[1]

Look at the results. How many queries did the recursive resolver make? What was the last nameserver queried?

Finally, let's trace a reverse lookup, and find out what the domain name of `192.0.32.7` is. To do a reverse lookup, we use the `-x` option immediately before the IP address: `dig +trace +nocrypto @8.8.8.8 -x 192.0.32.7`. The `PTR` record in the last response tells us what the "authoritative name" of this IP address is. Which nameservers did your query talk to (see the "`;; Received XXX bytes from`" lines in the answer section), and what was the final domain name response? Use a browser to learn about the significance of this domain to the question.

3. **Spoofing**. While packets on the internet can come from any source address (not necessarily the "true" address of the sender), they are generally delivered to the correct destination address. So a simple mitigation against packet spoofing is to include some kind of secret or identifier in each direction of a connection. If an attacker can't read traffic sent *to* their spoofed source address, they won't know this identifier, so a spoofing victim can ignore packets that don't have the right identifier. Let's think about how this defense might go wrong.

   (a) In DNS queries, each packet includes a short identifier, that was originally intended to let a resolver figure out which query a DNS response corresponds to. So in many resolvers, the `id` field was just increased by 1 for each query. Suppose an attacker could get a victim resolver to resolve their domain name (`evil.com`, of course) and wanted to convince the victim that `google.com` also pointed to their IP address (`42.42.42.42`). Also suppose that the attacker

[1] The `@8.8.8.8` option tells `dig` to run the recursive query through the google 8.8.8.8 DNS server, which is necessary in cselabs because cselabs hosts are configured to use a local "stub" resolver that can't do recursive lookups

knew that immediately after looking up `evil.com`, the victim would look up `google.com` (for instance, because `http://evil.com` loads an image from `google.com`). Draw a swim lane diagram illustrating how the attacker could convince the victim that `google.com` resolves to `42.42.42.42`. Your diagram should have lanes for the Victim, Evil.com Nameserver, and Google NS, and include enough details about the messages – the alleged source and contents – to see that it will work. (Assume that the attacker controls the nameserver for Evil.com, so can see messages sent to that nameserver, but cannot see messsages sent to other nameservers)

(b) Now suppose that instead of just increasing the `id` field by one for every query, the victim resolver has a secret odd value `k` and after every query the global id counter is incremented by `k`. How would the attack need to change in order to defeat this mitigation?

4. **Firewalls and Filtering**. Let's think some more about uses and limitations of firewalls.

(a) Firewall configuration: In class we saw that `iptables` is an interface to the linux software firewall. You can read the manual page for `iptables` online. Log in to your 4271 VM and let's test out a few rules. First, from your VM, test that you can reach `login02.cselabs.umn.edu` with `ping`:

```
$ ping -c 5 login02.cselabs.umn.edu
```

You should see an IP address for `login02` in the output of `ping` (`128.101.34.xxx`). Let's add a rule to DROP packets coming from `login02`. Your rule should be added to the `INPUT` chain, check the source address against `128.101.34.xxx`, and jump to the `DROP` target. Find the options to pass to `iptables` from the man page. Record it in your solution, then test the command on your VM. You'll have to run `iptables` with `sudo`, so you'll type something like `sudo iptables` <`options`>.

Once you've added the rule you can check that it's there by running

```
$ sudo iptables --list-rules INPUT
```

Test your rule by running ping again, e.g. `ping -c 5 login02.cselabs.umn.edu`. If the rule worked you should (after 5 seconds) see the output:

```
PING csel-remote-lnx-02.cselabs.umn.edu (128.101.34.33) 56(84) bytes of data.

--- csel-remote-lnx-02.cselabs.umn.edu ping statistics ---
5 packets transmitted, 0 received, 100% packet loss, time 4093ms
```

Per the manual page, you can remove this rule using the `--delete` option.

Now suppose we want to prevent outgoing connections to SSH (tcp protocol, destination port 22). What would be the command to accomplish this? (You may want to skim the list of iptables extensions) Test your command by running it on the VM (*make sure you add the rule to the OUTPUT chain* so that you don't block your existing ssh connection to your VM!), and then attempting to `ssh login02.cselabs.umn.edu`. If ssh doesn't print any output, your rule was correct. You can delete it from the OUTPUT chain, and move on to the next question.

(b) **Ingress** and **Egress** filtering: One somewhat common use of stateless packet filtering is ingress and egress filtering; these techniques can be deployed by ISPs as a mitigation for packet spoofing. The idea is to combine two rules:

Egress filtering drops packets leaving the network that do not have an IP address from inside the network. This prevents clients of the network from spoofing external addresses to the rest of the Internet.

Ingress filtering drops packets entering the network that have an IP address from inside the network. This prevents external attackers from spoofing local addresses to the hosts inside the ISP.

Suppose you are configuring the firewall router for an ISP using `iptables`, where packets from the local network arrive on interface `en0` and packets from the Internet arrive on interface `en1`, and let's suppose your ISP uses the address range `3.14.15.0/24`. What `iptables` rules would you use on the `FORWARD` chain to implement egress and ingress filtering?

(c) **Ingress and Egress Again**. If a particular network implements e/ingress filtering, would its clients be completely protected against packet spoofing attacks? What if all networks did so? Why or why not?

This assignment is based in large part on assignments originally by Prof. Nick Hopper, and is licensed under Creative Commons Attribution-ShareAlike 4.0.