Assignment 6 - DNS Amplification

In this assignment you will explore a class of Denial of Service (DoS) attacks known as amplification attacks. Although there are several protocols susceptible to amplification, for this project we will be using DNS amplification. After seeing the effects of the amplification attack on a target web server, you'll program an appliance to protect the web server from the attack.

Important:

Although the attack script we are providing cannot be used exactly as-is to attack a real server, the necessary principles are there. Therefore this project comes with two important caveats: 1) We are counting on you to be responsible and use the provided code _only_ to learn about DNS, amplification attacks and DoS, and protecting against these attacks, and _not_ to use the code or the knowledge you gain to do anything malicious to others. 2) **DO NOT DISTRIBUTE** this code to others, so they cannot use it for any malicious purposes either.

Before we begin, let's take a look at the topology we'll be using for this project:

```
     svr (10.0.0.4)
      |
      |
  firewall (s1)
      |
      |
  switch (s2)
  /   |   \
 |    |    |
h1   h2    dns (10.0.0.1)
```

The host named 'svr' will run the web server that we'll be attacking, and 'dns' will run the innocent DNS server that we'll take advantage of to amplify our attack. The hosts 'h1' and 'h2' are ordinary client / end hosts that we can use to retrieve web pages from the server or launch our DoS attack. The switch 's2' is an ordinary learning switch, but 's1' is just a passthrough - what goes in one end comes out the other. However, it has hooks in it to send DNS traffic to the controller where the function you write will inspect the traffic and either allow the DNS packets to pass through, or block them. The default behavior we've provided you to start with, though, is to simply let all traffic through.

The two switches are connected to each other by a high-bandwidth link. The two servers ('svr' and 'dns') are connected by moderate-bandwidth links, and the hosts ('h1' and 'h2') are connected by low-bandwidth links. Thus we'll see that a host with limited bandwidth can use amplification to saturate much larger network links.

First, update your git repository to get the assignment 5 code:

```
git commit -a -m "Saving work"
git pull --rebase
```

Before we begin, we need to configure a few things on your VM. Let's start by launching the class VM and running these steps.

First configure the bind9 DNS server software. We've provided the necessary configuraiton files in the bind/ directory of the assignment. One of these configuraiton files includes records for the

`www.cs6250.com_ domain` (which is a fake domain that we made up for this assignment).

1. `sudo apt-get install bind9`
2. `sudo cp bind/* /etc/bind/`
3. `sudo service bind9 restart`
4. `dig @localhost www.cs6250.com`

   From that last command, you should see a very large record for www.cs6250.com returned. If you do not get a good result from dig, go back and check your bind9 installation and configuration. You should also be able to dig real domains from your DNS server.

   When you're ready to continue, we next need to update your Ryu configuration. Ryu is used by the DNS parser that will help to parse the DNS messages for you so that your code can inspect their contents.

5. `pushd ryu-update; ./update-ryu-dns.sh ~/pyretic/pyretic/vendor/ryu; popd`

   Now that that's done, we can start the network and launch our attack. We created scripts to help launch the mininet topology and pyretic:

6. `./start-topology.sh` to start the mininet topology (topo.py)
7. in another terminal window, `./start-pyretic.sh` to start pyretic

   Note: In the course of running things, you may see some messages in the pyretic output that say "ERROR PUSHING MESSAGE". We're still working on tracking down this bug, but in our testing it did not affect everything working correctly, so we expect it is safe to ignore this if you see it.

   These scripts should start the DNS server and web server inside the mininet environment automatically. Let's test those an make sure they're working.

8. `mininet> h1 dig @10.0.0.1 www.cs6250.com` (If that doesn't work, you can try running dig locally on the DNS server to see if it's a problem with the DNS server or with connectivity between the hosts. `dns dig @localhost www.cs6250.com`)
9. `mininet> h2 ping -c 10 svr` to confirm connectivity and see what the latency to the web server is. Make a note of the RTT times you see here.
10. `mininet> h2 wget http://10.0.0.4` to make sure the web server is working. Make a note of how long it took to download the file. It should be somewhere around 4 seconds.

    Now that you've seen the latency (RTT time) and throughput (time to wget the web page) to the web server in normal conditions, let's launch the attack and see how the web server fairs when it's being DoSed. The attack script we've provided issues a query for the www.cs6250.com domain but spoofs the web server's IP as the source address in the IP header, so the DNS server's replies will be send to the web server instead of back to the attacking host. It takes two parameters. The first parameter is the address of the DNS server that will be used for the attack, and the second is the target of the attack (the web server, 'svr', in this case).

11. `mininet> h1 python ./amplify-dns.py 10.0.0.1 10.0.0.4 &` Note the & on the end runs it in the background so we can continue working in mininet while the attack continues.

    Now that the attack is going, let's see how the web server performs.

12. `mininet> h2 ping -c 10 svr` Note that the RTT latency is now much larger than before. (It's possible you may even see some packet loss.)
13. `mininet> h2 wget http://10.0.0.4` This should now take much longer than 4 seconds.

    Now that you've seen what the attack does, it's time to modify the firewall (s1) to block it. For

starters, let's just block all DNS traffic.

14. `dns_firewall.py` has a funciton that will be called every time 's1' receives a DNS packet. Modify it to change it's policy from allowing all DNS packets to blocking DNS responses. You can add your code to the funciton provided there, and you can also add any global variables you need. Some starter code is provided to parse the DNS packets for you, so you can use the variables the parser extracted from the packet for you. To allow a packet through, simply return it from the function; to block a packet, return `None`.

    Note: Be careful of putting output (e.g., print statements) in this function. The function is called on _every single DNS packet_ that the switch sees, which is **a lot** when the attack is underway. Since output statements are also relatively slow to execute (compare to other lines of code), print statements that are executing on every single funciton call can potentially overload the controller itself, turning the DoS attack against the web server into a DoS on pyretic (even if your policy correctly drops the attack traffic). Conditional output may be okay, if the condition is infrequent enough. It also may be okay to use some print statements for debugging, if you like to debug that way, so long as you take them back out before testing to see if your policy actually works under attack conditions.

15. Now start up mininet and test your policy. Run steps 6-13 again (you can skip step 8, running dig). If your policy works, you should now see comparable performance both before and during the attack.

16. There's just one problem now. Run `mininet> svr dig @10.0.0.1 www.cs6250.com`. Uh-oh! The web server can't issue legitimate DNS requests anymore! (You can also stop the attack and try this; it should still fail even when there is no attack.)

17. What we'd like to allow matched pairs of queries and their responses, while blocking an spurious responses (that is, responses to queries that we haven't seen). Modify `dns_firewall.py` to monitor DNS queries that pass through the firewall (and allow them to pass, of course), then to check any DNS responses against previously seen queries and allow them if they match an earlier query, but block them otherwise. DNS queries contain a transaction ID field with a randomly-assigned number. DNS responses also have a transaction ID that matches the transaction ID of the queries they are responding to. This is normally used to match responses to their queries, in the event that a host issues multiple queries in a short time, and you can also use it for matching responses to queries.

18. Now test your new policy! Start mininet and pyretic and make sure the server can make DNS queries to the DNS server (step 16). Then run steps 6-13 again (you can skip step 8) to make sure it is still effective at blocking the attack.

When you are finished, submit `dns_firewall.py` on T-Square. We will test that your code is able to stop a DNS-based DoS attack, and also that it still allows the web server to make DNS queries.

Files:

- amplify-dns.py - script that runs a DNS amplification attack
- topo.py - the topology for the mininet network
- run-topology.sh - shell script helper to launch the topology (topo.py) and start the DNS server and web server inside the mininet environment
- run-pyretic.sh - shell script helper to start pyretic
- dns_firewall.py - contains a function that gets called on every DNS packet that the firewall (s1) sees; this is the only file you need to turn in, and should be the only one you modify
- firewall.py - the firewall (s1) policy: calls the function on dns_firewall.py on every DNS packet, and simply forwards all non-DNS traffic
- pyretic_switch.py - a standard learning switch policy, which is used for the regular switch (s2)
- dns_amplification_prevention.py - the main file for the pyretic controller

- bind/ - this directory contains configuration files for the bind9 DNS server
- http/ - this directory contains the web server and a web page that can be retrieved from the server
- impacket/ - this directory contains the impacket library, which is used by the amplify-dns.py attack script
- ryu-update/ - this directory contains the files needed to install the DNS parser into your Ryu configuraiton

This page was last edited on 2015/07/20 16:18:20.