Gabe Small Nov 30, 2025

The first thing I did was review the slides and the DNS primer to review and make sure I had a grasp on DNS. Then I tried to start but could not get a grasp on where to start. I looked at the previous assignment, but that did not have anything on sniffing in scapy and then manipulating what was there. I read https://www.geeksforgeeks.org/python/packet-sniffing-using-scapy/ to understand how to use scapy better, but it was all in the terminal so that confused me. I went to office hours to see if the TA could help me figure out where to start and they helped me a lot. I was still a bit shaky so I watched https://www.youtube.com/watch?v=n3yyV96LTTI in order to get a better grasp on what exactly I needed to do.

Between all of that I understood that I needed to sniff for valid packets, take them apart, then put them back together with a few things reversed and of course a forged answer.

The other part was the interface. I got some advice from the TA to use argparse to read the arguments. I needed to check the interface against the list of valid interfaces and check if the hostnames file is there and use it if one was given. I also added an optional count argument as I did not like it constantly running and wanted a way to stop it after a certain amount of forging.

At this point I was lost on how exactly to test it as I am not the most familiar with Docker or virtual machines. I went to office hours again to try and solve this issue. Eventually we realized that the interface name is a name on Windows and not something like eth0. I still had some issues that the TA could not solve, but I ended up resorting to using AI to debug, but I wrote all of my code myself, only using AI to determine where the issue was, not how to solve it. In the end I used WSL as my VM to test against.

Thanksgiving came around and so I was away from Brandeis and could not test against Brandeis's DNS resolver until I got back. I tried to set up the VPN, but could not get it to work with WSL or any other VM I tried to use, so I just waited. In the meantime I was able to test against a variety of resolvers from different places. I used the command `dig @<DNS server> <query address>` , I was located in Lawrence, New York, and I attempted ten times each

IP	Location	Success Rate
8.8.8.8	Mountain View, California	100%
151.202.0.84	New York City, New York	80%
194.145.240.7	Manchester, England	100%
210.181.4.25	Seoul, South Korea	100%
128.238.2.38	Brooklyn, New York	100%

I wanted to test against a variety of locations to see how distance affected success rate. I saw that against the one in New York City I did not succeed every time and so I tried one that was even closer, but I succeeded all ten times. I suspect that the high rate of success for further locations is due to the fact it takes time to reach those locations and back, while my code is all

local. As for the one in New York City doing worse than the one in the closer Brooklyn, it could be that there is a direct route to New York City, while it needs to bounce around to reach the one in Brooklyn.

Finally I got back to Campus and was able to test against the University's resolver. An issue I encountered was that when I used dig with a specific DNS server listed then it went through vEthernet (WSL (Hyper-V firewall)), an interface only for WSL, but if I did not declare a target DNS server and it just uses the default, then it goes through the Wi-Fi interface. This was a problem as that would make it much harder to filter. The way I worked around that was to determine the Brandeis DNS server using `ipconfig /all` in powershell and using that as the DNS server. My injector did not work 100% as the Brandeis DNS resolver was able to give a response much faster than farther away resolvers.