Assignment: DNS Poisoning

【注意】作为小组作业，请大家一个小组上传一份就可以。

【作业文件命名规则】<Team Name>.pdf

In this assignment, you will develop

1. an on-path DNS poisoning attack tool, and
2. a passive DNS poisoning attack detector.

Both tools should support just plain (UDP) DNS traffic over port 53.

Part 1 (30%):

The DNS packet injector you are going to develop, named '**dnspoison**', captures the traffic from a network interface in promiscuous mode and injects forged responses to selected DNS A requests with the goal of poisoning the cache of the victim's resolver.

Your program should conform to the following specification:

[-i interface] [-f hostnames] [expression]

-i Listen on network device (e.g., eth0). If not specified, **dnspoison** should select a default interface to listen on. The same interface should be used for packet injection.

-f Read a list of IP address and hostname pairs specifying the hostnames to be hijacked. If '-f' is not specified, **dnspoison** should forge replies to all observed requests with the chosen interface's IP address as an answer.

The optional <expression> argument is a BPF filter that specifies a subset of the traffic to be monitored. This option is useful for targeting a single victim or a group of victims.

The <hostnames> file should contain one IP and hostname pair per line, separated by whitespace, in the following format:

10.6.6.6 foo.example.com

10.6.6.6 bar.example.com

192.168.66.6 http://www.cs.stonybrook.edu/

Pay attention to the time needed for generating the spoofed response. Your code should be fast enough so that the injected reply reaches the victim before the server's real response. The spoofed packet and content should be valid according to the initial DNS request, and the forged response should be accepted and processed normally by the victim. Failure to win the race will not affect your grade, but you should at least try (see hints below).

Part 2 (70%):

The DNS poisoning attack detector you are going to develop, named '**dnsdetect**', captures the traffic from a network interface in promiscuous mode and detects DNS poisoning attack attempts, such as those generated by your own **dnspoison**, or **dnsspoof** (https://www.monkey.org/~dugsong/dsniff/). Detection is based on identifying duplicate responses within a short time interval towards the same destination, which contains different answers for the same A request (i.e., the observation of the attacker's spoofed response and the server's actual response).

The order of arrival should not matter: you should raise an alert irrespectively of whether the attacker's spoofed response arrived before or after the real response. You should make every effort to avoid false positives, e.g., due to legitimate consecutive responses with different IP addresses for the same hostname due to DNS-based load balancing.

Your program should conform to the following specification:

[-i interface] [-r tracefile] expression

-i Listen on network device <interface> (e.g., eth0). If not specified, the program should select a default interface to listen on.

-r Read packets from <tracefile> (tcpdump format). Useful for detecting DNS poisoning attacks in existing network traces.

<expression> is a BPF filter that specifies a subset of the traffic to be monitored.

Once an attack is detected, **dnsdetect** should print to stdout a detailed alert containing a printout of both the spoofed and legitimate responses. You can format the output in any way you like. The output must contain the DNS transaction ID, the attacked domain name, and the original and malicious IP addresses, for example:

20210309-15:08:49.205618 DNS poisoning attempt

TXID 0x5cce Request www.example.com

Answer1 [List of IP addresses]

Answer2 [List of IP addresses]

What to submit:

An archive file with:

1. all required source code files
2. a pcap trace containing one or more successful poisoning attacks generated using your dnspoison tool

3. a short report with a brief description of your programs, the strategy you followed for DNS poisoning detection, and the output of your dnsdetect tool when fed with the above pcap trace

Hints

- Mind your spoofed packet's header fields and checksums!

- Think about what fields should remain the same or may differ between the spoofed and actual response packets.

- An easy way to test your implementation is using two hosts: the attacker can be either your real machine or a VM, and the victim can be a different VM. In both cases, the attacker will be in a position to sniff the victim's traffic and inject packets towards the victim, i.e., perform a man-on-the-side attack. Alternatively, you can also use ARP spoofing to perform a man-in-the-middle attack.

- Don't forget to disable DoH/DoT, if it is enabled in the victim's browser.

- If your host OS is configured to use a fast DNS server (e.g., Google's 8.8.8.8), these tend to have a very low RTT of just 3-4ms, so reliably winning the race will require carefully tuned code. Try ping'ing the DNS server to observe its RTT. To win the race more easily and reliably, you can pick a server that is "far away", e.g., 30-40ms away. Here is a list of free public DNS servers (https://twitgoo.com/best[1]free-dns-servers/ (https://twitgoo.com/best-free-dns-servers/)), many of them have very low latency from SBU, but there are some in the 30-40ms range (e.g., Alternate DNS, OpenNIC, SmartViper) so your attack should be reliable using those. Yandex or puntCAT should be even easier, as they are in the 100ms range.

- If you still cannot win the race in your setup, for testing purposes you can "slow down" or drop the incoming legitimate responses (e.g., using NetfilterQueue or any other method you prefer). Failure to win the race will not affect your grade, but the spoofed response should be correct and accepted by the victim (if it would arrive first).