

Report: DNS Poisoning and detection

Shubham Agrawal - 113166701

DNS Poisoning:

The program dnspoin.go provides implementation to sniff and poison live DNS Traffic with/ without interfaces, poison hosts file and BPF filter options.

Output and Program Usage:

1. `$ sudo go run dnspoin.go`

If we don't provide any parameters, it will take the default interface/ ethernet device and start poisoning all the DNS traffic which it can trace.

2. `$ sudo go run dnspoin.go -i en0`

"i" parameter takes the ethernet device as input. If the ethernet device is wrong, the program stops giving the error. If it is correct, it poisons all the DNS traffic for this particular ethernet device.

3. `$ sudo go run dnspoin.go -f "/assignment-3/poisonhosts"`

"f" value takes the poison host file as input. If the path is correct, it poisons only those requests which are mentioned in the file for the default interface/ ethernet device. If the path is wrong, the program stops specifying that the path is wrong.

Sample Poison Hosts File:

172.24.18.135 www.example.com

172.24.18.135 www.tcpdump.org

Sample Output:

Invalid File Name:

```
admins-mbp-2:awesomeProject admin$ go run dnspoin.go -f "/Users/admin/Downloads/SBU-US/Network Security/assignment-3/poisonho"
panic: open /Users/admin/Downloads/SBU-US/Network Security/assignment-3/poisonho: no such file or directory

goroutine 1 [running]:
main.poin(0xc420818274, 0x3, 0x7ffefbfff6f9, 0x44, 0x41ac464, 0x3)
/Users/admin/go/src/awesomeProject/dnspoin.go:58 +0x571
main.main()
/Users/admin/go/src/awesomeProject/dnspoin.go:47 +0x299
exit status 2
admins-mbp-2:awesomeProject admin$
```

4. `$ sudo go run dnspoisn.go -i en0 -f "/assignment-3/poisonhosts"`

If both 'i' and 'f' values are given, it will poison only those requests which are mentioned in the file for the specified interface/ ethernet device.

5. `$ sudo go run dnspoisn.go -i en0 -f "/assignment-3/poisonhosts" "icmp"`

A string value at the end of the above command specifies a BPF filter. Here, "icmp" is a BPF filter. BPF filter is not applied if no string is provided at the end of the command, for example, in the above four examples.

Any combination of the above can be used to run the go program.

Trace file (PCAP file) generated from the above dnspoisn.go with poisoned DNS packets: poisonpackets.pcap file attached in the submission

```
admins-mbp-2:assignment-3 admin$ sudo tcpdump -r before_upload/113166701/poisonpackets.pcap
reading from file before_upload/113166701/poisonpackets.pcap, link-type EN10MB (Ethernet)
19:05:57.780368 IP 172.24.16.119.50724 > dns.google.domain: 40183+ A? www.tcpdump.org. (33)
19:05:57.783591 IP dns.google.domain > 172.24.16.119.50724: 40183 1/0/0 A 172.24.18.135 (64)
19:05:57.803874 IP dns.google.domain > 172.24.16.119.50724: 40183 2/0/0 A 159.89.89.188, A 192.139.
46.66 (65)
19:05:59.588286 IP 172.24.16.119.61335 > dns.google.domain: 35267+ A? www.google.com. (32)
19:05:59.594902 IP dns.google.domain > 172.24.16.119.61335: 35267 1/0/0 A 172.217.12.196 (48)
19:05:59.767414 IP 172.24.16.119.60221 > dns.google.domain: 26059+ A? www.gstatic.com. (33)
19:05:59.781522 IP dns.google.domain > 172.24.16.119.60221: 26059 1/0/0 A 172.217.12.163 (49)
19:05:59.827603 IP 172.24.16.119.54127 > dns.google.domain: 64589+ A? apis.google.com. (33)
19:05:59.829196 IP 172.24.16.119.64388 > dns.google.domain: 41489+ A? ogs.google.com. (32)
19:05:59.843951 IP dns.google.domain > 172.24.16.119.64388: 41489 2/0/0 CNAME www3.l.google.com., A
172.217.11.46 (69)
19:05:59.849405 IP dns.google.domain > 172.24.16.119.54127: 64589 2/0/0 CNAME plus.l.google.com., A
172.217.10.46 (70)
19:06:03.929467 IP 172.24.16.119.58065 > dns.google.domain: 56365+ A? encrypted-tbn0.gstatic.com. (
44)
19:06:03.943679 IP dns.google.domain > 172.24.16.119.58065: 56365 1/0/0 A 172.217.9.238 (60)
19:06:07.084897 IP 172.24.16.119.55755 > dns.google.domain: 3379+ A? www.example.com. (33)
19:06:07.091831 IP dns.google.domain > 172.24.16.119.55755: 3379 1/0/0 A 93.184.216.34 (49)
19:06:07.095627 IP dns.google.domain > 172.24.16.119.55755: 3379 1/0/0 A 172.24.18.135 (64)
19:06:07.158154 IP 172.24.16.119.51141 > dns.google.domain: 55116+ A? www.iana.org. (30)
19:06:07.174878 IP dns.google.domain > 172.24.16.119.51141: 55116 2/0/0 CNAME ianawww.vip.icann.org
., A 192.0.32.8 (78)
admins-mbp-2:assignment-3 admin$
```

It contains two poisons:

1. www.tcpdump.org - here my program wins the race and my local server page is rendered -> TXID: 40183
2. www.example.com - here my program loses the race and my local server page is not rendered -> TXID:

3379

Code/ Implementation Walkthrough:

1. Accept input parameter values , -i (for interface name), -f (for poisonhosts file) and flag.Args (for BPF filter)
2. Depending on the input parameters, deciding whether to read from file and poison for a particular interface, or poison all the DNS packets, or if no parameters, take the default device and start poisoning DNS packets.
3. For default devices, **pcap.FindAllDevs()** returns all available interfaces. Took the first one in the list as the default.
4. For the default IP, devices[0].Addresses[1].IP gives the IP of the host machine where the go program is running. If there is no poisonhost file given, we have to route the DNS poison to the host IP
5. Now, if the poisonhosts file is provided, read the file and store the IP and value in a hashmap named poisonHosts
6. Used **pcap.OpenLive(interfacePointer, 1600, true, pcap.BlockForever)** to read from interface
7. After starting the handle to read from respective locations:
 1. Setting the BPF Filter using:
handle.SetBPFFilter(expressionPointer)
 2. Extracting each packet:
gopacket.NewPacketSource(handle, handle.LinkType())
 3. Running a loop for each packet
 4. Filtering only those packets where IP Layer is not null
 5. Filtering only those packets where UDP Layer is not null
 6. Filtering only those packets where UDP Destination port is 53 since DNS packets are associated with UDP Layer and Destination port as 53
 7. Extract DNS Layer
 8. Filter only Question packets in DNS Layer, i.e. Requests, as we need to only poison requests and not responses.
 9. If poison hosts file is not given as argument, send a DNS Packet with the default IP
 10. If the poison hosts file is given, run a loop for all the key-value pairs in the poisonHosts hashmap data structure.
 11. If the dns.Questions[0].Name is present as a key in the hasmap, send the DNS response packet with the IP provided in the poisonhosts file mapped for the particular question

12. In the DNS Response Packet: using the existing Layers in the Request and forming the response by changing the required placeholders.
13. Ethernet Layer: Setting Response Destination MAC address as the Request Source MAC address. Setting the Response Source MAC address as the system MAC address.
However, we can also use the Request Destination MAC address.
14. IP Layer: Interchanging the Source and Destination IP addresses.
15. UDP Layer: Interchanging the Source and Destination ports.
16. DNS Layer: Creating a DNS Answer Record by specifying the poison IP address as the answer. Also specifying other fields of DNS Answer Record such as:

```
var dnsAnswer layers.DNSResourceRecord
```

```
a, _, _ := net.ParseCIDR(value + "/24")
```

```
dnsAnswer.Type = layers.DNSTypeA
```

```
dnsAnswer.IP = a
```

```
dnsAnswer.Name = []byte(key)
```

```
dnsAnswer.Class = layers.DNSClassIN
```

```
dnsAnswer.TTL = 100
```

17. In the DNS Packet, setting the QR bit to true, which signifies that it is an DNS Response packet
18. Assigning the number of answers it contains and other fields such as:

```
dnsPacket.QR = true
```

```
dnsPacket.ANCount = 1
```

```
dnsPacket.OpCode = layers.DNSOpCodeQuery
```

```
dnsPacket.AA = false
```

```
dnsPacket.RA = true
```

```
dnsPacket.Answers = append(dnsPacket.Answers, dnsAnswer)
```

```
dnsPacket.ResponseCode = layers.DNSResponseCodeNoErr
```

19. Initialize a Serialize buffer. Set the Buffer options.
20. Set the Network Layer Checksum
21. Serialize the buffer with all the Layers and write the packet to the handle in bytes.

DNS Detection:

The program `dnsdetect.go` provides implementation to detect poisoned DNS Traffic in real time or by reading through a PCAP file.

Output and Program Usage:

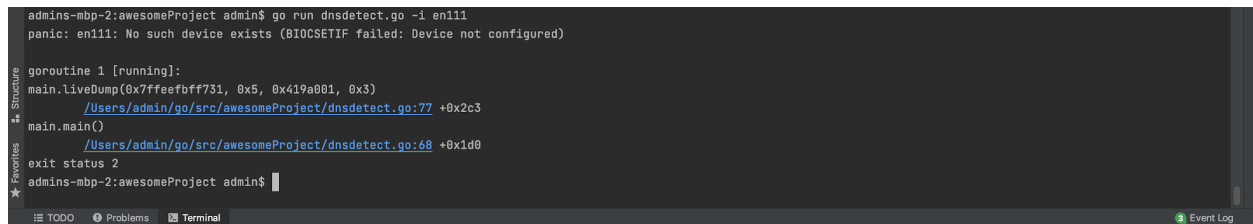
6. `$ sudo go run dnsdetect.go`

If we don't provide any parameters, it will take the default interface/ ethernet device and start detecting poison in the DNS traffic.

7. `$ sudo go run dnsdetect.go -i en0`

"i" parameter takes the ethernet device as input. If the ethernet device is wrong, the program stops giving the error. If it is correct, it starts detecting poison in the DNS traffic for the specified ethernet device.

Sample Output for invalid interface:



```
admins-mbp-2:awesomeProject admin$ go run dnsdetect.go -i en111
panic: en111: No such device exists (BIOCSSETIF failed: Device not configured)

goroutine 1 [running]:
main.liveDump(0x7ffefbfbf731, 0x5, 0x419a001, 0x3)
    /Users/admin/go/src/awesomeProject/dnsdetect.go:77 +0x2c3
main.main()
    /Users/admin/go/src/awesomeProject/dnsdetect.go:68 +0x1d0
exit status 2
admins-mbp-2:awesomeProject admin$
```

8. `$ sudo go run dnsposion.go -r "/assignment-3/poisonpackets.pcap"`

"r" value takes the pcap file as input. If the path is correct, it detects poison in the DNS packets in the pcap file for the default interface/ ethernet device. If the path is wrong, the program stops specifying that the path is wrong.

- Using the PCAP file generated from the `dnsposion.go` program above to detect the DNS poisons

Sample Output:

Valid PCAP file:

```
admins-mbp-2:awesomeProject admin$ go run dnsdetect.go -i en0 -r /Users/admin/Downloads/SBU-US/Network Security/assignment-3/before_upload/113166701/poisonpackets.pcap
20210403-19:05:57.803074 DNS poisoning attempt
TXID 40183 Request www.tcpdump.org
Answer 1 172.24.18.135
Answer 2 159.89.89.188 159.89.89.188
20210403-19:06:07.095627 DNS poisoning attempt
TXID 3379 Request www.example.com
Answer 1 93.184.216.34
Answer 2 172.24.18.135
admins-mbp-2:awesomeProject admin$
```

Invalid PCAP file:

```
admins-mbp-2:awesomeProject admin$ go run dnsdetect.go -r "/Users/admin/Downloads/SBU-US/Network Security/assignment-3/dnspoisn.pcap"
panic: /Users/admin/Downloads/SBU-US/Network Security/assignment-3/dnspoisn.pcap: No such file or directory

goroutine 1 [running]:
main.readFromPcap(0x7ffefbfff6f1, 0x49, 0x419a001, 0x3)
/Users/admin/go/src/awesomeProject/dnsdetect.go:104 +0x2a7
main.main()
/Users/admin/go/src/awesomeProject/dnsdetect.go:64 +0x197
exit status 2
admins-mbp-2:awesomeProject admin$
```

9. \$ sudo go run dnspoisn.go -i en0 -r "/assignment-3/dnspoisn.pcap"

If both ‘i’ and ‘r’ values are given, it will detect for DNS poisons in the given trace file.

10. \$ sudo go run dnspoisn.go -i en0 -r "/assignment-3/poisonhosts" “icmp”

A string value at the end of the above command specifies a BPF filter. Here, “icmp” is a BPF filter. BPF filter is not applied if no string is provided at the end of the command, for example, in the above four examples.

Note: Any combination of the above can be used to run the go program.

Output file from the above dnsdetect.go with poisoned DNS packets: poisonpackets.pcap file attached in the submission.

Output file Name: detectoutput.txt

Code/ Implementation Walkthrough:

1. Accept input parameter values , -i (for interface name), -r (for pcap file) and flag.Args (for BPF filter)
2. Depending on the input parameters, deciding whether to detect DNS poison from the pcap file, or detect DNS poisons for all the live DNS packets for the default or specified interface/ ethernet device.
3. For default devices, **pcap.FindAllDevs()** returns all available interfaces. Took the first one in the list as the default.
4. Initializing three hashmaps: dnsDetect - contains the key-value of Transaction ID of answer and DNS packet, dnsTime - contains the key-value of Transaction ID of answer and time at which the packet came, destinationIp - contains the key-value of Transaction ID of answer and the destination IP of the packet. All of

these hashmaps are Safe hashmaps, i.e. applying mutex synchronization operation if we are deleting or writing something. We are doing this, because of another thread which removes the data explained later on.

5. If the pcap file is provided, read the file and initialize handle to iterate over each packets
6. If the pcap file is not provided, read the live packets for the interface if provided or select the default interface. Initialize the handle to iterate over each packets
7. Used **pcap.OpenLive(interfacePointer, 1600, true, pcap.BlockForever)** to read from interface
8. After starting the handle to read from respective locations:
 1. Setting the BPF Filter using:
handle.SetBPFFilter(expressionPointer)
 2. Extracting each packet:
gopacket.NewPacketSource(handle, handle.LinkType())
 3. Running a loop for each packet
 4. Filtering only those packets where IP Layer, UDP Layer and DNS Layer is not null
 5. Extract DNS and IP Layer
 6. Filter only Answer packets in DNS Layer, i.e. Responses (QR bit set to true), as we need to detect poisons only in responses.
 7. Call a go thread function flushData which flushes all the DNS packets which is 5 seconds before. I am calling this function only when the length of the current hashmap is more than 500. We could have called the function every time as well, but this approach would have taken more CPU usage, as it would be trying to flush the data every time. Also, I have taken a default value of 500, we can increase or decrease this value based on the system metrics and RAM size. We can put larger values, if the RAM size is more and CPU is less or vice versa.
 8. Check whether the transaction ID of the DNS Response is present in the dnsDetect hashmap or not
 9. If the hashmap does not have the transaction ID, put the values in all the three hashmaps initialized above, i.e. dnsDetect, dnsTime, destinationIp locking and unlocking the safe hashmaps
 10. If the transaction ID is present in the hashmap, check whether the destination IP is the same as that of the previous DNS Answer Packet and whether the time difference

between the two DNS Responses is very close. (In this program, I have taken a default time difference value of 3 seconds between poison and actual response)

11. If the destination IP and time difference criteria is not satisfied, I overwrite the subsequent values in the HashMap with the latest timestamp, destination IP and DNS Packet locking and unlocking the safe hashmaps
12. If the criteria is satisfied, it means that the DNS answer has been poisoned and a DNS attack is detected.
13. Printing the details as required by the assignment. Here, sometimes what happens, is that the IP address becomes Nil as canonical names are present in the packet. In this case, we should ideally print the canonical names. For this assignment, I am still printing the IP which is nil as we need to print the list of IP addresses.

9. flushData thread function:

1. Create an array.
2. Store the Transaction IDs of all the DNS Packets which is more than 5 seconds before than the current packet in the above created array.
3. Run a loop of the this array and delete from all the hashmaps based on the transaction ID by locking and unlocking the safe hashmaps

Note: This thread function is not an infinite loop, but is called every time once the length of hashmaps increases over 500 in order to optimise the system usage.

Corner Case: Sometimes, what happens is that the actual DNS response from the server can come in 2 different packets. Hence, there will be a case where same transaction IDs of the two responses are coming but not due to poisoning. In such cases, we can check whether the DNS Answer/ Response is actually truncated or fragmented into sub packets and do not consider this to be a poison.

References:

1. <https://www.devdungeon.com/content/packet-capture-injection-and-analysis-gopacket>
2. <https://github.com/ggreer/dsniff/blob/master/dnsspoof.c>
3. <https://medium.com/@openmohan/dns-basics-and-building-simple-dns-server-in-go-6cb8e1cfe461>
4. <https://blog.davidvassallo.me/2016/11/03/simple-dns-sniffer-in-golang/>
5. <https://tour.golang.org/concurrency/1>

6. <https://blog.golang.org/maps>
7. <https://tour.golang.org/concurrency/9>

Submissions:

1. dnspoison.go - go program to poison DNS packets
2. dnsdetect.go - go program to detect DNS Spoofing
3. poisonhosts file - contains which packets to spoof
4. poisonpackets.pcap file - two spoofed DNS Response
5. Output of dnsdetect tool when fed with the above pcap trace file - detectoutput.txt
6. Report