**Ramaiah Institute of Technology**

(An Autonomous Institute, Affiliated to VTU)

MSR Nagar, MSRIT post, Bangalore-54

A Dissertation Report on

**DNS Flooding Mitigation**

Submitted to

**Sanjeetha R.**

Department of CSE

Submitted by

Atul Rustagi 1MS16CS025

Ankit Singh 1MS16CS017

Rahul Negi 1MS15CS159

***Bachelor of Engineering in Computer Science & Engineering***



**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

**M.S.RAMAIAH INSTITUTE OF TECHNOLOGY**

**(Autonomous Institute, Affiliated to VTU)**

**BANGALORE-560054**

[www.msrit.edu](http://www.msrit.edu/), **20177**

**Introduction**

DNS flood is a type of Distributed Denial of Service (DDoS) attack in which the attacker targets one or more Domain Name System (DNS) servers belonging to a given zone, attempting to hamper resolution of resource records of that zone and its sub-zones.DNS servers helps requestors find the servers they seek.A DNS zone is a distinct portion of the domain name space in the Domain Name System (DNS). For each zone, administrative responsibility is delegated to a single server cluster.In a DNS flood attack the offender tries to overbear a given DNS server with apparently valid traffic, overwhelming server resources and impeding the servers ability to direct legitimate requests to zone resources. DNS floods are symmetrical DDoS attacks. These attacks attempt to exhaust server-side assets (e.g., memory or CPU) with a flood of UDP requests, generated by scripts running on several compromised botnet machines. A DNS flood attack is considered a variant of the UDP flood attack, since DNS servers rely on the UDP protocol for name resolution, and is a Layer 7 attack. With UDP-based queries , a full circuit is never established, and spoofing is more easily accomplished. To attack a DNS server with a DNS flood, the attacker runs a script , generally from multiple servers. These scripts send malformed packets from spoofed IP addresses. Since Layer 7 attacks like DNS flood require no response to be effective, the attacker can send packets that are neither accurate nor even correctly formatted.

The attacker can spoof all packet information, including source IP and make it appear that the attack is coming from multiple sources. Randomized packet data also helps offenders to avoid common DDoS protection mechanisms, while also like IP filtering completely useless. Another common type of DNS flood attack is DNS NXDOMAIN flood attack, in which the attacker floods the DNS server with requests for records that are non-existent or invalid. The DNS server expends all its resources looking for these records, its cache fills with bad requests, and it eventually has no resources to serve legitimate requests. Large Layer 3 attacks like DNS floods are very difficult for on-premises solutions to mitigate.

Incapsula’s approach to mitigating DNS floods is simple and hassle-free. The solution is enabled by a dedicated anti-DDoS 'DNS Protection' service. Using DNS Protection, clients are able to deploy Incapsula's multi-datacenter network in front of their DNS authoritative server—doing so without making any changes to their existing zone file settings. With the DNS Protection in place, Incapsula becomes the destination for all incoming DNS queries, which are scrubbed on their way to their origin.

Incapsula users can then continue to set their own custom thresholds, with different values provided for the more likely-to-be-legitimate “safe” queries. In addition, users can also manually enforce DNS freshens by electing to refresh all cached data or by selectively refreshing specific DNS records.

**Literature survey**

**Mitigating DNS Query-Based DDoS Attacks with Machine Learning on Software-Defined Networking**

Due to *lack of stringent security* measures in IoT systems, *DNS protocol exploitation*, and *limitations of packet forwarding* in traditional networks, mitigation against DDoS attacks is a big challenge for military organizations.

Two famous DNS query-based DDoS attacks are:

1. DNS amplification

To launch such attack, attackers send large volume of forged DNS queries with spoofed IP address of victim to open DNS resolvers to prompt them to send massive volumes of DNS query responses to the victim with that address. To amplify such attack, DNS request can be sent using EDNS0 or DNSSEC DNS protocol extension or with spoofed queries of type ANY. These results in amplification factor of 70:1.

2. DNS flooding

These attacks attempt to target server-side resources with a flood of UDP requests, generated by scripts running on botnet devices. Layer 7 attacks and DNS NXDOMAIN flood attacks are more common. The DNS server expends all its resource looking for false records filling its cache thus no resources to serve legitimate requests.

Since SDN controller maintain global view of entire network and dynamically manage switches in an SDN environment, if seems effective to develop mitigation solutions for DDoS attacks. Features of SDN used for mitigation are: -

1. Decoupled control and data planes

2. Dynamic updating of forwarding rules

3. Software-based traffic analysis

Dirichlet Process Mixture Model (DPMM) is proposes as a nonparametric Bayesian approach for clustering traffic application where nonparametric means that no. of traffic app. Are unknown and may grow over time.

To study effectiveness of proposed DPMM approach for attack detection we evaluate following: -

1. Attack traffic classification accuracy

2. Overall classification accuracy

3. Misclassification rate

**Mitigating DNS DoS Attacks (existing solution)**

As explained in previous paper that DNS flooding attacks target the server-side resources and the caches. Therefore, proposed resolver modification focuses on change of caching behaviour of DNS resolvers so that they can shoulder more of the resolution burden, especially when nameservers are unavailable.

Proposed change in the operation of DNS resolvers are: -

1. Stale cache

Resolvers don’t completely expunge records whose TTL value has expired. Rather, such records are evicted from the cache and stored in a separate *stale cache.* In effect, the stale cache together with the resolver cache represents the part of global DNS database that has been accessed by the resolver.

2. Resolving queries

In our proposed modification, first two steps are as same as of todays DNS resolver but the third step is modified as:

In case the resolution process fails due to inability of the resolver to contact all the nameservers of the relevant zone at any step of traversal, search the stale cache for the required record and if found, the resolution process can continue on basis of this stale record.

3. Stale cache clean-up

Existing resolvers cache the responses to the queries made during the resolution process but in this proposed modification these responses are also used to evict the corresponding stale records from the stale cache.

Advantages of proposed modification: -

1. DNS Robustness

2. Simplicity

Does not change the basic protocol operation and infrastructure

Does not impose any load on DNS

Does not impact the latency of query resolution

3. Incremental Deployment

4. Motivation for Deployment

Objections of proposed modification: -

1. DNS caching semantics and the possibility of inaccurate information being used

2. Autonomy for zone operators

3. Attackers attempting to force the use of inaccurate information

4. Privacy concerns

5. Resolution latency in the face of an attack

**Denial of Service Attack and Prevention on SIP VoIP Infrastructure Using DNS Flooding**

The goal of attack is to make SIP infrastructure inoperable for as long as possible. These attacks can target any kind of SIP entity like user agent, proxy, registrars, and redirect proxy but are most effective against proxies and redirectors.

Scope of Attack: -

Whenever a SIP proxy encounters a fully qualified URL in a header field necessary for routing, it issues a query to the local name server to receive a valid address mapping. Usually it takes around 1.3 DNS queries to receive an answer with mean resolution latency of less than 100ms. But these values can be high due to configuration errors.

These relatively high processing time are major targets of the SIP DNS attack. It is done by specially crafted SIP messages containing URLs that will cause an even higher processing time at DNS server by using a URL in a routing header of which the attacker is sure that its mapping will not be in cache or will trigger to authoritative server having low response time.

Issuing SIP queries with a variation of URLs that are well formatted, complying with the SIP standards and can’t be filtered out by Intrusion Detection System, will stop operation at a SIP proxy for a considerable time. Since SIP proxy can continue its operation only after receiving an answer from DNS server, if no answer is received within timeout period, a negative reply is generated.

Existing solution against DNS attacks: -

General method is to trace the source of attack and block the traffic from is but for SIP network that runs at application layer, back tracing is costly and not much efficient. Thus, Synchronous and Asynchronous DNS implementation were proposed but they too are not sufficient to withstand a DNS attack with large no. of malicious messages per second.

Proposed solution against DNS attack: -

In proposed system, we use Non-Blocking Cache Design that is based on caching the results of successful DNS queries. This is implemented for a synchronous working server.

Steps involved are: -

1. DNS Attack Detection and Prevention (DADP)

2. Implementation of DADP

Performance Evaluation of DADP

Cache Replacement Policies Evaluation

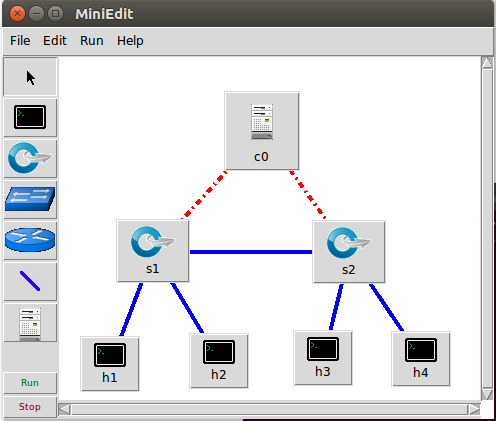
Evaluation of Cache Entry Numbers

**Proposed Solution**

The attack is done using the IP spoofing means the attacking nodes are sending the packets using IP of genuine hosts but those packets have the mac address of the attacking host only.

Thus, to stop the flooding attack we can add a rule in controller through which the flow of packets will install only for the genuine host. For this we need to make changes in the l2\_learning.py file in forwarding folder of pox.

**Implementation**



Code for topology

#!/usr/bin/python

from mininet.net import Mininet

from mininet.node import Controller, RemoteController, OVSController

from mininet.node import CPULimitedHost, Host, Node

from mininet.node import OVSKernelSwitch, UserSwitch

from mininet.node import IVSSwitch

from mininet.cli import CLI

from mininet.log import setLogLevel, info

from mininet.link import TCLink, Intf

from subprocess import call

def myNetwork():

net = Mininet( topo=None,

build=False,

ipBase='10.0.0.0/8')

info( '\*\*\* Adding controller\n' )

c0=net.addController(name='c0',

controller=RemoteController,

ip='127.0.0.1',

protocol='tcp',

port=6633)

info( '\*\*\* Add switches\n')

s1 = net.addSwitch('s1', cls=OVSKernelSwitch)

s2 = net.addSwitch('s2', cls=OVSKernelSwitch)

info( '\*\*\* Add hosts\n')

h1 = net.addHost('h1', cls=Host, ip='10.0.0.1', mac='00:00:00:00:00:01', defaultRoute=None)

h2 = net.addHost('h2', cls=Host, ip='10.0.0.2', mac='00:00:00:00:00:02', defaultRoute=None)

h4 = net.addHost('h4', cls=Host, ip='10.0.0.4', mac='00:00:00:00:00:04', defaultRoute=None)

h3 = net.addHost('h3', cls=Host, ip='10.0.0.3', mac='00:00:00:00:00:03', defaultRoute=None)

info( '\*\*\* Add links\n')

net.addLink(s1, h1)

net.addLink(s1, h2)

net.addLink(h3, s2)

net.addLink(s2, h4)

net.addLink(s1, s2)

info( '\*\*\* Starting network\n')

net.build()

info( '\*\*\* Starting controllers\n')

for controller in net.controllers:

controller.start()

info( '\*\*\* Starting switches\n')

net.get('s1').start([c0])

net.get('s2').start([c0])

info( '\*\*\* Post configure switches and hosts\n')

CLI(net)

net.stop()

if \_\_name\_\_ == '\_\_main\_\_':

setLogLevel( 'info' )

myNetwork()

Here in above topology, h4 is server, h2 and h3 are attacking hosts and h1 is genuine host.

Server.py

import socket

import optparse

import time

class DNSQuery:

def \_\_init\_\_(self, data):

self.data=data

self.dominio=''

self.DnsType=''

HDNS=data[-4:-2].encode("hex")

if HDNS == "0001":

self.DnsType='A'

elif HDNS == "000f":

self.DnsType='MX'

elif HDNS == "0002":

self.DnsType='NS'

elif HDNS == "0010":

self.DnsType="TXT"

else:

self.DnsType="Unknown"

tipo = (ord(data[2]) >> 3) & 15 # Opcode bits

if tipo == 0: # Standard query

ini=12

lon=ord(data[ini])

while lon != 0:

self.dominio+=data[ini+1:ini+lon+1]+'.'

ini+=lon+1

lon=ord(data[ini])

def respuesta(self, ip):

packet=''

if self.dominio:

packet+=self.data[:2] + "\x81\x80"

packet+=self.data[4:6] + self.data[4:6] + '\x00\x00\x00\x00' # Questions and Answers Counts

packet+=self.data[12:] # Original Domain Name Question

packet+='\xc0\x0c' # Pointer to domain name

packet+='\x00\x01\x00\x01\x00\x00\x00\x3c\x00\x04' # Response type, ttl and resource data length -> 4 bytes

packet+=str.join('',map(lambda x: chr(int(x)), ip.split('.'))) # 4bytes of IP

return packet

if \_\_name\_\_ == '\_\_main\_\_':

parser = optparse.OptionParser()

parser.add\_option("-f", "--filename", action="store", type="string",dest="SaveFile", help="input a filename to log output too")

(options, args) = parser.parse\_args()

ip='192.168.1.1'

print('pyminifakeDNS:: dom.query. 60 IN A %s' % ip)

udps = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

udps.bind(('',53))

try:

while 1:

data, addr = udps.recvfrom(1024)

p=DNSQuery(data)

udps.sendto(p.respuesta(ip), addr)

print( 'Respuesta: %s -> %s -> %s -> %s' % (addr[0], p.DnsType, p.dominio, ip))

if options.SaveFile:

MyDate=time.strftime('%Y %m %d')

MyTime=time.strftime('%H:%M:%S')

logfile = open(options.SaveFile,"a")

logfile.write('%s,%s,%s,%s,%s,%s\n' % (MyDate,MyTime,addr[0], p.DnsType,p.dominio,ip))

logfile.close

except KeyboardInterrupt:

print ('Finalizando')

udps.close()

client code to send packet

from scapy.all import \*

def synFlood(src,tgt):

for sport in range(1024,1500):

L3=IP(src=src , dst=tgt)

L4= TCP(sport=sport , dport=1337)

pkt=L3/L4

send(pkt)

for i in range(0,250):

src = "10.0.0.1"

i+=1

tgt = "10.0.0.4"

synFlood(src,tgt)

**Implementation**

In l2\_learning.py file add

if((packet.src==”00:00:00:00:00:01”)or(packet.dst==”00:00:00:00:00:01”)) before flow installation code.

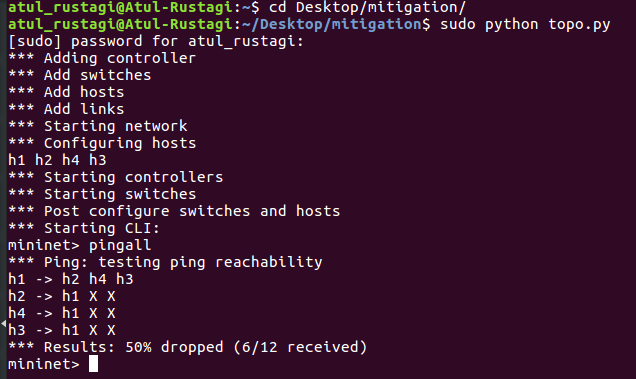
In terminal change directory to pox and run “sudo ./pox.py log.level –DEBUG forwarding.12\_learning” command to start controller

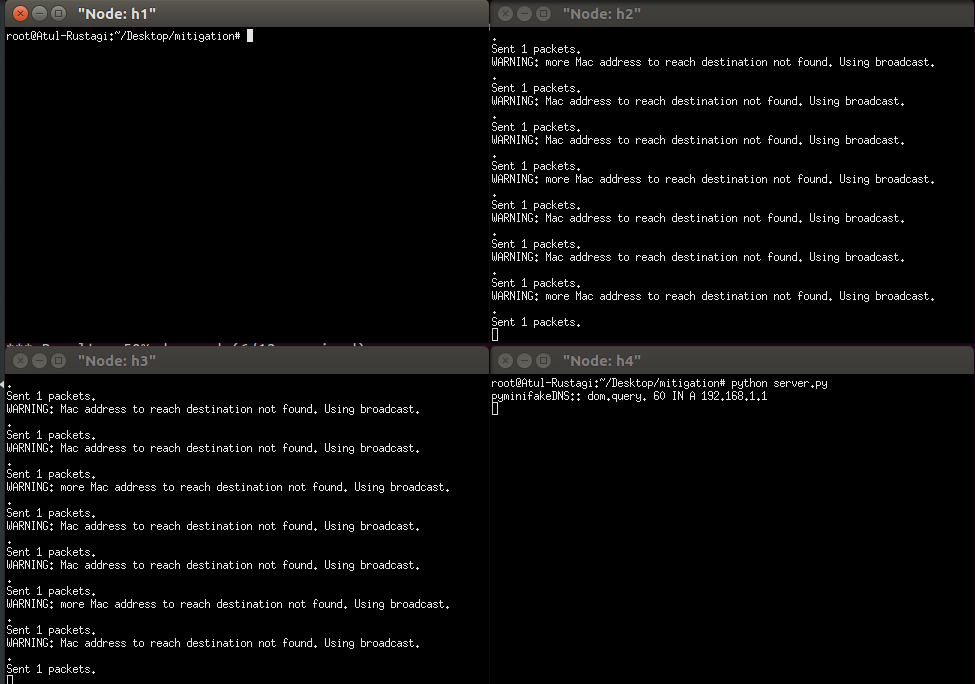
And in another terminal, change directory to folder containing client, server and topology file and run “sudo python topology.py” to start topology

To check flow of packets run “pingall”

Open xterm h1 h2 h3 h4 and run server file in h4 and client file in h2 and h3 to attack

Since these are blocked thus packets will go till controller with message that no more mac to forward packet but when client runs in h1, packets are sent without any warning.





**Result**

When attack is done, and wireshark is opened from host h1. We can see that packets catched are sent by attacking hosts h2, and h3 thus shown in red (flood attack) and for them flows are getting installed in the controller as the packets catched have the source mac address is of host h3 and h4 but IP of host h1 (IP spoofing).

In its graph, every second nearly 40 packets are sent which are getting acknowledged by server which are not actually sent by genuine host h1

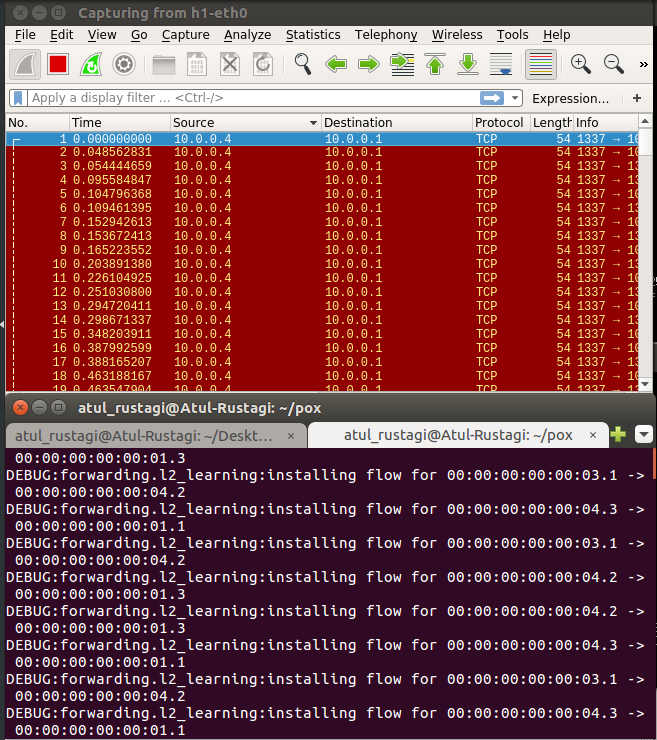
But in mitigation wireshark is catching packets sent by host h2 and h3 as they are having IP of host h2 but are not transmitted to server (host h4) and thus again and again sent for retransmission.

In its graph, initially 3-5 packets are sent which do not get any acknowledgement thus are dropped and sent again and again for retransmission.

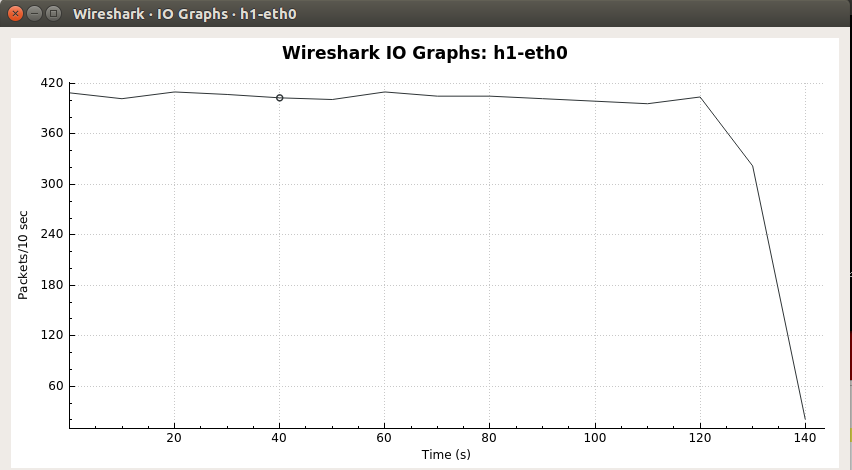
Also, no flows are installed for those packets in the controller.

**Packets catched and flow installation**

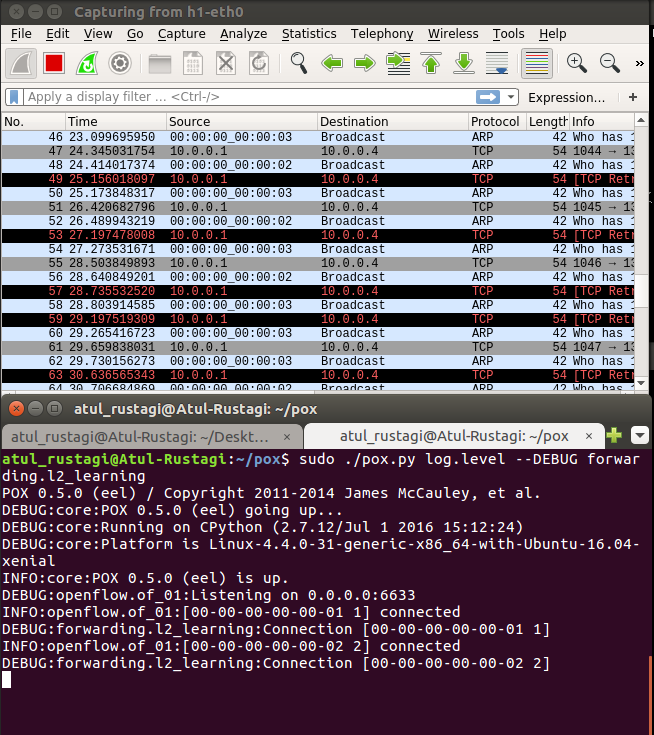
Attack



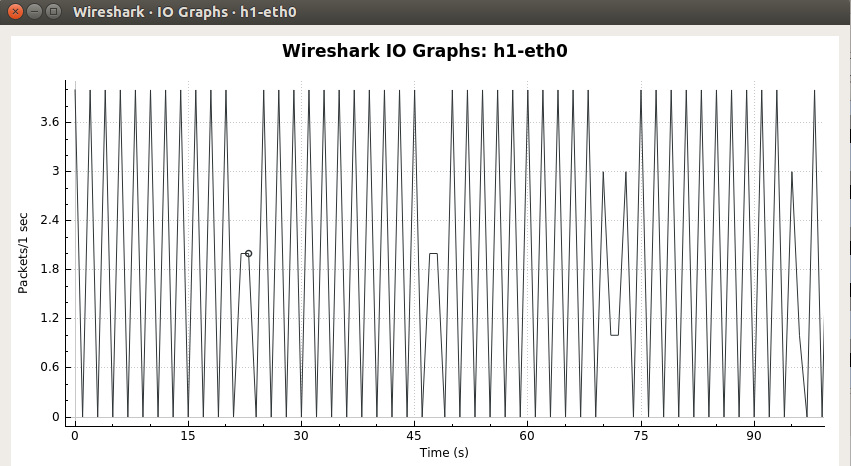
Graph



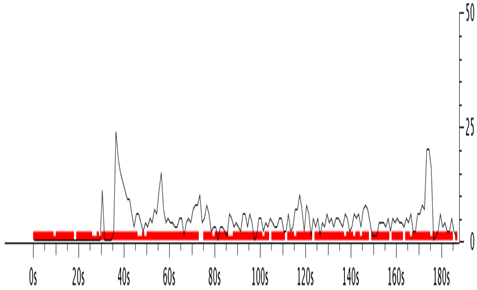
Mitigation



Graph



Normal flow



**References**

Klein, Amit, and Zohar Golan. "System and method for detecting and mitigating DNS spoofing trojans." U.S. Patent No. 8,266,295. 11 Sep. 2012.

Klein, A., & Golan, Z. (2012). *U.S. Patent No. 8,266,295*. Washington, DC: U.S. Patent and Trademark Office.

Klein, Amit, and Zohar Golan. "System and method for detecting and mitigating DNS spoofing trojans." U.S. Patent 8,266,295, issued September 11, 2012.

Klein, A. and Golan, Z., EMC Corp, 2012. *System and method for detecting and mitigating DNS spoofing trojans*. U.S. Patent 8,266,295.

Klein A, Golan Z, inventors; EMC Corp, assignee. System and method for detecting and mitigating DNS spoofing trojans. United States patent US 8,266,295. 2012 Sep 11.