CAuth – Protecting DNS application from spoofing attacks

NM SAHRI† and Koji OKAMURA††

Summary
UDP-based DNS packet is a perfect tool for hackers to launch a well-known type of distributed denial of service (DDoS). The purpose of this attack is to saturate the DNS server availability and resources. This type of attack usually utilizes a large number of botnet and perform spoofing on the IP address of the targeted victim. Therefore, it is hard for the DNS provider to differentiate between legitimate and attack DNS query packets. We take a different approach for IP spoofing detection strategy to protect the DNS server by utilizing Software Defined Networking (SDN). In this paper, we present CAuth, a novel mechanism that autonomously block the spoofing query identified with no impact on legitimate queries. By manipulating Openflow control message, whenever a server controller receives query packet, it will send an authentication packet back to the client network and later the client controller also responds via authentication packet back to the server controller. The server controller will only forward the query to the DNS server if it receives the replied authentication packet from the client. From the evaluation, CAuth instantly manage to block spoofing query packet while authenticate the legitimate query as soon as the mechanism started. Most notably, our mechanism designed with no changes in existing DNS application and Openflow protocol.

Key words:
DNS flooding attack, spoofing detection, authentication, network security, Openflow, SDN

1. Introduction

One of common cyber-attacks is Denial of Service (DoS) which aim to cripple down the attacked application online resources. By doing this, the offered services will not be available to intended users. Usually, the attack attempt to temporarily or suspending the services of its hosting provider. Making it worse, the UDP DDoS which is an aggregate of attacks scenario that exploit UDP protocol does not require any connection to be established prior the communication between the client and server. Each of this packet requires computational resources from the targeted server, thus this overloads the victim's capabilities.

Majority of recent large volume of DDoS attack use flooding technique, such as exploiting DNS servers and Network Time Protocol (NTP) [1]. Since almost every Internet services depend on DNS, it is much more damaging than the others are. The introduction of Open Recursive DNS server such as OpenDNS spike security threat higher to this type of attack since it permits any IP addresses to access their service for IP resolution. DNS, a UDP based network and always considered as a fundamental Internet service is crucial and the services need to be offer to the whole public. However, it is much easier for hackers to spoof the source IP address as UDP itself is connectionless and does not require a handshake like TCP does.

The hackers can utilize a large number of botnet army with spoofing the victim IP address and make a large number of DNS query attempt to flood the DNS server with request for the services. By spoofing the victim IP address, it will make it harder for the application servers to distinguish between the attack and legitimate query, since the application are designed to accept any range of IP addresses to process the DNS query. As a result, it would simply process all the DNS query from both of them and send the responses which will limit the resources of the DNS server to process other legitimate request. The other type of attack that manipulate the open recursive DNS server known as DNS Amplification attacks. This attack aims to amplify the attack traffic to a targeted victim. Since the open recursive server accept any source to send query packet, hacker include victim IP address in the DNS query packet (spoofing) and the query packet size is much smaller than the response packet, so the attack is amplified to the victim with higher impact. Current protection against this type of threat, such as IDS or firewall, is having difficulties of differentiate which response packet is attack or legitimate. Furthermore, by using only the network statistic behavior to identify the malicious attempts, it is not enough to separate or blocking such intelligent attacks and most of the preventive action has been left to the victim side [2].

The emergence of Software Defined Network (SDN) that promise the simplicity in managing networks seems to be the future of the current Internet architecture. By separating the control plane that orchestrated by logical network controller platform and data plane as a forwarding drive, SDN, can play an important role as a network protection tool against DDoS attacks. An effective defense against spoofing UDP based DDoS attacks on DNS servers requires source address spoofing detection. Assuming the SDN-managed network is implemented in where the DNS server reside can distinguish between spoofed DNS packets from real queries, it can selectively drop those spoofed packets and authenticate the legitimate DNS query packet with little collateral damage.

Manuscript received June 5, 2016
Manuscript revised June 20, 2016

In this paper, we present a novel spoof detection mechanism that exploit the UDP protocol to launch the attack on the DNS server. This mechanism autonomously blocks the "unwanted" DNS query packet that force the DNS servers to amplify the unnecessary legitimate traffic to the victims. In our approach, a server controller term is introduced and this controller reside in the DNS server domain. The server controller responsible to send an authentication packet to each hosts that request for the DNS services previously. By validating the "authentication packet" that replied by the requesting client network, the server controller is able to determine if a DNS query launch from any of source IP address is indeed a legitimate query or an attack packets. We developed a module called CAuth, which can be implement without any changes in DNS application servers. CAuth can be deployed at any time during DNS server runtime without require dataset training or manual tuning from the administrators. Furthermore, we did not use any statistical analysis for the anomalous flow behavior detection.

In Section 2, we discuss background, some existing approaches and related works for defending against UDP flood and amplification attacks for the DNS service specifically. With the granularity and the flexibility promised by SDN, we state the objective of our proposed method and introduce the idea for the protection against IP spoofing attack on DNS services in Section 3. In Section 4, we empirically evaluate the effectiveness of the approach and present the analysis of the experimental results. Finally, in Section 5, we discussed some of the important finding in our experiment and Section 6 summarize the paper.

2. Background

DNS, a UDP based network services is crucial and the services need to be offer to the whole public. However, it is much easier for hackers to spoof the source IP address as UDP itself is connectionless and does not require a handshake like TCP does. Therefore, it is worthwhile to design a countermeasure against DDoS flooding that suited the DNS traffic. Through the years, quite a number of DDoS attack detection method against the DNS server has been proposed, but these methods have certain drawbacks such as no strong incentives for the providers to employ since it protects the others network but not protect themselves from the flooding attacks, false positive and false negative, etc. In [3], the author proposed a method to detect the spoofing DNS query packets. The method requires the DNS server to generate some type of cookies embedded in the DNS response packets. However, it only can authenticate the requests between DNS servers. The main attack tools that is generated from the general DNS clients, cannot been verified.

Network Ingress Filtering [4], a mechanism that deploy filters at the border of the network to block incoming source IP addresses that not origin from the network itself. Unfortunately, the effectiveness of this method depends on the global deployment across the Internet. This method is "neighborhood policy" than require all ISP to participate to provide the list of IP addresses that does not belong to their network. No doubt, the spoofing IP problem can be solved by this method but it requires the needed information to be pass between the ISP efficiently. Furthermore, the information is passed between the ISP manually. This means that the attack packet might traverse through the network undetectable at the first time and it is too little too late to block the packets.

In [2], the author proposed amplification attack detection where they detect the attack using one to one mapping process between the DNS query and response. However, vast amount of database size could increase rapidly when traffic rate is high make the approach is not scalable. In Pushback [5], the mechanism allows network routers to limit the effect of DDoS attack to some destinations. In this work, every router has the capability to detect and drop suspicious packet. Pushback act locally and impose a rate limit on that particular traffic. SIFF [6], Stateless Internet Flow Filter permit the packet receiver to inform the router and selectively discard the flows from reaching their network. All the above-discussed related works is a network based solutions and detections method. The main issue in this type of solutions is the complexity of distributed environment, which require quite a number of network resources to be sacrifice.

2.1 SDN

Network management cover the area of security, network performance and reliability. It is often cause a headache for the network operator. They are not only responsible to maintain many of connected network devices, but also to satisfy the demand of increasing number of users. The current network management is lack of capabilities to support the demands and for this reason, SDN is introduced. SDN definition is described in the Open Networking Foundation (ONF) white paper [12], ''In the SDN architecture, the control and data plane are decoupled, network intelligence and state are logically centralized, and the underlying network infrastructure is abstracted from the applications''. SDN decouple the network devices to act as a forwarding plane only while the programmable control plane is done centrally. With this feature, the network configuration is performed with simple software update centrally and pushed to the whole connected devices rather than performing reconfiguration for the whole devices one by one. In reality, many research groups and industry parties has put their interest into SDN such as

Ethane [13] and OpenFlow [14]. Many commercial switch vendors such as HP and NEC support OpenFlow. For this reason, many network operators are implementing SDN in their network [15,16].

2.2 Security with SDN

With the programmability and resiliency of SDN architecture, it can be utilized to improve the network security. The network operator has the ability to deploy any security features on the fly to their network. Once any anomaly reported to the central controller, it can reactively response to analyze the situation. Based on this report, the controller can propagate the action to entire network at once. Many security researchers have started to recognize and proof that SDN is a workable tool to secure the network such as [17], the author proposed a DDoS detection method based on network flow features monitored by NOX/OpenFlow switches. Suspicious flows are investigated by analyzing the flow features with an artificial neural network which is Self-Organizing Maps (SOM). In [18], the author proposed a protection method to block attack traffic. In their work, they utilize the ability to redirect certain suspected traffic to certain ports after they detect some attacks in a specific service.

In [19], the author uses multiple type of anomaly detection algorithm in their research test where the author validates their algorithm in Small Office/Home Office (SOHO) environment. The author utilized Openflow for detecting network security problem close to the source of abnormality using the idea of decentralization control of the network devices. The author also uses periodic sampling for the flow statistics collection. Contrast to previous work, author [20] decoupled the controller communication channel with Openflow switches where the sFlow flow statistics collection method is used and the native Openflow communication channel is used only for the forwarding purposed separately. The experimental results show significant reduction in flow table size and the control plane load.

Few effort is given from researchers to explore the IP spoofing detection and mitigation mechanism with SDN. Most notably, SEFA [21] is route-based spoofing filtering that simply applied the existing IDPF [22] as the filtering application in the controller. VASE [23] is also spoofing detection method which utilize sampling and on the fly filtering configuration. This work is the extension from their previous work [24] which aim to detect the network attack by validating the source IP address of the incoming packets.

In this paper, we take a different approach for IP spoofing detection strategy to protect the DNS server by utilizing Software Defined Networking (SDN). In this paper, we present CAuth, a novel mechanism that autonomously block the spoofing query identified with no impact on legitimate queries. CAuth mechanism is designed without any changes to the applications itself. CAuth mechanism allow filtering rules to be created on demand, thus can save some expensive resources in the network for other purposes.

3. UDP Spoofing Detection Approach

In this section, we elaborate our approach that aims to detect the spoofing DNS query packet which gave high damage on the server side resources and performance. This approach exploits the ability of Openflow protocol [7] that provide secure communication channel between the network controller and the router/switches inside their network. The DNS query packet is generated using random source port number, make every query is unique to the DNS server and make it harder for the server side network to differentiate between legitimate and attacks query packet. We depict the main components of our architecture in Figure 1.

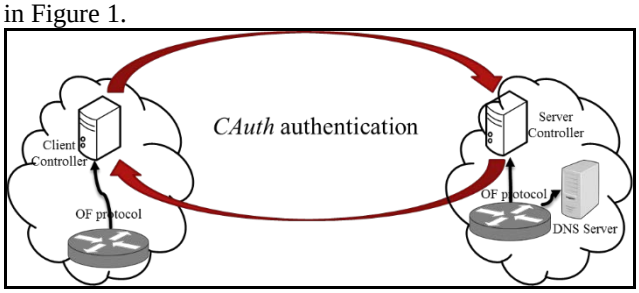

Figure 1: Main CAuth Architecture

3.1 Main Component of the Proposed Architecture

Based on Figure 1, our protection architecture against DNS flood attack composed of an SDN network controller in both client and server network domain, POX controller [8], the CAuth application that include as the main component in both of the network controller and Openflow switches that communicate with the controller via Openflow protocol. Our system require collaboration between client and server side network. Meaning that both client and server network is SDN managed network.

We define the term for the server side controller as server controller and client side controller as client controller. We make several assumptions about the architecture components in order for our protection application to suit the real environment. The assumptions are as follows:

- The *client controller* holds a list of legitimate internal network IP addresses. The controller uses the addresses to identify the inbound and outbound traffic that traverse through their network. The lists play a crucial part for the *client controller* decision making whether to perform

packet redirection or to accept the incoming legitimate DNS response packet. In real practice, IDS/firewall might also have the list of legitimate internal IP addresses to make filtering decision.

- The DNS flooding attack are initiated from botnet by using the spoofed IP address of the victim which made the decision to differentiate legitimate or attack traffic become more challenging task. Moreover, the botnet did not show any anomalous behavior that the network operator could easily detect and mitigate. Furthermore, current type of attackers tries to launch an attacks that aim to stay under the threshold value to avoid getting detected and dropped.

Based on these assumptions and the system architecture depicted in Figure 1, we design our SDN-based DNS spoofing blocking application CAuth component. The objective of this application is for the DNS service provider to be able to differentiate which DNS query is attack packets or legitimate queries which aim to protect itself from spoofing attack and able to protect the client resources at the same time.

3.2 CAuth Main Table Structure

In our proposed work, CAuth application record any inbound flow information into an active flow table. This main flow table contain all current active inbound flows record in their network. The structure of the table described as in Figure 2. In this table, two important field created to record specific information about a particular inbound flow. In $flow_l$, we record the 5-tuples (source port, destination port, source IP, destination IP and the protocol) of any inbound flow. Our approach records the 5-tuples packet header information that sent via Packet-In message as the input. The 5-tuples information is provided by the Openflow switch to the controller when the incoming flow has no match to be found in their flow table entries list.

In Openflow, when packets received by the datapath and sent to the controller, a control message OFPT_PACKET_IN is used [9]. This control message embeds the packet header together with other important fields to the controller for further decision-making by the central network controller. When the Openflow switches has no match for any incoming packet, it will also buffer the packet in their switch and provide a buffer ID, B_{id} to the central controller. It is used as a reference for the switches to find the match packet information that sent by the central network controller previously.

***CAuth* App Flow Table**	
Inbound Flow $(src_{ip}, dest_{ip}, src_{port}, dest_{port}, protocol)$	Pi_{hit} **Counter** $(min = 0, max = 2)$
$flow_l, B_{id}$	

Figure 2. Controller Inbound Table Structure

Next, we introduced Pi_{hit} counter. This counter is critical spoofing detection indication in our detection method. This field record how many times the same $flow_l$ information that was sent via Packet-In has been receive by both client and server controller. We elaborate in more details about this counter in the next section.

3.3 DNS Query Spoofing Protection Workflow

Initially, when a new query from any client (Figure 3, step 1) arrives at Openflow switches at the server network, it does not match any flow entry in the switch flow table. The DNS query packet is generated using random source port number, make every query is unique information to the DNS server. As a result, the switch will buffer the packet and send the copy of the packet to controller via Packet-In. When the server controller receives a Packet-In (Figure 3, step 2), assuming there is no information about this particular flow, CAuth will save the 5-tuples together with the packet buffer-id, B_{id} in the CAuth App Table (Figure 3, step 3). The buffer-id is unique value used to track the location of the buffered packet in the switch. Note that in this work, we did not consider the buffer arrangement problem in the open Vswitches. Since the flow information is not in the server controller list before, CAuth update the Pi_{hit} counter for $flow_l$ to 1. Initially, the value of Pi_{hit} is zero and we purposely set maximum value to two.

Next, we manipulate the ability of the Openflow protocol to make changes to the packet header information. Once the server controller records the $flow_l$ information in their flow table, in (Figure 3, step 4), CAuth replicate the Packet-In flow tuples received with no changes to the original buffered packet. Then, CAuth modify the flow information at the previous replicated packet to send this packet back to the client. The server controller modifies the UDP and IP header information. As example, the originally initiated DNS query packet source IP address is 1.1.1.1 and the destination IP address is 2.2.2.2 with source port number 1234 and destination port number is port 53. Now, server controller then swaps the IP and UDP information of the replicated packet, with the objective to send this packet back to the originator where the packet source IP address is now change to 2.2.2.2 and the destination IP address is 1.1.1.1 with source port number 53 and destination port number is port 1234. We provide the command line that replicate and modify the incoming

packet in Figure 3. Then, CAuth install the entry in the respected switch with the action to forward this packet back to the source of the query. The modified packet is used by our method as an authentication mechanism to detect the spoofed DNS query packet. It is worth to note that the original query packet still buffered in switch and the flow entry is yet to be installed.

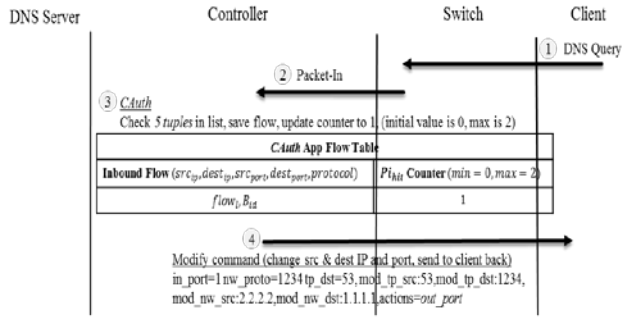

Figure 3. Initial CAuth defense strategy in server network

The client controller also implements CAuth App Table (Figure 2) in their database. When this authentication packet arrives at the client network, there will also no match for the packet since it is new flow information to the client network, thus this packet is sent to client controller for further decision making via Packet-In (Figure 4, step 1). When the client controller receives the modified packet from the server network, the authentication process started. First, the client controller will check the destination IP address of the packet header (Figure 4, step 2). The client controller will only perform the authentication process if the destination IP is in their network domain; otherwise, it will forward the authentication packet to the next hop until it reaches the destination. Here we assume that all client controller has a list of all known IP addresses in their domain. Furthermore, the client controller also checks the source IP address of the authentication packet (Figure 4, step 3). If the source IP address is from the server, the client controller further checks in their CAuth App Table. Since it is new information for the client controller, the 5-tuples information from the authentication packet is copied into the table and the Pi_{hit} counter updated to 1.

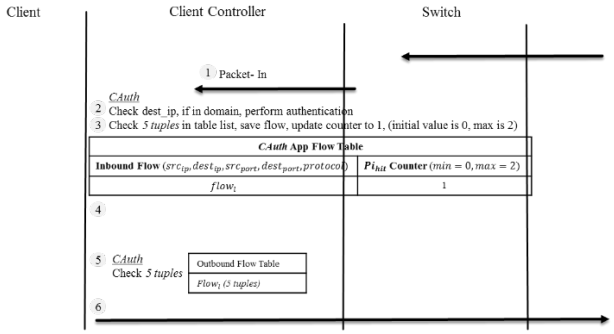

Figure 4. Initial CAuth defense strategy in client network

Next, the client controller modify the authentication packet received by swapping back the IP and UDP information in the packet header (Figure 4, step 4). This time, this authentication packet is aim to be redirect back to the DNS server. The client controller then checks the modified authentication packet 5-tuples information with their existing outbound flow table (Figure 4, step 5). If the client controller found a match, then the client controller confirms that there are DNS query packet previously sent to server but still waiting for the reply. Finally, client controller installs the flow entry in the switch with the action to forward the packet to the server (Figure 4, step 6). By this means, the authentication packet that originally sent by the server controller is redirect back by the client controller after the authentication verification process as explained before.

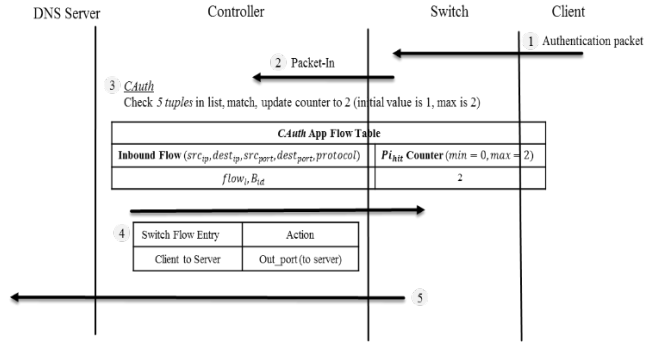

Figure 5. Server controller receive CAuth authentication packet

When the server controller receive the authentication packet sent from client, this packet information is not match any entry in the switch, since we did not install the flow entry for the original DNS query packet yet. As a result, this packet is being forward to the controller as Packet-in again (Figure 5, step 2). Then, the server controller refers the incoming authentication packet header with the CAuth App Table (Figure 5, step 3). This time, server controller will found a match for the authentication packet with the previously created flow information. Since there is exactly genuine first DNS query packet attempt to

the server, the second time controller receive the packet with the same flow 5-tuples information, it will update the Pi_{hit} counter to 2. After that, the server controller will free the previously buffered authentic DNS query packet by installing the flow entry into the switch in order to forward the DNS query to the respected DNS server (Figure 5, step 4). Subsequently, CAuth authenticate the first attempt query packet to use the DNS server services (Figure 5, step 5). In our work, we simply drop the packet that we used for our authentication mechanism. The authentication packet that created originally by the server controller and redirect back by the client controller are used only for the server network to authorize the DNS query packet to reach the DNS server.

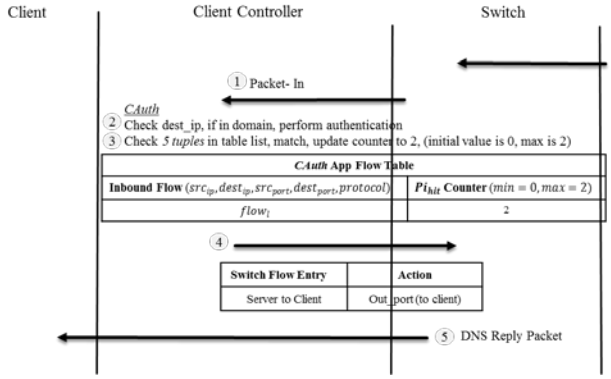

Figure 6. Client controller forward the DNS reply packet to client

When the DNS server process the received query, it will find the related query information as usual and reply with the DNS reply packet. Here, we did not modify any application packet field for the CAuth authentication process. The server controller will forward the DNS reply packet to the client based on the header information. When the client network receives the authentic DNS reply packet from server, again there will be no match found in the switch and the packet sent to controller (Figure 6, step 1). This time the client controller check that 5-tuple information is already created previously in the CAuth App Flow Table (Figure 6, step 3). From this, the client controller update the Pi_{hit} counter and install the flow entry in switch to enable the DNS reply packet to reach the client successfully (Figure 6, step 4). We provide CAuth authentication method pseudo-code for server and client network as in Figure 7 and 8.

```
Initialization: In_flow, Out_flow, flow_i, Pi_hit = 0/*(Pi_hit min = 0, max =
2), In_flow, Out_flow table record incoming and outgoing flow 5 tuple data
structure */
Input : p: PacketIn

For PacketIn
  If dest_ip is SERVER,
  Check flow_i in In_flow
    If empty /*not created yet*/
      flow_i = {src_ip, dest_ip, src_port, dest_port, protocol}/*Extract 5_tuple
from PacketIn into In_flow table if the 5 tuple not in table yet*/
      flow_id = buffer_id /*Extract buffer ID from PacketIn into In_flow
table*/
      update Pi_hit=1 for flow_i
      For flow_i, flow_id
        replicate flow_i to (flow_i)_a
        (flow_i)_a = flow_i (src_ip = dest_ip, dest_ip = src_ip, src_port =
dest_port, dest_port = src_port) /* modify(flow_i)_a tuple*/
        send(flow_i)_a to outport /*install entry for modified packet to out
from network)
    ElseIf not empty /*flow tuples info is already in switch*/
        if (flow_i)_a then check match flow_i /*matching the tuples in
incoming flow table */
        if match then update Pi_hit=2 for flow_i
        if Pi_hit=2 then
          forward flow_i, flow_id to server /*forward buffered flow
waiting in switch to server*/
          drop/ignore (flow_i)_a
          delete flow_i, flow_id from In_flow table
        End if
    End if
  End if
```

Figure 7. CAuth server controller defense

```
Initialization: In_flow, Out_flow, flow_i, Pi_hit = 0/*(Pi_hit min = 0, max =
2), In_flow, Out_flow table record incoming and outgoing flow 5 tuple data
structure */
Input : p: PacketIn

For PacketIn
  If dest_ip is in domain,
   If src_ip is from SERVER,
    Check flow_i in In_flow /*to check whether the same flow info has been
created in table or not*/
    If empty /*not created yet*/
      flow_i = {src_ip, dest_ip, src_port, dest_port, protocol}/*Extract 5_tuple
from PacketIn into In_flow table if the 5 tuple not in table yet*/
      update Pi_hit=1 for flow_i
      For flow_i,
        replicate flow_i to (flow_i)_a
        (flow_i)_a = flow_i (src_ip = dest_ip, dest_ip = src_ip, src_port =
dest_port, dest_port = src_port) /* modify(flow_i)_a tuple*/
        send(flow_i)_a to outport /*install entry for modified packet out to
SERVER)
    ElseIf not empty /*flow tuples info is already in switch*/
        if (flow_i)_a then check match flow_i /*matching the tuples in
incoming flow table */
        if match then update Pi_hit=2 for flow_i
        if Pi_hit=2 then
          forward flow_i to client
          delete flow_i from In_flow table
        End if
    End if
  ElseIf dest_ip is NOT in domain
    forward flow_i to destination
  End if
```

Figure 8. CAuth client controller defense

4. Performance Evaluation

4.1 Emulation Parameter

To validate the effectiveness of the proposed method, we develop the application of the mechanism to be part of POX controller module. To emulate a large scale of DNS spoofing attack, we use Mininet [10], a network emulator

that illustrate SDN environment with the support of virtual hosts, switches and network controller. The Mininet emulator use the real code for both the Openflow and Open vSwitch code and with the great functionality, as it can easily connect to the real networks. For our test, we use Mininet version 2.1.0 with the Openflow v1.3 supported. To emulate a real DNS traffic, well-known BIND 9.8.1 server is configured in separate Linux virtual machine to serve as the DNS server. Then, we bridge the Mininet SDN network domain to the BIND server virtual machine using the code function provided by the Mininet.

For our experiments, we create two types of DNS enquirers, legitimate clients and spoof attackers. The clients, attackers and the DNS server are design to be on different SDN network domain and have their own POX controller. We simulate a large-scale botnet to launch DDoS attacks on the protected DNS server in SDN environment. We purposely set the legitimate clients to issue DNS query rate, $\bar{x}_{norm}$ every 3 seconds whereas the bots, $\bar{x}_{attack}$ every 1 seconds. The reason why we set query rate higher for the botnet attacks is for the botnet to more active than the legitimate clients. The DNS query issued with the Poisson exponential inter-arrival time distribution. For the DNS query packet, we employ Scapy [11], a packet manipulation program that able to forge DNS packets easily. In our test, we set the number of legitimate client, n_{users} = 100 where one of the host in this network is configured to be the victim of spoofing IP addresses. The number of botnet attackers, n_{bots} = 400 is configured so that issue the DNS query at their given ferocities. All of this 400 hosts are configured to use the single victim IP address. We configure the BIND server to act as recursive server so that whenever the clients send DNS query packet, the BIND server is assumed to be responsive, so the DNS response returned as soon as possible. Every DNS queries generated by the Scapy is using a script where it randomly generates source port number and query DNS name, thus every incoming DNS query packet to the DNS server network is unique.

4.2 Number of Blocked Versus Number of Authenticated Packets

This section describes the evaluation results of the spoofing DNS query attack scenario; we configure all botnet to send the query as soon as the emulation start. We set all of each of the DNS queries generated from the bot is unique; random source port numbers and random DNS query name. By doing this, all of the DNS name and their IP addresses information requested by the bots is not cached in the local DNS server. By doing this, every time the server controller receives DNS query packet which different information and unique. This condition is also true for the legitimate clients. We deliberately generate a high throughput of DNS query packet to test the effectiveness of our proposed CAuth mechanism that implemented as a module in POX controller at both DNS server and client side.

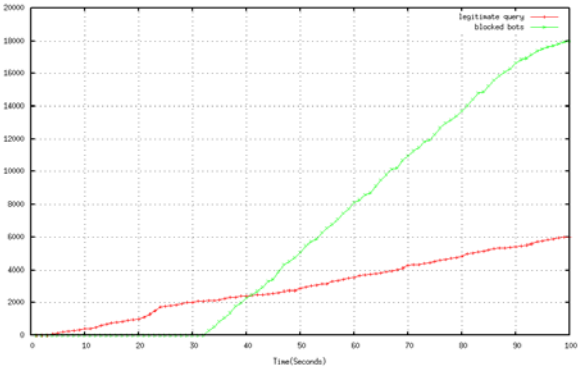

Figure 9. Number of blocked versus authenticated DNS queries packet

In this experiment, we measure the detection time that the server controller takes to block the spoofed DNS query packet while authenticate the query that issued by the legitimate clients. The botnets and legitimate clients launch DNS query packet that have randomly generated source ports and query name so that every-time the DNS server receive these query, it will act as a recursive. In Figure 9, the vertical axis is the number of measured substance, which is the number of blocked bots and the number of authenticated queries from the legitimate clients. We purposely launch the botnets to attack the DNS server at t=30 seconds to investigate whether it has effect on the legitimate query packets.

As can be seen at approximately t=33.2 seconds, the CAuth has start to perform the detection mechanism to block the spoofed queries. This indicate that CAuth are able to classify and block the spoofed attempt at any time before the server controller forward the queries to the DNS server. Even the attack packet generated at every t=1 second, the server controller depends on the preconfigured timeout to drop the identified spoofed packet. In our case, the server controller decides to drop the incoming DNS queries packet if it did not receive the authentication packet reply from the clients in 2 seconds. CAuth managed to block all botnets by roughly t=100 seconds. From the Figure 9 however, as the legitimate queries inter-arrival rate generated at every 3 seconds, even though CAuth can successfully differentiate between the spoofing and legitimate DNS queries, it takes a significant amount of time to forward the legitimate packet to the DNS server. The reason of this behavior is that CAuth require additional one round trip time (RTT) in order to authenticate the clients to use the DNS services. In this experiment, we found that the detection of botnet packets

has no direct impact on the legitimate queries packets. Starting from near t=4.1 seconds onwards, CAuth are able to forward all of the legitimate queries to the DNS server.

5. Discussion

5.1 DNS Provider Bandwidth Consumption

As we can observe, once the DNS server controller receive the query packet, it will send an authentication packet out to the clients that initiate the communication. Our mechanism ensure that the authentication packet sent by the server controller that was intended to reach the source will never reach the spoofed IP address. Here, we assume that no spoofing botnets army will receive the authentication packet sent by the server controller. Since the well-known objective of spoofing is to saturate the legitimate victim resources, the botnets will never receive any packet from the DNS server. In our works, the critical spoofing identification is when CAuth decide to drop any $flow_1.B_{id}$ that buffered and unprocessed DNS queries after the server controller did not receive the authentication packet back from the client network in time t=2 seconds.

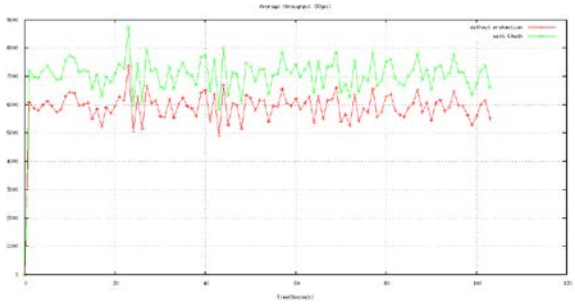

Figure 10. DNS provider bandwidth usage

Even though CAuth is effective in defending against spoofing IP addresses, another important issue to be highlighted is the effect of the proposed method to the DNS provider bandwidth. Since our method requires one additional RTT for any client to perform DNS query, it obviously increase a significant delay for the DNS resolution process. As mentioned before, this additional delay used as an authentication process in order for the DNS provider to be able to recognize between legitimate and attack packets. From the same emulation, we study the average bandwidth usage of the DNS provider bandwidth. The purpose is to find the impact to the provider bandwidth. As in Figure 10, as expected, even though our method can differentiate between legitimate and attack packets and able to block the attacks before it reaches the DNS server, our method consumes the usage of the provider bandwidth. From the experiment, our method increases on average 1.2 times higher comparing when there is no protection introduced. This is clearly a trade-off between the accuracy of the attacks packet detection and network performance of the DNS provider.

5.2 Client Bandwidth Effect

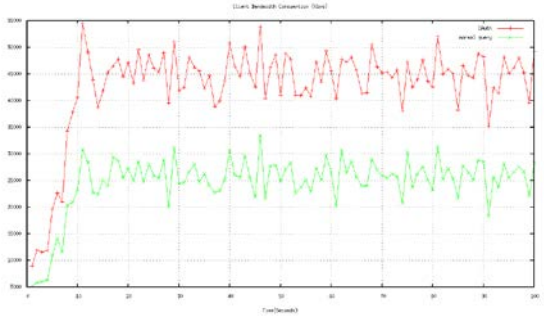

Figure 11. Client bandwidth usage

We also perform another emulation to find the effect of our method to the client network. Our method requires the server controller to send an authentication packet back to the source. This critical function introduced in order for the server network to make decision whether to accept or block the incoming packets that attempt to perform DNS queries. Botnets that launch the attack packets will never receive the authentication packet sent by the server network, but the legitimate client will. In this test, 100 hosts with different set of IP addresses are configured to launch DNS query packets towards the DNS server where the inter-arrival time of each packet is every 1 seconds. From Fig. 11, the client bandwidth consumption on average increased 1.7 times higher than a network without protection.

6. Conclusion

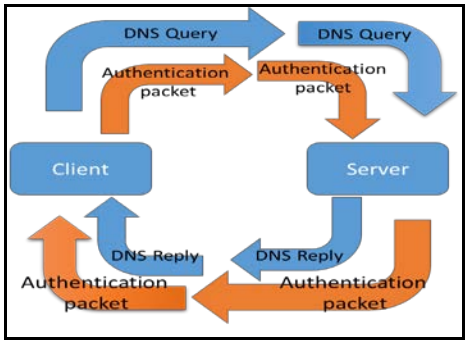

Figure 12. Summary of CAuth authentication process

We simplify the details of the *CAuth* workflow in Figure 12. In this paper, we proposed an efficient spoofing detection mechanism to detect spoofed DNS query packets against DNS servers thus ensuring the DNS operator to be

able to protect their server resources. The key thwart of our spoofing detection method is to block all queries from clients that did not reply back the authentication packet sent previously by the *server controller*. Our method authenticates the DNS queries that enter the server network domain for the second time. From this, the *server controller* makes a decision to forward the previously original DNS query that was buffered in the Openflow switches and simply drop the authentication packet received from the clients.

From the experiments, we can conclude that our method effectively blocks all of the DNS queries that was sent by the botnet. At the same time however, our method increases the bandwidth consumption on both the client and the server network. It is worth to note that, the UDP attack is fast since the size of the attack packets is small but it was not design to attack bandwidth but to consume the victim resources. This scheme is well suited to an SDN-administered network since the client and server controllers need to collaborate for the process of creating the authentication packets. Furthermore, worth to note that our scheme does not require high computation algorithm to detect the spoofed packet such as public key cryptography that usually used for authentication purpose. Moreover, we did not introduce any new protocols and all interaction between the client and server networks use standard Openflow protocol, make it as a lightweight spoofing detection method.

Acknowledgments

The author would like to thank the Ministry of Higher Education of Malaysia and University of Technology MARA Malaysia for financially supporting this research under Contract KPT(BS)790405085321.

References

[1] Arbor Networks. Q1 2015 Infrastructure Security Report. [Online]. Available: http://preview.tinyurl.com/kvacqcv

[2] G. Kambourakis, T. Moschos, D. Geneiatakis, and S. Gritzalis, “Detecting DNS Amplification Attacks,” in Workshop on Critical Information Infrastructures Security (CRITIS), vol. 5141. Springer, 2008, pp. 185–196.

[3] RFC 2827, “Network Ingress Filtering. Defeating Denial of Service Attacks Which Employ IP Source Address Spoofing.” Network Working Group, IETF

[4] F. Guo, J. Chen, and T. Chiueh, “Spoof detection for preventing DoS attacks against DNS servers,” in IEEE ICDCS, 2006, pp. 37–37.

[5] J. Ioannidis and S. M. Bellovin, “Implementing pushback: Router-based defense against ddos attacks,” in Proc. of the Symposium on Network and Distributed Systems Security (NDSS 2002), 2002.

[6] A. Yaar, A. Perrig, and D. Song, “SIFF: A Stateless Internet Flow Filter to Mitigate DDoS Flooding Attacks,” in Proc. of IEEE Symposium on Security and Privacy, 2004, 2004.

[7] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: Enabling innovation in campus networks. ACM SIGCOMM Computer Communications Review, 38(2):69–74, 2008

[8] POX. [Online]. Available: http://www.noxrepo.org/pox/about-pox/

[9] OpenFlow Switch Specification, Version 1.3.0 (Wire Protocol 0x04). [Online]. Available: https://www.opennetworking.org/images/stories/ downloads/specification/openflow-spec-v1.3.0.pdf

[10] Mininet. (2013, Mar). An Instant Virtual Network on your Laptop (or other PC). [Online]. Available: http://mininet.org

[11] SCAPY. <http://hg.secdev.org/scapy>.

[12] O.M.E. Committee et al., Software-defined Networking: The New Norm for Networks, ONF White Paper, Open Networking Foundation, Palo Alto, US.

[13] M. Casado, M.J. Freedman, J. Pettit, J. Luo, N. McKeown, S. Shenker, Ethane: taking control of the enterprise, ACM SIGCOMM Comput. Commun. Rev. 37 (4) (2007) 1–12.

[14] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, J. Turner, OpenFlow: enabling innovation in campus networks, ACM SIGCOMM Comput. Commun. Rev. 38 (2) (2008) 69–74.

[15] H. Kim, N. Feamster, Improving network management with software defined networking, IEEE Commun. Mag. 51 (2) (2013) 114–119.

[16] G. Gibb, H. Zeng, N. McKeown, Outsourcing network functionality, in: Proceedings of the First Workshop on Hot Topics in Software Defined Networks, ACM, 2012, pp. 73–78.

[17] R. Braga, E. Mota, A. Passito, Lightweight DDoS flooding attack setection using NOX/OpenFlow, in: IEEE 35th Conference on Local Computer Networks (LCN), IEEE, 2010, pp. 408–415.

[18] S. Lim, J. Ha, H. Kim, Y. Kim, S. Yang, A SDN-oriented DDoS blocking scheme for botnet-based attacks, in: Sixth International Conf on Ubiquitous and Future Networks (ICUFN), IEEE, 2014, pp. 63–68.

[19] Syed Akbar Mehdi, Junaid Khalid, Syed Ali Khayam, Revisiting traffic anomaly detection using software defined networking, in: RAID’11 Proceedings of the 14th International Conference on Recent Advances in Intrusion Detection, 2011, pp. 161–180.

[20] K. Giotis, C. Argyropoulos, G. Androulidakis, D. Kalogeras, and V. Maglaris, “Combining openflow and sflow for an effective and scalable anomaly detection and mitigation mechanism on sdn environments,” Computer Networks, vol. 62, no. 0, pp. 122 – 136, 2014.

[21] B. Zhang, J. Bi, T. Feng, P. Xiao, D. Zhou, Performing software defined route- based IP spoofing filtering with SEFA, in: the 23rd IEEE International Conference on Computer Communications and Networks (ICCCN14), IEEE, 2014.

[22] Z. Duan, X. Yuan, J. Chandrashekar, Constructing inter-domain packet filters to control IP spoofing based on BGP updates, in: Proceedings of IEEE Infocom, 2006.

[23] G. Yao, J. Bi, P. Xiao, Vase: filtering IP spoofing traffic with agility, Comput. Netw. 57 (1) (2013) 243–257.
[24] G. Yao, J. Bi, P. Xiao, Source address validation solution with OpenFlow/NOX architecture, in: 19th IEEE International Conference on Network Protocols (ICNP), IEEE, 2011, pp. 7–12.

Nor Masri Sahri received his first Bachelor Degree (B. of Information Technology) from Northern University of Malaysia on 2001 and obtained his Master Degree (MSc. of Information Technology) from University of Technology MARA on 2006. He has 6 years of industrial experience in one of the largest network service provider in Malaysia working as Senior Network Engineer. He is currently a Ph.D. student and belong to the department of Advanced Information Technology, Graduate School of Information Science and Electrical Engineering, Kyushu University, Japan.

Koji Okamura is a Professor at Department of Advanced Information Technology and also at Computer Center Kyushu University, Japan. He received B.S. and M.S. Degree in Computer Science and Communication Engineering and Ph.D. in Graduate School of Information Science and Electrical Engineering from Kyushu University, Japan in 1988, 1990 and 1998, respectively. He has been a researcher of MITSUBISHI Electronics Corporation Japan for several years and has been a Research Associate at the Graduate School of Information Science, Nara Institute of Science and Technology, Japan and Computer Center, Kobe University, Japan.