<PROJECT 최종보고서**>**

Hash puzzle based
DDoS mitigation solution

양석훈**,** 선민준**,** 전진용

신영길 교수님

권태경 교수님 **(MMLAB)**

이현민 박사님 **(MMLAB)**

Table of Contents

1. Abstract

DDoS 공격은 공격자가 다량의 Packet을 Host에게 전송해서 부하를 야기시켜 Host server가 정상적으로 동작하지 못하게 만드는 공격 방식이다. DDoS 공격은 이론적인 난이도가 높지 않음에도 불구하고 탐지가 어렵고 명확한 대응책이 없기 때문에 서비스의 가장 큰 위협이 되는 공격 중 하나로 손 꼽힌다. 현재 널리 사용되는 대응책으로는 경험적으로 수집한 데이터를 통해 특정 출처로부터의 요청을 무시하거나, 중간에 요청을 지연시키는 다른 서버를 거치게 하여 서비스에 가해지는 부하를 감소시키는 방법 등이 있으나 이런 방법들 모두 안전하지 않다고 말할 수 있다. 공격자가 트래픽을 증가시키거나, IP 주소를 변경해가며 공격할 경우 방어하지 못하기 때문이다.

이렇듯 현재 여러가지 DDoS 방어기법들이 연구되고 있지만, Kernel 단에서 실질적인 통신과정이나, 네트워크의 구조 자체를 변경한다면, 널리 사용되는 다른 서비스 들과의 호환성을 보장할 수 없기에 위에서 말한 방어기법들이 주로 사용되는 실정이다. 실질적인 환경을 변화시키는 방법 중에 한 가지는, Puzzle을 이용해서 DDoS 공격을 방어하는 기법이다. 이는, 서버는 Puzzle을 제공하고, 사용자는 이 Puzzle에 대한 올바른 Solution을 구해야만 서버가 사용자의 query를 수락하도록 디자인된 방법이다. 하지만, 기존의 연구에서 제안한 End-Host 통신사이에서 이루어지는 DDoS 방지 방법은 공격자가 Puzzle을 계속해서 요청하는 방법으로 공격을 시도할 경우 서버에 가해지는 부하를 효과적으로 줄이지 못한다는 문제점이 있다. 또한, 이 방법도 이론적으로 제안만 이루어진 방법이고, 실질적인 서비스나 구현체는 존재하지 않는다.

본 프로젝트에서는 위에서 말한 End-host puzzle solution의 기법을 차용하여 실질적인 DDoS mitigation solution을 제안한다. 또한, 기존 방법에 더불어 효율적인 Hash function을 제작하여 Puzzle로써 활용할 것이다. 위에서 제시한 기존 방안의 문제점을 해결하기 위해 Local DNS를 DDoS mitigation system의 참여자로서 추가하는 방법을 제안한다. 이러한 방안들을 실제 kernel 단에서 구현하여 하나의 System으로서 제작하여 구현하고, 해당 방법의 성능을 측정하여, 실제 DDoS mitigation System으로써의 제언을 하고자 한다.

2. Introduction

서비스 거부 공격Denial of Service Attack, DoS Attack)은 시스템을 악의적으로 공격해 해당 시스템의 리소스를 부족하게 하여 원래 의도된 용도로 사용하지 못하게 하는 공격이다[1]. 그 중

[1] Seo, Jung-Woo; Lee, Sang-Jin. “A study on the detection of DDoS attack using the IP Spoofing”. 《Journal of the Korea Institute of Information Security and Cryptology》 25 (2014): 147–153.

분산적으로 배치된 여러 공격자들이 동시에 목표를 공격하게 하여 탐지를 어렵게 하고 방지를 무력화하는 방법을 분산 서비스 거부 공격(Distributed Denial of Service Attack, DDoS Attack)이라고 한다. 이러한 DDoS 공격은 사전 탐지가 불가능에 가깝고, 공격자만을 분류해 낼 뚜렷한 방법도 없기 때문에 뚜렷한 대응책이 없는 상황이다.

웹 상에 대부분의 정보는 자신의 위치를 의미하는 고유의 Internet Protocol address(IP 주소)를 가지고 있다. 하지만 숫자와 점으로 이루어진 주소들을 외우는 것이 어렵고, 혼동하기 쉽기 때문에 비교적 외우기 쉬운 별칭을 설정했으며 이를 Domain Name이라고 한다. Domain Name Server (DNS)는 일종의 주소록으로서 Domain Name을 질의로 보냈을 때 해당 Domain Name이 의미하는 IP 주소를 반환해주는 역할의 서버이다. 만약 질의한 Domain Name에 대한 정보를 모를 경우 다른 DNS에게 질의하여 정보를 갱신하며, 이미 알고 있더라도 정보 최신화를 위해 일정 주기로 다른 DNS와 소통한다. 이러한 DNS에는 여러 종류가 있다. Local DNS는 특정 지역 혹은 통신사에 속한 DNS로서 자신이 담당하고 있는 구역에서 사용자가 질의하는 서버이다. 반대로 특정 구역을 담당하지 않고 자유롭게 사용할 수 있는 DNS를 Open DNS 라고 한다. Authoritative DNS는 Domain과 IP 주소의 관계가 기록, 저장 및 변경되는 DNS로서, 서비스 사용자가 자신의 Domain Name 과 IP 주소를 전파하기 위해 사용하는 DNS가 대표적인 예시이다.

DDoS 공격을 막기 위해 연구되는 방법 중 한 가지는 Puzzle을 이용한 방법이다. 서버에 최초 요청이 올 경우 서버가 Puzzle과 함께 요청 거부 패킷을 보낸다. 사용자는 요청에 제공받은 Puzzle과 함께 유효한 Solution을 함께 제시해야 하며 제시하지 못했을 경우 무시함으로써 1차적으로 부적절한 요청에 응답하는데 걸리는 시간을 줄이고, 두번째로 사용자가 Puzzle을 푸는데 걸리는 시간을 이용해 요청의 Frequency를 줄이는 데 목적을 둔다.

이번 프로젝트에서는 위에서 소개한 두 가지 방법을 병합하여 기존 Puzzle 기반의 DDoS 방어 기법에 DNS를 참여시켜 새로운 DDoS Defense System을 제안하고자 한다. 먼저, 기존에 이론적으로만 연구되었던 Puzzle 기반의 DDoS System을 보완하여, 새로운 Architecture로 디자인하고, 이를 모든 OS에서의 호환성이 보완되도록 Kernel 단에서 구현해내었다. 또한, 기존 DDoS Defense System의 참여자에는 Host (Target server)와 Client (Bot)만이 존재하였는데, 여기에 Local DNS를 참여자로서 추가시켜 더욱 효율적인 DDoS Defense System을 제한하고자 한다. 이를 위해, 해당 보고서에는 Hash puzzle을 기반으로 디자인된 System의 Architecture에 대해 소개하고, 이를 직접 구현해서 테스트한 결과를 통해 해당 System의 성공 여부에 대해 논할 것이다.

이 프로젝트의 의의는 이번 프로젝트가 잘 성공한다면, 이론적으로만 연구되던 기법을 실제로 테스트해서 확인해볼 수 있다는 점에 있으며, 기존보다 더욱 발전된, 여러 참여자들이 포함된

Network 구조를 만들었기에, 추후 실제 서비스로서 제언할 수 있음에 존재한다.

3. Background Study

A. 관련 접근방법/기술 장단점 분석

1) Cloud Flare

DDoS 공격을 방어하기 위해 가장 쉽게 생각할 수 있는 방법은 시간당 Host에 전송되는 Packet 수를 감소시키기 위해 기존 통신을 묶어 두는 것이 될 것이다. 현재 DDoS 방어를 가장 잘 한다고 알려진 기업인 Cloud Flare에서는 통신을 통해 query를 요청할 경우 해당 query를 약 5초 동안 freeze 시킨 다음, 해당 query가 여전히 유효하다면 query를 수락하는 방식의 방어기법을 사용한다. 이러한 방법은 대개 Syn packet만 보내는 DDoS 공격 기법을 잘 막아낼 수 있다는 장점이 존재한다. 그러나, 실제 유저 역시도 5초간 자신의 요청이 응답되지 않는 불편함을 겪는다는 단점이 존재하며, Bot의 개수 및 종류에 따라 난이도 조절을 하지 못한 채 동일한 policy를 적용할 수 없고, Bot이 증가할 경우 이를 효과적으로 방어하지 못한다는 여러 단점들을 가지고 있다.

2) Puzzle based DDoS mitigation Solution

기존의 Puzzle을 이용한 DDoS Mitigation Solution은 대체로 어떠한 Puzzle을 사용하는 것이 더 효과적인가를 탐구하고 있으며 시스템 자체에 대한 연구는 그리 많지 않다. 2005년 Feng et al은 Network Layer에서의 Puzzle을 이용한 Mitigation Solution을 제시하였다.[2] 이후 2009년 Abliz et al 이 제안한 Guided tour puzzle[3]이나 2011년 Abraham et al이 Traceback을 이용한 Puzzle의 개선[4] 등을 제안하는 등 여러 연구가 있었다. 하지만 이 방법이 적용되기 위해서는 모든 서버와 모든 사용자가 Puzzle을 내고 Solution을 구하는 과정을 통신 과정에 추가해야 하기 때문에 적용되지 못하고 이론

[2] Feng, Wu-chi, Edward Kaiser, and Antoine Luu. “Design and implementation of network puzzles.” Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies.. Vol. 4. IEEE (2005)

[3] Abliz, Mehmud, and Taieb Znati. “A Guided tour puzzle for denial of service prevention,” 2009 Annual Computer Security Applications Conference. IEEE (2009)

[4] Abraham. Anup Mathew, and Shweta Vincent. “Defending DoS Attackers Using a Puzzle-Based Approach and Reduction in Traceback Time towards the Attacker.” Global Trends in Computing and Communication Systems.: 4th International Conference, ObCom (2011)

연구에만 그치고 있다. 특히 공격자가 Puzzle을 풀지 않고 Puzzle 제공 요청만 계속해서 보낼 경우 서버는 다른 사용자로 인식하여 계속해서 Puzzle을 만들고 전송해야 하기 때문에 서버에 가해지는 부하는 크게 줄어들지 않는다는 문제점 또한 가지고 있다.

B. 프로젝트 개발환경

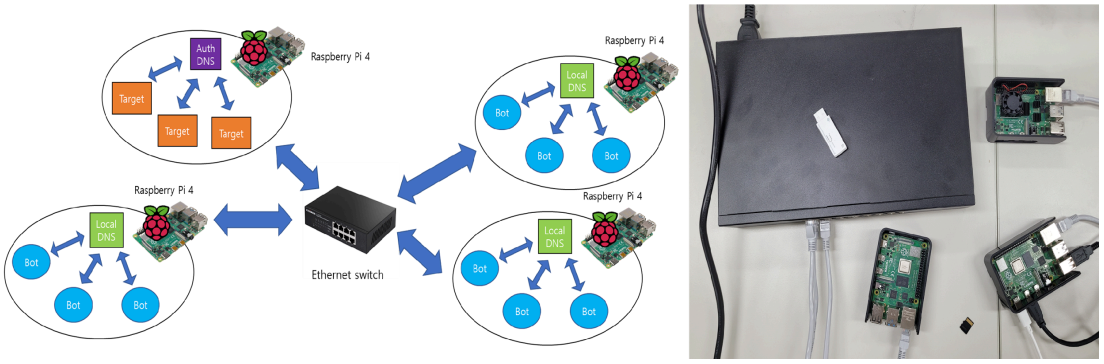

그림 **1. Diagram**

i. 공격 대상이 될 Host (Target server)와 Authoritative DNS, Local DNS의 역할을 할 Server 들을 자체적으로 구현한다. 이 server 들은 실제 OS 위에 올라가서 동작할 예정이며, Raspberry pi 위에 올려서 Server로서의 역할을 하게 한다. 자세한 다이어그램은 위의 그림을 참고하기 바란다.

ii. 공격자 Client들을 역시도 자체적으로 구현하여서 Raspberry pi 위에 올려 실험을 진행한다. Raspberry pi가 4코어이므로, 공격자의 역할을 하는 Raspberry pi 하나당 1개의 Local DNS, 3개의 Client를 할당하여 Concurrent한 결과를 확인하고자 한다. 자세한 다이어그램은 위의 그림을 참고하기 바란다.

iii. Puzzle 기반의 System을 Linux kernel을 직접 수정하여 구현해 낸다. 이는 OS간의 이식성 및 난이도 조절, 하위호환성 등을 위한 것이며, 이렇게 수정된 kernel을 바탕으로 Raspberry pi를 부팅시켜 해당 System의 성능을 파악할 것이다.

iv. Ethernet이 Bottleneck이 되지 않도록 별도로 Ethernet switch를 준비하여 사용한다.

4. Goal/Problem & Requirements

1. DNS 와 Host가 협력하는 DDoS Mitigation System 구축

    1) Local DNS가 사용자에게 Hash puzzle의 Seed값을 출제한다.

    2) 사용자는 DNS서버에서 받은 Puzzle의 Solution을 구해 Target server에 요청을 보낼 때 함께 전송한다.

    3) Target server는 올바른 Puzzle 및 Solution이 전송되었는지 검증하고, 올바를 경우에 대해서만 연결을 수락하고, 아닌 경우에는 연결을 거부한다.

2. 자체적인 Test Bed 구성
    a. Client와 DNS 들을 잘 구현해낸다.
    b. Bot을 구현하여 Raspberry pi 위에 올려 테스트를 진행한다.
    c. Server와 Bot들이 정상적으로 TCP 및 UDP 통신을 진행하도록 구현한다.
    d. 외부에 영향을 끼치지 않도록 자체적인 Test bed를 구성한다.
3. DDoS Mitigation System 구현 및 테스트 진행
    a. Puzzle기반 DDoS Mitigation System을 Linux kernel을 수정하여 구현해낸다.
    b. DDoS Mitigation System 적용시 Host가 받는 Traffic을 효과적으로 떨어트릴 수 있도록 구현한다.
    c. Puzzle의 난이도를 조절하여 효과적인 Host의 Rate limiting이 가능하도록 한다.
    d. Puzzle을 통하여 Host server가 다운되는 것을 방지할 수 있도록 한다.

5. Approach

A. Assumption

1. DNS는 신뢰할 수 있으며 공격에 참여하지 않는다.
2. DNS는 Local DNS Server와 Target Server의 Authoritative DNS만 존재한다.
3. Target Server와 사용자 간의 통신은 TCP로 이루어진다.
4. 이외의 통신은 모두 UDP로 이루어진다.

B. Approach

DNS 서버는 사용자의 컴퓨터에 찾고자 하는 Domain Name - IP Address 정보가 없을

경우 해당 정보를 질의하기 위한 서버로서 부하가 심하지 않고, 특히 Local DNS의 경우 담당하고 있는 사용자가 제한적이기 때문에 통신 과정이나 관리에 들어가는 비용이 적다. 또한 DDoS를 막고자 하는 서버 입장에서도 Local DNS를 통해 관리한다면 사용자들을 쉽게 묶어 관리할 수 있고, 이에 따라 어느정도 공격자와 일반 사용자 사이에 차등적인 정책을 관리할 수 있게 된다.

공격자 입장에서도 대부분의 경우 모든 이용자가 장애를 겪게 하는 것이 목적이 아닌, 특정 서버를 무력화하는 것이 목적이기 때문에 Local DNS에 많은 요청을 보낼 당위성이 적으며, 오히려 Local DNS가 무력화될 경우 공격자가 서버를 공격하는 데 방해가 될 뿐이기 때문에 Local DNS가 담당해야 할 비용이 크지 않을 것으로 예상된다.

따라서 서버가 문제를 내고 정답을 검증하는 과정을 모두 처리하는 기존의 방법에 Local DNS를 문제를 내는 역할로서 참여시킨다면, 서버의 부하를 크게 줄이면서 Local DNS가 담당해야 할 부하는 의미 없을 정도로 작을 것이라고 판단된다.

6. Project Architecture

A. Architecture Diagram

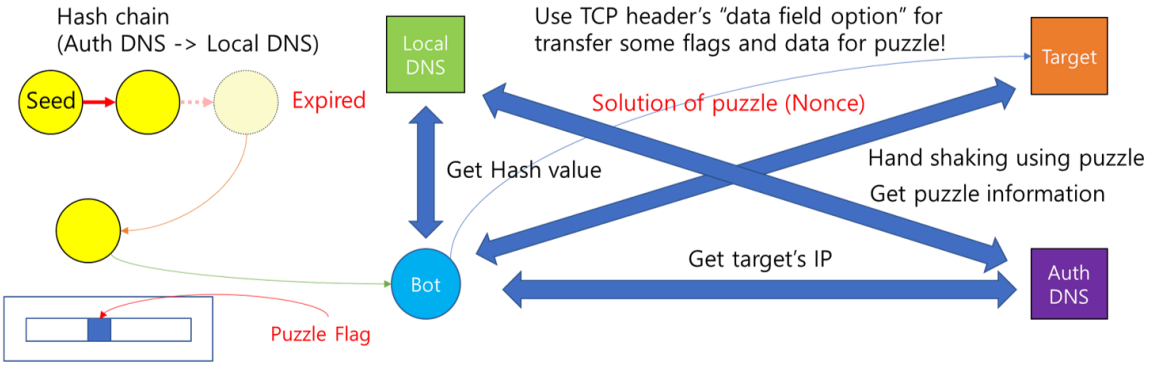

그림 **2. Architecture**

B. Architecture Description

1. Target Server가 DDoS attack을 감지하여 방지하기로 결정하고, Puzzle의 난이도를 설정한다.

2. 사용자(Bot)이 보낸 요청을 Target Server가 Puzzle Flag를 포함하는 패킷을 보내며 거부한다.
3. Bot이 Local DNS에 Target Server에 대한 Puzzle을 요청한다.
4. Local DNS가 Target Server에 Hash Chain의 마지막 값을 Puzzle로서 제공하고 사용한 값을 제거한다.
    a. 만약 남은 유효한 Puzzle이 없다면 Local DNS가 Target Server의 Authoritative DNS에 새로운 Puzzle 발급을 요청한다.
    b. Authoritative DNS는 Local DNS에 새로운 Hash seed와 유효 개수를 발급한다.
    c. Local DNS는 Hash seed를 이용해 유효 개수만큼의 길이를 가지는 Hash Chain을 생성한다.
5. 사용자가 Puzzle을 풀고, Solution 과 Puzzle을 포함해 Target Server에 요청을 보낸다.
6. Target Server는 Puzzle과 Solution이 유효한지 확인하고 요청을 처리한다.
    a. 유효하지 않을 경우 처리하지 않는다.

7. Implementation Spec

A. Input/Output Interface

이번 프로젝트는 System을 Desing하고 구현하여 평가하는 것이기에, I/O Interface의 경우 GUI 보다 CUI로 구현하였다. 작업이 기본적으로 Kernel 단에서 이루어졌기에 User level에서 이에 접근하기 위하여 System call을 활용하였다. 여기서, System call을 CUI 명령어처럼 사용할 수 있도록 구현하였다.

편의를 위해 System call을 여러 종류 구현해 놓았는데, 실행시 명령어를 보여주며, Arguments를 인자로 줄 경우 해당하는 system call을 출력한다. 퍼즐 정책 확인 및 변경, Caching된 퍼즐 정책 확인, 설정된 DNS IP 가져오기 등 총 12가지 System call을 사용할 수 있도록 구현해 놓았다. System call에 의해서 변경된 Puzzle policy들은 Socket 프로그램이 connect() 혹은 accept()을 호출할 경우 변경된 kernel logic에 따라 자연스럽게 Puzzle policy가 적용되도록 디자인하였다.

B. Inter Module Communication Interface

내부 모듈 간의 통신은 TCP, UDP 통신을 통해서 이루어진다. UDP 통신의 경우, Puzzle 관련 정보를 전달하는 역할을 하기에 대부분 기존 통신 구조를 거의 그대로 따라가도록

구현하였다. TCP 통신의 경우에는 Puzzle 정보의 전달에 더불어 Puzzle을 풀고 검증하는 과정 등이 추가되어야 했기에 변경된 부분이 많은데, 변경점을 제외한다면 일반적인 통신 규약을 따라 네트워크 통신을 진행한다.

TCP segment header

Offsets		0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
0	0	Source port																Destination port															
4	32	Sequence number																															
8	64	Acknowledgment number (if ACK set)																															
12	96	Data offset				Reserved 0 0 0 0				CWR	ECE	URG	ACK	PSH	RST	SYN	FIN	Window Size															
16	128	Checksum																Urgent pointer (if URG set)															
20	160	Options (if *data offset* > 5. Padded at the end with "0" bits if necessary.)																															
⋮	⋮																																
56	448																																

그림 **3. TCP Segment Header**[5]

변경점의 경우, 위 그림에서 표시한 Data offset field를 이용하여 TCP header를 확장한다. 이렇게 확장한 TCP header에 Puzzle Flag, Puzzle, Solution 등에 대한 정보를 추가하여 통신시 자연스럽게 해당 정보들이 전달되도록 구현하였다. 1 word를 Token hash (Puzzle)에, 1 word를 nonce(Solution)에 할당한다. TCP 연결 과정은 3-Way Handshaking으로 구현한다.

TCP Handshaking 과정에서 Puzzle data의 전송 및 검증 등을 위해 과정을 약간 수정하였다. 아래의 그림에서 검은색 선이 기존의 TCP Handshaking 과정이고, 빨간색으로 구성된 선이 새롭게 추가한 과정이다. 기존 TCP Handshaking 과정에서 Puzzle 관련된 작업들이 자동적으로 실행되도록 추가한 것이다. Design을 보면 Solution이 틀릴 경우 Socket을 연결하기 이전에 Query를 Discard해야 하기에 Handshaking 과정을 수정하여 추가적으로 몇 개의 작업을 실행하도록 한 것이다.

[5] 출처 : https://en.wikipedia.org/wiki/Transmission_Control_Protocol

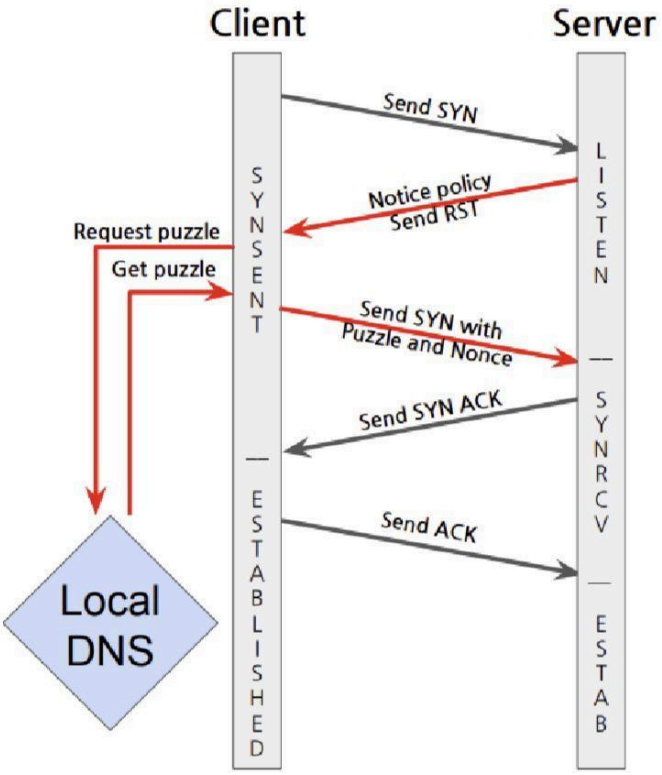

그림 **4. Modified TCP Handshaking**

C. Modules

1) Client (Bot)

Client는 Query를 요청하고, 이 Query가 Puzzle solution의 부재로 인해 거절된다면, DNS로부터 Puzzle token을 받아와서 풀고 solution을 함께 보내는 과정을 가진다. 이 모든 과정은 Kernel에서 자동적으로 진행되도록 구현하였다. Puzzle을 풀 때는 단순한 Naïve한 알고리즘을 통해서 푸는 것으로 계획을 세웠다. 이 모든 과정을 Kernel에서 자동적으로 일어나게 모듈화 함으로써 User는 이 과정을 알 수 없고, 관여하지 못하도록 디자인하여 안정성 높은 System을 구현하고자 하였다.

2) Host (Target server)

Host는 Client가 전송한 Query들을 처리하기 전에 Puzzle 및 Puzzle solution을 검증하는 과정을 거친다. 이 과정 역시도 Kernel단에서 자동적으로 검증이 진행되고, 이후에 Socket을 열도록 구현하였다. 이는 Socket programming의 과정을 수정해서

구현되었다.

3) DNS server

구현한 DNS server는 크게 Local DNS와 Authoritative DNS로 나뉜다. 이들 DNS는 모두 Puzzle 관련 정보들을 받아서 다른 주체에게 정보를 전달해주는 역할을 한다. 디자인한 DDoS Mitigation System에서는 여러 Local DNS 및 Authoritative DNS를 활용하는데 이들을 모듈화 시켜서 전체적인 큰 동작들은 모두 비슷하게 구현하였다. Hash Chain 생성의 경우는 역시 Kernel단에서 자동적으로 생성이 이루어지도록 구현해 놓았다.

8. Solution

A. Implementations Details

기본적으로 Linux kernel을 수정하여 TCP 통신 과정을 수정해서 Puzzle과 관련된 여러 정보들을 통신 한 번으로 주고받을 수 있도록 구현하였다. Kernel에서 수정한 파일들에 대한 간략한 설명들은 아래 표에 정리해 놓았다. 아래 표와 별개로 동작을 위해 추가적인 파일들을 수정하여 자료구조를 수정하거나, 알고리즘 및 동작 구조를 수정하였지만, 아래 표에는 이 중 핵심적인 Detail들만 골라서 작성하였다.

표 **1. Linux Kernel Modification**

net/ipv4/tcp_ipv4.c	- 함수 호출 순서 조절 - tcp_v4_send_reset() 퍼즐 정보를 포함하여 발송하도록 수정
net/ipv4/ip_output.c	- ip_send_unicast_replay() 퍼즐 정보를 포함하여 발송 할 수 있도록 수정
net/ipv4/tcp_input.c	- tcp header option 파싱 추가 - tcp_rcv_state_process() 수정 LISTEN일 경우 puzzle 과 nonce 값 확인 후 틀렸으면

	거절 - tcp_rcv_synsent_state_process()의 반환값에 따른 처리 추가 puzzle 정보 갱신후 재발송시 연결을 그만두지 않고 재시도 - tcp__synsent_state_process() 수정 puzzle 정보 갱신 요청시 갱신 후 재발송
net/ipv4/tcp_output.c	- __tcp_transmit_skb() 함수에서 puzzle 정보를 포함하도록 변경 - tcp header option 작성시 puzzle 관련 항목 추가
net/puzzle.c	- 퍼즐 정책 관련 데이터 구조 추가 - 관리할 수 있도록 시스템콜 추가 - 퍼즐에 사용할 hash function 추가

위의 표에 추가하여 다른 동작들도 구현해내었다. Host server와 DNS server들은 모두 Socket programming을 통해서 구현하였다. DNS server가 참가하는 통신은 현실을 반영하기 위해 모두 UDP 통신을 사용하도록 구현해내었고, 다른 통신들의 경우 위에서 Kernel을 통해 수정한 TCP 통신을 활용하도록 구현하였다.

B. Implementations Issue

Kernel을 직접 수정하는 만큼 구현하며 매우 많은 문제들에 직면하게 되었다. 이로 인해 여러가지 Approach를 시도해보며 계속 구현 방식을 조금씩 변경하게 되었다.

첫 번째로는, Kernel을 직접 수정하지 않고, Module을 통해 Hooking하는 방법을 사용하고자 하였다. 이는 Netfilter를 이용해서 Linux Kernel을 수정하는 것이 아닌 Module을 구현했을 때 Module이 자연스레 Kernel에 올라가도록 하는 방법이다. 이는 Kernel을 직접

수정하지 않기에 Kernel panic 발생시 복구가 용이하고, 매번 Kernel의 빌드를 할 필요가 없기에 빌드 시간이 획기적으로 단축되어서 빠른 개발을 할 수 있었기에 이 방법을 선택하였었다. 그러나, 이 방법의 경우 보안 문제로 인해 TCP Handshaking 과정에는 접근할 수 없도록 규정이 정해져 있었고, 이 규정을 수정한다면 오히려 Kernel을 수정하는 것보다 더 큰 문제들이 발생해서 배보다 배꼽이 더 큰 상황에 직면하였다. 이로 인해 5월 중순까지 개발하던 해당 Approach를 모두 포기하고 다시 구현에 착수하게 되었다.

두 번째로는, TCP Handshaking을 Kernel단에서 수정하고, 이 과정에서 여러 함수와 구조체들을 직접 수정하는 방식을 시도하였다. 그러나, 실제 동작시에 Puzzle 관련된 Option들이 자꾸 누락되는 결과를 보았는데, 이는 SYN packet과 관련이 있었다. SYN packet이 없을 경우, TCP option 중 일부는 고려되지 않으며, 이로 인해 기본적으로 call-by-value 방식으로 TCP header 값을 받아서 사용하도록 구현되어 있었다. 따라서, 전체적인 과정을 보았을 때, Puzzle policy가 적용된다면 이 방식대로 구현하는 것에 문제가 발생하였으며 이 문제점으로 인해 5월 말까지 시도하였던 이 방식을 포기하게 되었다.

마지막으로, 구현을 하던 도중 TCP 통신에서 TCP header만을 사용하는 것이 아니라 TCP 통신을 할 경우 이 통신이 IP layer로 올라가서 오히려 IP header 및 Buffer의 값들이 실제 통신에 사용되는 것을 볼 수 있었다. 이로 인해 기존의 TCP header를 수정하던 방식에는 문제점이 생겨서 구현 방법을 한 번 더 수정하게 되었다.

9. Results

A. Experiments

1) Puzzle policy를 적용하지 않은 경우의 DDoS Attack Test

Puzzle policy를 적용하지 않고, 비교를 위해 기존의 DDoS attack을 자체 구현한 Test bed에서 실험해본 결과는 아래 표와 같이 나타난다. 여기서, 시간당 전송하는 Packet의 수는 Client 12개를 모두 활용한 경우에 나타난 결과이며, Server의 부하를 측정하는 과정에서는 Client의 개수마다 Server의 부하를 측정해보았지만, 아래 표에는 편의를 위해서 그중 일부만을 표에 기록하였다. 측정 방법은 Packet을 1000000번 보낸 것을 기준으로, 걸린 시간을 측정하여 시간당 전송한 Packet의 수를 측정하고, CPU Resource는 해당 시간의 평균을 구하여 기록하였다.

표 **2. DDoS without Puzzle Policy**

Packet / sec (1 Client)	4249 Packet/s
Server CPU Resource (1 Client)	1.235%
Server CPU Resource (2 Client)	1.605%
Server CPU Resource (3 Client)	2.0665%
Server CPU Resource (12 Client)	6.8376%

2) Puzzle의 효율성 검증

Puzzle system을 Linux kernel에서 구현해낸 후, 이 구현이 잘 되었는지를 확인해보았다. Threshold를 설정할 경우, 각 Threshold에 따라 Puzzle을 푸는데 걸리는 시간을 측정하여서 아래 그래프로 그려 놓았다.

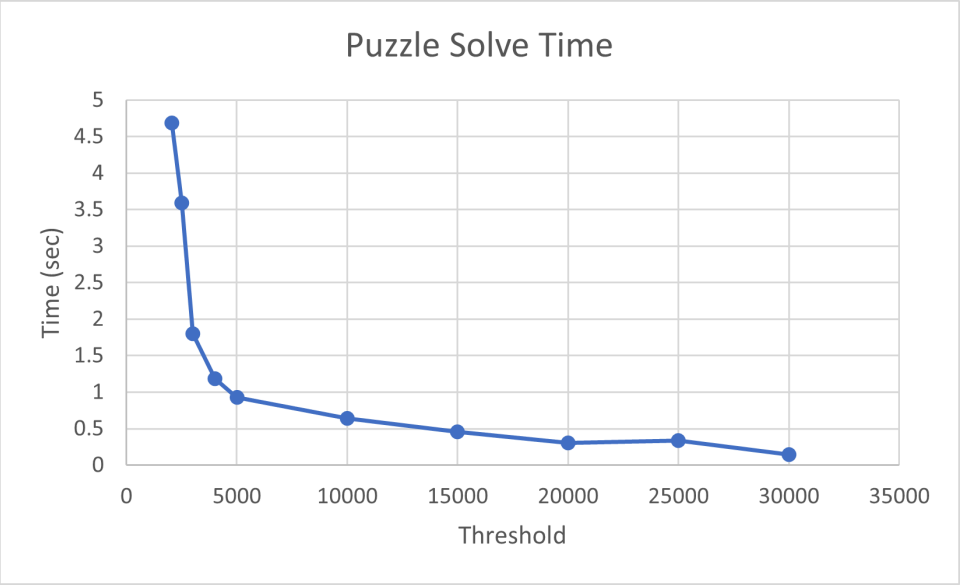

그림 **4. Puzzle Solve Time**

3) Puzzle의 난이도에 따른 DDoS Attack Test

Puzzle policy를 적용한 경우에 대하여 Puzzle의 난이도에 따라 시간당 Packet의 수, CPU Resource를 계산한 결과는 아래와 같다. 실험방법은 위의 실험과 동일하며,

결과는 그래프로 그려 놓았다. Packet Rate는 1 Client에 대해 측정한 결과이며, CPU Resource는 12 Client를 모두 활용한 경우에 대한 결과이다.

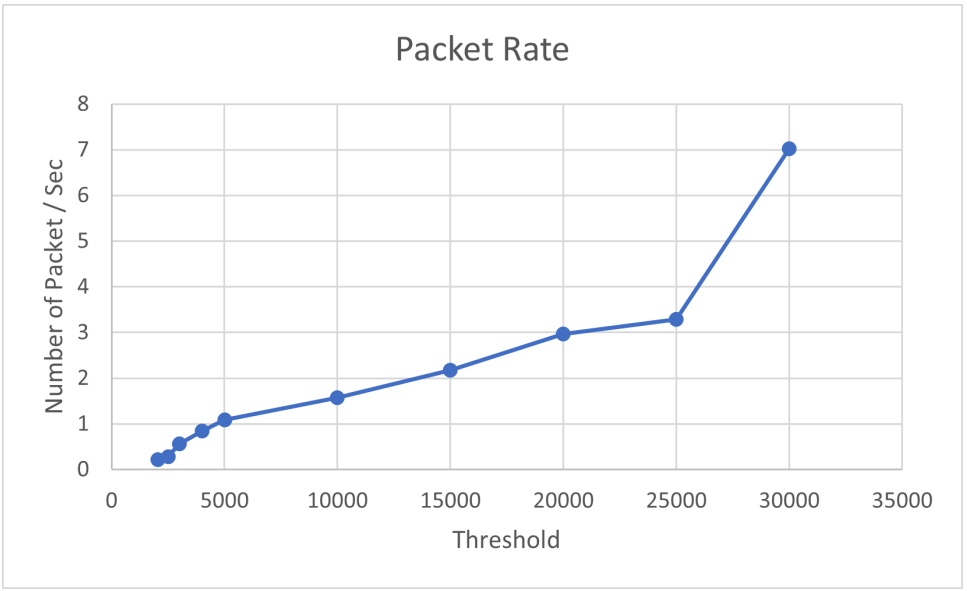

그림 **5. Packet Rate**

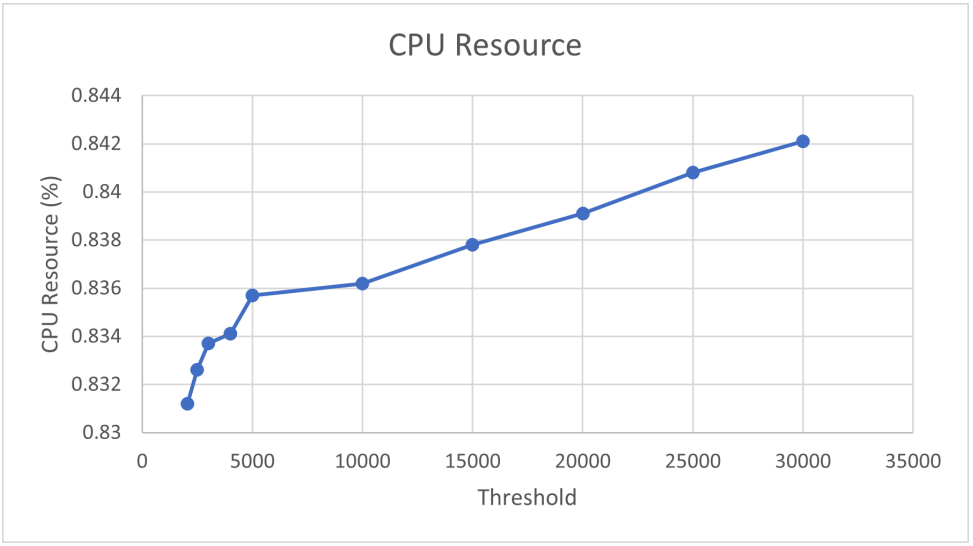

그림 **6. CPU Resource**

4) Puzzle policy를 적용한 DDoS Mitigation System

위에서 구현하고 검증한 Puzzle 기반의 DDoS Mitigation System을 실제 작동에 적용시킨 결과이다. 실제 Traffic을 감지하여 Puzzle policy를 수정하는 System의 결과이다.

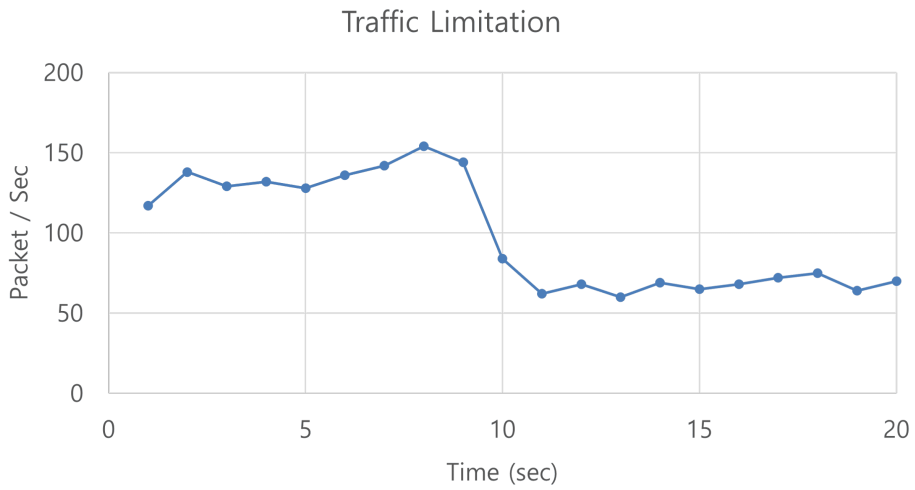

그림 **7. Traffic Limitation**

B. Result Analysis and Discussion

1) Puzzle policy를 적용하지 않은 경우의 DDoS Attack Test

해당 실험은 모든 Client가 최대한 많은 Packet을 전송할 수 있도록 구성한 환경하에서 실험을 진행하였다. 이는 Puzzle policy를 적용한 DDoS Mitigation System의 비교군으로 삼고자 함과 동시에, 구현한 Bot, Host, DNS server가 정상적으로 동작하는지를 확인하고자 하는 목표를 가지고 진행된 실험이다.

결과를 보면, Bot 1개가 초당 4249개의 Packet을 전송하고 있음을 확인할 수 있다. 즉, Client 12개를 모두 활용할 경우 초당 약 51000개의 Packet을 전송하여 Host에 부하를 가할 수 있게 된다. 이 경우 Host의 CPU를 약 6.8퍼 사용하게 된다.

기존의 목표 중 하나는 Bot을 이용하여 Server의 CPU Resource를 모두 사용하여 Server를 다운시키는 것이었다. Raspberry pi의 CPU가 생각보다 좋은 성능을 자랑해서, 최대한으로 Packet을 전송할 경우, 오히려 Client의 CPU가 먼저 최대의 Resource를 사용하게 되어 Client의 CPU가 먼저 다운되는 결과를 관측하였다. 기존의 DDoS attack들이 100만대가 넘어가는 Bot들을 사용하는 것을 생각해보면, 최대 12개의 불과한 자체 Test bed의 Bot들로는 Raspberry pi를 기반으로 한 Host server를 완전히 다운시키는 것은 불가능하다는 사실을 알 수 있었다. 따라서, 이 부분은 추후 연구로 남겨두고, 이번 프로젝트에서는 다운시킨다의 정의를 최대한으로 차지하게 만들 수 있는

CPU resource로 정하고, 이와 비교하여 효율을 측정하였다.

2) Puzzle의 효율성 검증

그래프에서 확인할 수 있는 것처럼 Puzzle이 효과적으로 동작한다는 사실을 알 수 있다. 간단히 말하자면, Threshold를 감소시킬수록 Puzzle을 푸는데 걸리는 시간이 유효한 정도로 증가한다는 사실을 알 수 있다. Threshold를 2048로 설정할 경우, Puzzle을 푸는 시간이 약 5초가 걸리게 된다. 결국, Puzzle을 사용해서 효과적으로 Rate limiting을 할 수 있다는 결과를 얻을 수 있다.

3) Puzzle의 난이도에 따른 DDoS Attack Test

첫 번째로, Threshold를 조절할 경우에 Packet rate가 유효하게 감소하였다는 사실을 알 수 있다. 위의 실험에서 Puzzle policy가 존재하지 않는다면 초당 4249개의 Packet을 보낼 수 있다는 결과를 얻었다. 이번 실험의 경우 Threshold 값을 2048로 정하면 초당 약 0.2개의 Packet을 전송할 수 있게 된다. Puzzle을 사용해서 효과적으로 Packet을 감소시킬 수 있다는 것이다. 즉, Puzzle 기반의 DDoS Mitigation System이 효율적으로 동작한다는 결과를 얻을 수 있다. 특히 2048의 Threshold로 높은 난이도를 사용할 경우, Packet rate를 약 99.95% 감소시킨 0.05%로 줄일 수 있다. 30000의 Threshold를 사용하더라도 Packet rate를 약 0.16%로 줄일 수 있다.

두 번째로, CPU resource를 확인할 경우, 12 Client를 모두 사용하는 경우에 CPU resource가 확연히 감소하였다. 기존에 Puzzle policy를 사용하지 않는 경우에는 약 6.8%의 CPU를 사용하였다. Puzzle policy를 사용한다면, Threshold를 30000으로 설정하는 경우에는 약 0.842%로 기존대비 약 88%를 개선한 12.5%만의 CPU를 사용하고, Threshold 2048의 경우에는 기존대비 약 12.2%만의 CPU를 사용한다. 즉, 기본적으로 Client와의 통신에 사용하는 CPU 소모량을 확실하게 감소시킬 수 있다는 것이다.

마지막으로, 실험의 결과로는 넣지 않았지만, Local DNS가 CPU를 사용하는 비율은 약 0.0%로 너무 작아서 측정할 수 없는 결과가 나타났다. 여기서 확인할 수 있는 것은 기존의 가정처럼 Local DNS가 Puzzle을 전달해주어도 Local DNS 자체의 부하는 얼마되지 않고, Puzzle을 전해줌으로써 Host의 부하는 감소시킬 수 있는 것이다. 즉, DDoS Mitigation System을 위해서 디자인한 Architecture가 효율적이라는 사실을 알 수 있다.

4) Puzzle policy를 적용한 DDoS Mitigation System

위의 그래프를 보면, Host가 받는 Packet 수가 일정수준 이상이 되면, Host가 DDoS attack이 존재함을 감지하고, Puzzle의 난이도를 증가시킨다. 위 그래프의 경우 Threshold가 30000이던 상황에서 20000로 증가시킨 것이다. 이렇게 증가시킬 경우, 이후의 Packet rate가 확실하게 감소하는 경향성을 관측할 수 있다. 이를 실제 상황에 빗대어 생각해볼 경우, DDoS attack을 감지하면 Host가 자율적으로 Packet rate를 조절하여 자체적으로 Server가 다운되는 것을 방지하고, 안정성 있는 운영을 할 수 있다는 사실을 알 수 있다. 즉, 상황에 따라 Threshold를 조절함으로써 User friendly하고, 효율적인 DDoS Mitigation System을 구축할 수 있다

10. Division & Assignment of Work

항목	담당자
Local DNS 구현	양석훈
Target Server 구현	양석훈
DDoS Attack Bot 구현	선민준
Puzzle 제작	전진용
TCP Packet 구현	선민준
테스트 환경 구축	선민준, 양석훈
데이터 및 결과 분석	선민준, 양석훈

11. Conclusion

1) Conclusion

이번 프로젝트의 목표는 DDoS Mitigation System을 기존에 이론적으로만 존재하던 Puzzle기반의 System을 실제 상황에 적용할 수 있도록 Design하고, 실질적으로 구현해내는 것이었다. OS간의 호환성, 기존 통신 시스템의 유지, 개별적인 정책 적용 등을 위하여 Linux kernel 단에서의 수정을 통해 DDoS Mitigation System을 구현하였다. 구현한 DDoS Mitigation System의 성능을 확인해본 결과, 초당 들어오는 Packet의 수를 0.1%이하로 제한할 수 있고, CPU의 사용량은 약 10%로 감소시킴으로써 효율적으로 DDoS 공격을 방어해낼 수 있다. 특히, 초기의 목표였던 10배의 성능을 개선시키는 것에 성공하였으며, Puzzle을 사용하지 않는 것과 비교하였을 때 40%를 훨씬 넘기는 성능을 확인하였다. 결국, 처음에 세웠던 프로젝트의 목표를

대부분 달성해내는 것을 성공하였다.

지금까지 이론적으로만 연구되던 Puzzle 기반의 DDoS defense system을 이번 프로젝트를 통해 실질적으로 구현해보았다. 이 프로젝트에서 성공적으로 DDoS 공격을 방어해낼 수 있다는 사실을 확인하였기에 앞으로 이 System을 현실에 적용해서 DDoS 공격을 방어해내는 System을 구축할 수 있다는 제안을 할 수 있다. 이 점에서 이번 프로젝트의 의의를 볼 수 있다.

2) Limitation & Future Work

앞에서 말했듯이 이번 프로젝트의 가장 큰 한계는 사용가능한 자원의 한계이다. 이번 실험에서 Bot의 한계로 인해 더욱 더 큰 사이즈에서의 정확한 측정을 하지 못하였는데 추후에 더 많은 자원을 확보하여 실험을 다시 진행한다면 더욱 정확한 실험을 진행할 수 있고, 효율을 파악하기 더 쉬워질 것으로 예상된다. 특히, Bot을 증가시킨다면 실질적으로 Server를 다운시키고, 이를 방지하는 상황의 Demo를 확실하게 보여줄 수 있을 것이라 기대된다.

추가적으로, 현재는 Threshold를 사용하여 Rate limiting을 하는 것이 핵심이었기에 최적화 방안은 생각해보지 않았다. 따라서, 추후 연구를 진행함에 있어서 Bot의 수 및 Packet의 수 등에 따른 상황에서 어떤 Threshold의 Puzzle을 사용하는 것이 좋을지 User, Host, Client 등이 모두 존재하는 상황에서의 게임이론 적인 분석을 통해 효율을 최적화해보는 연구를 해보아도 좋을 것 같다.

마지막으로, 이번 실험에서는 Local DNS 만이 존재하는 상황을 상정하고 진행하였다. 그러나, 실제는 Open DNS와 같은 더 다양한 주체들이 존재하기에 이들이 모두 존재하는 상황에 대한 연구를 진행한다면, 실제 현실에 적용하기 더욱 적합한 상황에 대해 실험하고 결과를 얻을 수 있을 것이라고 기대된다.