Pusan National University
Computer Science and Engineering
Technical Report 2024-10

클라우드 컴퓨팅 공격 및 방어 플랫폼 개발

저자 201513137 이강빈

저자 202155599 장진영

저자 202055505 강수민

지도교수 김태운

목 차

1. 서론

1.1. 연구 배경

팬데믹 이후 클라우드 서비스의 사용이 급격하게 증가하였고[1], 플랫폼 기업들은 온프레미스[1] 환경에서 클라우드 환경으로의 마이그레이션을 확대했다[2]. 이에 따라 많은 사람들이 클라우드에 관심을 갖게 되었고, 해커들 또한 이러한 환경에 주목하게 되었다[3]. 그 결과, 과거와는 다른 클라우드 환경에 특화된 새로운 공격들이 등장하게 되었다. 이러한 보안 위협이 본 연구의 중요한 배경이 되었다.

그 중에서도 EDoS(Economic Denial of Sustainability)는 클라우드 서비스 환경에서 점점 더 심각한 위협으로 떠오르고 있다. 이는 클라우드 자원의 자동 확장(Auto Scaling) 기능을 악용하여 서비스 제공자에게 재정적 피해를 입히는 새로운 형태의 공격이다. EDoS 에는 여러 유형이 있는데, 그 중 YoYo Attack 은 HTTP Request 를 이용한 공격이다. 해당 공격 방식은 주로 웹 서버가 클라이언트 요청을 처리하는 방식을 악용하여 서버 자원을 과도하게 소비하게 만든다. 공격자는 HTTP 요청을 지속적으로 보내고, 서버가 해당 요청을 처리하기 위해 지속적으로 자원을 사용하게 한다. [4][5]

클라우드 서비스는 사용률 기반으로 과금 되기 때문에, 공격자는 합법적인 트래픽처럼 보이는 가짜 요청을 지속적으로 발생시켜 피해 기업의 자원이 과도하게 사용되도록 유도하고, 결과적으로 과도한 비용을 발생시킨다.[6] 이는 기존에 존재하던 방어 기법들로 막을 수 없는 공격 방식이며, 클라우드 기반의 서비스를 이용하는 환경에서 발생 할 수 있는 새로운 취약점이라 볼 수 있다.

우리는 IT 기업들의 다양한 서비스들이 온프레미스 환경에서 클라우드 서비스로 전환하는 시점에서, 클라우드 서비스의 취약점 중 하나로 대두되고 있는 YoYo Attack 에 대해 연구를 진행하고자 한다. 본 연구에서는 로컬 환경에서 Auto Scaling 을 유도하는 실제 공격 방식을 분석하고, 방어자가 이러한 공격을 어떻게 감지할 수 있는지, 효과적인 방어 방법은 무엇인지에 대해 탐구하고자 한다.

[1] On-premise : 기업의 서버를 자체적으로 보유한 전산실 서버에 직접 설치에 운영하는 방식

1.2. 기존 DDoS 방어 매커니즘의 한계

DDoS 공격은 네트워크나 서버의 트래픽 처리 용량을 초과 시켜 서비스를 마비시키는 것을 목표로 하며, 여러 대의 컴퓨터를 동원해 짧은 시간 내에 대규모 트래픽을 발생시키는 것이 특징이다. 반면, EDoS는 클라우드 서비스나 종량제 시스템에서 리소스 사용률을 과도하게 늘려 경제적 손실을 일으키는 것이 주된 목표이다. 따라서 대규모 트래픽이 아닌, 지속적으로 트래픽을 발생시켜 경제적 손실을 유도하는 것이 특징이다. 이로 인해 EDoS는 장기간에 걸친 저강도 공격이 될 수 있으며, 이러한 공격은 정상적인 트래픽과 구분하기 어려워 기존 DDoS 방어 방식이 효과적이지 않을 수 있다.

DDoS의 방어 기법으로는 대표적으로 SYN Cookies와 Rate Limiting이 있다. SYN Cookies는 서버가 TCP 연결 요청을 받을 때 TCP 상태 정보를 메모리에 저장하지 않고, SYN-ACK 패킷의 시퀀스 번호에 연결 정보를 암호화해 포함시키는 방식이다. 이 방법은 소프트웨어적으로 쉽게 적용할 수 있지만, SYN Flooding 공격에만 효과적이라는 한계가 있다. 또한, Rate Limiting은 서버나 네트워크 장비에서 특정 IP 주소나 전체 트래픽에 대한 요청 빈도를 제한하는 방식이다. 추가적인 하드웨어나 복잡한 시스템 없이도 구현이 가능하지만, EDoS 공격은 정상 사용자와 유사한 트래픽을 발생시켜 서버 자원을 소모 시키기 때문에 임계 값을 잘못 설정하면 정상 사용자까지 차단되는 문제가 있을 수 있다.

결론적으로, 기존의 DDoS 방어 방식만으로는 정상 트래픽과 유사한 EDoS 공격을 방어하는 데 한계가 있다.

1.3. 연구 목표

본 연구는 클라우드 환경에서의 자원 자동 확장 기능을 악용한 YoYo Attack 공격을 감지하고, 이를 방어할 수 있는 실시간 알고리즘을 개발하는 것을 목표로 한다.

클라우드 서비스에 대한 공격을 실시간으로 모니터링하고, 공격자의 공격 성공 여부에 대한 판단에 혼란을 야기하고, 서버의 자원 관리와 비용 효율성을 동시에 보장할 수 있는 방어 매커니즘을 개발하는 것을 목표로 한다.

이 연구는 YoYo Attack의 심각성이 증대되고 있는 상황에서, 기업이 직면할 수 있는 재정적 위협을 줄이기 위한 중요한 기여를 할 수 있을 것이다.

2. 배경 지식

공격자는 YoYo Attack이 성공적으로 진행되고 있는지 판단하기 위한 지표를 가지고 있으며, 서버(방어자)는 비정상 트래픽을 감지하고, YoYo Attack을 효과적으로 방어하기 위한 지표를 가지고 있다. 2장에서는 YoYo Attack과 연구의 이해를 위해 필요한 단어를 정의하고 해당 지표에 대해서 알아본다.

2.1. 용어 정리

- Auto Scaling: 클라우드 컴퓨팅에서 서버 자원을 자동으로 조정하는 기능

  Scale Out: 시스템 부하가 증가했을 때 서버 인스턴스의 개수를 늘리는 과정

  Scale In: 시스템 부하가 줄어들었을 때 필요 없는 서버 인스턴스의 개수를 줄여 자원을 회수하는 과정

- YoYo Attack: Auto Scaling 을 활용하여 자원을 확장(Scale Out)하고 축소(Scale In)하는 사이클을 반복하게 만들어 서버에 경제적 피해를 입히는 공격 방식
- 더미 서버: 방어 매커니즘을 위해 정상 서버와 동일한 핵심 기능을 가지면서 Auto Scaling 을 지원하지 않고 응답 시간을 교란하는 서버

2.2. YoYo Attack 방식 정의

YoYo Attack 은 Auto Scaling 과 관련이 있고, Auto Scaling 은 CPU 사용률과 연관이 있다. 따라서 YoYo Attack 방식은 CPU 사용률이 높은 비지니스 로직을 실행하는 Application Layer Attack 으로 한정한다.

HTTP GET Flood

- 공격자가 타겟 Web Application 에 다량의 HTTP GET 요청을 전송하는 공격이다.
- Web Application 서버는 다량의 GET 요청을 처리하는 과정에서 CPU, Memory 등의 자원을 소모하게 되고, 과부하가 걸리게 된다.

HTTP POST Flood

- 공격자가 타겟 Web Application 에 다량의 HTTP POST 요청을 전송하는 공격 방식이다.

- Web Application 서버는 다량의 POST 요청을 처리하는 과정에서 CPU, Memory 등의 자원을 소모하게 되고 과부하가 걸리게 된다.
- GET 요청을 처리하는 과정보다 POST 요청을 처리하는 과정이 복잡할 수 있으며, 효율적인 공격 방식이 된다.

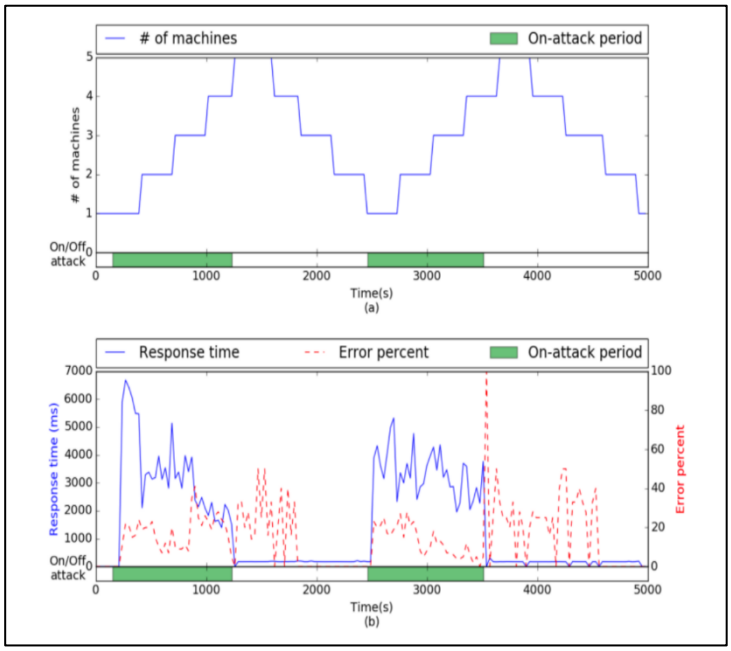
2.3. 공격자 주요 지표: 응답시간

공격자의 주요 지표는 HTTP 요청에 대한 응답시간이다.

- 공격자 측에서 공격 성공 여부 및 대상 시스템의 상태 유추에 사용 할 수 있는 지표로 응답시간의 증감을 활용한다.

공격자 입장에서의 응답시간 변화는 다음과 같이 해석된다.

- Scale Out이 일어나면 Pod 수가 증가하며, 응답 시간이 감소한다.
- Scale Out이 최대로 일어나면, 응답 시간 변화 없거나 증가한다.



[그림 1] YoYo Attack on system with discrete scale policy [7]

[그림 1]의 상단 그래프는 YoYo Attack 공격 여부에 따른 Pod 수의 변화를 나타낸다. 공격이 활성화 된 시간동안 Scale Out 이 발생하여 Pod 수가 증가하고, 공격이 멈춘 시간동안 Scale In 이 일어나 Pod 수가 감소한다.

[그림 1]의 하단 그래프는 YoYo Attack 공격 여부에 따른 응답 시간의 변화를 나타낸다.

상단 그래프와 하단 그래프를 통해 Pod 수가 증가하면 응답 시간이 감소하는 경향성을 알 수 있다. 따라서 응답 시간의 증감을 이용하여 공격자는 공격의 성공 여부를 알 수 있다.

2.4. 서버(방어자) 주요 지표: 요청 횟수, CPU 사용률, 응답 시간

서버(방어자) 측에서 비정상 트래픽 관측을 위해 사용할 수 있는 지표

요청 횟수

갑작스럽게 다량의 요청이 발생하면 시스템에 비정상적인 트래픽이 유입되고 있을 가능성이 높다.

요청 횟수를 이용해 특정 시간 동안 비정상적으로 많은 트래픽을 감지할 수 있다.

CPU 사용률

갑작스러운 다량의 요청으로 인해 CPU 사용률이 급격히 상승할 수 있다.

CPU 사용률을 이용해 비정상적인 트래픽이 유입되고 있음을 빠르게 감지할 수 있다.

요청 횟수와 CPU 사용률

CPU를 많이 사용하는 특정 API에 갑작스럽게 많은 요청이 들어오는 경우, 비정상적인 트래픽 유입을 감지할 수 있다.

⇒ 요청 횟수와 CPU 사용률을 종합적으로 분석하여 비정상 트래픽을 탐지할 수 있다.

서버(방어자)가 비정상 트래픽 방어를 위해 사용할 수 있는 지표

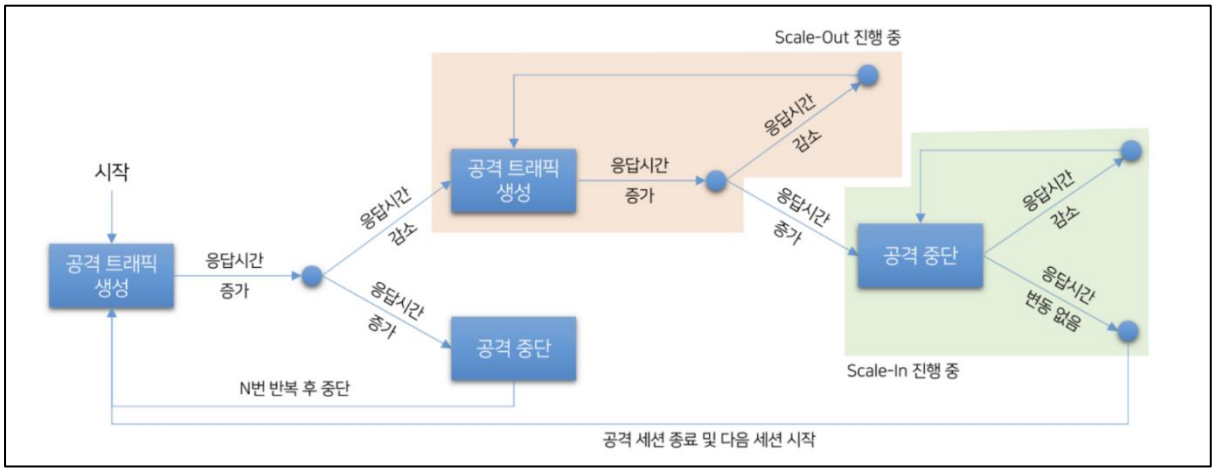
공격자 응답에 대한 응답 시간 교란

서버는 공격 성공 여부를 혼란시키기 위해 의도적으로 응답 시간을 지연시키거나 변동시킬 수 있다. 이를 통해 공격자가 공격 성공 여부를 명확히 알 수 없게 된다.

3. 제안하는 공격 및 방어 매커니즘 개요

3장에서는 실험 이해를 위한 개념과 이론적 배경을 다룬다. 공격자가 사용하는 공격 매커니즘과 이에 대응하는 서버(방어자)의 방어 방법을 설명한다. 이후, 서버가 공격자 의심 IP를 탐지하는 매커니즘을 설명한다.

3.1. 공격자 공격 매커니즘



[그림 2] 공격자 입장에서 공격 매커니즘 개요

[그림 2] 설명

1. 공격 트래픽 생성
   HTTP 요청을 N회 비동기로 전송

2. 응답 시간 기록
   N회 요청에 대한 평균 응답 시간을 기록

3. 응답 시간 증가, 감소 측정

   1, 2번 과정을 M회 반복하여 진행

   M번 반복한 요청의 평균 응답 시간 기록

   이전 평균 응답 시간과 비교

   - 응답 시간 감소: Scale Out 진행 중이라 판단하여 공격 지속
   - 응답 시간 동일 or 증가: Scale Out 이 최대로 일어났다고 판단 후 Scale In 유도하기 위해 잠시 공격 중단
   - 일정 시간 이후 공격을 다시 했음에도, 응답 시간이 동일하거나 증가하는 것이 반복되면 Auto Scaling을 지원하지 않는 서버로 판단 공격 중단

3.2. 서버 방어 매커니즘

3.2.1. Sigmoid 함수를 활용한 동적 방어 매커니즘

단위 시간 동안의 요청 횟수를 통해 방어 매커니즘을 동적으로 적용한다.

Sigmoid 함수를 활용하여, 요청을 적게 보내는 정상 사용자는 방어 매커니즘에 영향을 적게 받고, 요청을 많이 보내는 비정상 사용자는 방어 매커니즘의 영향을 많이 받도록 한다.

3.2.2. 특정 IP SLEEP

단위 시간동안 특정 IP의 요청 횟수에 따라 해당 IP를 처리하는 Thread에 Sleep을 동적으로 적용해 응답 시간을 교란하는 방법이다.

알고리즘 설명

- 서버에 들어오는 각 사용자의 단위 시간당 요청 횟수를 기록한다.
- 요청 횟수에 따라 Thread Sleep 시간을 동적으로 적용한다.
- 애플리케이션 미들웨어 레벨에서 Thread마다 개별적으로 Thread Sleep을 적용한다.
- 횟수가 많아질수록 Thread Sleep 시간을 길게 적용하여 과도한 요청을 억제하고, 시스템 자원 사용을 줄인다.

기대 효과

- 요청 횟수가 높은 IP에 대해서 더 긴 시간의 Thread Sleep을 적용해 공격자가 Scale Out이 끝났다고 판단하게 하여, 공격 중단을 유도 한다.
- 정상 사용자의 요청은 바로 처리되어 서비스 품질 저하를 막을 수 있다.

3.2.3. VM 방화벽 설정을 통한 패킷 드랍

단위 시간 동안 특정 IP의 요청 횟수에 비례하는 확률로 해당 IP를 서버가 실행되고 있는 VM에 전달하고, 방화벽 설정을 통해 일정 시간 해당 IP의 패킷을 드랍하여 응답 시간을 교란하는 방법이다.

알고리즘 설명

- 서버에 들어오는 각 사용자의 단위 시간당 요청 횟수를 기록한다.
- 요청 횟수가 많아질수록 해당 사용자가 방어 매커니즘에 영향을 많이 받게 된다.
- 요청 횟수에 비례하는 확률로 IP를 VM에 전달하여 방화벽 설정을 적용한다.
- VM이 IP를 전달받으면 방화벽 규칙을 변경하고, 해당 IP의 네트워크 패킷을 일정 시간 동안 드랍한다. (Linux iptables 활용)

기대 효과

- 요청 횟수가 많은 비정상 사용자는 점차 높은 확률로 네트워크 패킷이 차단되므로 공격 효과가 줄어든다.
- 정상 사용자는 요청 횟수가 적으므로 방화벽 적용을 거의 받지 않아 서비스가 원활하게 유지된다.
- VM의 방화벽을 통해 직접 패킷 드랍을 처리하므로, 서버로 불필요한 요청이 전달되지 않으며 서버 자원 소모를 최소화할 수 있다.
- 공격자의 네트워크 패킷을 드랍함으로써 공격자의 응답 시간을 지연시켜 응답 시간 교란을 할 수 있다.

3.2.4. 더미 서버 기반의 확률적 트래픽 리다이렉트

단위 시간 동안 특정 IP의 요청 횟수에 비례하는 확률로 해당 IP의 요청을 더미 서버로 리다이렉트하여 응답 시간을 교란하는 방법이다.

알고리즘 설명

- 서버에 들어오는 각 사용자의 단위 시간 당 요청 횟수를 기록한다.
- 요청 횟수가 많아질수록 해당 사용자가 방어 매커니즘에 영향을 많이 받게 된다.
- 요청 횟수에 비례하는 확률로 IP의 요청을 더미 서버로 리다이렉트 한다.
- 더미 서버로 들어오는 요청을 고의로 일정 시간 느리게 응답한다. (Linux 의 TC 를 활용한다)

기대 효과

- 정상 사용자는 서버에 요청을 많이 보내지 않으므로 더미 서버로 갈 확률이 낮다.
- 만약 정상 사용자가 더미 서버로 가더라 하더라도 새로고침 등을 활용한 재 요청 시 정상 서버로 갈 확률이 매우 높다.
- 공격자는 정상 사용자에 비해 많은 요청을 보내므로 더미 서버에 갈 확률이 높아진다.
- 정상 사용자의 피해를 적게 하면서, 공격자의 응답 시간 교란 할 수 있다.

3.2.5. OpenWRT를 활용한 중개방어

단위 시간 동안 특정 IP의 요청 횟수가 특정 조건이 되었을 때 해당 IP를 OpenWRT가 설치된 공유기에서 방화벽 설정을 활용하여 해당 IP 요청에 대한 응답을 의도적으로 지연시키는 방법이다.

알고리즘 설명

- 서버에 들어오는 각 사용자의 단위 시간 당 요청 횟수를 기록한다.
- 요청 횟수가 많아질수록 해당 사용자 요청에 대한 지연 시간이 증가한다.
- 특정 요청 횟수 특정 조건이 되면 공유기에서 방화벽 설정을 활용하여 응답 시간을 지연한다. (Linux TC 명령어 활용)

기대 효과

- 정상적인 사용자는 서버에 대량의 요청을 보내지 않음으로 트래픽 제어 규칙에 따라 적용될 가능성이 낮다.
- 서버의 앞 단에 있는 공유기 자체에서 패킷을 지연시키므로 서버와 방어 매커니즘을 분리 할 수 있어 효율적이다.

3.3. 서버(방어자)의 공격자 의심 IP 색출 기법

3.3.1. 서버(방어자) 주의점

서버에 사용자가 공격자인지 정상 사용자인지 정확하게 구분할 수 없다.

공격자로 의심이 되는 IP를 색출하면서도, 정상 사용자일 수 있는 가능성을 절대 배제하면 안된다.

3.3.2. 공격자 의심 IP 색출

공격자가 IP 변경 하지 않을 때 or 단일 공격자 일 때

- 서버에 요청을 보내는 사용자의 요청 횟수 로그 기록을 활용한다.
- 로그 데이터를 바탕으로 머신러닝 알고리즘 이상 탐지(Outlier Detection)를 활용해 정상 사용자와 비정상 사용자를 구분한다. 비정상 사용자로 구분된 IP는 방어시스템 적용 대상이 된다.
- 한계 :
  - 공격자가 IP를 변경하면 이전에 비정상으로 탐지된 기록이 무의미해진다.
  - 다수의 PC를 이용한 공격은 공격 횟수가 분산되므로 탐지하기 어렵다.

공격자 IP 변경이 예상 될 때 or 공격자가 다수의 PC로 공격할 때

서버 자신만 아는 정보를 활용:

- Pod 또는 Node의 Scale Out을 1회 일으킬 수 있는 요청 횟수 : IP를 교체하거나 공격자가 여러 PC를 사용하더라도, Scale Out을 최대한으로 유도하려면 Pod 하나에 대해 Scale Out을 일으킬 요청 횟수보다 더 많은 요청을 보내야 한다. 따라서 Scale Out을 1회 일으킬 수 있는 요청 횟수를 기준으로 설정한다.
- Scale Out 및 Scale In 시간 활용: 기준 시간에 과한 요청을 보내는 IP를 탐지해야 한다. 따라서 Scale Out이 최대로 발생하고 이후 Scale In이 이루어지는 시간을 기준으로 설정한다. 해당 시간 내에 과도한 요청이 발생할 경우 이를 공격자로 의심한다.

결론

이상 탐지 기법과 서버 자신만 아는 정보를 함께 사용하면, 공격자의 수에 관계없이 모든 경우에 대해 대처할 수 있다.

4. 구현 상세 및 검증

4장에서는 실험에서 사용된 알고리즘 구현 설명과 알고리즘 검증 결과를 제시한다. 사전 정의를 통해 실험에서 활용된 주요 변수와 기준을 정의한다. 이후 공격자와 방어자의 알고리즘 구현을 설명한다. 그리고 해당 알고리즘에 대한 실험 및 검증 결과를 제시하여 각 방어 매커니즘의 성능을 분석하고 방어 매커니즘에 의한 정상 사용자의 피해 정도를 알아본다.

4.1. 응답시간 기준 설정

구체적인 알고리즘 구현에 앞서 필요한 정보를 먼저 정의하고자 한다.

4.1.1. CPU 사용률이 가장 높은 API 평균 응답 시간 기준 설정

Google은 200ms 이하의 응답시간을 유지하는 게 이상적이라고 한다. [8]
실험에 사용될 서버의 성능은 구글 서버 성능보다 좋지 않음을 고려해 CPU 사용률이 가장 높은 API의 평균 응답 시간을 300ms으로 설정하고 로직을 작성한다.

4.1.2. 비 정상적 응답 시간 및 서버 Spec 기준 설정

애플리케이션 성능 지표인 **APDEX(Application Performance Index)** 사용 하여 사용자 만족도에 대한 지표로 활용한다.[9]

- 임계값 0.3초로 설정

  이는 서버 측에서 임의로 설정하는 값이다. CPU 사용률이 가장 높은 API의 평균 응답 시간이 300ms이므로 0.3초로 설정한다.

- 만족 (Satisfied,S): 업무처리에 전혀 문제 없음 ≤ 0.3초 (만족 S 기본값)
- 허용 (Tolerating,T): 사용자가 지연을 느끼나 업무처리는 가능 ≤ 1.2초 (만족 S * 4)
- 불만 (Frustrated,F): 업무처리가 불가능 > 1.2초 (허용 T 초과 및 오류)

서버 측 성능

현재 직접 실험 해 볼 수 있는 AWS Instance 가 t2.micro와 t3.medium 정도의 성능이다. 따라서 서버의 Spec 을 t3.medium 정도의 성능으로 구축한다. [10] [11] [12]

공격자 입장 서버 측 Spec 기준

공격자는 서버 측의 Spec을 알 수 없다. 하지만 실험을 진행하는데 있어 실험 도구의 성능에 제한이 있다. 따라서 공격자 입장에서는 서버 측 Spec을 AWS Instance 중 t3에 해당한다고 가정했다.

4.2. 공격자 알고리즘

4.2.1. 공격 트래픽 생성 및 응답 시간 기록

HTTP 요청을 8회 비동기로 전송

8회인 이유

- 비동기 요청을 처리하기 위해서는 서버 측 Spec이 중요하다.
- t3 인스턴스 기준 제일 많은 vCPU 는 8개이다.
- 공격자는 서버측 인스턴스의 정보를 알지 못하기 때문에 vCPU 개수가 가장 많은 8개를 기준으로 공격한다.
- 다수의 공격자 일 때는 8개의 요청을 나눠서 전송한다. 예를 들어 공격 PC가 3대 일 때 각각 3개의 요청을 전송한다.

HTTP 요청을 보내고 응답이 오는 시간 차이를 계산해 요청에 대한 응답 시간을 기록

응답 시간 기록 시 중요한 점 [13]

새로 생긴 Pod 에서 서버 실행이 다 되지 않았을 때 요청을 처리하지 못하고 HTTP 400, 500번대 상태 코드를 반환 한다. 그러므로 HTTP 응답 코드를 기준으로 200번 또는 300번대 코드에 대한 응답 시간만 기록한다.

공격 트래픽 생성 & 응답 시간 기록 슈도 코드

Algorithm 1 URL에 대해 N회 비동기 HTTP 요청 처리 및 응답 시간 기록

입력: URL url
출력: 응답 시간 리스트 $responseTimes$
$responseTimes \leftarrow []$ {빈 리스트 초기화}
for $i = 1$ **to** N **do**
  $startTime \leftarrow current\_time()$ {현재 시간 기록}
  $response \leftarrow async\_http\_request(url)$ {비동기 HTTP 요청}
  $endTime \leftarrow current\_time()$ {응답 후 시간 기록}
  if $response.status \in [200, 300]$ **then**
    $responseTime \leftarrow endTime - startTime$ {응답 시간 계산}
  else
    $responseTime \leftarrow NULL$ {상태 코드가 200 또는 300이 아닌 경우}
  end if
  $responseTimes.append(responseTime)$ {결과를 리스트에 저장}
end for
return $responseTimes$ {리스트 반환}

4.2.2. 평균 응답 시간 기록

공격자는 서버에 8회의 비동기 요청을 전송하고 8회 요청이 전부 종료될 때까지 기다린다.

모든 요청에 대한 응답을 받게 되면 8회에 대한 평균 응답 시간을 측정한다.

8회 평균 응답 시간 측정 시 중요한 점 [13]

- 새로운 Pod에서 서버가 완전히 실행되지 않아 HTTP상태 코드가 200번, 300번대가 아니라서 응답 시간이 정확히 기록 되지 않는 상황이 발생할 수 있다.
- 새로운 Pod에서 실행된 서버가 안정화가 되지 않아서 응답 시간이 매우 높게 나오는 상황이 발생할 수 있다.
- 위 두 상황에서는 보정을 위해 적절한 값을 응답 시간에 더해주어야 하는데, 공격 전 8회 비동기 요청 응답시간의 평균 값 1600ms를 더해준다.

슈도 코드

Algorithm 2 N회 비동기 HTTP 요청의 평균 응답 시간 계산

입력: N회 비동기 응답시간 리스트 $responseTimes$
출력: N회 비동기 응답시간의 평균 $averageTime$
for $time$ **in** $responseTimes$ **do**
  if $time = NULL$ **then**
    $totalTime \leftarrow totalTime + 1600$ {NULL 값을 1600ms으로 대체}
  else
    $totalTime \leftarrow totalTime + time$
  end if
end for
$averageTime \leftarrow totalTime/N$ {평균 응답 시간 계산}
return $averageTime$ {평균값 반환}

4.2.3. 응답 시간 증가, 감소 측정

8회 비동기 요청을 20회 반복

t3 인스턴스 기준 CPU 부하가 높은 API에 160회 정도 요청을 보내면 CPU 사용률을 충분히 높일 수 있다.

20회 비동기 요청에 대한 평균 응답 시간 비교

애플리케이션에 따라 크게 달라질 수 있지만, 평균적으로 Pod 1개가 늘어났을 때 성능이 10%~30% 정도 개선된다. Pod가 1개일 때 평균 응답시간은 1600ms이므로 성능이 10% 개선되면 응답시간이 150ms 정도 줄어든다고 판단했다. 따라서 이전 20회 비동기 요청의 평균 시간과 현재 20회 비동기 요청의 평균 응답 시간을 비교해서 150ms 이상 차이가 난다면 Scale Out 이 일어난다고 판단했다.

4.2.4. 공격 지속, 중단

연속해서 응답 시간의 감소가 측정되지 않으면 Scale Out이 최대로 일어났다고 판단한다.

Scale In을 유도하기 위해 CPU Sleep을 한다.

- Sleep Time은 Pod 1개당 1분으로 설정한다.
- 1분으로 설정한 이유는 AWS에서 제공하는 Auto Scaling기준 default Scale In시간은 5분이지만 AWS에서는 해당 시간이 너무 길기 때문에 적절하게 조절이 필요하다고 했다. 따라서 1분으로 설정했다. [14]
- 응답 시간이 감소된 횟수 x 1분 만큼 공격을 중단하고 Scale In 유도한다.

응답시간의 감소가 없고 계속 증가만 한다면 Auto Scaling이 지원하지 않는 서버로 판단하고 공격을 중단한다.

응답 시간 증가, 감소 측정, 공격 지속, 중단 코드

Algorithm 3 평균 응답 시간 분석 및 조건에 따른 대기 처리

```
trueStop ← 0 {trueStop (공격 완전 종료 변수) 초기화}
temporaryStop ← 0 {temporaryStop (공격 임시 종료 변수) 초기화}
decreaseSection ← 0 {decreaseSection (응답 시간 감소 부분 체크 변수) 초기화}
responseTimes ← [] {응답 시간을 기록할 리스트 초기화}
averageResponseTimes ← [] {평균 응답 시간을 기록할 리스트 초기화}
while trueStop < 2 do
  for i = 1 to REPEAT_NUMBER do
    responseTime ← get_average_response() {2번 알고리즘으로 부터 평균 응답 시간 가져오기}
    responseTimes.append(responseTime) {응답 시간 리스트에 추가}
  end for
  sumResponse ← ∑ responseTimes {응답 시간 합산}
  average ← sumResponse/REPEAT_NUMBER {응답 시간의 평균 계산}
  averageResponseTimes.append(average) {평균 응답 시간을 리스트에 추가}
  if len(averageResponseTimes) ≥ 2 then
    if averageResponseTimes[−1] − averageResponseTimes[−2] ≥ 150 then
      temporaryStop ← temporaryStop + 1 {응답 시간이 감소하지 않았을 경우}
    else
      decreaseSection ← decreaseSection+1 {응답 시간이 감소했을 경우}
    end if
  end if
  if temporaryStop ≥ 2 then
    responseTimes.clear() {responseTimes 리스트 초기화}
    averageResponseTimes.clear() {averageResponseTimes 리스트 초기화}
    temporaryStop ← 0 {temporaryStop 초기화}
    if decreaseSection = 0 then
      trueStop ← trueStop+1 {decreaseSection이 0일 경우 trueStop 증가}
    end if
    wait(decreaseSection × 1000) {decreaseSection에 따른 대기}
    decreaseSection ← 0 {decreaseSection 초기화}
  end if
end while
```

4.3. 방어자 알고리즘

4.3.1. 비정상 IP 색출 및 Sigmoid 함수 적용

비정상 사용자 요청 횟수 결정

- 실험에 사용한 서버와 애플리케이션의 성능상 Pod 1개를 늘리기 위해 필요한 연속된 요청의 횟수는 240회이다. 따라서 240회 이상의 요청을 보냈을 때 Sigmoid 함수가 0.5의 값을 가지도록 설정했다.
- 이때 서버 사용자에 대한 과거 요청 횟수에 대한 로그 기록이 있다면 같이 사용하는 것이 좋다. 예를 들어 이상 탐지를 활용해 본 결과 CPU 사용률이 높은 API에 100회 이상의 요청이 들어올 때 이상 요청으로 탐지하면, 100회에서 Sigmoid 함수가 0.5의 값을 가지고 240회에서 Sigmoid 함수가 1의 값을 가지게 하면 훨씬 효율적이다.

제안하는 Sigmoid 함수

애플케이션의 로그기록이 없기 때문에 Pod 1개를 늘리기 위한 요청 횟수인 240회에 0.5의 값을 가지게 했다. 300회에 1의 가까운 값을 가지도록 함수의 파라미터를 설정했다.

슈도 코드

Algorithm 4 Sigmoid 함수

입력: x
출력: Sigmoid 함수의 결과값
$X0 \leftarrow 240$
$K \leftarrow 0.05$
함수 $sigmoid(x)$:
    return $\frac{1}{1+\exp(-K\times(x-X0))}$

4.3.2. 특정 IP SLEEP

알고리즘 설명

- 단위 시간 내 특정 IP가 보낸 요청 횟수를 조회한다.
- 횟수를 입력으로 sigmoid 함수를 호출하여 0과 1사이의 값을 얻고 1000을 곱해 ms 단위로 변경한다.
- 계산된 지연 시간만큼 서비스 로직을 수행하기 전 애플리케이션 미들웨어에서 Thread Sleep을 활용하여 응답시간을 교란한다.

슈도 코드

Algorithm 5 IP 요청 횟수를 기반으로 Sigmoid 함수 적용 및 스레드 대기

1: **입력:** 단위 시간 동안의 IP 요청 횟수 $requestCount$
2: $delayTime \leftarrow sigmoid(requestCount) \times 1000$ {4번 알고리즘 Sigmoid 결과에 1000을 곱한 값}
3: **스레드** $sleep(delayTime)$ {스레드를 $delayTime$ ms만큼 대기}

4.3.3. VM 방화벽 설정을 통한 패킷 드랍

알고리즘 설명

- 단위 시간 내 특정 IP가 보낸 요청 횟수를 조회한다.
- 요청 횟수를 입력으로 sigmoid 함수를 호출하여 0과 1사이의 값을 얻는다.
- Sigmoid 함수를 통해 얻은 값을 활용하여 서비스 로직을 수행하기 전 애플리케이션 미들웨어에서 VM에 확률적으로 IP를 전달한다. HTTP POST 요청을 활용하여 IP를 JSON 형식으로 전달한다.

서버 슈도 코드

Algorithm 6 IP 요청 횟수를 기반으로 Sigmoid 함수 적용 및 IP 확률적으로 VM에 전달

입력: 단위 시간 동안의 IP 요청 횟수 $requestCount$
$probability \leftarrow sigmoid(requestCount)$ {4번 알고리즘 Sigmoid 결과를 확률로 저장}
$randomValue \leftarrow random(0, 1)$ {0과 1 사이의 임의의 값 생성}
if $randomValue \leq probability$ **then**
    전달: 요청 IP를 VM에 전달 {확률적으로 IP를 VM에 전달}
end if

VM 방화벽 설정 슈도 코드

Algorithm 7 IP 차단 및 해제 프로세스

입력: POST 요청 데이터 ip
IP에 대해 VM 방화벽 실행:
    a. VM 방화벽 설정을 위한 차단 명령어 생성:
    $blockCommand \leftarrow$ `sudo iptables -A INPUT -s {ip} -j DROP`
    b. 차단 명령어 실행: $execute(blockCommand)$
    c. 차단된 IP 목록에 해당 IP를 차단 상태로 표시
    d. 일정 시간 대기: $sleep(time)$
    e. VM 방화벽 설정을 위한 해제 명령어 생성:
    $unblockCommand \leftarrow$ `sudo iptables -D INPUT -s {ip} -j DROP`
    f. 해제 명령어 실행: $execute(unblockCommand)$
    g. 일정 시간 대기: $sleep(time)$
    h. 차단된 IP 목록에 해당 IP를 해제 상태로 표시
종료

Algorithm 8 명령어를 실행하는 `execute` 함수

입력: 명령어 $command$
함수 execute($command$):
    try:
        $subprocess.run(command)$ {명령어 실행}
함수 종료

4.3.4. 더미 서버 기반의 확률적 트래픽 리다이렉트

알고리즘 설명

- 단위 시간 내 특정 IP가 보낸 요청 횟수를 조회한다.
- 요청 횟수를 입력으로 sigmoid 함수를 호출하여 0과 1사이의 값을 얻는다.
- Sigmoid 함수를 통해 얻은 값을 활용하여 서비스 로직을 수행하기 전 애플리케이션 미들웨어에서 확률적으로 더미 서버로 리다이렉트 한다.
- 더미 서버에는 TC 명령어를 활용하여 응답 시간을 1000ms 지연한다.

슈도 코드

Algorithm 9 IP 요청 횟수 기반 Sigmoid 함수 적용 및 IP 리다이렉션

1: **입력:** 단위 시간 동안의 IP 요청 횟수 $requestCount$
2: $probability \leftarrow sigmoid(requestCount)$ {4번 알고리즘 Sigmoid 결과를 확률로 저장}
3: $randomValue \leftarrow random(0, 1)$ {0과 1 사이의 임의의 값 생성}
4: **if** $randomValue \leq probability$ **then**
5: **리다이렉션**: 요청 IP를 더미 서버로 리다이렉션
6: **end if**

더미 서버 tc 명령어를 활용한 응답 시간 지연

```
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.20.10.9  netmask 255.255.255.240  broadcast 172.20.10.15
        inet6 fe80::884:7451:b40:283  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:0c:b9:5c  txqueuelen 1000  (Ethernet)
        RX packets 27  bytes 4121 (4.1 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 86  bytes 9260 (9.2 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

```
sudo tc qdisc add dev enp0s3 root netem delay 1000ms
```

4.3.5. OpenWRT를 이용한 중개방어

알고리즘 설명

- 단위 시간 내 특정 IP가 보낸 요청 횟수를 조회한다.
- 요청 횟수가 특정 조건이 되면 응답 시간 지연 명령어를 적용한다.
- 이때 요청 횟수 조건은 150회, 240회, 350회이다.
  - 150회일 때는 sigmoid 값이 0.1이 나오므로 100ms 응답 시간 지연 설정
  - 240회일 때는 sigmoid 값이 0.5가 나오므로 500ms 응답 시간 지연 설정
  - 350회일 때는 sigmoid 값이 0.99가 나오므로 1000ms 응답 시간 지연 설정

OpenWRT 명령어 응답 시간 1000ms 적용 예시

```
# traffic control 명령어를 이용해서 qdisc에 특정 interface에 있는 규칙을 조회하는 코드
tc qdisc show dev {interface}


# 단일 IP에 대해서 1000ms manually defense
tc qdisc add dev {interface} root handle 1: htb default 12
tc class add dev {interface} parent 1: classid 1:1 htb rate 1mbit
tc qdisc add dev {interface} parent 1:1 handle 10: netem delay 1000ms
tc filter add dev {interface} protocol ip parent 1:0 prio 1 u32 match ip src {specific_ip} flowid 1:1

# 다중 IP에 대해서 1000ms manually defense
tc qdisc add dev {interface} root handle 1: htb default 12
tc class add dev {interface} parent 1: classid 1:1 htb rate 1mbit
tc qdisc add dev {interface} parent 1:1 handle 10: netem delay 1000ms
tc filter add dev {interface} protocol ip parent 1:0 prio 1 u32 match ip src {specific_ip} flowid 1:1
tc filter add dev {interface} protocol ip parent 1:0 prio 1 u32 match ip src {specific_ip} flowid 1:1
```

4.4. 실험 및 검증

4.4.1. 실험 서버 사양 및 파라미터

가상 머신 환경		
가상화 소프트웨어	Oracle VM VirtualBox 7.0.18	
가상머신(VM) 구성	VM 개수: 1개	
VM 사양	CPU: 4 core	CPU 실행 제한: 100%
	메모리	3092MBn
	디스크 크기	70GB
	설치된 OS	Ubuntu 24.04 LTS

[표 1] 가상 머신 환경 파라미터

위 설정을 통해 서버가 AWS Instance t3.medium 과 비슷한 성능을 가지도록 한다.

네트워크 환경	
네트워크 설정	어뎁터에 브릿지
대역폭	1Gbps

[표 2] 네트워크 환경 파라미터

애플리케이션 서버	
언어	JAVA 21
프레임워크	Spring Boot 3.3.3

[표 3] 애플리케이션 서버 버전

Kubernetes 환경	
Kubernetes	Minikube v1.30.0

[표 4] Kubernetes 버전

Minikube는 로컬 환경에서 Kubernetes 클러스터를 간편하게 실행할 수 있도록 돕는 도구이다. 단일 노드로 구성된 작은 규모의 클러스터를 실행하며, 주로 Kubernetes 학습이나 테스트 용도로 사용된다. 클러스터를 빠르게 생성하고 삭제할 수 있어 개발 환경에서 유용하다.

HPA 설정	
스케일링 대상 리소스 종류	Deployment
스케일링 기준	최소 Pod 수 - 1개
	최대 Pod 수 - 5개

[표 5] HPA 파라미터

행정안전부 정보 자원 효율성 측정 기준에 따르면, 전용 서버의 CPU 활용률이 20% 이하이면 여유롭다고 판단 한다. 따라서 전체 가용 CPU의 20%를 각 Pod의 CPU 사용률 한계로 설정하였다. 한 Pod당 전체 가용 CPU의 20%까지 쓸 수 있기 때문에, Pod는 최대 5개까지 만들 수 있다.[15]

목표 메트릭 값	
메트릭 종류	CPU 사용률
타입	Utilization
평균 사용률 목표치	60%

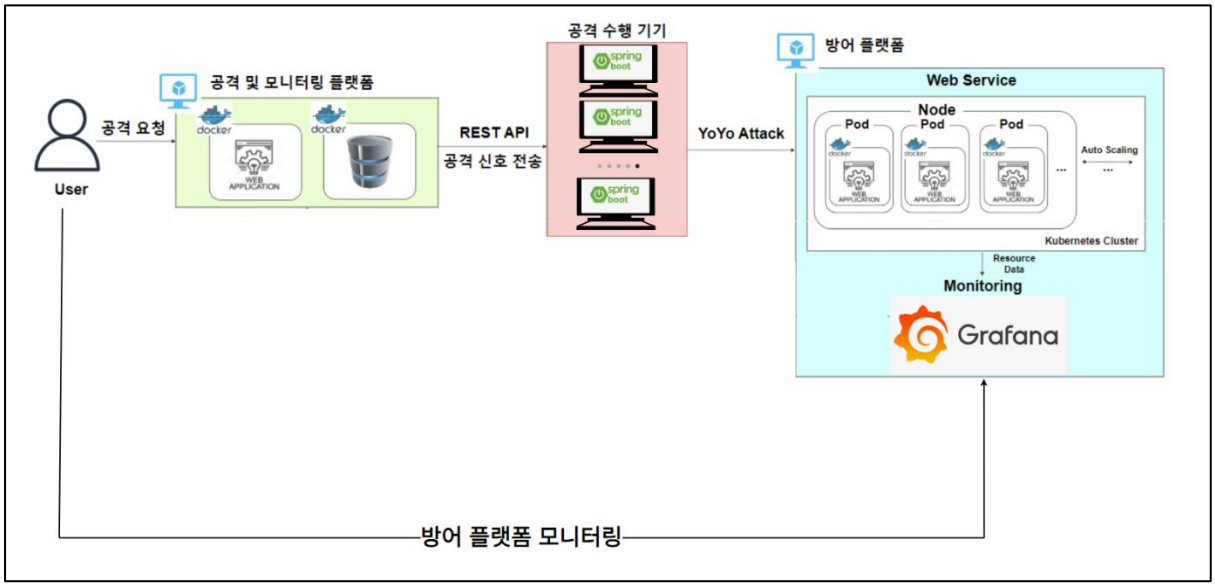
[표 6] 목표 메트릭 파라미터

행정안전부의 정보 자원 효율성 측정 기준, 70% 초과시 서버 사용률이 과다하다고 판단한다. 따라서 60%일 때, Auto Scaling이 일어나도록 설정한다. [15]

HPA 정책			
Scale In (줄일 때)	stabilizationWindowSeconds (안정화시간): 60초		
	정책	종류	Pod 수
		값	1
		periodSeconds(관찰 기간)	60초
Scale In (줄일 때)	stabilizationWindowSeconds (안정화시간): 15초		
	정책	종류	Pod 수
		값	1
		periodSeconds(관찰 기간)	15초

[표 7] HPA 정책 파라미터

4.4.2. 전체 시스템 구성도



[그림 3] 전체 시스템 구성 다이어그램

요소	내용
UI	HTML, CSS, Java Script를 사용하여 공격자 대시보드, 방어자 대시보드를 개발한다
방어 플랫폼 (백엔드 서버)	사용자들이 사용하는 웹 서비스이다. 비정상 트래픽이 감지되면 방어 매커니즘을 적용한다. Spring Boot 를 활용하여 개발한다.
공격 수행 기기	방어 플랫폼에 HTTP 요청을 보내 YoYo Attack 을 수행한다.
Docker	컨테이너 이미지를 기반으로 웹 애플리케이션을 다양한 환경에 배포한다
Kubernetes	컨테이너화 된 애플리케이션을 리소스 사용률에 따라 Auto Scaling 한다.
Grafana	시스템 모니터링 도구로 웹 애플리케이션의 상태와 성능 지표 수집 및 시각화를 수행한다.

[표 8] 시스템 구성요소

4.4.3. 실험 환경

공격자 PC1			
OS	Window 10		
하드웨어	제품명	LG gram	
	CPU	i5-1135G7 @ 2.40GHz	쿼드코어
	메모리	16GB	

[표 9] 공격자 PC1 Spec

공격자 PC2			
OS	Window 10		
하드웨어	제품명	GE75 8RF	
	CPU	i7-8750H @ 2.2GHz	헥사코어
	메모리	16GB	

[표 10] 공격자 PC2 Spec

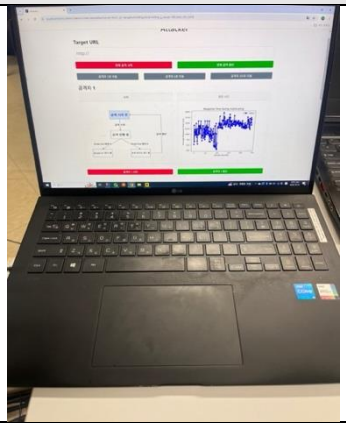
공유기				
소프트웨어	OpenWRT ramips/mt7621 23.05.5 (r24106-10cc5fcd00)			
하드웨어	제품명	ipTIME - AX2004M		
	CPU	미디어텍 MT7621 - 듀얼코어		
	메모리	256MB		
	디스크 크기	128MB		
	연결 단자	LAN	4port	1Gbps
		WAN	1port	1Gbps
		WAN Channel	2.4GHz	5GHz

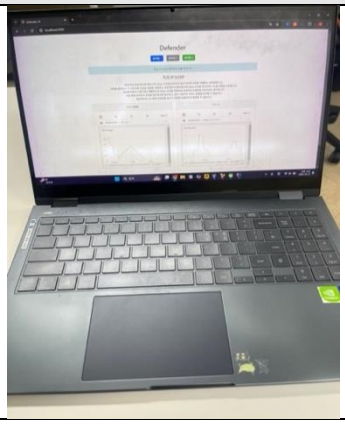
[표 11] 공유기 Spec

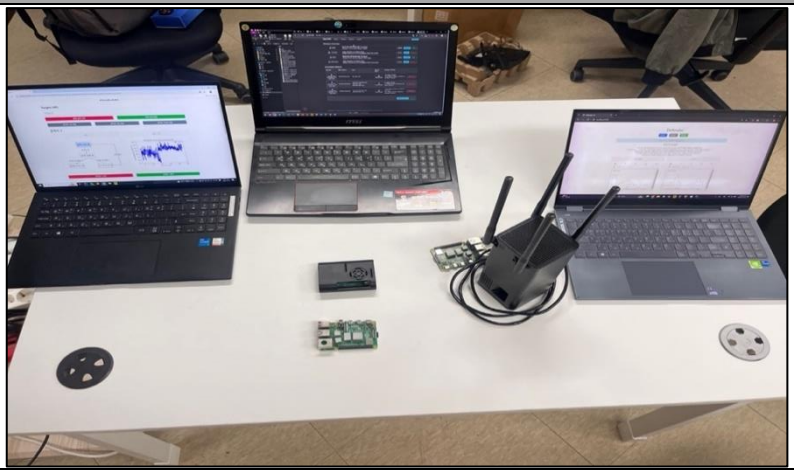
Raspberry Pi			
소프트웨어	Raspberry Pi OS (64-bit)		
하드웨어	제품명	Raspberry Pi 4 1GB	
	MCU	BCM2711	브로드컴(Broadcom)
	프로세서	쿼드 코어 Cortex-A72(ARM v8) 64비트SoC @ 1.5GHz.	
	메모리	1GB	
	디스크 크기	Sdcard 64GB	

[표 12] Raspberry Pi Spec

실험 구성 기기 사진

공격자 구성		
공격자 PC1	공격자 PC2	공격자 PC3
		
[그림 4] 공격자 PC1	[그림 5] 공격자 PC2	[그림 6] 공격자 PC3

방어 매커니즘 구성	
방어 서버	OpenWRT
	
[그림 7] 서버 PC	[그림 8] OpenWRT 설치된 공유기

전체 구성도

[그림 9] 실험 전체 구성도

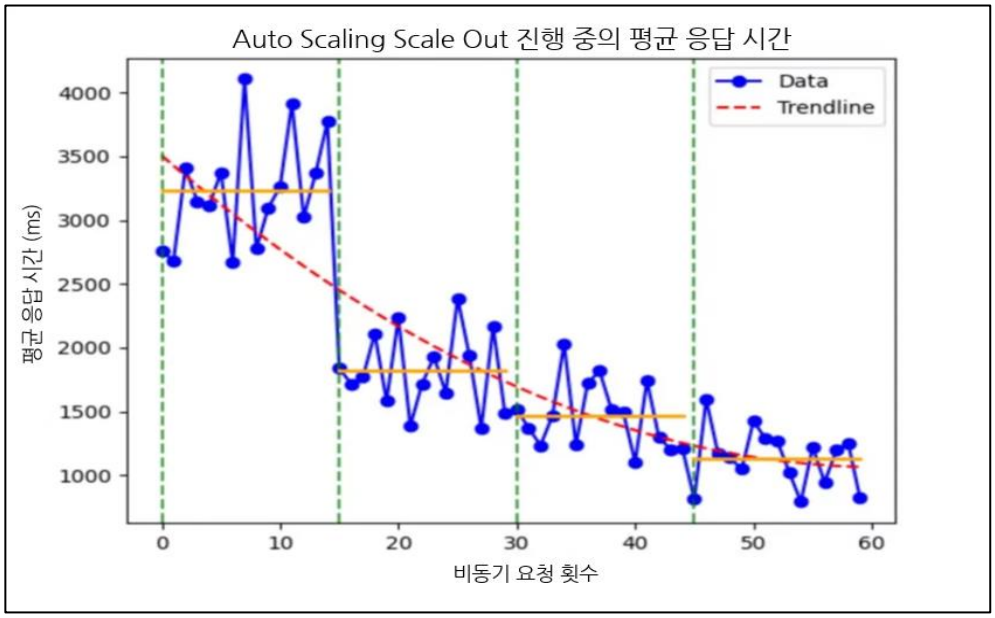
4.4.4. 서버 Pod 개수에 따른 응답 시간 측정

공격자, 방어자 모두 응답 시간을 활용한다.

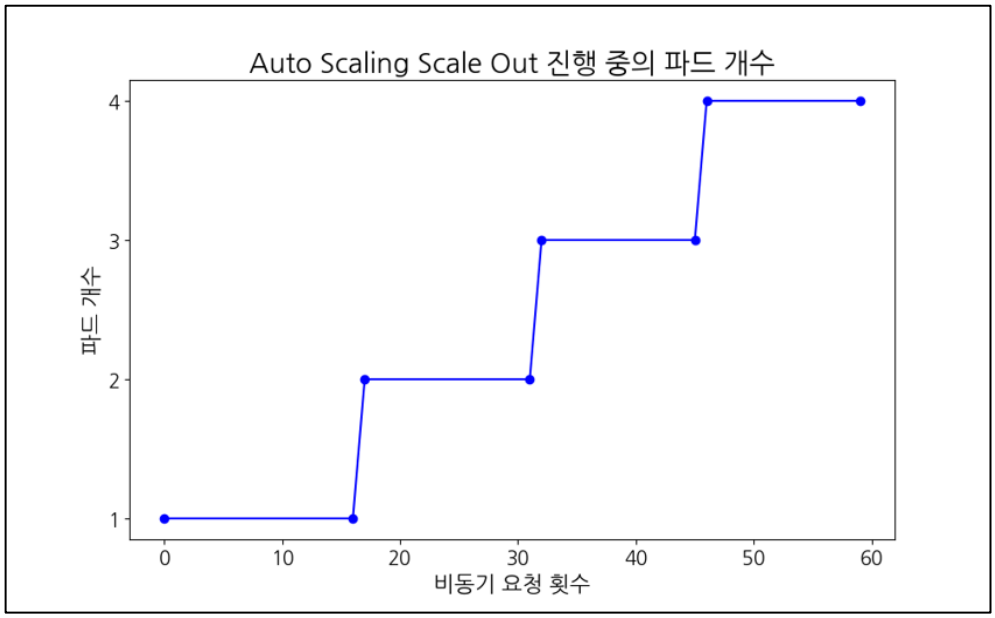
- 공격자는 응답 시간을 활용해 공격의 성공 유무를 판단한다.
- 방어자는 응답 시간을 활용해 공격자의 공격 중단을 유도한다.

따라서 실제로 실험을 위해 구축한 서버에서 Auto Scaling Scale Out 여부에 따른 응답 시간 변화가 배경 지식에서 소개한 [그림 2]와 같이 나타나는지 확인 해보려 한다.

Auto Scaling Scale Out 진행에 따른 응답 시간 변화 측정



[그림 10] Scale Out 진행 중 평균 응답 시간



[그림 11] 요청 횟수 별 Pod 개수

그래프 설명

[그림 10]: Auto Scaling Scale Out 진행 중의 평균 응답 시간

x축: 비동기 요청을 보낸 횟수

y축: 각 비동기 요청에 대한 평균 응답 시간 (단위: ms)

주황색 수평선: 동일 Pod 개수에서의 평균 응답 시간

빨간색 점선: 전체적으로 응답 시간이 어떻게 변하는지에 대한 일반적인 경향을 나타내는 트렌드 라인

녹색 점선: Pod 수가 증설된 지점

[그림 11]: Auto Scaling Scale Out 진행 중의 Pod 개수

x축: 비동기 요청을 보낸 횟수

y축: Pod 수

주요 해석 및 결론

Pod가 증가할 때마다, 평균 응답 시간이 점차 감소하는 경향을 보인다. 이는 Auto Scaling을 통해 처리 능력이 향상되어 서비스의 응답 속도가 빨라지는 것을 나타낸다.

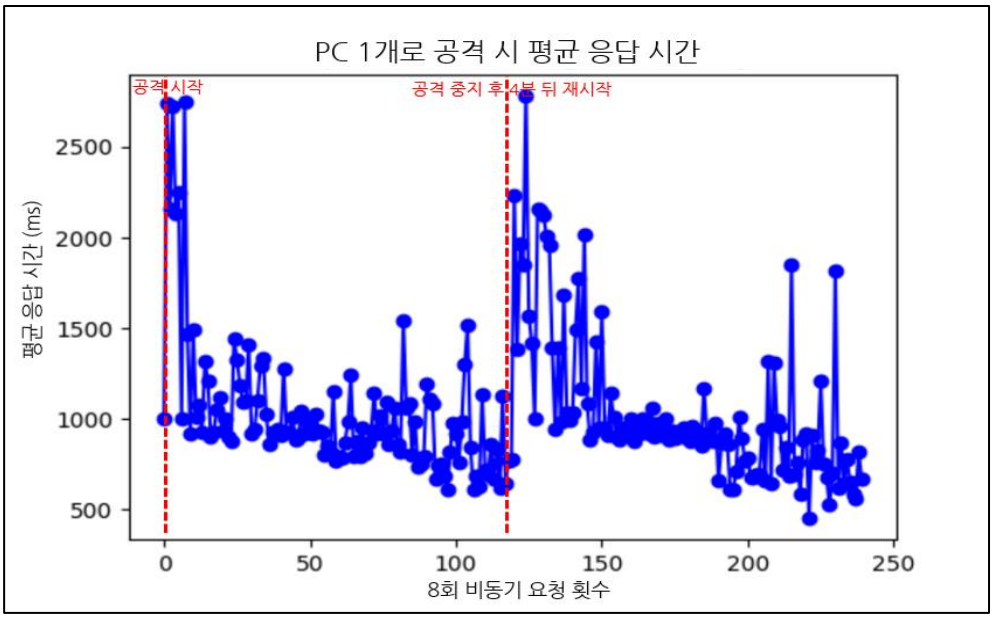
Pod 수가 증가하는 구간마다 응답 시간이 뚜렷하게 감소하는 모습이 보인다.

배경 지식에서 소개한 [그림 2]와 같은 결과가 나온다.

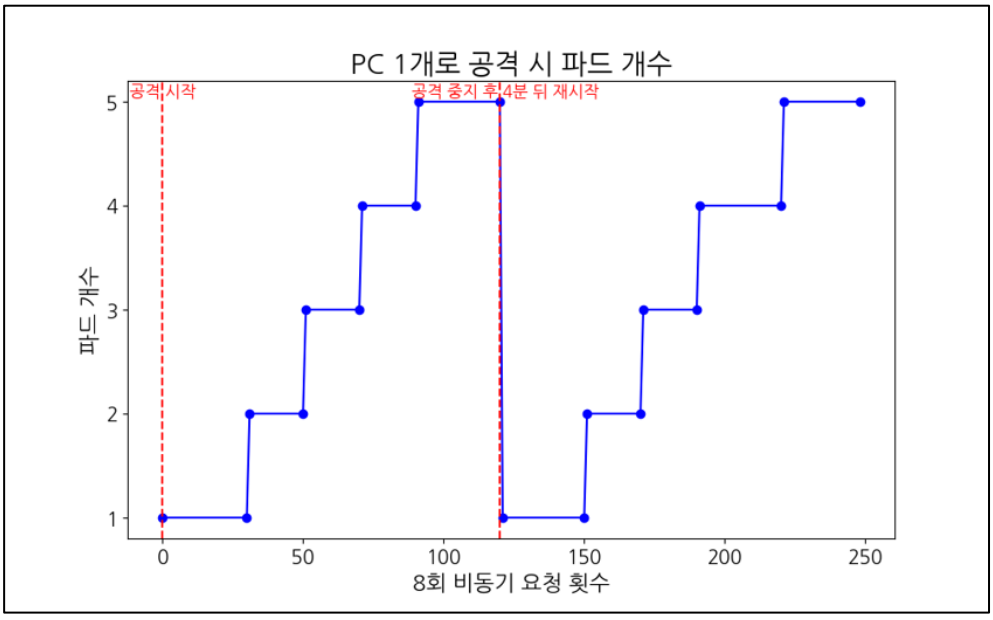
4.4.5. 공격자 알고리즘 실험 결과

작성한 공격자 알고리즘이 Auto Scaling Scale Out 을 유도하여, 서버에 피해를 입히는지 실험해 보았다.

단일 공격자 서버 공격 그래프



[그림 12] 방어 매커니즘 없는 서버 공격 중 평균 응답 시간



[그림 13] 방어 매커니즘이 없는 서버 공격 중 Pod 개수 변화

그래프 설명

[그림 12]: 방어 매커니즘이 없는 서버 공격 중 평균 응답 시간

x축: 8회 비동기 요청 횟수

y축: 8회 비동기 요청에 대한 평균 응답 시간 (단위: ms)

빨간 점선: 공격 시작 지점

[그림 13]: 방어 매커니즘이 없는 서버 공격 중 POD 수

X축: 8회 비동기 요청 횟수(상단 그래프)와 동일

Y축: Pod 수

주요 해석 및 결론

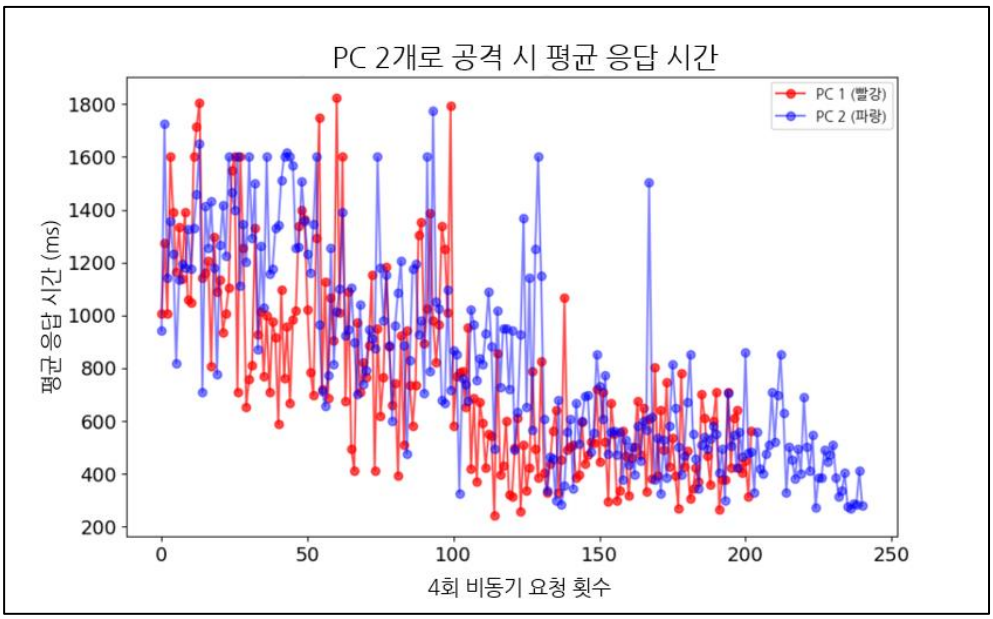
공격이 Auto Scaling을 유발하여 부하가 증가할 때 Pod 수가 증가하고, 공격이 중단되면 Pod 수가 감소하는 패턴을 보여준다. 이는 공격으로 인해 자원의 할당이 증가하여 경제적 비용이 발생할 수 있다.

공격자는 서버에 공격을 시작하고 8회 비동기 요청을 120회 반복 했을 때 평균 응답 시간의 변화가 없다는 것을 감지하고 공격을 중단한다.

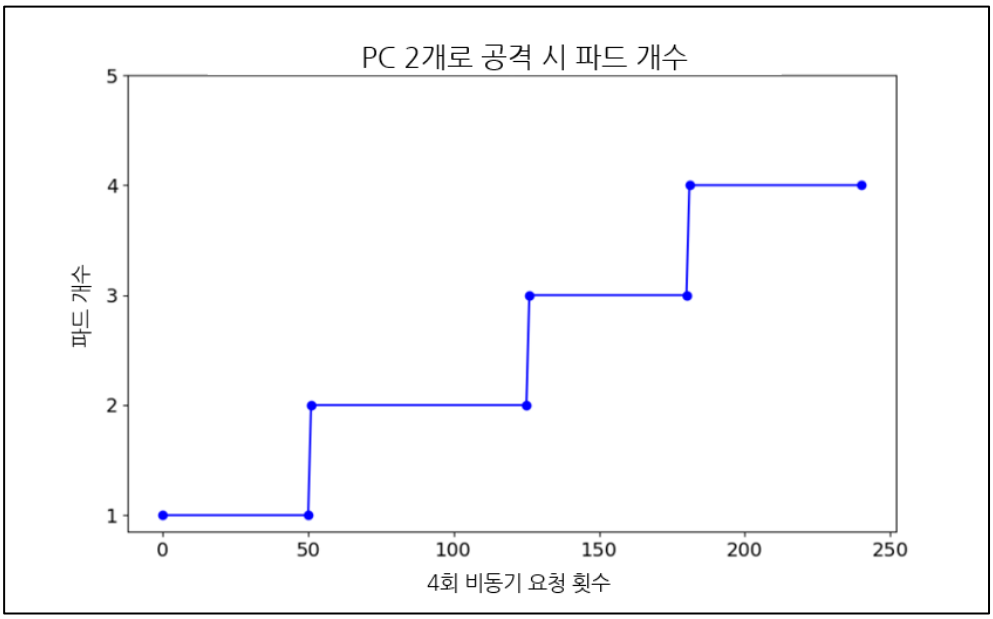
공격을 중단하고 Scale In 을 유도하기 위해 응답 시간이 4회 감소했으므로 4분 대기한다. 4분 후 다시 공격을 진행하고 공격 1회차의 상황과 동일한 그래프를 나타낸다.

구현한 알고리즘은 서버에 경제적 영향과 성능 영향을 준다.

다중 공격자 (2대 PC 활용, 경향성이 반복되므로 1회차 공격만 그래프에 표시)



[그림 14] 방어 매커니즘이 없는 서버 다중 공격 중 평균 응답 시간



[그림 15] 방어 매커니즘이 없는 서버 다중 공격 중 Pod 변화

그래프 설명

[그림 14]: 방어 매커니즘이 없는 서버 다중 공격 중 평균 응답 시간

x축: 4회 비동기 요청 횟수

y축: 4회 비동기 요청에 대한 평균 응답 시간 (단위: ms)

빨간색 점: PC 1의 비동기 요청의 평균 응답 시간

파란색 점: PC 2의 비동기 요청의 평균 응답 시간

[그림 15]: 방어 매커니즘이 없는 서버 다중 공격 중 Pod 수

X축: 4회 비동기 요청 횟수(상단 그래프)와 동일

Y축: Pod 수

주요 해석 및 결론

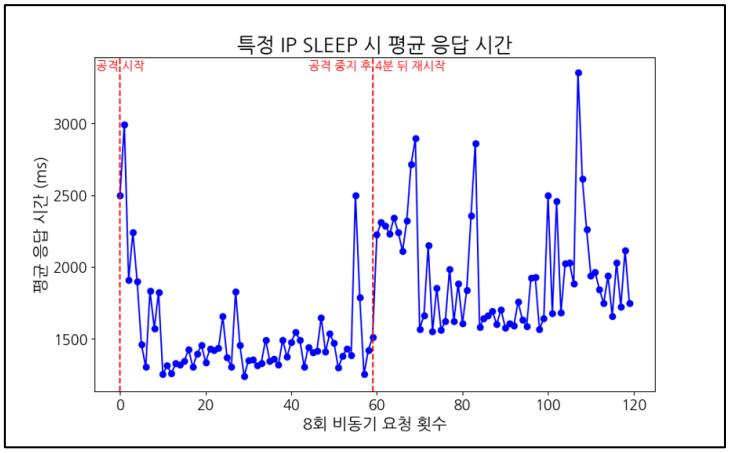
두 PC의 응답 시간 추이가 단일 공격자일 때와 동일한 양상을 보인다.

Scale Out을 최대로 일으키고, Scale In을 유도한 후 다시 공격을 반복하는 양상이 계속해서 반복되어 나타난다.

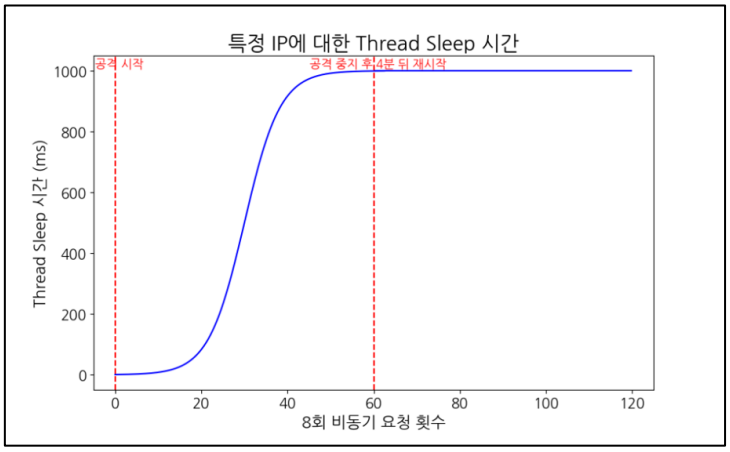
구현한 알고리즘은 다중 공격자일 때도 서버에 경제적 영향과 성능 영향을 준다.

4.4.6. 방어 매커니즘 실험 결과

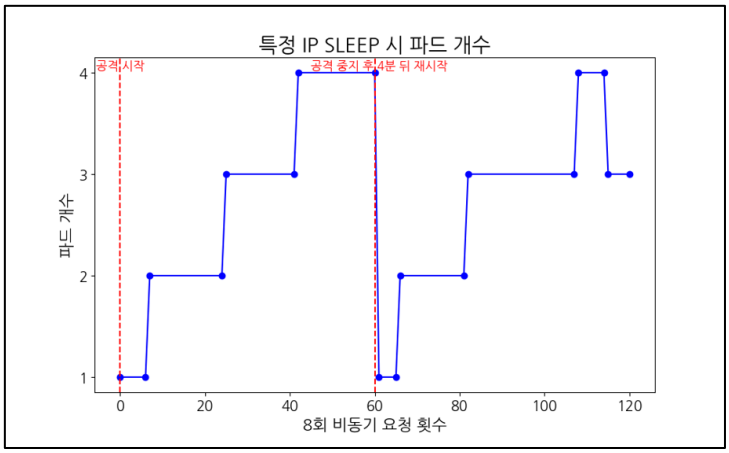
특정 IP SLEEP - 단일 공격자



[그림 16] 특정 IP Sleep 방어법을 적용했을 때 공격자의 평균 응답 시간



[그림 17] Thread Sleep이 적용되는 시간



[그림 18] 특정 IP Sleep을 적용했을 때 Pod 수

그래프 설명

[그림 16]: 특정 IP Sleep 방어법을 적용했을 때 공격자의 평균 응답 시간

x축: 8회 비동기 요청 횟수

y축: 8회 비동기 요청에 대한 평균 응답 시간 (단위: ms)

파란색 점: 비동기 요청의 평균 응답 시간

[그림 17]: Thread Sleep이 적용되는 시간

x축: 8회 비동기 요청 횟수

y축: Thread Sleep이 적용되는 시간 (단위: ms)

[그림 18]: 특정 IP Sleep을 적용했을 때 Pod 수

x축: 8회 비동기 요청을 보낸 횟수

y축: Pod 수

주요 해석 및 결론

1회차 공격에서는 공격자가 요청을 많이 보낼수록 Thread Sleep 시간이 증가한다. 이로 인해 응답 시간이 증가한다. 공격자는 Scale out이 모두 완료되었다고 착각하고, 공격을 일시 중단한다.

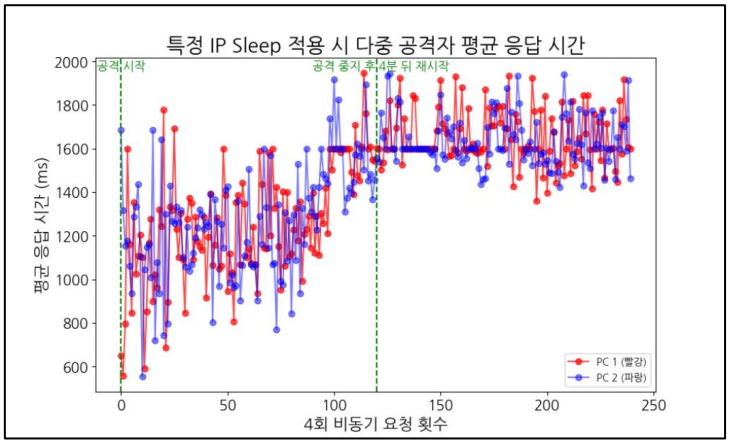
공격자의 요청 횟수가 증가함에 따라 Thread의 Sleep 시간이 증가하고, CPU 사용률이 감소한다.

2회차 공격에서는 서버가 Scale in이 완료된 상태이지만 Thread Sleep에 의해 공격자의 응답시간의 변화가 적다. 이로 인해 공격자는 Scale in이 아직 완료되지 않았다고 오인하여 공격을 중단한다.

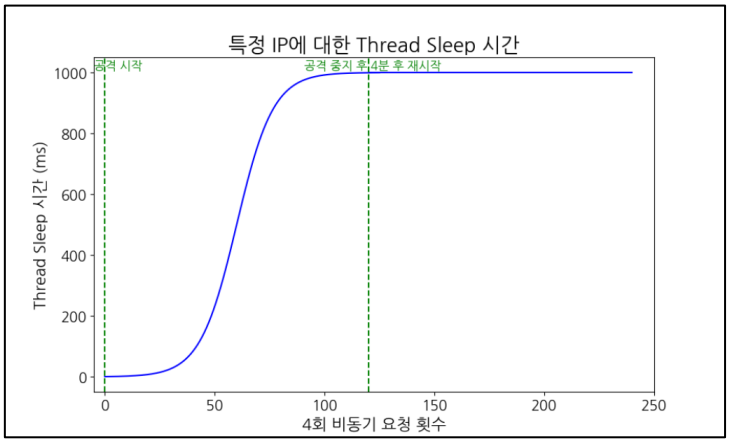
2회차 공격부터는 Thread Sleep이 최대로 적용되고 공격자의 요청이 지연되어 들어와 CPU 사용률이 감소한다. 이로 인해 Pod의 수는 3개까지 늘어난다.

특정 IP Sleep 방어법은 응답 시간을 교란하여 공격자가 공격을 중단하게 한다.

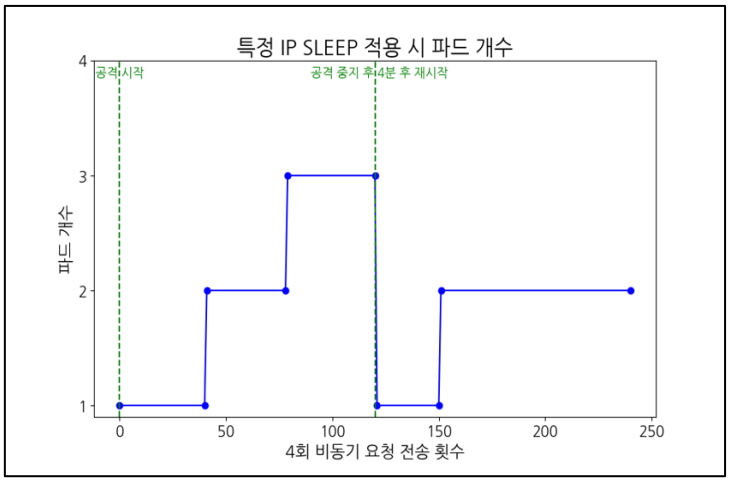
특정 IP Sleep - 다중 공격자



[그림 19] 특정 IP Sleep 방어법을 적용했을 때 공격자들의 평균 응답 시간



[그림 20] 특정 IP Sleep 방어법이 적용되는 시간



[그림 21] 특정 IP Sleep 방어법을 적용했을 때 Pod 수

그래프 설명

[그림 19]: 특정 IP Sleep 방어법을 적용했을 때 공격자들의 평균 응답 시간

x축: 4회 비동기 요청 횟수

y축: 4회 비동기 요청에 대한 평균 응답 시간 (단위: ms)

빨간색 점: PC 1의 비동기 요청의 평균 응답 시간

파란색 점: PC 2의 비동기 요청의 평균 응답 시간

[그림 20]: 특정 IP Sleep 방어법이 적용되는 시간

x축: 4회 비동기 요청 횟수

y축: Thread Sleep이 적용되는 시간 (단위: ms)

[그림 21]: 특정 IP Sleep 방어법을 적용했을 때 Pod 수

x축: 4회 비동기 요청을 보낸 횟수

y축: Pod 수

주요 해석 및 결론

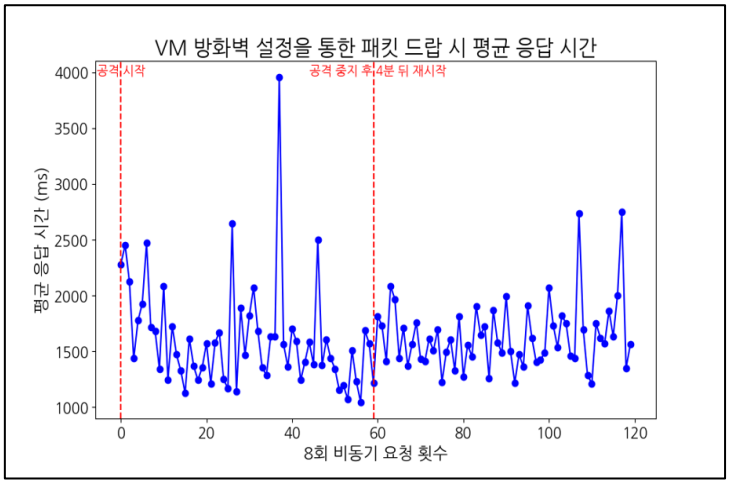
단일 공격과 마찬가지로 1회차 공격에서는 공격자가 요청을 많이 보낼수록 Thread Sleep 시간이 증가한다. 이로 인해 응답 시간이 증가한다. 공격자는 Scale out이 모두 완료되었다고 착각하여 공격을 일시 중단한다.

공격자의 요청 횟수가 증가함에 따라 Thread의 Sleep 시간이 증가하고, CPU 사용률이 감소한다. 결과적으로 Pod의 수는 최대 Pod 수인 4까지 증가하지 않고 3로 유지된다.

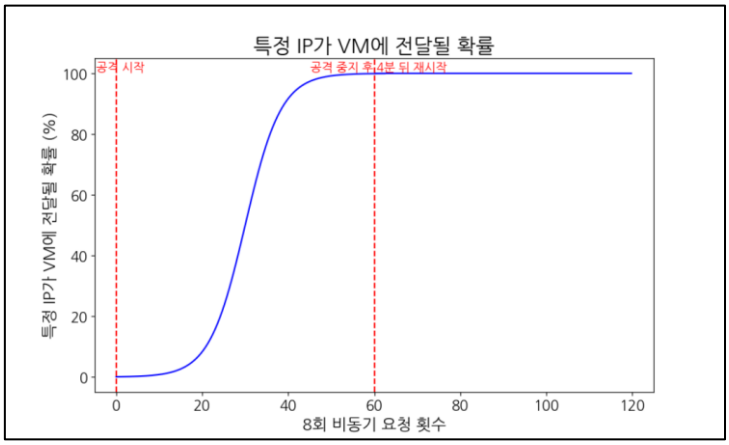
단일 공격과 마찬가지로 2회차 공격에서는 서버가 Scale in이 완료된 상태이지만 Thread Sleep에 의해 공격자의 응답시간의 변화가 적다. 이로 인해 공격자는 Scale in이 아직 완료되지 않았다고 오인하여 공격을 중단한다.

특정 IP Sleep 방어법은 다중 공격자인 경우에도 응답 시간을 교란하여 공격을 중단하게 한다.

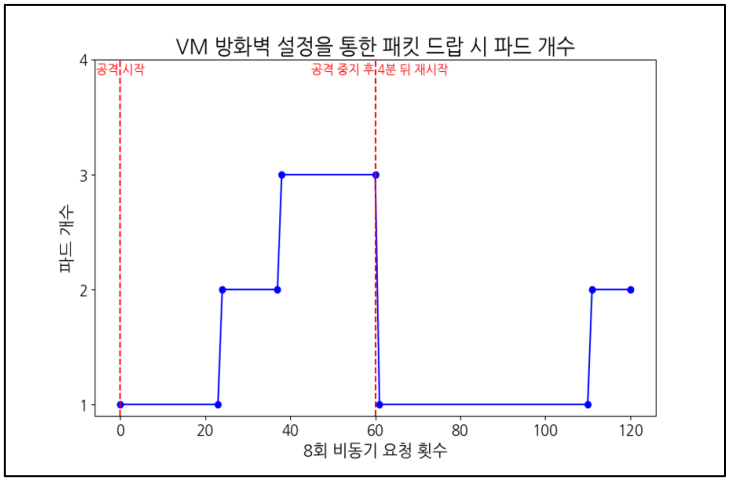
VM 방화벽 설정을 통한 패킷 드랍



[그림 22] VM 방화벽 설정을 통한 패킷 드랍 방어법을 적용 시 평균 응답 시간



[그림 23] 특정 IP가 VM에 전달될 확률



[그림 24] VM 방화벽 설정을 통한 패킷 드랍 방어법을 적용했을 때 Pod 수

그래프 설명

[그림 22]: VM 방화벽 설정을 통한 패킷 드랍 방어법을 적용 시 평균 응답 시간

x축: 8회 비동기 요청 횟수

y축: 8회 비동기 요청에 대한 평균 응답 시간 (단위: ms)

파란색 점: 비동기 요청의 평균 응답 시간

[그림 23]: 특정 IP가 VM에 전달될 확률

x축: 8회 비동기 요청 횟수

y축: VM이 특정 IP를 차단할 확률 (단위: %)

[그림 24]: VM 방화벽 설정을 통한 패킷 드랍 방어법을 적용했을 때 Pod 수

x축: 8회 비동기 요청을 보낸 횟수

y축: Pod 수

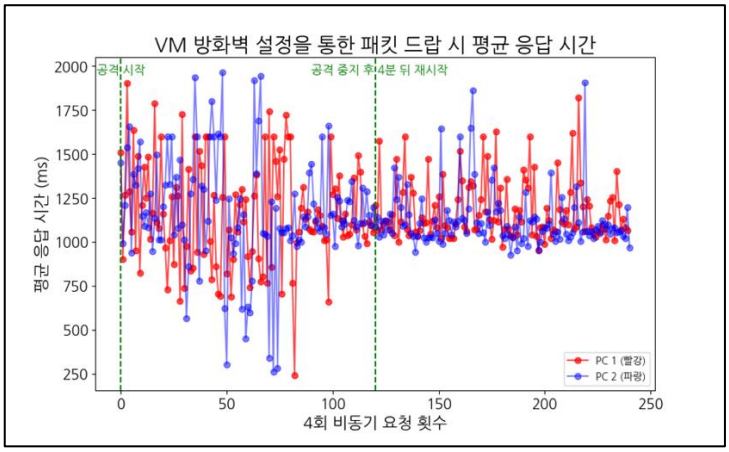
주요 해석 및 결론

1회차 공격에서는 공격자가 요청을 보낼수록 공격자의 IP가 VM에 전달될 확률이 점차 높아진다. VM에 IP가 전달되면, 해당 IP의 네트워크 패킷이 1초 동안 차단된다. 이러한 차단으로 인해 요청의 평균 응답시간이 점차 늘어나게 된다. 공격자는 Scale out이 모두 일어났다고 착각하여 공격을 중단한다.

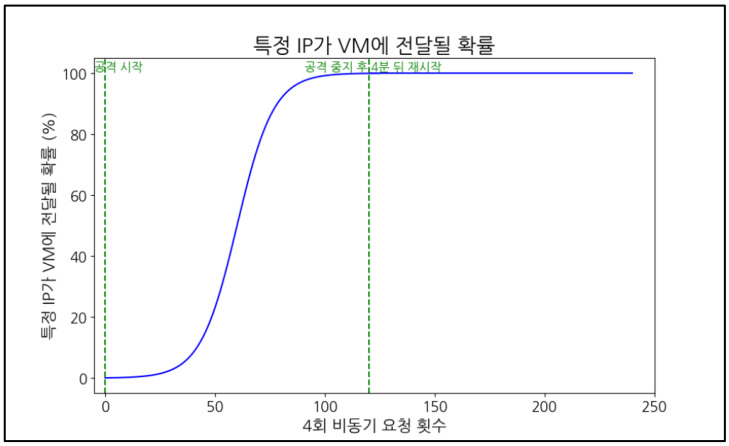
2회차 공격부터는 공격자의 IP가 VM에 전달될 확률이 100%이다. 따라서 공격자의 요청이 반복적으로 차단된다. 이로 인해 CPU 사용률이 감소하게 되고, Pod 수는 2개로 유지된다.

VM 방화벽 설정을 통한 패킷 드랍 방어법은 응답 시간을 교란시켜 공격자가 공격을 중단하도록 유도한다.

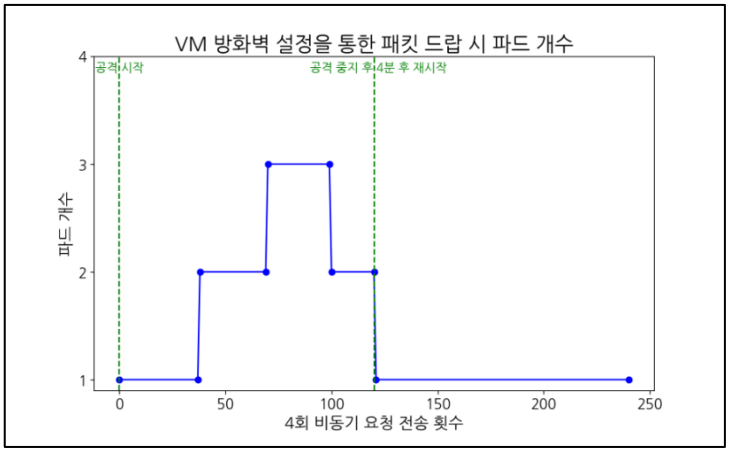
VM 방화벽 설정을 통한 패킷 드랍 - 다중 공격자



[그림 25] VM 방화벽 설정을 통한 패킷 드랍을 적용했을 때 공격자들의 평균 응답 시간



[그림 26] 특정 IP가 VM에 전달될 확률



[그림 27] VM 방화벽 설정을 통한 패킷 드랍을 적용했을 때 Pod 수

그래프 설명

[그림 25]: VM 방화벽 설정을 통한 패킷 드랍을 적용했을 때 공격자들의 평균 응답 시간

x축: 4회 비동기 요청 횟수

y축: 4회 비동기 요청에 대한 평균 응답 시간 (단위: ms)

빨간색 점: PC 1의 비동기 요청의 평균 응답 시간

파란색 점: PC 2의 비동기 요청의 평균 응답 시간

[그림 26]: 특정 IP가 VM에 전달될 확률

x축: 4회 비동기 요청 횟수

y축: VM이 특정 IP를 차단할 확률 (단위: %)

[그림 27]: VM 방화벽 설정을 통한 패킷 드랍을 적용했을 때 Pod 수

x축: 4회 비동기 요청을 보낸 횟수

y축: Pod 수

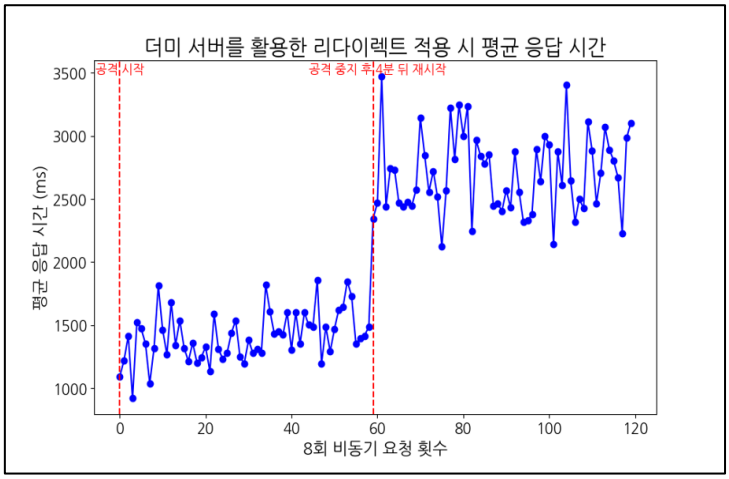
주요 해석 및 결론

단일 공격과 마찬가지로 공격자가 요청을 많이 보낼수록 공격자의 IP가 VM에 전달되어, 패킷이 1초 동안 차단될 확률이 점차 높아진다. 이로 인해 응답 시간이 1초로 일정해진다. 공격자는 Scale out이 모두 완료되었다고 착각하여 공격을 중단한다.

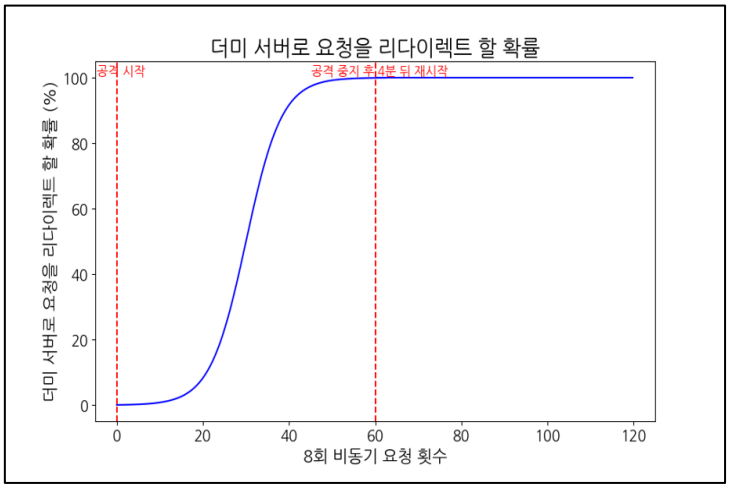
이후 공격에서는 공격자의 요청 횟수가 많아져 IP가 VM으로 전달될 확률이 높아지고 그에 따라 패킷이 짧은 주기로 차단된다. 이로 인해 CPU 사용률이 감소하게 되고, Pod의 수가 증가하지 않는다.

VM 방화벽 설정을 통한 패킷 드롭 방어법은 다중 공격자가인 경우에도 응답 시간을 지연시켜 공격자가 공격을 중단하도록 유도한다. 또한 요청 횟수가 증가할수록 IP를 확률적으로 차단함으로써 CPU 사용률을 낮게 유지한다.

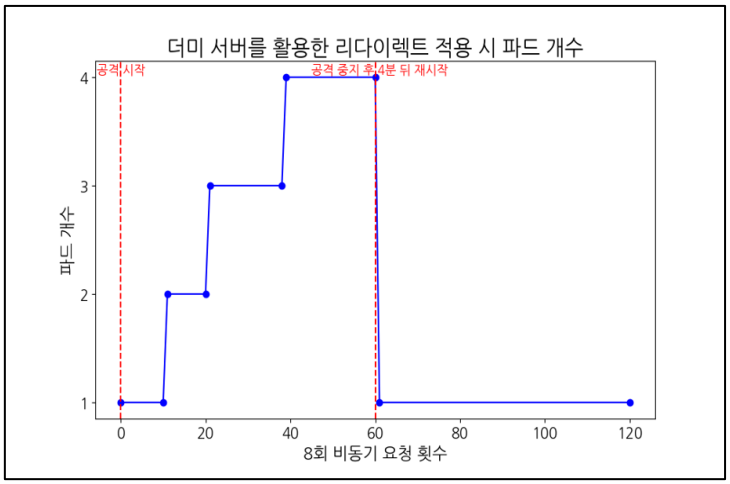
더미 서버 기반의 확률적 트래픽 리다이렉트



[그림 28] 더미 서버 기반 확률적 트래픽 리다이렉트 방어법을 적용했을 때 평균 응답 시간



[그림 29] 더미 서버로 요청을 리다이렉트 할 확률



[그림 30] 더미 서버 기반 확률적 트래픽 리다이렉트 방어법을 적용했을 때 Pod 수

그래프 설명

[그림 28]: 더미 서버 기반 확률적 트래픽 리다이렉트 방어법을 적용했을 때 평균 응답 시간

x축: 8회 비동기 요청 횟수

y축: 8회 비동기 요청에 대한 평균 응답 시간 (단위: ms)

파란색 점: 비동기 요청의 평균 응답 시간

[그림 29]: 더미 서버로 요청을 리다이렉트 할 확률

x축: 8회 비동기 요청 횟수

y축: 더미 서버로 요청을 리다이렉트 할 확률 (단위: %)

[그림 30]: 더미 서버 기반 확률적 트래픽 리다이렉트 방어법을 적용했을 때 Pod 수

x축: 8회 비동기 요청을 보낸 횟수

y축: Pod 수

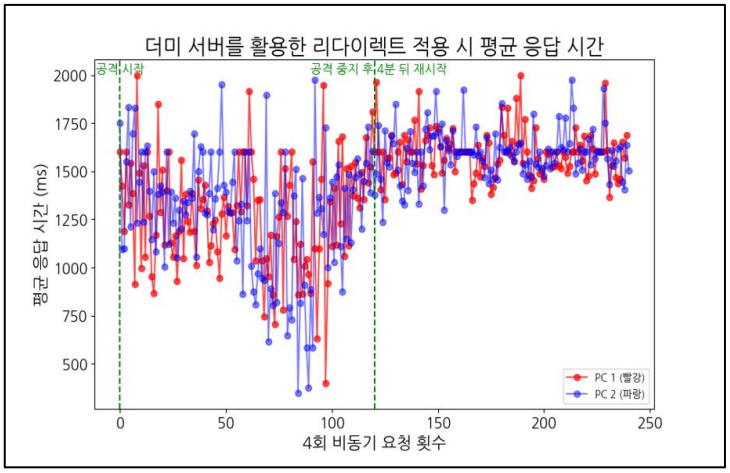
주요 해석 및 결론

1회차 공격에서는 공격자가 요청을 보낼수록 요청이 더미 서버로 리다이렉트 될 확률이 점점 증가한다. 이때 더미 서버는 고의적으로 응답시간을 1초 증가한다. 이로 인해 요청의 평균 응답 시간은 증가하며 공격자는 Scale out이 완료되었다고 판단해 공격을 중단한다.

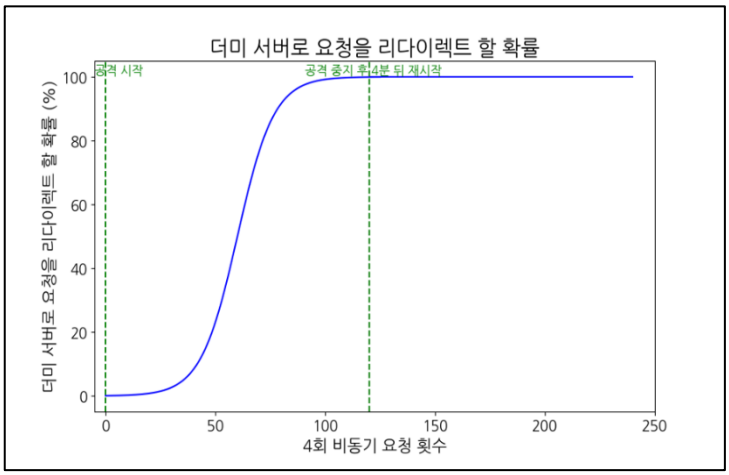
2회차 공격에서는 공격자의 요청이 더미 서버로 갈 확률이 100%이다. 정상 서버로는 공격자의 요청이 들어오지 않아 CPU 사용률이 발생하지 않으며 Pod 수가 2개로 유지된다.

더미 서버 활용 리다이렉트 방어법은 응답 시간을 교란시켜 공격자가 공격을 중단하도록 유도한다. 또한 기준 횟수를 초과하는 요청은 모두 더미 서버로 보내므로 정상 서버의 CPU 자원을 소모하지 않도록 한다.

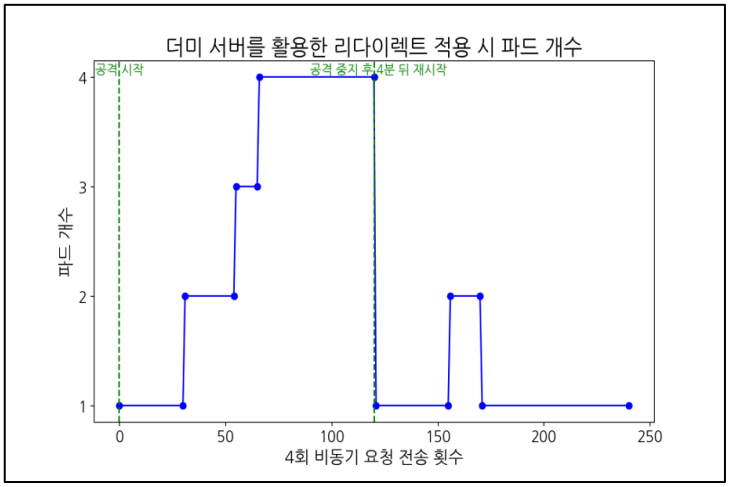
더미 서버 기반 확률적 트래픽 리다이렉트 - 다중 공격자



[그림 31] 더미 서버 기반 확률적 트래픽 리다이렉트를 적용 시 공격자들의 평균 응답 시간



[그림 32] 더미 서버로 요청을 리다이렉트할 확률



[그림 33] 더미 서버 기반 확률적 트래픽 리다이렉트를 적용했을 때 Pod 수

그래프 설명

[그림 31]: 더미 서버 기반 확률적 트래픽 리다이렉트를 적용 시 공격자들의 평균 응답 시간

x축: 4회 비동기 요청 횟수

y축: 4회 비동기 요청에 대한 평균 응답 시간 (밀리초 단위)

빨간색 점: PC 1의 비동기 요청의 평균 응답 시간

파란색 점: PC 2의 비동기 요청의 평균 응답 시간

[그림 32]: 더미 서버로 요청을 리다이렉트할 확률

x축: 4회 비동기 요청 횟수

y축: 더미 서버로 요청을 리다이렉트할 확률 (단위: 퍼센트)

[그림 33]: 더미 서버 기반 확률적 트래픽 리다이렉트를 적용했을 때 Pod 수

x축: 4회 비동기 요청을 보낸 횟수

y축: Pod 수

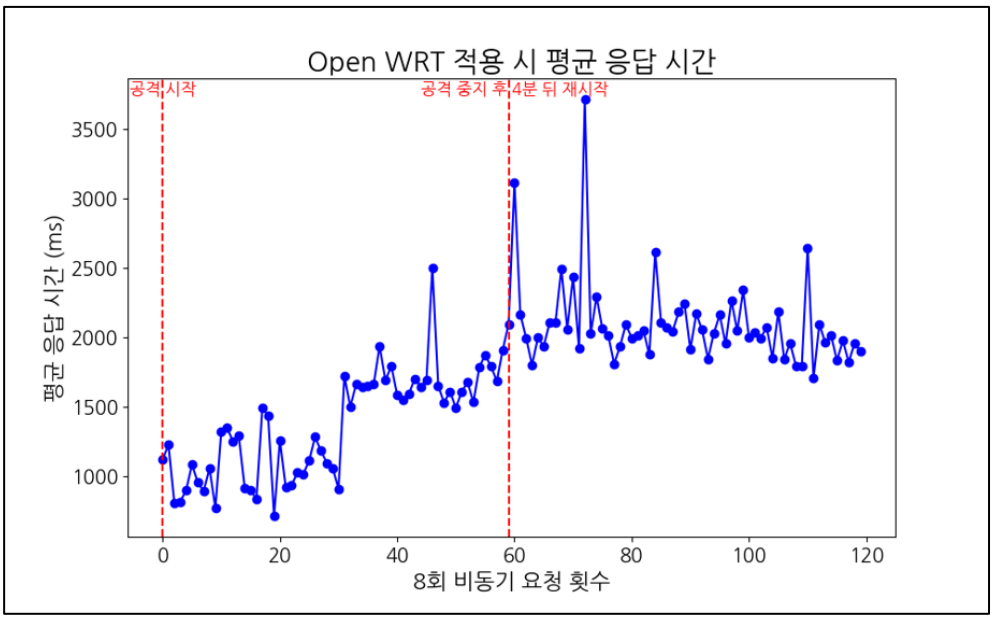
주요 해석 및 결론

단일 공격과 마찬가지로 공격자가 요청을 지속적으로 보내면 요청이 점차 더미 서버로 리다이렉트된다. 이때 더미 서버는 의도적으로 응답 시간을 1초씩 증가시키며, 평균 응답 시간이 유지되거나 점차적으로 늘어난다. 공격자는 Scale out 이 완료되었다고 판단하여 공격을 멈춘다.

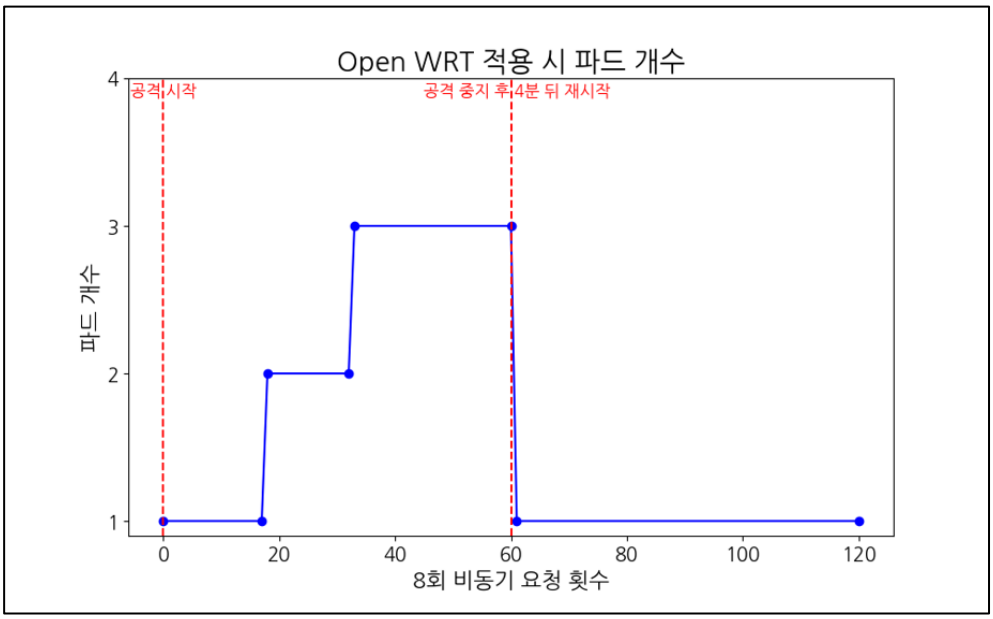
이후 공격에서는 공격자의 요청 횟수가 증가하여 요청의 90% 이상이 더미 서버로 리다이렉트된다. 이로 인해 정상 서버의 CPU 사용률은 감소하게 되며 Pod 가 확장되지 않고 일정한 개수를 유지한다.

더미 서버 기반 확률적 리다이렉트 방어법은 다중 공격자 시나리오에서도 효과적으로 작동한다. 공격자가 요청을 지속할수록 응답 시간과 리다이렉트 확률이 증가하며, 결국 공격자가 공격을 중단하도록 유도한다. 또한 정상 서버의 CPU 사용률을 보호하고, 서버 자원을 효율적으로 관리하는 데 기여한다.

OpenWRT를 이용한 중개방어



[그림 34] OpenWRT를 이용한 중개방어를 적용했을 때 평균 응답 시간



[그림 35] OpenWRT를 이용한 중개방어를 적용했을 때 Pod 수

그래프 설명

[그림 34]: OpenWRT를 이용한 중개방어를 적용했을 때 평균 응답 시간

x축: 8회 비동기 요청 횟수

y축: 8회 비동기 요청에 대한 평균 응답 시간 (단위: ms)

파란색 점: 비동기 요청의 평균 응답 시간

[그림 35]: OpenWRT를 이용한 중개방어를 적용했을 때 Pod 수

x축: 8회 비동기 요청을 보낸 횟수

y축: Pod 수

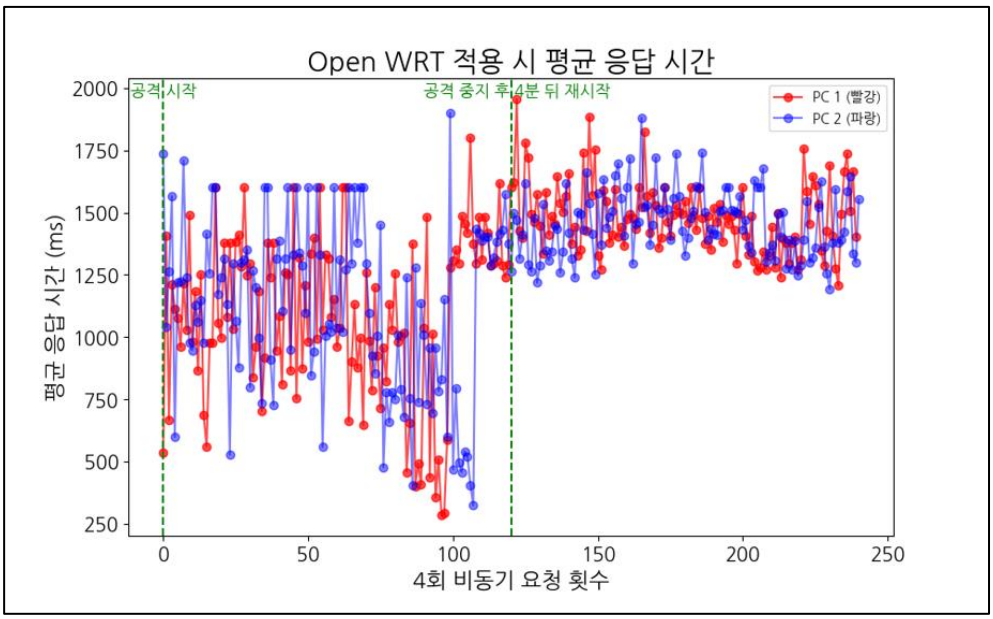
주요 해석 및 결론

요청 횟수가 150, 240, 350 이 되었을 때 OpenWRT를 활용하여 공유기에서 해당 IP에 대한 응답 시간을 각각 100ms, 500ms, 1000ms 지연시킨다. 이로 인해 공격자가 응답시간 감소 구간을 감지하지 못하고, Scale Out이 완료되었다고 판단해 공격을 중단한다.

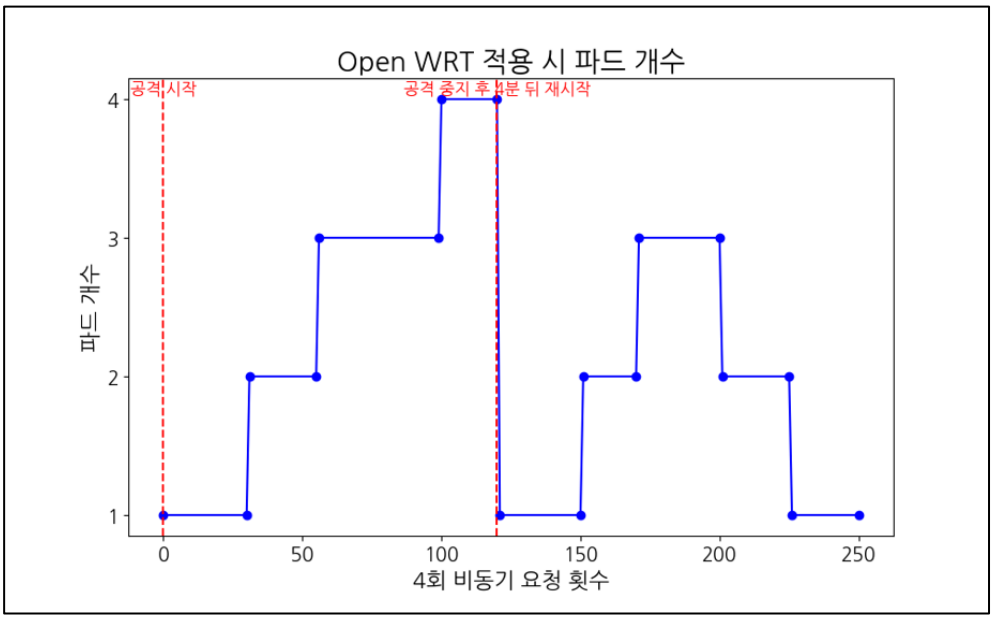
응답시간이 증가함에 따라 요청이 들어오는 시간도 느려져, CPU 사용률이 방어 시스템이 없을 때에 비해 낮기 때문에, Scale out이 일어나지 않는다.

OpenWRT를 이용한 중개방어는 응답 시간을 교란시켜 공격자가 공격을 중단하도록 유도한다. 또한 응답 시간 지연으로 인해 요청 횟수가 감소하여 CPU 사용률을 낮게 유지한다.

OpenWRT를 이용한 중개방어 - 다중 공격자



[그림 36] OpenWRT를 이용한 중개방어를 적용했을 때 공격자들의 평균 응답 시간



[그림 37] OpenWRT를 이용한 중개방어를 적용했을 때 Pod 수

그래프 설명

[그림 36]: OpenWRT를 이용한 중개방어를 적용했을 때 공격자들의 평균 응답 시간

x축: 4회 비동기 요청 횟수

y축: 4회 비동기 요청에 대한 평균 응답 시간 (단위: ms)

빨간색 점: PC 1의 비동기 요청의 평균 응답 시간

파란색 점: PC 2의 비동기 요청의 평균 응답 시간

[그림 37]: OpenWRT를 이용한 중개방어를 적용했을 때 Pod 수

x축: 4회 비동기 요청을 보낸 횟수

y축: Pod 수

주요 해석 및 결론

단일 공격과 마찬가지로 요청 횟수가 150, 240, 350회일 때 응답 시간이 증가한다. 따라서 공격자는 응답시간 감소 구간을 감지하지 못하여 공격을 중단한다.

응답시간이 증가함에 따라 요청이 들어오는 횟수가 줄어들고, 이로 인해 CPU 사용률이 줄어 Scale out이 최대로 일어나지 않는다.

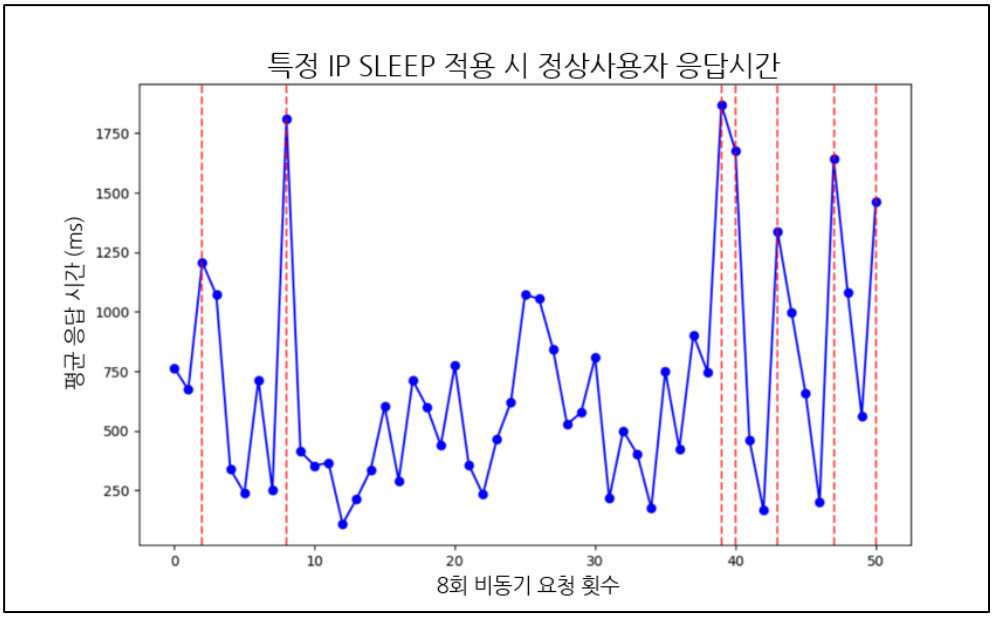
OpenWRT를 이용한 중개방어은 다중 공격자인 경우에도 응답 시간을 교란하여 공격을 중단하게 한다.

4.4.7. 정상 사용자 피해 정도

정상 사용자의 방어 매커니즘에 의한 피해 상황을 정확히 측정하기 위해 다음과 같이 변인을 통제했다

- Pod 수를 4개로 고정
- 공격자가 방어 매커니즘에 최대 강도로 적용됨

특정 IP Sleep



[그림 38] 특정 IP Sleep 방어 매커니즘 적용 시 정상 사용자 피해 정도

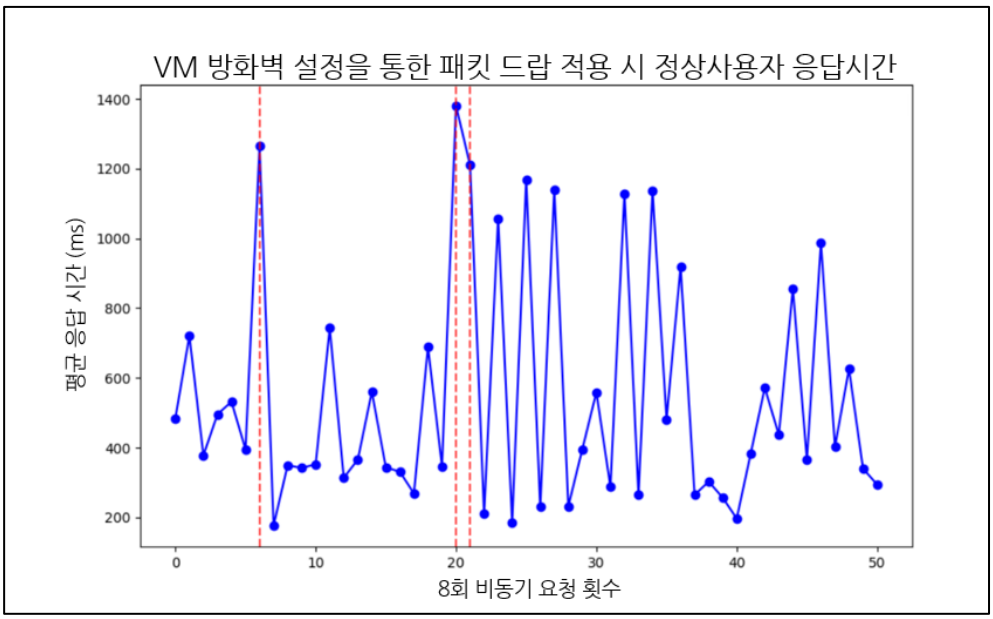
[그림 38] 설명:

응답시간이 1200ms 이상인 요청이 간헐적으로 보인다. (1200ms: apdex 기준 사용자가 서비스에 불만을 느끼는 응답시간)

방어 매커니즘 작동으로 가용 할 수 있는 Thread가 전부 Sleep 할 때 정상 사용자는 기다려야한다. 이때 간헐적으로 응답시간이 증가한다.

전체적인 평균 응답시간은 687.0ms 로 apdex 기준 사용자가 서비스에 만족하는 응답시간이다.

VM 방화벽 설정을 통한 패킷 드랍



[그림 39] VM 방화벽 설정 매커니즘 적용 시 정상 사용자 피해 정도

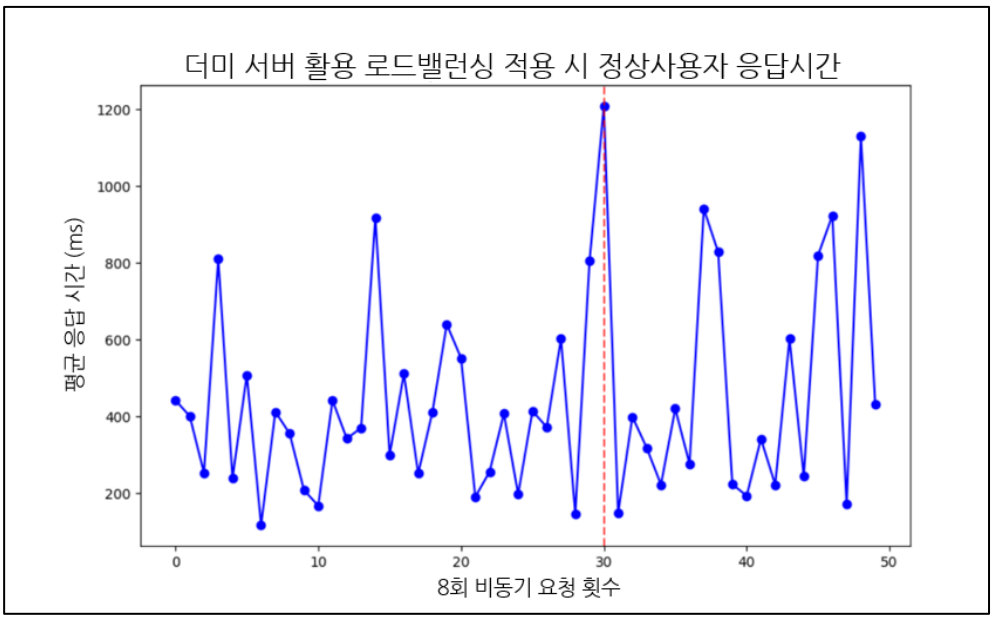
[그림 39] 설명:

응답시간이 1200ms 이상인 요청이 간헐적으로 보인다. (1200ms: apdex 기준 사용자가 서비스에 불만을 느끼는 응답시간)

정상 사용자의 서비스 경험이 저하되는 순간은, 공격자의 방화벽 차단이 해제되어 공격자의 요청이 갑자기 몰려들 때 정상 사용자의 요청이 겹치는 경우이다.

전체적인 평균 응답시간은 543.0 로 apdex 기준 사용자가 서비스에 만족하는 응답시간이다.

더미 서버 기반 확률적 리다이렉트



[그림 40] 더미 서버 기반 확률적 리다이렉트 매커니즘 적용 시 정상 사용자 피해 정도

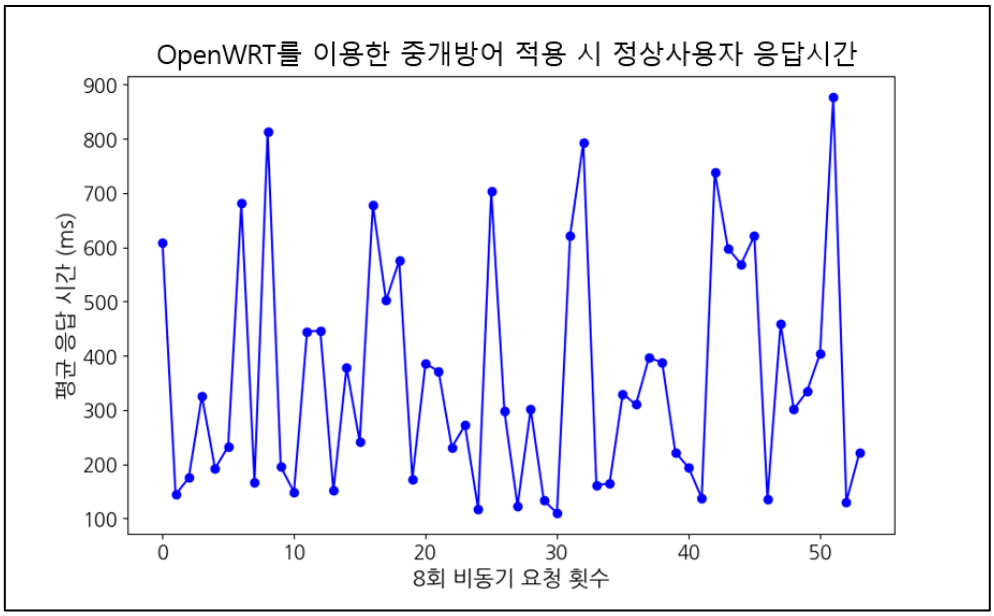
[그림 40] 설명:

응답시간이 1200ms 이상인 요청이 거의 보이지 않는다. (1200ms: apdex 기준 사용자가 서비스에 불만을 느끼는 응답시간)

정상 사용자의 서비스 경험이 저하되는 순간은 낮은 확률로 더미 서버로 리다이렉트 되는 순간이다.

전체적인 평균 응답시간은 442.0ms 로 apdex 기준 사용자가 서비스에 만족하는 응답시간이다.

OpenWRT 를 이용한 중개방어



[그림 41] OpenWRT 를 이용한 중개방어 매커니즘 적용 시 정상 사용자 피해 정도

[그림 41] 설명:

응답시간이 1200ms 이상인 요청이 보이지 않는다.

OpenWRT 를 이용한 중개방어 적용 시 요청 횟수가 적은 정상 사용자는 방어법에 적용될 확률이 거의 없다.

정상 사용자는 사용자 경험을 좋게 유지할 수 있다.

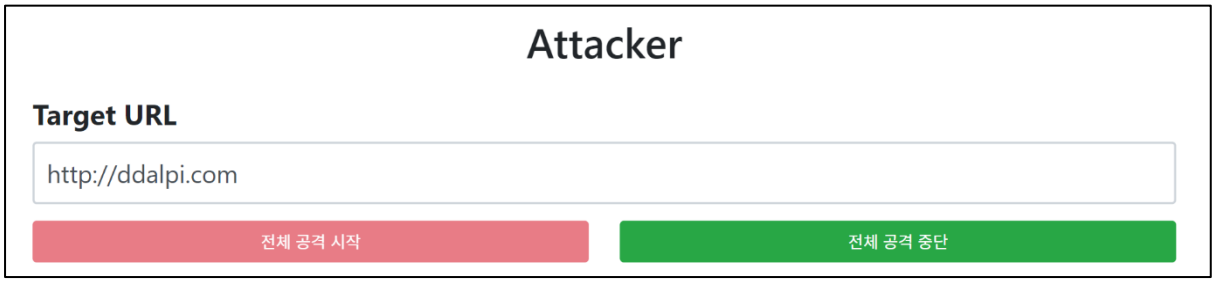
전체적인 평균 응답시간은 359.9ms 로 apdex 기준 사용자가 서비스에 만족하는 응답시간이다.

4.4.8. 공격자와 방어자 시스템 제어 및 모니터링 대시보드

공격자 전체 화면

[그림 42] 공격자 전체 화면

공격자 대시보드 기능 1 - 공격 진행



[그림 43] 공격자 대시보드 기능1

[그림 43] 설명:

Target URL에 공격을 보낼 URL을 입력할 수 있다.

전체 공격 시작 버튼을 눌러서 등록된 공격자에게 공격 요청을 보낸다.

전체 공격 중단 버튼을 눌러 현재 공격을 진행 중인 공격자에게 공격 중단 요청을 보낸다.

공격자 대시보드 기능 2 - 등록된 공격자 상태, 응답 시간 확인

[그림 44] 공격자 대시보드 기능 2

[그림 44] 설명:

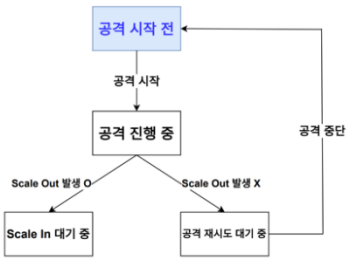
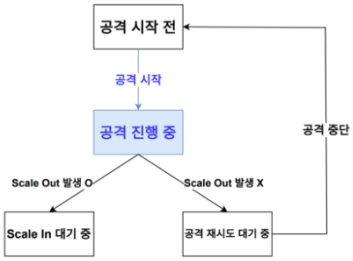
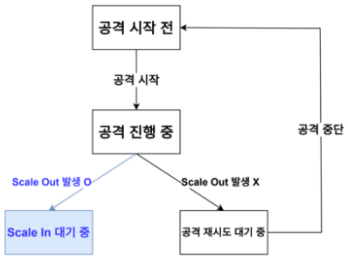
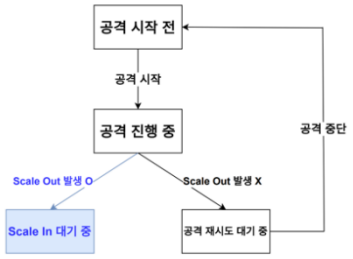
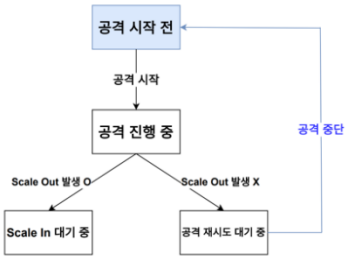
공격자 이동 버튼을 활용하여 등록된 특정 공격자로 이동할 수 있다.

공격자의 상태에서는 현재 공격 진행 상황을 확인할 수 있다. - 5가지 상황 존재

응답 시간 그래프를 활용하여 요청 횟수에 따른 응답 시간 추이를 확인할 수 있다.

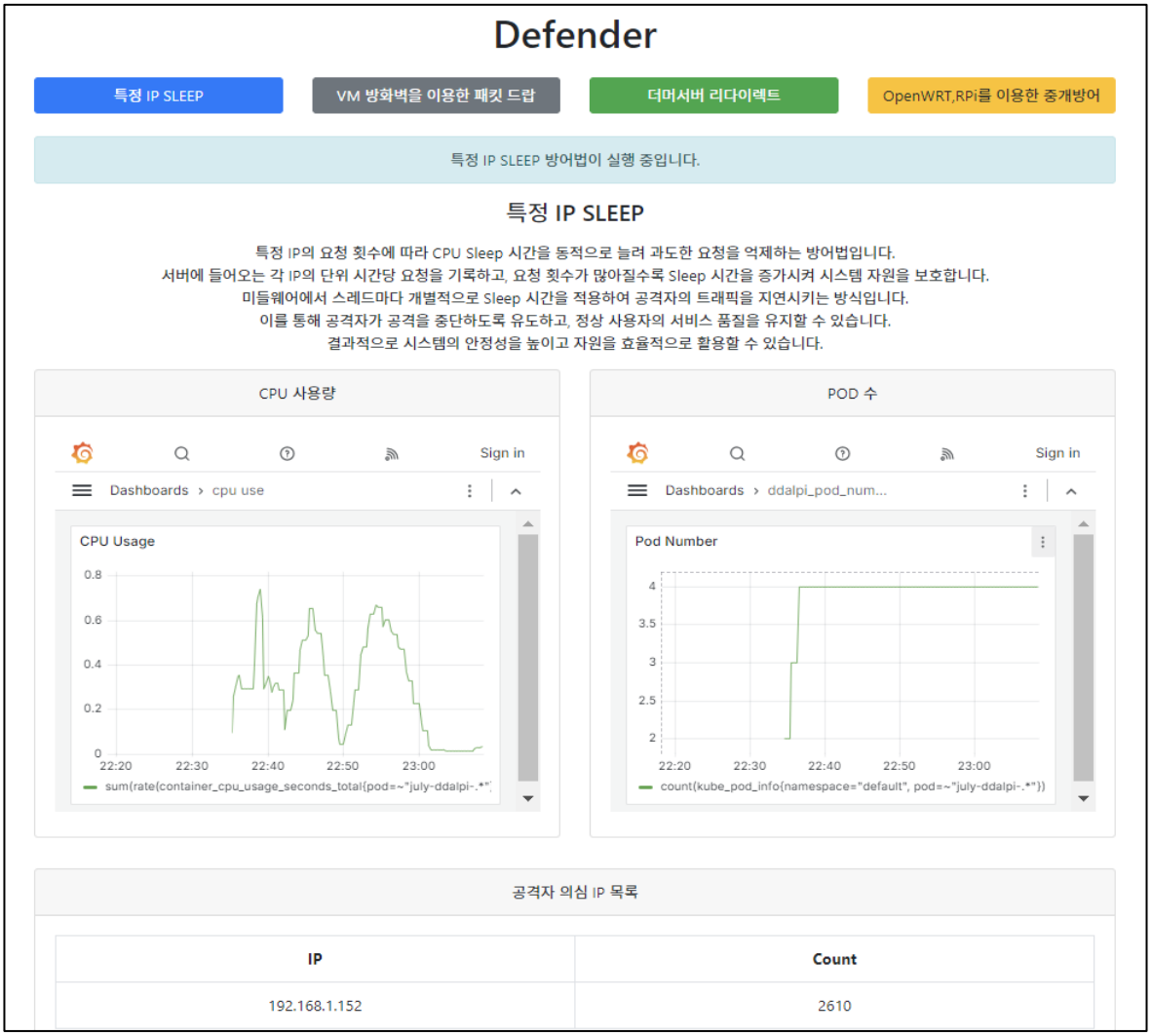
공격자 개별로 공격을 시작, 중단 할 수 있다.

공격 진행 5가지 상황

1. 공격 시작 전	2. 공격 진행 중
	
3. Scale Out 발생 후 Scale In 대기	**4. Scale Out 발생 x 공격 대기**
	
5. 응답 시간 감소 탐지 되지 않아 공격중단	
	

[그림 45] 공격 진행 5가지 상황

방어자 대시보드 전체 페이지



[그림 46] 방어자 대시보드 전체 화면

방어자 대시보드 기능 1 - 방어 방법 선택 및 방어방법

[그림 47] 방어자 대시보드 기능 1

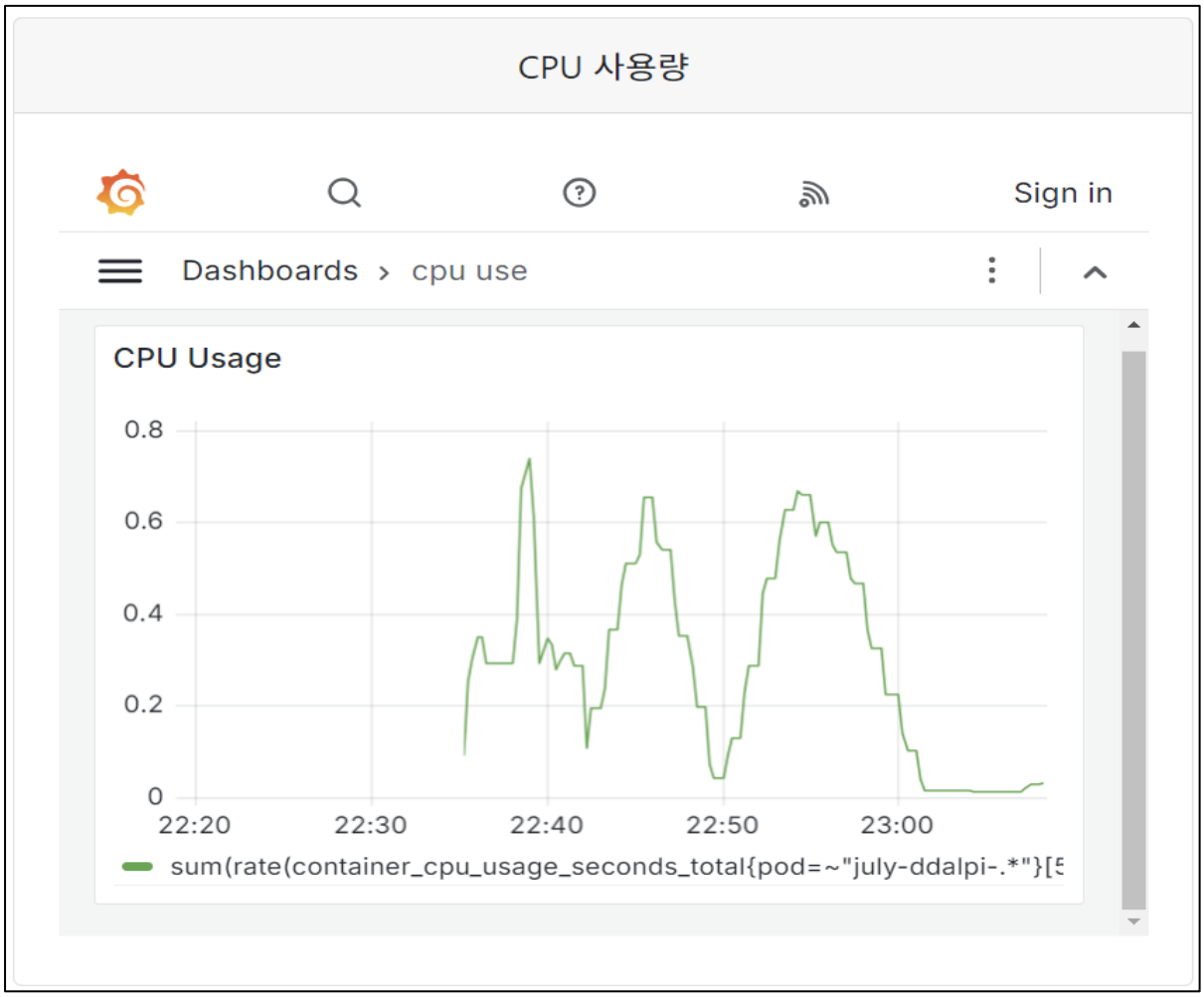
[그림 47] 설명:

사용자는 버튼을 통해 방어 방법을 선택할 수 있다.

파란 박스를 통해 현재 적용 중인 방어 방법을 확인할 수 있다.

현재 적용 중인 방어 방법의 설명을 간략하게 볼 수 있다.

방어자 대시보드 기능 2 - 서버 CPU 사용률 확인



[그림 48] 방어자 대시보드 기능 2

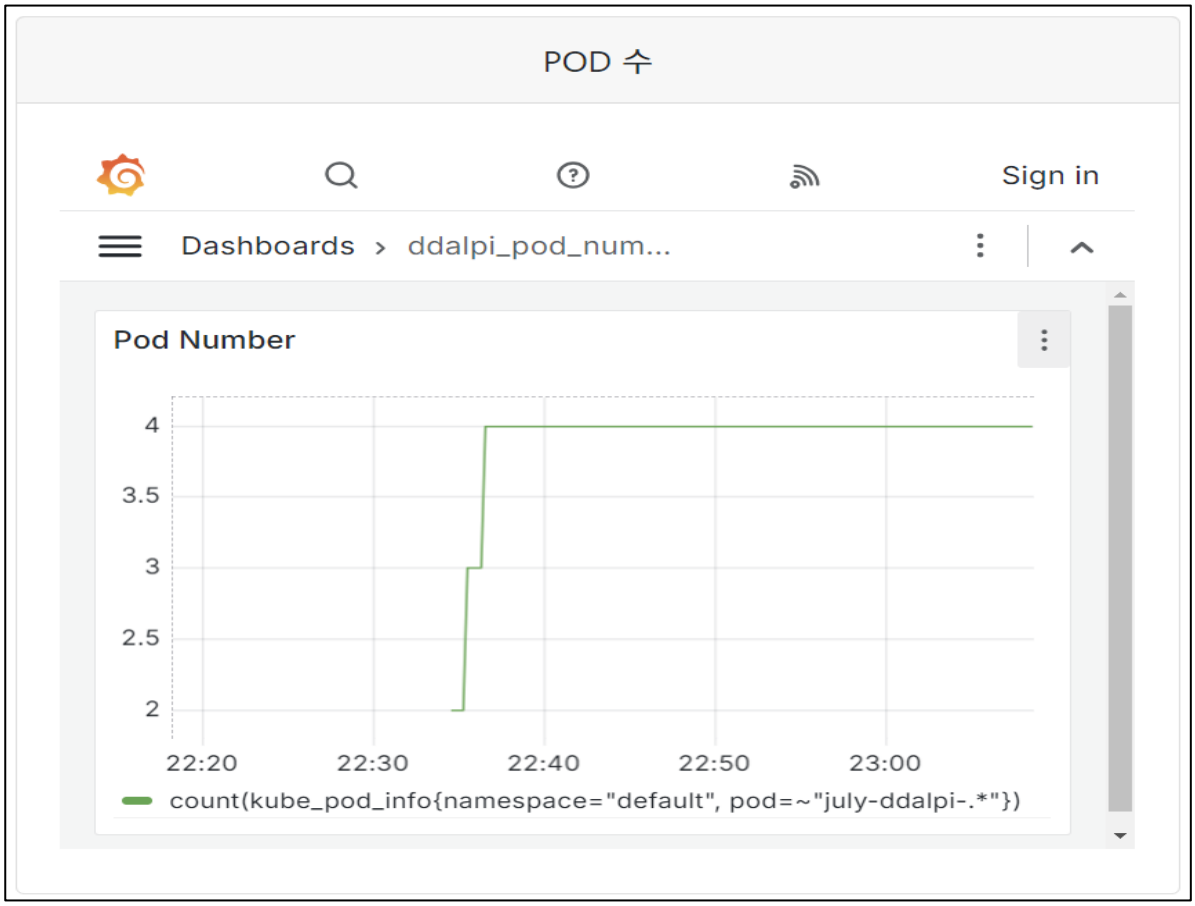
[그림 48] 설명:

서비스의 시간에 따른 CPU 사용률을 확인할 수 있다.

x축은 시간을 나타낸다.

y축은 CPU 코어를 나타낸다.

방어자 대시보드 기능 3 - 서비스 Pod 수 확인



[그림 49] 방어자 대시보드 기능 3

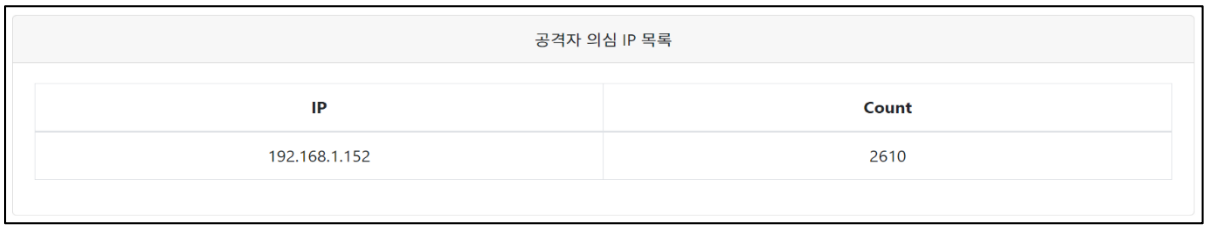
[그림 49] 설명:

서비스의 시간에 따른 Pod 개수를 확인할 수 있다.

x 축은 시간을 나타낸다.

y 축은 Pod 수를 나타낸다.

방어자 대시보드 기능 4 - 공격자 의심 IP 조회



[그림 50] 방어자 대시보드 기능 4

[그림 50] 설명:

현재 서비스에서 공격자로 의심되는 IP 목록을 나타낸다.

단위시간당 240회 이상의 요청을 보내면 공격자 의심 IP 목록에 등록된다.

5. 결론 및 향후 연구 방향

본 연구에서는 클라우드 환경에서 발생할 수 있는 EDoS 공격 중 하나인 YoYo Attack의 공격 방식을 공격자 알고리즘을 통해 분석하고, 이를 효과적으로 방어할 수 있는 플랫폼을 제시한 후 실험을 통해 그 성능을 검증했다. 공격자는 Pod의 Scale Out과 Scale In에 따른 응답 시간 변화를 기반으로 공격 지속 여부를 결정한다. 서버(방어자)는 공격자의 요청에 대한 응답을 의도적으로 지연시켜 응답 시간을 교란함으로써 공격을 중단하도록 유도한다. 또한, 공격자의 다수 요청을 즉시 처리하지 않고 지연시켜 CPU 사용률을 낮추어 Pod 수를 안정적으로 유지할 수 있다.

본 연구에서는 응답 시간을 교란하는 방안으로 특정 IP에 대해 Sleep 적용, VM 서버 방화벽을 활용한 패킷 드랍, 더미 서버 기반의 확률적 트래픽 리다이렉트, OpenWRT를 이용한 중개 방어 등 네 가지 방법을 제시한다. 또한, 정상 사용자의 피해를 최소화하기 위해 Sigmoid 함수를 활용하여 방어 매커니즘이 요청 횟수에 비례해 작동하도록 설계한다. 실험 결과, 공격 의심 IP에 대해 확률적으로 패널티를 부과함으로써 방어 매커니즘이 정상 사용자에게 미치는 영향을 최소화하면서도 공격자를 효과적으로 차단할 수 있음을 확인했다. 또한, 공격자의 응답 시간을 교란하여 공격을 중단시키고 서버(방어자)의 자원 소모를 줄이는 데 성공했다.

향후 연구에서는 본 연구에서 사용된 t3.medium 인스턴스를 넘어 성능이 더 우수한 서버 환경에서 추가적인 실험과 검증을 진행할 계획이다. 특히, 더 높은 자원을 제공하는 서버에서 방어 매커니즘의 성능을 심층적으로 평가할 예정이다. 또한, 본 연구에서 공격자로 사용된 PC 수가 제한적이었다는 점을 보완하여, 더 많은 공격자가 참여하는 대규모 공격 시나리오에서 방어 매커니즘의 효과를 확인하고자 한다. OpenWRT를 활용한 중개 방어 기법 역시 실제 클라우드 환경에 적용 가능한지에 대한 추가적인 실험과 검토가 필요하다. 이 방법이 클라우드 인프라에서 실제로 효율적으로 작동할 수 있는지, 그리고 그 과정에서 발생할 수 있는 성능적 한계나 최적화 방안을 탐구할 계획이다. 아울러, 본 연구에서는 MiniKube를 활용해 로컬 환경에서 Kubernetes 클러스터를 운영하였으나, 제안된 방어 매커니즘이 상용 클라우드 환경의 Kubernetes에서도 안정적으로 동작하는지에 대한 후속 연구가 필요하다. 이를 통해 상용 클라우드 환경에서의 적용 가능성과 시스템 신뢰성을 확인하고, 실제 환경에서의 안정적인 운영 방안을 모색할 예정이다.

6. 구성원별 역할 및 개발 일정

이름	분류	세부 역할 분담
이강빈	OpenWRT 개발 Front - End 개발 회로 구성	Open WRT 방어법 개발 공격자 대시보드, 방어자 대시보드 UI 개발 공격자, 방어자 회로 구성
장진영	알고리즘 개발 공격 플랫폼 개발	공격 알고리즘 개발 방어 알고리즘 개발 - IP_SLEEP 방어법 - 더미 서버 활용 리다이렉트 방어법 - VM 방화벽 활용 패킷 드랍 방어법 공격 플랫폼 구성
강수민	실험 환경 구축 Full Stack 개발	실험 환경 구축 - 도커, Kubernetes 구성 - Prometheus, Grafana 구성 방어 플랫폼 구성 대시보드 UI 개발
공통	보고서 작성	초안, 중간, 최종 보고서
	자료 조사	연구 자료 조사

구분	작업일정																						
	5월			6월				7월				8월					9월				10월		
	3	4	5	1	2	3	4	1	2	3	4	1	2	3	4	5	1	2	3	4	1	2	3
YoYo Attack 스터디																							
VM 환경 YoYo Attack 실습																							
VM 실습 환경 구축																							
공격 알고리즘 구현																							
방어 알고리즘 구현																							
중간 보고서 작성																							
공격 알고리즘 세부구현																							
특정 IP Sleep 알고리즘 세부구현																							
VM 방화벽 패킷 드랍 알고리즘 세부구현																							
더미 서버 리다이렉트 알고리즘 세부구현																							
UI/UX 개발																							
모니터링 시스템 개발																							
OpenWRT 알고리즘 세부구현																							
디버깅 및 테스트																							
최종 보고서																							

7. 멘토링 결과 반영 내역

멘토 피드백에 따라 YoYo Attack 공격/방어 매커니즘의 방향성에 대한 의견은 크게 없었지만, 중간 보고서에서 공격 모델과 방어 매커니즘을 구체적으로 서술할 필요가 있었다는 지적을 받았다. 이에 실험에 사용되는 파라미터들을 다음과 같이 명확하게 제시하고, 해당 값들에 대한 근거도 추가했다.

또한, 실험 결과 그래프 설명이 부족하다는 피드백에 따라 각각의 그래프에 직관적인 이름과 설명을 추가하고, 공격 및 Auto Scaling 시점 등 주요 이벤트 발생 시점도 그래프에 표기했다.

공격자와 방어자의 제어 및 모니터링 시스템, 시스템 자원 사용률 측정 도구를 Spring Boot를 활용하여 대시보드를 개발하여 웹 서비스를 사용할 수 있게 하고, 설명을 4.4.8절에 추가하였다.

더불어, 단일 공격자 환경 뿐만 아니라 다중 공격자 환경에서도 실험을 진행하고 그 결과를 추가해 실험 설계를 보완했다.

지난 연구에서 누락되었던 OpenWRT 중개 방어 기법은 3.3.5절에 알고리즘과 설명을, 4.3.5절에 실험 결과를 추가했다.

마지막으로, 보고서의 구체성이 부족하고 서식이 통일되지 않았다는 의견을 반영해 내용의 구체성을 높이고 서식의 양식을 통일했다.

8. 참고 문헌

[1] Mariusz Michalowski (2024, Mar 27) spacelift [Online] https://spacelift.io/blog/cloud-computing-statistics

[2]Dana Krook (2024,Agu 13) Auvik [Online] https://www.auvik.com/franklyit/blog/cloud-migration-statistics/

[3]문가용 (2024, Agu 8) 보안뉴스 [Online] https://m.boannews.com/html/detail.html?idx=131924

[4] F. Z. Chowdhury, L. B. M. Kiah, M. M. Ahsan and M. Y. I. B. Idris, "Economic denial of sustainability (edos) mitigation approaches in cloud: Analysis and open challenges", 2017 International Conference on Electrical Engineering and Computer Science (ICECOS), pp. 206-211, 2017 .

[5] Adrian Grigorof, K (2020, Oct 12) Managedsentinel [Online] https://www.managedsentinel.com/edos-attack-azure-sentinel/

[6] Gilad David Maayan (2022, Mar 22) FORTRA [Online] https://www.tripwire.com/state-of-security/edos-the-next-big-threat-to-your-cloud

[7] A. Ali-Eldin, M. Kihl, J. Tordsson and E. Elmroth, "Efficient provisioning of bursty scientific workloads on the cloud using adaptive elasticity control", Proceedings of the 3rd Workshop on Scientific Cloud Computing Date, pp. 31-40, 2012.

[8] "Improve 서버 Response Time" Google Developers, [Online]. Available: https://developers.google.com/speed/docs/insights/서버. [Accessed: Oct. 1, 2024].

[9] "Application Performance Index (Apdex)," Apdex.org, [Online]. Available: https://www.apdex.org/. [Accessed: Oct. 1, 2024].

[10] "Amazon EC2 T2 인스턴스," Amazon Web Services, [Online]. Available: https://aws.amazon.com/ko/ec2/instance-types/t2/. [Accessed: Oct. 1, 2024].

[11] "Amazon EC2 T3 인스턴스," Amazon Web Services, [Online]. Available: https://aws.amazon.com/ko/ec2/instance-types/t3/. [Accessed: Oct. 1, 2024].

[12] "Amazon EC2 인스턴스 유형," Amazon Web Services, [Online]. Available: https://aws.amazon.com/ko/ec2/instance-types/. [Accessed: Oct. 1, 2024].

[13] "Horizontal Pod Autoscaler를 사용하여 포드 배포 규모 조정" Amazon Web Services, [Online]. Available: https://docs.aws.amazon.com/ko_kr/eks/latest/userguide/horizontal-Pod-autoscaler.html. [Accessed: Oct. 1, 2024].

[14] "Amazon EC2 Auto Scaling의 스케일링 쿨다운" Amazon Web Services, [Online]. Available: https://docs.aws.amazon.com/ko_kr/autoscaling/ec2/userguide/ec2-auto-scaling-scaling-cooldowns.html. [Accessed: Oct. 1, 2024].

[15] 행정안전부, "정보시스템 운영 성과관리 지침 개정(안)," 2017. [Online]. Available: [https://www.mois.go.kr/cmm/fms/FileDown.do?atchFileId=FILE_00069233jb4EKhm&fileSn=2]. [Accessed: Oct. 1, 2024].