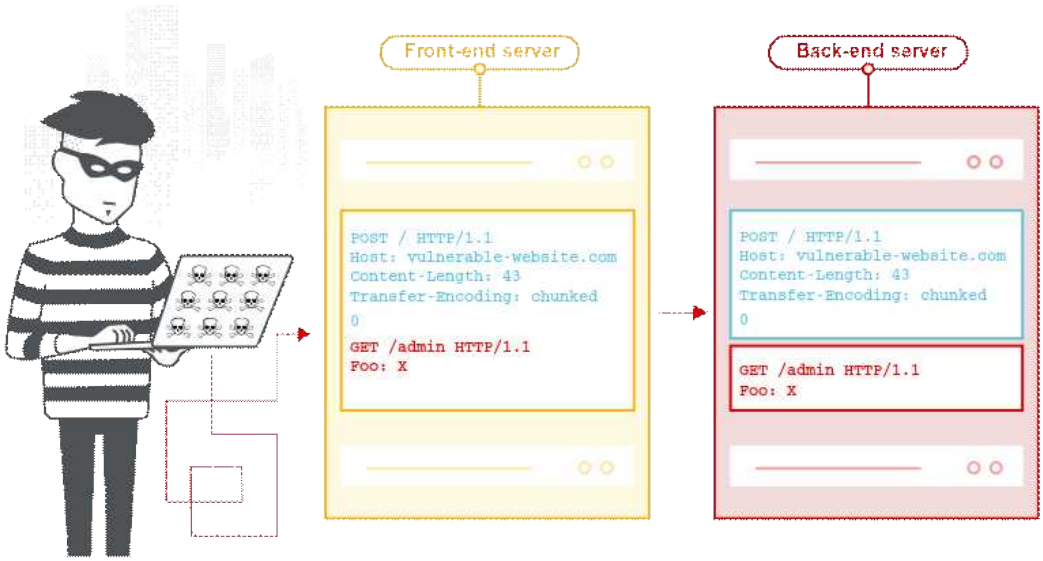
What is HTTP request smuggling?

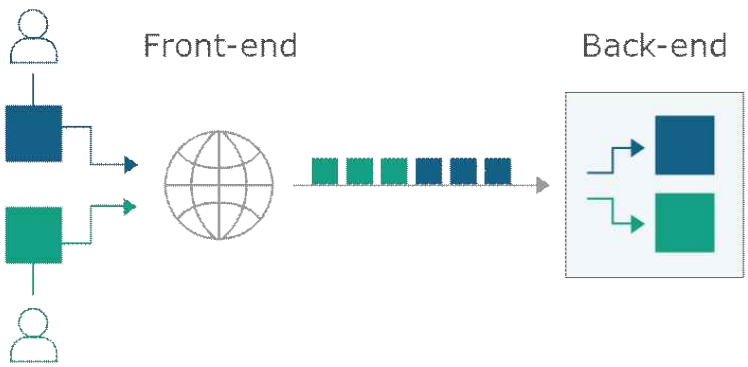
HTTP request smuggling은 사용자로부터 하나 또는 여러개의 수신된 HTTP 요청의 처리 순서 방식을 방해하는 기술이다. Request smuggling 취약점은 공격자가 보안 장치를 우회할 수 있고, 인가되지 않은 민감한 데이터에 접근하여 탈취할 수 있고, 직접적으로 다른 애플리케이션 사용자들에게 손상시킬 수 있는 매우 치명적인 공격이다.



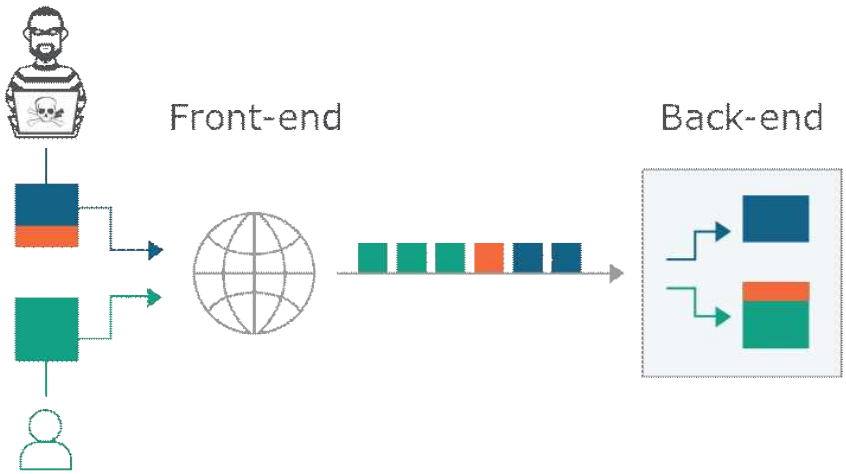
What happens in an HTTP request smuggling attack?

오늘날의 웹 애플리케이션은 사용자들과 애플리케이션 로직 사이에 빈번하게 HTTP 서버 체인을 사용한다. 사용자는 front-end 서버 (load balancer 또는 reverse proxy)에 요청을 보내고, 이 서버는 back-end 서버에 하나 이상의 요청을 포워딩 한다. 이러한 유형의 아키텍쳐는 현대 클라우드 기반 애플리케이션에서 점점 더 일반화되고, 불가피한 경우도 있다.

front-end 서버가 HTTP 요청을 back-end 서버로 포워딩 하면, 일반적으로 동일한 back-end 네트워크 연결(한번 연결된 3wayhandshake)을 통해 여러개의 요청을 보내는데, 이유는 이렇게 하는 것이 더 효율적이고 고능률적이기 때문이다. HTTP 프로토콜을 매우 간단하다.
HTTP 요청은 차례대로 전송되고, 요청을 받은 서버는 한 개의 요청이 어디가 끝이고 다음 요청의 시작 부분을 알아야 하기 때문에 HTTP 요청 해더를 분석한다.



이러한 상황에서, front-end와 back-end 시스템은 요청간의 경계에 대해 동의하는 것이 중요하다. 그렇지 않으면, 공격자가 front-end와 back-end 시스템에서 다르게 해석되는 모호한 요청을 보낼 수 있다.



위 사진을 참고해서 설명하자면, 공격자는 back-end 서버에서 front-end 요청의 일부를 다음 요청의 시작으로 해석한다. 이것은 다음 요청의 앞에 효과적으로 추가되므로, 애플리케이션 처리 요청 방식에 방해할 수 있다. 이것이 HTTP request smuggling attack이고 이것은 재앙을 일으키는 결과물을 보여준다.

How do HTTP request smuggling vulnerabilities arise?

대부분 HTTP request smuggling 취약점은 HTTP specification 가 요청의 끝을 명시하는 2개의 다른 방식을 제공한다. 이는 바로 "Content-Length" 해더와 "Transfer-Encoding" 헤더이다.

"Content-Length" 해더는 간단하다. 이 해더의 값은 bytes로 body message의 길이를 나타낸다.
예를 들어,

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

q=smuggling
```

"Transfer-Encoding" 해더는 message body가 chunked encoding을 사용한다는 것을 명시하기 위해 사용될수 있다. 이 말의 의미는 message body가 하나 이상의 data chunks를 포함되어 있음을 의미한다.
각 chunk는 byte 단위(16진수)에 이어 새로운 줄, chunk 콘텐츠로 구성된다. 이 메시지는 크기가 0인 chunk로 종료된다. 예를 들어,

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

b
q=smuggling
0
```

HTTP specification은 HTTP 메시지의 길이를 명시하기 위해 두 개의 다른 방법을 제공하기 때문에, 단일 메시지에 두 개의 방법을 사용하여 서로 충돌을 일으킬 수 있다. HTTP specification은 "Content-Length" 와 "Transfer-Encoding" 해더가 존재한다면 "Content-Length" 헤더를 무시해야 한다고 명시되어 있어 이러한 문제를 방지하려고 한다. 싱글 서버만 동작중일 때는 이러한 애매모호한 것을 피하기에 충분하지만, 두 개 이상의 서버는 서로 연결되어 있는 경우에는 그렇지 않다. 이러한 상황에서 2가지의 문제가 발생한다.

- 몇몇의 서버는 요청에서 "Transfer-Encoding" 헤더를 지원하지 않는다.
- "Transfer-Encoding" 해더를 지원하는 몇몇의 서버는 해더가 난독화 된다면 처리하지 않을 수 있다.

만약 front-end 와 back-end 서버가 (난독화 될 수 있는) Transfer-Encoding 헤더와 관련하여 다르게 동작하는 경우, 연속적인 요청 사이 간 경계에 대해 동의하지 않아 request smuggling 취약점이 발생할 수 있다.

How to perform an HTTP request smuggling attack

Request smuggling attack은 하나의 HTTP 요청에 "Content-Length" 헤더와 "Transfer-Encoding" 해더 둘다 존재해야 하고 front-end 와 back-end 서버가 요청을 다르게 처리하게 이 해더들을 조작해야 한다. 이를 수행하는 정확한 방법은 두 서버의 동작에 따라 다르다.

- CL.TE: front-end 서버가 Content-Length 해더를 사용하고, back-end 서버가 Transfer-Encoding 해더를 사용하는 경우
- TE.CL: front-end 서버가 Transfer-Encoding 해더를 사용하고, back-end 서버가 Content-Length 해더를 사용하는 경우
- TE.TE: front-end 와 back-end 서버가 Transfer-Encoding 해더를 지원하지만, 두 서버중 하나가 난독화된 해더를 처리하지 못하는 경우

CL.TE vulnerabilities

아래 요청처럼 간단한 HTTP request smuggling attack을 수행 할 수 있다.

```
POST / HTTP/1.1
Host: vulnerable-website.com
Content-Length: 13
Transfer-Encoding: chunked

0\r\n
\r\n
SMUGGLED
```

front-end server는 "Content-Length" 해더를 처리하고 request body가 13bytes 인지 확인한다.
이 요청은 back-end 서버로 포워딩 된다.

back-end 서버는 "Transfer-Encoding" 해러를 처리하게 되고, 따라서 message body를 chunked encoding을 사용하는 것으로 취급한다. 길이가 0인 첫 번째 청그를 처리하므로 요청을 종료하는 것으로 간주된다.
"SMUGGLED"는 처리되지 않고 남아있게 되고, back-end 서버는 다음 요청의 시작으로 취급하게 된다.

TE.CL vulnerabilities

```
POST / HTTP/1.1
Host: vulnerable-website.com
Content-Length: 3
Transfer-Encoding: chunked

8
SMUGGLED
0
```

front-end 서버는 "Transfer-Encoding" 헤더를 처리하고, message body를 chunked encoding을 사용하는 것으로 취급한다. 다음 행의 시작 부분까지 8bytes 길이로 표시된 첫 번째 chunk를 처리한다. 다음은 길이가 0인 두 번째 chunk를 처리하므로 요청이 종료된 것처럼 취급된다. 이 요청은 back-end server로 포워딩 된다.

back-end 서버는 "Content-Length" 해더로 처리하고 request body 가 다음 행의 시작 부분까지 3bytes 인지 확인한다. 남은 데이터는 처리되지 않고 back-end 서버에 다음 요청의 시작부분으로 취급된다.

TE.TE behavior: obfuscating the TE header

"Transfer-Encoding" 해더를 난독화하기 위한 방법은 끝이 없다. 아래 예를 보자.

```
Transfer-Encoding: xchunked

Transfer-Encoding : chunked

Transfer-Encoding: chunked
Transfer-Encoding: x

Transfer-Encoding:[tab]chunked

[space]Transfer-Encoding: chunked

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

Transfer-Encoding
: chunked
```

각각의 이러한 기술은 HTTP specification으로부터 미묘한 차이가 있다. 프로토콜 specification을 구현하는 Real-world 코드는 절대적으로 정확하게 준수되는 경우가 거의 없고. 다른 규현에서는 규격과 다른 변동을 용인하는 것이 일반적이다. TE.TE 취약점을 발견하기 위해서는, front-end 와 back-end 서버 중 하나만 "Transfer-Encoding" 해더를 처리하고 다른 서버는 이를 무시 하도록 헤더의 변형을 찾아야 한다.
난독화된 Transfer-Encoding 헤더를 처리하지 않도록 유도 할 수 있는 front-end 서버인지 또는 back-end 서버인지에 따라 나머지 공격은 위 설명 CL.TE 또는 TE.CL 취약점과 동일한 형태를 갖는다.

How to prevent HTTP request smuggling vulnerabilities

HTTP request smuggling 취약점은 front-end 서버가 같은 네트워크 연결일 때 back-end 서버로 여러개의 요청을 포워딩 할 때 발생하고, back-end 연결에 사용된 프로토콜이 두 서버 간의 경계에 대해 동의하지 않을 위험이 있는 경우에 발생한다. HTTP request smuggling 취약점을 막기위한 방법은 아래와 같다.

- back-end 연결의 재사용을 비활성화 해서, 각각의 back-end 요청이 분리된 네트워크 연결로 보내지게 된다.
- back-end 연결을 위해 HTTP/2를 사용한다. 이 프로토콜은 요청간의 경계에 대한 모호함을 방지한다.
- front-end 와 back-end 서버에서 정확히 같은 웹 서버 소프트웨어를 사용하여, 요청 간에 경계에 대해 동의하게 된다.

경우에 따라, 취약점은 front-end 서버가 모호한 요청을 정규화하거나 back-end 서버가 모호한 요청을 거부하고 네트워크 연결을 닫아 취약점을 피할 수 있다. 그러나 이러한 방법은 위에서 설명한 일반적인 완화 방법보다 오류가 발생하기 쉽다.

interfere 방해하다
unavoidable 불가피한
interprete 해석하다
specify (구체적으로) 명시하다
straightforward 간단한, 쉬운, 복잡하지 않은
obfuscate 애매하게 만들다. (난독화하다.)
induce 설득하다, 유도하다, 유발하다
successive 연속적인
subtle 미묘하다, 미묘한
departure 벗어남, 떠남, 출발