7장 - 웹을 안전하게 지켜주는 HTTPS

👉 HTTP의 약점

평문(암호화 하지 않은) 통신이기 때문에 도청 가능

HTTP를 사용한 리퀘스트나 리스폰스 통신 내용은 `HTTP 자신을 암호화하는 기능이 없기 때문에` 통신 전체가 암호화 되지는 않는다. → 평문(암호화되지 않은 메시지)으로 HTTP 메시지를 보내게 된다.

1. TCP/IP는 도청 가능한 네트워크

- TCP/IP의 구조의 통신 내용은 전부 `통신 경로의 도중에 엿볼 수 있다.`
- 암호화 통신은 메시지 속의 의미는 간파할 수 없을 수도 있겠지만 `암호화된 메시지 자체는 열볼 수 있다.`
- 네트워크 상을 흐르고 있는 패킷을 수집하는 것만으로 도청할 수 있다.
    - 패킷을 수집하려면 패킷을 해석하는 `패킷 캡처(WireShark)`나 `스니퍼`라는 툴을 사용

2. 암호화로 도청을 피하다

도청으로부터 정보를 지키기 위한 방법 중 가장 보급되어 있는 기술은 암호화

→ 암호화의 몇 가지 대상

1. **통신 암호화**
    - 통신을 암호화 하는 방법
    - `HTTP에는 암호화 구조는 없지만 SSL(Secure Socket Layer)이나 TLS(Transport Layer Security)이라는 다른 프로토콜을 조합함으로서 HTTP의 통신 내용을 암호화`
    - SSL 등을 이용해 `안전한 통신로를 확립`하여 HTTP 통신
    - SSL을 조합한 HTTP를 `HTTPS`(HTTP Secure)나 `HTTP overSSL`이라고 명칭

> SSL & TLS 란?
>
> `네트워크 통신에서 데이터의 보안성을 제공`하기 위해 사용되는 프로토콜

> SSL은 초기에 개발된 프로토콜로, TLS는 SSL의 후속 버전으로 개발
> 그러나 일반적으로 SSL과 TLS라는 `용어는 상호교환 가능`하게 사용
> 서버 간의 통신을 암호화하고, 데이터의 무결성과 인증을 보장하는 역할
>
> 1. 암호화(Encryption): SSL/TLS는 데이터를 암호화하여 제3자가 통신 내용을 엿볼 수 없도록 보호합니다. `클라이언트와 서버 간의 모든 데이터는 암호화되어 전송`되므로, 중간에 끼어들거나 도청당하는 위험이 줄어듭니다.
> 2. 무결성(Integrity): SSL/TLS는 데이터 무결성을 유지합니다. 데이터 전송 중에 변경되지 않도록 보장하고, 변조되지 않았음을 확인하기 위해 해시 기반의 메시지 인증 코드(Message Authentication Code)를 사용합니다.
> 3. 인증(Authentication): SSL/TLS는 클라이언트와 서버 간의 상호 인증을 수행합니다. 클라이언트와 서버는 `서로의 신원을 확인`하고, `신뢰할 수 있는지 검증`하는 과정을 거칩니다. 이를 통해 중간에 가짜 서버에 접속하는 등의 위험을 방지할 수 있습니다.
> 4. 호환성(Compatibility): SSL/TLS는 다양한 암호화 및 인증 알고리즘을 지원하며, 다른 프로토콜과의 통합을 가능하게 합니다. 이는 웹 브라우저와 서버, 이메일 서버와 클라이언트 등에서의 상호 운용성을 보장합니다.
>
> SSL/TLS는 HTTPS(Hypertext Transfer Protocol Secure)에서 가장 일반적으로 사용
>
> HTTPS는 HTTP 프로토콜 위에 SSL/TLS 암호화 계층을 추가하여 웹 사이트 간의 보안 연결을 제공합니다. 따라서 사용자가 웹 사이트와 상호 작용할 때 개인 정보가 암호화되어 보호됩니다.

2. **콘텐츠 암호화**

통신하고 있는 `콘텐츠의 내용 자체를 암호화`해 버리는 방법

HTTP에 암호화를 하는 기능이 없기 때문에 HTTP를 사용해서 `운반하는 내용을 암호화`하는 것

`즉, HTTP 메시지에 포함되는 콘텐츠만 암호화 하는 것`

콘텐츠의 암호화를 유효하게 하기 위해서는 클라이언트와 서버가 콘텐츠의 암호화나 복호화 구조를 가지고 있는 것이 전제가 되므로, 평상시에 유저가 사용하는 브라우저와 웹 서버에서는 이용하는 것이 어렵다.

주로 `웹 서비스 등에서 이용되는 방법`이다.

통신 상대를 확인하지 않기 때문에 위장 가능

HTTP를 사용한 리퀘스트나 리스폰스에서는 `통신 상대를 확인하지 않는다.`

즉, 리퀘스트를 보낸 서버가 정말로 URI에서 지정된 호스트인지, 리스폰스를 반환한 클라이언트가 정말로 리퀘스트를 출력한 클라이언트인지를 확인이 불가

1. 누구나 리퀘스트할 수 있다.

- HTTP에 의한 통신에는 상대가 누구인지 확인하는 처리가 없기 때문에 누구든지 리퀘스트를 보낼 수 있다.
- HTTP는 리퀘스트를 보내면 리스폰스가 반환되는 매우 심플한 구조 (`상대를 확인하지 않는 점이 약점`)
    - 리퀘스트를 보낸 곳의 웹 서버가 원래 의도한 리스폰스를 보내야 하는 웹 서버인지 아닌지를 확인할 수 없다. (`위장한 웹 서버`일 우려가 있다.)
    - 리스폰스를 반환한 곳의 클라이언트가 원래 의도한 리퀘스트를 보낸 클라이언트인지 아닌지를 확인할 수 없다. (`위장한 클라이언트`일 우려가 있다.)
    - 통신하고 있는 상대가 접근이 허가된 상대인지 아닌지를 확인할 수 없다. (중요한 정보를 가진 웹 서버에서는 `특정 상대만 통신을 허가하고 싶을 경우`가 있다.)
    - 어디의 누가 리퀘스트를 했는지 확인할 수 없다.
    - 의미없는 리퀘스트라도 수신하게 된다. (`대량의 리퀘스트에 의한 DoS 공격`(서비스 불능 공격)을 방지할 수 없다.)

2. 상대를 확인하는 증명서

- HTTP에서는 통신 상대를 확인할 수 없지만 `SSL로 상대를 확인할 수 있다.`

- SSL은 암호화뿐만 아니라 상대를 확인하는 수단으로 증명서를 제공 (디지털 인증서)
- 증명서는 신뢰할 수 있는 제 3자 기관에 의해 발행되는 것이기 때문에 서버나 클라이언트가 실재하는 사실을 증명한다.
    - 디지털 인증서는 인증 기관(Certificate Authority, CA)에 의해 발급되며, 발급 기관의 디지털 서명에 의해 인증
- 증명서를 위조하는 것은 기술적으로 상당히 어렵다.
- 클라이언트가 증명서를 가짐으로써 본인 확인을 하고, 웹 사이트 인증에서 이용할 수도 있다.

> ### SSL/TLS 인증 과정의 단계
>
> 1. 클라이언트 Hello: 클라이언트는 서버에 연결 요청을 보내면서 `지원 가능한 암호화 및 인증 알고리즘 목록, 클라이언트 랜덤(임의의 값)을 포함`한 Hello 메시지를 전송합니다.
> 2. 서버 Hello: 서버는 클라이언트 Hello 메시지를 받고, `클라이언트와의 암호화 및 인증 알고리즘, 서버 랜덤 값을 포함`한 Hello 메시지를 응답합니다.
> 3. 인증서 교환: 서버는 클라이언트에게 `서버의 디지털 인증서를 전송`합니다. 이 인증서에는 `서버의 공개키와 발급 기관의 디지털 서명이 포함`되어 있습니다.
> 4. 클라이언트 신원 확인: 클라이언트는 서버의 `인증서를 수신하고, 발급 기관의 디지털 서명을 확인`합니다. 클라이언트는 발급 기관의 공개키를 가지고 있어야 합니다. 클라이언트는 또한 인증서의 유효 기간, 도메인 이름 등과 같은 정보를 확인하여 서버의 신원을 검증합니다.
> 5. 클라이언트 공개키 교환: 클라이언트는 서버의 공개키를 사용하여 세션 키(일회용 대칭 키)를 암호화하여 서버에 전송합니다.
> 6. 서버 신원 확인: 서버는 클라이언트로부터 받은 암호화된 세션 키를 복호화하여 세션 키를 얻습니다. 이를 통해 클라이언트와 서버는 상호 인증을 완료하고, 안전한 통신을 위한 세션 키를 얻게 됩니다.

완전성을 증명할 수 없기 때문에 변조 가능

완전성이란 정보의 정확성을 가리킨다.

`완전성을 증명할 수 없다는 것은 정보가 정확한지 아닌지를 확인할 수 없음을 의미`

1. 수신한 내용이 다를지도 모른다.

- HTTP가 완전성을 증명할 수 없다는 듯은 리퀘스트나 리스폰스가 발신된 후에 상대가 수신할 때까지의 사이에 `변조되었다고 하더라도 이 사실을 알 수 없다`는 뜻이다.
- 공격자가 도중에 리퀘스트나 리스폰스를 빼앗아 변조하는 공격을 중간자 공격(Man-in-the-middle)이라고 부른다.

2. 변조를 방지하려면 ?

- HTTP를 사용해서 완전성을 확인하기 위한 방법은 있지만, 확실하면서 편리한 방법은 아직 존재하지 않는다.
- 그 중 자주 사용되고 있는 방법은 `MD5나 SHA-1 등의 해시 값을 확인하는 방법과 파일의 디지털 서명을 확인하는 방법`이다.
- 파일 다운로드 서비스를 제공하고 있는 웹 사이트에서는 PGP(Pretty Good Privacy)에 의한 서명과 MD5에 의한 해시 값을 제공하는 일이 있다.
    - PGP는 파일을 작성했다는 증명을 위한 서명
    - MD5는 단방향성 함수에 의한 해시 값
- 어느 쪽을 사용하더라도 클라이언트를 이용하고 있는 유저 자신이 다운로드 받은 파일을 토대로 검사할 필요가 있다.
- 브라우저에서 자동적으로 검사가 진행되는 것은 아니다.
- 그러나 이 방법도 확실히 확인할 수 있는 방법이 아니다.
- `확실히 방지하기에는 HTTPS를 사용할 필요`가 있다.
- SSL에는 `인증이나 암호화, 그리고 다이제스트 기능을 제공`하고 있다.
- HTTP만으로 완전성을 보증하는 것이 어렵기 때문에 다른 프로토콜을 조합함으로써 실현하고 있다.

그 외,

- 특정 웹 서버나 특정 웹 클라이언트의 구현상의 약점(취약성 또는 시큐리티 홀)

- Java나 PHP 등으로 구축한 웹 어플리케이션 취약성 등

> 💡 위 약점은 HTTP만이 아닌, 다른 암호화하지 않은 프로토콜에도 공통되는 문제이다.

👉 HTTP + 암호화 + 인증 + 완전성 보호 = HTTPS

HTTP에 암호화와 인증과 완전성 보호를 더한 HTTPS

`HTTP 통신은 암호화되지 않은 평문으로 실시`된다.

- ex) 웹페이지에서 신용 카드 번호를 입력했을 때 통신이 도청되면 신용 카드 번호를 도청당하게 된다.

HTTP 통신 상대의 서버나 클라이언트를 인증하는 수단이 없다.

메시지가 도중에 변조되어 있을 가능성도 있다.

→ 이러한 문제를 해결하기 위해 암호화와 인증과 완전성 보호 같은 구조를 HTTP에 추가할 필요가 있다.

→ HTTP에 암호화나 인증 등의 구조를 더한 것을 HTTPS(HTTP Secure)라고 부른다.

HTTPS를 사용하는 통신

- 웹 페이지의 로그인
- 쇼핑의 결제 화면

HTTPS는 SSL의 껍질을 덮어쓴 HTTP

`HTTPS는 새로운 애플리케이션 계층의 프로토콜이 아니다.`

HTTP 통신을 하는 소켓 부분을 SSL이나 TLS라는 프로토콜로 대체

보통 `HTTP는 TCP와 직접 통신`

SSL을 사용한 경우, HTTP는 SSL와 통신하고 `SSL이 TCP와 통신`

SSL을 사용함으로써 `암호화, 증명서, 완전성 보호`를 이용할 수 있다.

상호간에 키를 교환하는 공개키 암호화 방식

SSL에서는 `공개 키 암호화 방식`이라 불리는 암호화 방식을 채용하고 있다.

암호는 알고리즘이 공개되어 있고 키를 비밀에 부침으로써 안전성을 유지하여 암호화나 복호화할 때 이 키를 사용

→ 단, 키가 노출되면 암호화가 의미를 잃어버리게 된다.

공통키 암호의 딜레마

공통키 암호 : 암호화와 복호화에 `하나의 키를 같이 사용`하는 방식

- 상대방에기 키를 넘겨 줄 때 안전하게 배송해야 한다.
- 받은 키를 안전하게 보관해야 한다.

두 개의 키를 사용하는 공개키 암호

공개키 암호 : `서로 다른 두 개의 키 페어(쌍)을 사용`

→ 공통키 암호의 문제를 해결하기 위해 나온 방식

→ 한 쪽은 비밀키(Private Key), 다른 한 쪽은 공개키(Public Key)라고 명칭

- 공개키 암호를 사용한 암호화는 `암호를 보내는 측이 상대의 공개키를 사용해 암호화`를 한다.
- 암호화된 정보를 받아들인 상대는 `자신의 비밀키를 사용해 복호화`를 실시
- 암호를 푸는 비밀키를 통신으로 보낼 필요가 없기 때문에 도청에 의해서 키를 빼앗길 걱정 X
- 암호문과 공개키라는 정보에서 평문을 구하는 것은 매우 어려운 수학적인 특징이 잇기 때문에 간단하지 않다.

HTTPS는 하이브리드 암호 시스템

HTTP는 공통키 암호와 공개키 암호의 양쪽 성질을 가진 하이브리드 암호 시스템이다.

`공개키` 암호는 `공통키` 암호에 비해서 `처리 속도가 늦다.`

위의 단점을 보안하기 위해

- 키를 교환하는 곳에서는 `공개키` 암호를 사용

- 그 후의 통신에서 메시지를 교환하는 곳에서는 `공통키` 암호를 사용

공개키가 정확한지 아닌지를 증명하는 증명서

공개키 암호의 문제점

- 공개키가 진짜인지 아닌지를 증명할 수 없다.

위 문제를 해결하는 데는 `인증 기관(CA)`과 그 기관이 발행하는 `공개키 증명서`가 이용되고 있다.

- 인증 기관이란 클라리언트와 서버가 모두 신뢰하는 제 3자 기관
  - ex) VeriSign 등
  -

공개키 증명서 인증 단계

- 서버의 운영자가 `인증 기관에 공개키를 제출`
- 인증 기관은 제출된 `공개키에 디지털 서명`하고 `서명이 끝난 공개키를 생성`
- 공개키 인증서에 서명이 끝난 공개키를 담는다.
- 서버는 이 인증 기관에 의해서 작성된 공개키 인증서를 `클라이언트에 보내고 공개키 암호로 통신`
  - 공개키 인증서는 디지털 증명서나 주령서 증명서라고 명칭
- 증명서를 받은 클라이언트는 `증명 기관의 공개키를 사용`해서 서버의 공개키를 인증한 것이 진짜 인증 기관이라는 것과 서버의 공개키가 신뢰할 수 있다는 것을 확인

> 사용되는 인증 기관의 공개키는 안전하게 클라이언트에 전달되어야 한다.
> 통신중에는 어떤 방법을 사용하더라도 안전하게 전달하는 것은 어렵기 때문에 많은 브라우저가 `주요 인증 기관의 공개키를 사전에 내장한 상태로 제품을 출시`

조직의 실제성을 증명하는 EV SSL 증명서

증명서의 역할

- 서버가 올바른 통신 상대임을 증명하는 것
- 상대방이 실제로 있는 기업인지를 확인하는 역할

→ 역할을 가진 증명서를 `EV SSL 증명서`라고 명칭

EV SSL 증명서는 `세계 표준의 인정 가이드라인에 의해서 발행`되는 증명서 → 신뢰성이 높다.

> 브라우저의 주소창의 색이 녹색으로 변하면 EV SSL 증명서로 증명된 웹 사이트이다.
> 주소창의 옆에는 SSL 증명서에 기재되어 있는 조직명 및 증명서를 발행한 인증 기관 명이 표시

클라이언트를 확인하는 클라이언트 증명서

HTTPS에서는 클라이언트 증명서도 이용할 수 있다.

서버가 통신하고 있는 상대가 의도한 클라이언트인 것을 증명하는 `클라이언트 인증`이 가능하다.

클라이언트 증명서 문제점

- 증명서의 입수와 배포 → 유저가 클라이언트 증명서를 인스톨 해야 한다. (유료)

안전성이 매우 높은 인증 기능을 제공할 수 있지만 특정 용도로만 사용되는 실정

- ex) 은행 인터넷 뱅킹

자기 인증 기관 발행 증명서는 '나야 나' 증명서

`OpenSSL` 등의 소프트웨어를 사용하면 `누구든지 인증 기관을 구축하여 서버 증명서를 발행`할 수 있다.

그러나, 이 서버 증명서는 인터넷 상에서는 증명서로서 구실을 하지 못하며 `쓸모가 없다.`

독자적으로 구축한 인증 기관을 `자기 인증 기관`이라 명칭

→ 위 증명서를 비하하는 말이 '나야 나 증명서'

브라우저에 액세스 하면 아래와 같은 경고 메세지가 표시된다.

- [접속의 안전성을 확인할 수 없습니다.]
- [이 사이트의 보안 증명서에는 문제가 있습니다.]

`위장의 가능성을 불식할 수 없기 때문에` 쓸모가 없다.

안전한 통신을 하는 HTTPS의 구조

서버 측의 공개키 증명서(서버 증명서)만을 이용한 HTTPS에 의한 통신을 시작하여 HTTP에 의한 통신을 개시하는 곳까지의 통신의 수순

1. 클라이언트가 Client Hello 메시지를 송신하면서 SSL 통신을 시작
    a. 메시지에는 클라이언트가 제공하는 SSL의 버전을 지정하고,

    b. 암호 스위트(Cipher Suite)로 불리는 리스트(사용하는 암호화의 알고리즘이나 키 사이즈 등) 등이 포함
2. 서버가 SSL 통신이 가능한 경우에는 Server Hello 메시지로 응답
    a. 클라이언트와 같이 SSL 버전과 암호 스위트를 포함
    b. 서버의 암호 스위트 내용은 클라이언트에서 받은 암호 스위트 내용에서 선택된 것
3. 서버가 Certificate 메시지를 송신
    a. 메시지에는 공개키 증명서가 포함
4. 서버가 Server Hello Done 메시지를 송신하여 최초의 SSL 네고시에이션 부분이 끝났음을 통지
5. SSL의 최초 네고시에이션이 종료되면 클라이언트가 Client Key Exchange 메시지로 응답
    a. 메시지에는 통신을 암호화하는데 사용하는 Pre-Master Secret이 포함
    b. 3. 의 공개키 증명서에서 꺼낸 공개키로 암호화
6. 클라이언트는 Change Cipher Spec 메시지를 송신
    a. 이 메시지는 `이 메시지 이후의 통신은 암호키를 사용해서 진행한다는 것을 나타낸다.`
7. 클라이언트는 Finished 메시지를 송신
    a. 이 메시지는 접속 전체의 체크 값을 포함
    b. 네고시에이션이 성공했는지 어떤지는 서버가 이 메시지를 올바르게 복호화 할 수 있는지 아닌지가 결정
8. 서버에서도 마찬가지로 Change Cipher Spec 메시지를 송신
9. 서버에서도 마찬가지로 Finished 메시지를 송신
10. 서버와 클라이언트의 Finished 메시지 교환이 완료되면 SSL에 의해서 접속은 확립
    a. 통신은 SSL에 의해서 보호되고 있다.
    b. 이제부터는 애플리케이션 계층의 프로토콜에 의해 통신을 진행 → HTTP 리퀘스트를 송신
11. 애플리케이션 계층의 프로토콜에 의한 통신 → HTTP 리스폰스를 송신
12. 마지막에 클라이언트가 접속을 끊는다.
    a. 접속을 끊을 경우에는 close_notify 메시지를 송신 후 TCP FIN 메시지를 보내 TCP 통신을 종료

> 애플리케이션 계층의 데이터를 송신할 때에는 MAC(Message Authentication Code)라고 부르는 메시지 다이제스트를 덧붙힐 수 있다.
> MAC를 이용해서 변조를 감지할 수 있어서 `완전성 보호를 실현` 가능

SSL과 TLS

- SSL은 본래 브라우저 개발 회사였던 넷스케이프 커뮤니케이션스사가 내놓은 프로토콜
- `TLS는 SSL을 바탕으로 한 프로토콜`이지만, 이 프로토콜을 `총칭해서 SSL`이라고 부르기도 함

SSL은 느리다 ?

`SSL를 사용하면 처리가 늦어지게 된다.`

지연되는 이유 두 가지

1. 통신 속도가 떨어지는 것
    - 네트워크의 부하는 HTTP를 사용하는 경우에 비해 2배에서 100배 정도 지연될 수 있다.
    - TCP 접속과 HTTP의 리퀘스트/리스폰스 이외에 `SSL에 필요한 통신이 추가되기 때문에` 전체적으로 처리해야 할 통신이 증가
2. CPU나 메모리 등의 리소스를 다량으로 소비함으로써 처리가 느려지는 것
    - SSL은 반드시 암호화 처리를 하고 있기 때문에 서버나 클라이언트에서는 `암호화나 복호화를 위한 계산을 진행`
    - 서버나 클라이언트의 리소스를 소비하게 되어 HTTP에 비해 부담이 증가

> → SSL 엑셀레이터라는 하드웨어(appliance 서버)를 사용해서 이 문제를 해결
> **SSL 엑셀레이터란 ?**
> - SSL을 처리하기 위한 전용 하드웨어
> - 소프트웨어로 SSL을 처리할 때 보다 몇 배 빠른 계산이 가능
> - SSL 처리만 SSL 엑셀레이터에 맡김으로써 부하를 분산
