TLS Protocol 구현 방식

작성자: 박지혜
소속: DevUnion
제출일: 2025 년 8 월 3 일

목차

---

1. 서론

인터넷 통신의 기밀성과 보안을 확보하는 것은 정보화 사회에서 필수적인 요소가 되었다. 그중에서도 TLS(Transport Layer Security) 프로토콜은 웹을 비롯한 다양한 네트워크 환경에서 가장 널리 사용되는 암호화 통신 프로토콜이다. 본 보고서는 2018 년 van der Merwe 의 박사 논문을 중심으로 TLS 의 구조와 보안적 한계, 그리고 프라이버시 개선을 위한 연구 내용을 살펴본다. 또한 구현을 시도하였으나 환경적 제약으로 인해 실패한 경험을 간단히 언급하며, 실제 적용의 어려움도 함께 반영하고자 한다.

---

2. TLS 프로토콜 개요

TLS 는 애플리케이션 계층과 전송 계층 사이에서 동작하는 보안 프로토콜로, 주로 다음과 같은 기능을 제공한다.

- **기밀성 (Confidentiality):** 데이터 암호화를 통해 중간자 공격(MITM)을 방지
- **무결성 (Integrity):** 메시지 변경 탐지를 위한 MAC(Message Authentication Code) 사용
- **인증 (Authentication):** 공개키 기반 인증서를 통해 서버 및 클라이언트 신원 확인

현재는 TLS 1.3(RFC 8446)이 최신 표준으로 채택되고 있으며, 이전 버전들에서 발생한 다양한 보안 취약점을 개선하였다.

---

3. 연구자가 지적한 TLS 의 한계 (van der Merwe, 2018)

3.1 메타데이터 노출

TLS 는 패킷의 본문을 암호화하지만, 헤더 및 일부 핸드셰이크 데이터는 여전히 평문으로 전송된다. 특히 Server Name Indication(SNI)은 접속하고자 하는 도메인 이름이 그대로 노출되어, 사용자의 접속 목적지가 외부에 드러나는 문제가 발생한다.

3.2 트래픽 분석 위험

암호화된 트래픽이라 해도 패킷의 크기, 타이밍, 빈도 등을 분석함으로써 사용자의 활동을 유추할 수 있다. 이는 **트래픽 분석 공격(Traffic Analysis Attack)**으로, 공격자는 암호화된 데이터를 해독하지 않고도 행동 패턴을 파악할 수 있다.

3.3 중앙 집중형 인증 구조(PKI)의 신뢰성

TLS 의 인증 체계는 공인 인증 기관(CA)에 의존하는 구조다. 이로 인해 하나의 CA 가 공격을 받거나 내부적으로 부정행위를 저지르면 전체 TLS 생태계가 위협받을 수 있다. 실제로 2011 년

DigiNotar 사건에서는 공격자가 유효한 구글 인증서를 발급받아 사용자의 로그인 정보를 탈취했다.

---

4. 연구자의 개선 제안

4.1 Encrypted SNI (ESNI) 및 ECH

메타데이터 보호를 위해 SNI 를 암호화하는 기술이 제안되었으며, 이후 ECH(Encrypted ClientHello)로 발전되었다. 이를 통해 접속하려는 서버 정보조차도 외부에서 확인할 수 없도록 한다.

4.2 트래픽 패턴 숨기기

패킷 크기 무작위화, 더미 패킷 삽입 등으로 통신 패턴을 숨기는 기술이 제안되었다. 이러한 접근은 VPN 이나 Tor 등 익명 네트워크와 유사한 원리로, 공격자가 의미 있는 분석을 어렵게 만든다.

4.3 인증 인프라 개선

PKI 의 대안으로 블록체인 기반 분산 인증, 인증 투명성(Certificate Transparency) 로그 기록 등 탈중앙화된 보안 인프라가 제시되고 있다.

---

5. 구현 시도 및 한계

5.1 구현 방법 (시도 계획)

본인은 TLS 프로토콜의 실제 동작 원리를 실습적으로 확인하기 위해 Ubuntu 기반 가상 머신(VM)을 두 개 구성하였다. 이 두 VM 은 각각 클라이언트와 서버 역할을 담당하며, TLS 통신을 OpenSSL 도구를 사용하여 구현하고자 했다.

🖥️ **구성 환경**

- **VM1 (클라이언트)**: Ubuntu + OpenSSL
- **VM2 (서버)**: Ubuntu + OpenSSL
- **네트워크 설정**: NAT 또는 호스트 전용 어댑터를 이용한 VM 간 직접 통신

🛠️ **구현 목표**

- 자체 서명 인증서(Self-signed certificate)를 통한 TLS 서버 구축
- 클라이언트가 서버에 TLS 연결 요청하여 암호화된 세션 수립
- Wireshark 를 이용한 패킷 캡처 및 핸드셰이크 분석

🔧 **기본 명령어 (예시)**

서버에서:

```
bash
openssl req -x509 -newkey rsa:2048 -keyout key.pem -out cert.pem -days 365 -nodes
openssl s_server -key key.pem -cert cert.pem -port 4433
```

클라이언트에서:

```
bash
openssl s_client -connect [서버 IP]:4433
```

🧪 **의도한 결과**

- TLS 핸드셰이크 로그 확인 (ClientHello, ServerHello 등)
- 암호화된 애플리케이션 데이터 전송 시 Wireshark 에서 확인
- 인증서 관련 경고 메시지 분석

5.2 구현 중 발생한 문제 및 한계

실제 환경에서 구현을 진행하던 중 다음과 같은 문제들이 발생하였다:

- 가상머신(UTM) 리눅스 Ubuntu 연결중 오류로 제대로 된 실행 실패
- OpenSSL 라이브러리 버전 충돌로 s_server 실행 실패
- 클라이언트 연결 시 인증서 불일치로 인한 경고 및 연결 거부
- Wireshark 에서 암호화 이후 데이터 내용 해석 불가 및 Wireshark 실행 불가
- TLS 1.3 사용 시 OpenSSL 설정이 복잡하여 설정 파일 추가 필요

이러한 문제들로 인해 실제 구현 결과를 얻는 데에는 실패하였지만, 실습 과정을 통해 TLS 가 단순한 암호화 이상으로, 복잡한 설정과 환경이 맞물리는 **다층적인 보안 기술**임을 깊이 이해할 수 있었다. 이는 본 보고서가 이론에만 국한되지 않고 실무적 관점을 반영할 수 있도록 하는 데 큰 도움이 되었다.

---

6. 결론

TLS 는 현대 인터넷 통신의 핵심 보안 프로토콜로, 암호화와 인증 기능을 통해 데이터 보호를 실현한다. 그러나 여전히 메타데이터 노출, 트래픽 분석 가능성, 중앙화된 인증 구조 등 다양한 한계를 지닌다. van der Merwe 의 연구는 이와 같은 문제점을 개선하고자 실제적인 제안을 제시하였으며, 이는 TLS 의 진화 및 차세대 보안 설계에 많은 시사점을 제공한다.

본 보고서는 이론 중심으로 구성되었지만, 향후에는 실제 구현과 실험을 통해 이론적 논의가 실무에 어떻게 반영되는지도 심층적으로 탐구해보고자 한다.

---

7. 참고 문헌

1. van der Merwe, T. (2018). *Improving the Privacy of Transport Layer Security*. PhD Dissertation.
2. RFC 8446 – The Transport Layer Security (TLS) Protocol Version 1.3.
3. Cloudflare. (2020). *Encrypted Client Hello*.
4. Mozilla Security Blog. (2018). *Improving TLS Security*.
5. EFF. (2011). *What is SNI and why does it matter?*

---

8. 구현 방법

TLS 구현 방법

본인은 Mac 환경에서 **UTM 가상머신**을 활용하여 TLS 통신을 구현하고자 하였다. 총 2 개의 가상 머신(Ubuntu)을 설치하여 클라이언트와 서버로 구성하였고, OpenSSL 을 이용해 TLS 통신을 실습하려 했다.

1.가상 머신 환경 구성

- **호스트 OS**: macOS
- **가상화 도구**: UTM
- **게스트 OS**: Ubuntu 22.04 LTS (x86_64)
- **네트워크 설정**: 양쪽 VM 모두 같은 내부 네트워크에 연결 (Bridge 또는 Shared Network 사용)

2.OpenSSL 설치 (각 VM 내부에서)

```
bash
sudo apt update
sudo apt install openssl libssl-dev
```

3.TLS 서버 설정 (서버 역할 VM 에서)

1. **자체 서명 인증서 생성**

```
bash
openssl req -x509 -newkey rsa:2048 -keyout key.pem -out cert.pem -days 365 -nodes
```

2. **OpenSSL 서버 실행**

```
bash
openssl s_server -key key.pem -cert cert.pem -port 4433
```

4.TLS 클라이언트 접속 (클라이언트 역할 VM 에서)

```
bash
openssl s_client -connect 〔서버 VM IP〕:4433
```

- 연결 성공 시 TLS 핸드셰이크 로그와 서버 인증서 정보가 출력됨
- 인증서 검증 실패 메시지가 뜰 수 있으나 이는 테스트 환경에선 무시 가능

5.트래픽 분석 (선택사항)

- UTM 에서 브리지 네트워크로 설정하면 **Wireshark** 로 TLS 핸드셰이크 패킷 분석 가능