HTTP, SSH, SMTP, DNS

SSH

SSH(Secure Shell) 시큐어 셸

기존 Telnet, Rlogin, RSH를 대체하기 위해 만들어졌으며 네트워크 상의 다른 PC나 서버에 로그인, 원격 명령 실행, 파일 전송을 수행할 수 있는 프로토콜.

기존의 원격 접속 프로토콜인 Telnet, Rlogin, RSH는 데이터 전송 시 평문으로 전송되기 때문에 스니핑을 통해 데이터가 노출되기 쉬운 문제점을 갖고 있었다.

SSH 프로토콜은 안전한 원격 접속과 보호되지 않은 네트워크에서 안전한 네트워크 서비스를 제공하기 위해 암호화를 사용한다. 암호화를 통해 호스트(클라이언트)와 원격지(서버)간의 연결(사용자 인증, 명령, 파일 전송)을 네트워크 공격으로부터 보호할 수 있다.

# SSH 프로토콜의 일반적인 사용

원격 접속 사용자 및 자동화 프로세스의 접근 시

자동화된 파일 전송 시

원격 명령 실행 시

네트워크 인프라와 중요 시스템 관리 시

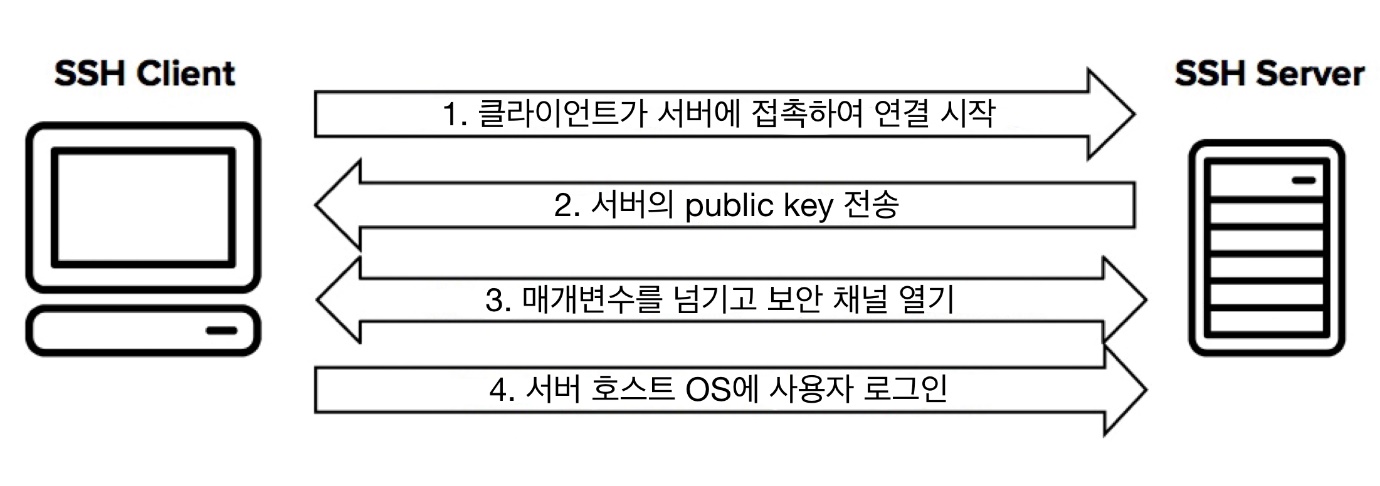
# SSH 작동 원리

SSH 프로토콜은 클라이언트-서버 모델로 동작하며 대칭키 방식, 비대칭키 방식, 해시 알고리즘을 사용하여 인증 및 암호화를 수행한다.

대칭키 방식은 클라이언트-서버 간 전체 연결을 암호화에 사용

비대칭키 방식은 키 교환, 클라이언트 인증, 서버 인증에 사용

해시 알고리즘은 패킷의 무결성을 확인하기 위해 사용



1. 클라이언트는 서버에 원격 접속하기 위해 연결을 설정하는 프로세스를 시작

SSH 프로토콜은 기본적으로 TCP 22번 포트를 사용하여 통신-> 클라이언트가 서버에 원격 접속하기 위해 서버의 TCP 22번 포트로 SSH 접속 요청을 보냄 -> 서버는 클라이언트에게 서버가 지원하는 프로토콜의 버전을 응답 -> 클라이언트는 서버가 지원하는 프로토콜의 버전 중 자신과 일치하는 것이 있다면 연결을 지속

2. 서버는 자신의 공개키를 클라이언트에게 전송

서버는 클라이언트로부터 SSH 접속 요청을 받고 자신의 공개키를 클라이언트에게 전송하고 클라이언트는 서버로부터 받은 공개키를 로컬에 저장-> 클라이언트는 원격 접속하는 서버들의 공개키를 로컬 사용자 홈 디렉터리의 .ssh 경로 내의 known\_hosts 파일에 저장하고 있다

3. 클라이언트와 서버는 여러 Parameter들을 주고 받으며 보안 채널을 확립

3.1 올바른 서버인지 확인 (클라이언트 관점)

클라이언트는 SSH로 원격 접속하려고 하는 서버가 올바른 서버인지 확인

이를 위해 클라이언트는 known\_hosts 파일에 존재하는 서버의 공개키를 통해 정상적인 서버인지 확인하는 작업을 수행한다.

1. 클라이언트에서 난수 생성, 난수 해시값 생성 및 저장
2. 난수를 서버의 공개키로 암호화 후 서버에 전송
3. 서버에서 서버의 개인키로 데이터를 복호화하여 난수 추출
4. 서버에서 복호화된 난수 해시값을 생성 후 클라이언트에게 전송
5. 클라이언트에 저장된 난수 해시값과 서버에서 받은 난수 해시값을 비교
6. 동일할 시 올바른 서버 확인

3.2 암호화된 통신을 위한 세션키 생성 (대칭키 생성)

세션키는 대칭키로 전체 세션을 암호화하는데 사용되며 모든 통신을 암호화한다.

대칭키는 비대칭키에 비해 빠르고 컴퓨팅 파워가 더 적게 든다는 장점을 가지고 있다. 그러나 대칭키가 유출되었을 경우 공격자가 암호화된 모든 통신을 복호화할 수 있는 치명적인 문제점을 가지고 있다. 이를 해결하기 위해 클라이언트와 서버는 키 교환 알고리즘을 통해 안전하게 대칭키를 공유

3.3 서버에 접근할 수 있는 클라이언트인지 확인 (서버 관점)

서버 또한 자신에게 접속하려는 클라이언트가 자신에게 접근할 수 있는 권한이 있는지 확인하는 단계가 필요

가장 간단한 방법으로 패스워드 인증

서버는 단순히 로그인하려는 계정의 암호를 묻고 클라이언트가 입력한 비밀번호는 세션키를 통해 암호화되고 전송되어 외부로부터 안전하게 보호됨, 패스워드가 암호화되지만 패스워드의 복잡성 설정의 한계가 있기 때문에 일반적으로 이 방법을 사용하지 않는 것이 좋다. 자동화된 스크립트를 통해 일반적인 길이의 패스워드는 공격에 의해 해제될 수 있다.

가장 많이 사용되고 권장되는 방법은 SSH 키 쌍을 사용하는 것

이 방법을 사용하기 위해서는 클라이언트 측에서도 SSH 키 쌍을 생성해야 한다. SSH 키 쌍을 통한 클라이언트 인증은 앞서 살펴본 올바른 서버인지 확인하는 과정과 비슷하다.

1. 클라이언트는 인증할 키 쌍의 ID를 서버에 전송
2. 서버는 클라이언트가 접속하고자 하는 계정의 .ssh/authorized\_keys 파일을 확인
3. ID에 매칭되는 공개키가 있을 시, 서버는 난수를 생성하고 클라이언트의 공개키로 암호화
4. 서버는 클라이언트에게 암호화된 메시지 전송
5. 클라이언트의 개인키를 통해 암호화된 메시지를 복호화하여 난수 추출
6. 클라이언트는 난수를 세션키와 결합하여 해시값 계산 후 서버 전송
7. 서버는 저장된 난수와 세션키를 결합하여 해시값 계산 후 비교
8. 일치할 시 클라이언트 인증

4. 클라이언트가 서버에 원격 접속을 할 수 있다.

이제 세션키를 통해 클라이언트와 서버는 안전한 네트워크 통신을 수행할 수 있다.

# 비대칭키 방식

공개키로 암호화한 내용은 공개키로 복호화할 수 없고 개인키로 복호화 가능

개인키로 암호화한 내용은 개인키로 복호화할 수 없고 공개키로 복호화 가능

# 정리

기본적으로 SSH는 한 쌍의 Key를 통해 접속하려는 컴퓨터와 인증 과정을 거치게 됩니다.

Private Key, Public Key

먼저 Public Key는 단어 뜻 그대로 공개되어도 비교적 안전한 Key

이 Public Key를 통해 메시지를 전송하기 전 암호화를 하게 됩니다. Public Key로는 암호화는 가능하지만 복호화는 불가능합니다.

그리고 이와 쌍을 이루는 Private Key는 절대로 외부에 노출이 되어서는 안되는 Key로 본인의 컴퓨터 내부에 저장하게 되어있습니다. 이 Private Key를 통해 암호화된 메시지를 복호화 할 수 있습니다.

이러한 Private Key와 Public Key를 통해 다른 컴퓨터와 통신을 하기 위해서는 먼저 Public Key를 통신하고하는 컴퓨터에 복사하여 저장 -> 요청을 보내는 클라이언트 사이드 컴퓨터에서 접속 요청을 할 때 응답을 하는 서버 사이드 컴퓨터에 복사되어 저장된 Public Key와 클라이언트 사이드에 해당 Public Key와 쌍을 이루는 Private Key와 비교를 하여 서로 한 쌍의 Key인지 아닌지를 검사 -> 이렇게 서로 관계를 맺고 있는 Key라는 것이 증명이 되면 비로소 두 컴퓨터 사이에 암호화된 채널이 형성이 되어 Key를 활용해 메시지를 암호화하고 복호화하며 데이터를 주고 받을 수 있게 됩니다.

ssh출처 :

<http://www.ktword.co.kr/test/view/view.php?m_temp1=2524>

<https://leejiwoo2002.tistory.com/21>

<https://velog.io/@ragnarok_code/SSH-%EB%8F%99%EC%9E%91%EC%9B%90%EB%A6%AC-%EC%9D%B4%ED%95%B4%ED%95%98%EA%B8%B0>

<https://baked-corn.tistory.com/52>

**- SMTP**

SMTP는 Simple Mail Transfer Protocol

메일 전송 프로토콜, 인터넷에서 이메일을 보내기 위해 이용되는 TCP/IP 프로토콜

사용하는 TCP 포트번호는 25..

이메일과 SMTP 는 관련 있는 개념이지만 동일하지는 않습니다. 이메일은 편지에 비유할 수 있고, SMTP는 편지를 전송하는 과정에 사용되는 기술을 의미.

# SMTP 서버

SMTP 프로토콜에 따라서 이메일 전달 과정을 처리해 주는 ‘이메일 서버’가 SMTP 서버

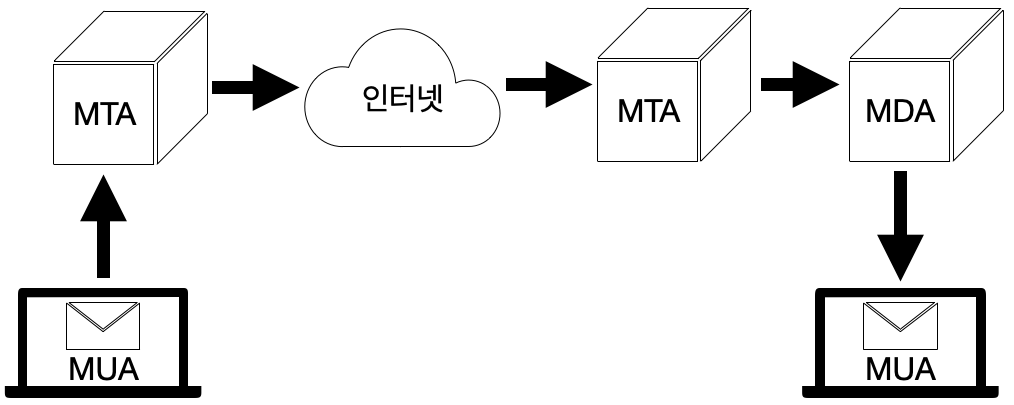
. SMTP 서버는 이메일 클라이언트가 보낸 메일을 수신하여 다른 메일 서버로 전달해 주는 역할을 합니다. 쉽게 설명한다면 우체국과 같은 역할

이를테면, 전라도 광주에 사는 김영희씨가 서울에 사는 박철수씨에게 편지를 보내기 위해서는 중개자인 우체국을 이용합니다. 그러면 우체국에서 편지를 받고, 분류하고, 서울에 사는 박철수씨의 집 앞 우편함까지 배달 해 주는 것입니다.

# SMTP의 동작 원리

SMTP의 동작 원리를 이해하려면 먼저 메일 전송에 관여하는 주요 구성 요소를 알아야 합니다. 이메일 시스템에는 크게 세 가지 구성 요소가 있습니다.

* 메일 사용자 에이전트(MUA, Mail User Agent): 사용자의 이메일 클라이언트로, 메시지를 작성하고 받은 메일을 읽을 수 있습니다.
* 메일 전송 에이전트(MTA, Mail Transfer Agent): 메시지를 수신하고 전달하는 서버로, SMTP를 사용해 메시지를 전송합니다.
* 메일 전달 에이전트(MDA, Mail Delivery Agent): 메시지를 최종 수신자의 메일박스에 배달하는 서버입니다.



1. 사용자가 MUA를 통해 이메일을 작성하고 발송을 요청합니다.

2. MUA는 이메일을 발신자의 MTA로 전송합니다.

3. 발신자의 MTA는 DNS를 사용하여 수신자의 메일 서버의 주소를 확인합니다.

4. 발신자의 MTA는 SMTP를 사용하여 수신자의 MTA로 이메일을 전달합니다.

5. 수신자의 MTA는 이메일을 MDA로 전달합니다.

6. MDA는 이메일을 수신자의 메일박스에 배달합니다.

7. 수신자가 자신의 MUA를 통해 메일박스를 확인하고 이메일을 읽습니다.

**# SMTP 인증과 보안**

SMTP 자체에는 별도의 인증 기능이 포함되어 있지 않지만, 이메일 전송 중 스팸 및 악성 소프트웨어로부터의 보호를 위해 추가적인 인증 및 보안 기능이 도입

- SMTP 인증(SMTP Auth): 이메일 발송을 위해 사용자가 자신의 계정 정보를 제공해야 하는 기능으로, 이메일 서비스 제공자는 사용자를 인증하여 스팸 전송을 방지합니다.

- SSL/TLS: 전송 계층 보안(TLS) 또는 그 이전 버전인 보안 소켓 계층(SSL)을 사용하여 이메일 전송 도중 발생할 수 있는 정보 유출 위험을 최소화합니다. 일반적으로 SMTPS(SMTP over SSL)는 465번 포트를 사용하며, STARTTLS는 587번 포트를 사용합니다.

**# SMTP 사용 포트**  
예전에는 SMTP에 주로 25번 포트를 사용

최근에는 보안상의 이유로 587번 포트를 사용하는 경우가 많습니다.

- 25번: SMTP 기본 포트로, 아직도 많이 사용되고 있지만 봇넷 스팸 공격에 악용되는 경우가 많습니다. 그래서 25번 포트 사용을 점차 차단하는 추세입니다.

- 587번: 현재 25번 포트 대용으로 사용하는 포트입니다. TLS 연결을 지원하므로 이메일을 안전하게 보낼 수 있습니다.

- 465번: 한때 이메일 시스템에 사용되었지만 현재는 SMTP 시스템에 사용하지 않습니다. 하지만 레거시 시스템(예전에 만든 노후화된 시스템)에서는 찾아볼 수 있습니다.

- 2525번: 공식적인 SMTP 포트는 아니지만, 587번 포트를 사용할 수 없을 때 대신 사용할 수 있습니다.

**# SMTP와 IMAP, POP의 차이점**

메일 시스템에는 2가지 종류의 프로토콜이 필요 그 중 하나는 여태까지 공부한 SMTP입니다. 하지만 SMTP만으로 최종 수신자가 이메일에 접근까지는 할 수 없고, 도착한 이메일을 받아 오기 위해서 IMAP이나 POP이라는 프로토콜을 이용해야 합니다. 하지만 최종 수신자가 우편함에 접근해서 편지를 찾아야만 내용을 볼 수 있겠죠? 그 과정을 하는 프로토콜이 바로 IMAP이나 POP라고 하는 것입니다.

IMAP과 POP는 둘 다 메일을 이메일 클라이언트까지 끌어오는 역할을 하는데, 서로 기능상 약간의 차이점이 존재

- SMTP: 메일을 보내기 위한 목적으로 사용됩니다. 상대 이메일 메일서버까지 이메일이 잘 전송되도록 푸쉬하는 역할입니다.

- IMAP: 도착한 이메일을 이메일 클라이언트로 끌어오는 역할입니다. 메일 서버에 접속해서 수신자의 이메일 클라이언트(아웃룩, 브라우저 등)로 이메일을 받아온 후 정리합니다. 이메일을 받아온 이후에도 메일 서버에서 원본을 삭제하지 않습니다.

- POP: 앞에서 말한 IMAP과 같이 이메일 클라이언트로 메일을 받아오는 역할이지만, 이메일을 받아온 후 메일 서버에 있는 내용을 삭제합니다. 그러므로 이메일 클라이언트에서 메일을 삭제하였다면 나중에 서버에서 되찾기가 힘듭니다.

출처

<https://nordvpn.com/ko/blog/smtp-protocol/>

<https://ccusean.tistory.com/entry/SMTP-%EA%B8%B0%EB%B3%B8-%EA%B0%9C%EB%85%90%EA%B3%BC-%EB%8F%99%EC%9E%91-%EC%9B%90%EB%A6%AC-%EC%9D%B4%EB%A9%94%EC%9D%BC-%EC%A0%84%EC%86%A1%EC%9D%98-%ED%95%B5%EC%8B%AC-%ED%94%84%EB%A1%9C%ED%86%A0%EC%BD%9C-%EC%95%8C%EC%95%84%EB%B3%B4%EA%B8%B0>

- **DNS 프로토콜**

1. DNS (Domain Name System)

1) DNS 등장 배경

인터넷 표준 프로토콜은 TCP/IP이다.

TCP/IP 프로토콜을 사용하는 네트워크 안에서 Host들을 식별하기 위한 목적으로 IP 주소를 사용한다. 사람의 경우 숫자보다 문자를 사용하는 것이 더 편하기 때문에 도메인 이름을 사용하여 Host들을 식별한다.

도메인 이름을 사용하는 경우에도 최종적으로 IP주소를 알고 있어야 상대방 장비와 연결이 가능하다. 네트워크에서 도메인이나 호스트 이름을 숫자로 된 IP 주소로 해석해 주는 TCP/IP Network Service인 DNS가 등장하였다.

2) DNS 포트 번호

UDP와 TCP 포트 53번을 사용

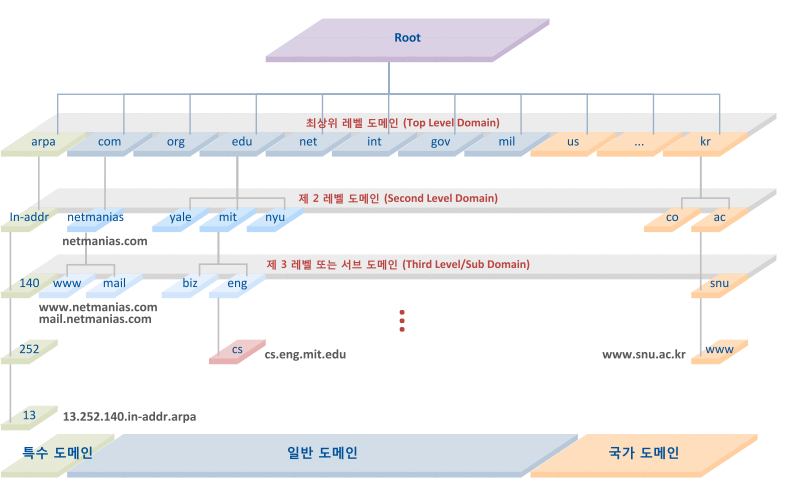
UDP: 일반적인 DNS 조회를 할 경우

TCP: Zone Transfer(영역 전송)을 수행할 경우 또는 512Byte를 초과하는 DNS패킷을 전송해야 할 경우

DNS의 구성 요소

1) 도메인 네임 스페이스 (Domain Name Space)

DNS가 저장,관리하는 계층적 구조를 의미



최상위에 루트 DNS 서버가 존재하고, 그 하위로 인터넷에 연결된 모든 노드(네모 표시)가 연속해서 이어진계층 구조로 구성되어 있다. PC에서 사용하는 디렉토리 구조와 유사함을 알 수 있는데, 각 레벨(Top level, Second level 등)의 도메인은 그 하위 도메인 에 관한 정보를 관리하는 구조이다. (계층적 구조)

2) 네임 서버 (Name Server)

문자열로 표현된 도메인 이름을 실제 컴퓨터가 통신할 때 사용하는 숫자로 표현된 IP 주소로 변환시켜 주기 위해서는 도메인 네임 스페이스의 트리 구조 에 대한 정보가 필요하며, 이러한 정보를 가지고 있는 서버를 네임 서버라고 한다. 도메인 이름을 IP 주소로 변환하는 것을 네임 서비스라고 한다.

리졸버(Resolver)로부터 요청 받은 도메인 이름에 대한 IP 정보를 다시 리졸버로 전달해주는 역할을 수행한다.

- Master Name Server (Primary Name Server)

해당 도메인을 관리하는 주 네임 서버이다.

Zone 파일을 관리하는 네임서버이다.

- Slave Name Server (Secondary Name Server)

master 네임 서버 의 고장 등의 이유로 동작하지 못하는 경우 이를 대신하여 네임 서버 역할을 수행하는 서버이고 주기적으로 master 네임 서버로부터 Zone Transfer를 통해 자신의 정보를 갱신하여 전체 네임 서버의 정보가 일관성 있게 유지 및 관리된다.

Authoriactive Name Server의 경우 부하분산, 가용성을 위해 보통 2대 이상으로 운영한다.

**3) 리졸버 (Resolver)**

웹 브라우저와 같은 DNS 클라이언트의 요청을 네임 서버로 전달하고 네임 서버로부터 정보(도메인 이름과 IP 주소)를 받아 클라이언트에게 제공하는 기능을 수행한다.

하나의 네임 서버 에게 DNS 요청을 전달하고 해당 서버에 정보가 없으면 다른 네임 서버에게 요청을 보내 정보를 받아 온다.

수많은 네임 서버에 접근하여 사용자로부터 요청 받은 도메인의 IP 정보를 조회하는 기능을 수행할 수 있어야 한다.

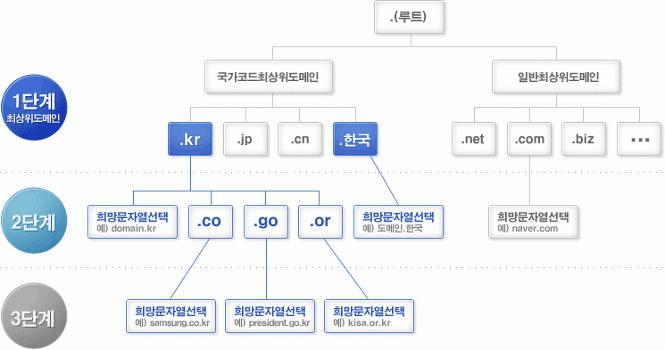
**4) 스터브 리졸버(Stub Resolver)**

리졸버의 모든 기능을 PC와 같은 클라이언트 호스트에 구현하는 것은 단말 시스템 자원의 한 계와 같은 제약이 있다.

리졸버의 대부분의 기능을 DNS 서버에 구현하고, 클라이언트 호스트에는 리졸버의 단순한 기능만을 지닌 리졸버 루틴을 구현한것이다.

스터브 리졸버는 수 많은 네임 서버의 구조를 파악할 필요 없이 리졸버가 구현된 네임 서버의 IP 주소만 파악하면 된다.

도메인에 대한 질의를 받은 스터브 리졸버는 설정된 네임 서버로 DNS 질의를 전달하고 네임 서버로부터 최종 결과를 응답 받아 웹 브라우저로 전달하는 인터페이스 기능만을 수행한다.

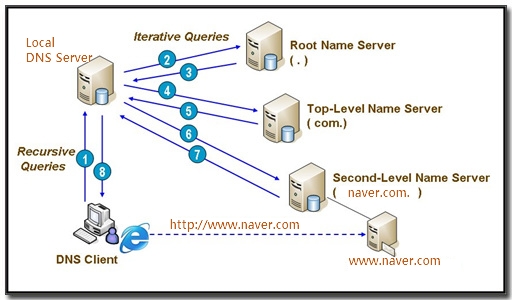


1) ROOT DNS 서버

2) 최상위 도메인 DNS 서버

3) 책임 DNS 서버

#동작 과정



1) DNS Client(웹 브라우저등) 로컬 DNS에게 www.naver.com을 질의한다

2) 로컬 DNS가 Root DNS에게 www.naver.com을 묻는다

3) Root DNS가 com을 인식하고 com을 관리하는 최상위 도메인 DNS 서버의 IP를 로컬 DNS에게 알려준다

4) 다시 로컬 DNS가 com도메인을 관리하는 최상위 도메인 DNS서버에게 www.naver.com을 질의한다

5) 최상위 도메인 DNS서버가 naver.com을 인식하고 naver.com을 관리하는     책임DNS 서버의 IP를 로컬 DNS에게 알려준다

6) 로컬 DNS는 책임 DNS서버에게 www.naver.com을 질의하고

7) 책임 DNS서버에서 www.naver.com 호스트에 대한 IP를 알려준다

8) 로컬 DNS가 DNS Client에게 www.naver.com의 IP를 알려준다

출처:

<https://galid1.tistory.com/53>

<https://peemangit.tistory.com/52>

<https://rooftoproom-whale.tistory.com/36>

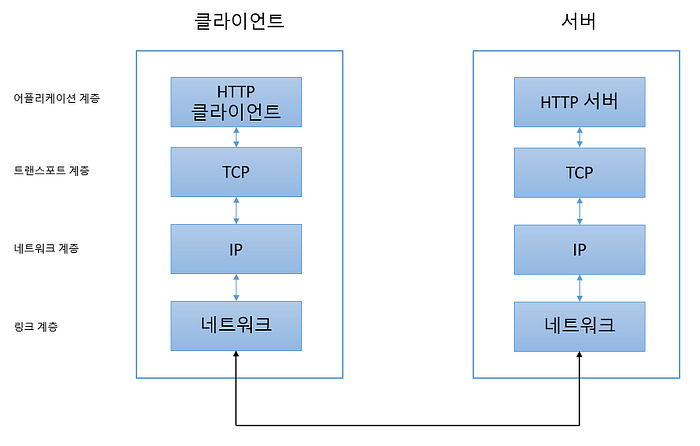
- HTTP ( Hyper Text Transfer Protocol)

HTTP 개요

HTTP는 HTML 문서와 같은 리소스들을 가져올 수 있도록 해주는 프로토콜.

서버와 클라이언트간의 TCP/IP 통신 위에서 메시지를 교환하기 위해 사용되는 프로토콜

HTTP는 웹에서 이루어지는 모든 데이터 교환의 기초이며, 클라이언트-서버 프로토콜이기도 합니다. 클라이언트-서버 프로토콜이란 (보통 웹브라우저인) 수신자 측에 의해 요청이 초기화되는 프로토콜을 의미합니다. 하나의 완전한 문서는 텍스트, 레이아웃 설명, 이미지, 비디오, 스크립트 등 불러온(fetched) 하위 문서들로 재구성됩니다.



HTTP 프로토콜에서는 서버와 클라이언트 간의 통신

웹브라우저의 통신 과정

클라이언트에서 요청(request)를 보내면 서버는 요청을 처리해서 응답(response)한다.

URL 분석 및 접속

웹 브라우저는 URL을 분석해 서버의 IP 주소와 포트(기본은 80)를 이용해 서버와 TCP/IP 연결을 요청합니다.

Request 헤더 전송

브라우저에서 요청 파일명 등이 기술된 헤더를 전송합니다.

Request 바디 전송

필요한 경우에, 로그인 폼에 입력한 데이터나 첨부 파일 등의 추가적인 데이터를 전송합니다.

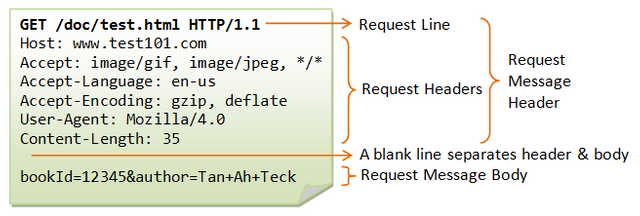
Response 헤더 해석

서버에서 헤더를 수신하고 응답 상태(404 등)를 확인하며, 바디의 Content-Type 등을 확인합니다.

Response 바디 해석

바디가 있는 경우에, 서버에서 수신한 바디를 헤더에 기술된 Content-Type에 따라서, text/html인 경우에 HTML을 렌더링하고, image/jpeg인 경우에는 그림을 띄우는 등 적절히 해석합니다.

**# Request(요청):Client to Server**



요청은 메서드, URI, 프로토콜 버전, 헤더, 바디로 구성

웹 브라우저는 웹 서버에 데이터를 "요청"하는 "클라이언트 프로그램" 이다. 요청은 서버가 인식할 수 있는 약속된 형식 (HTTP 형식)을 따라야 한다. 요청 데이터는 "HEADER"와 "BODY"로 구성된다.

필수 요소로 요청의 제일 처음에 와야 하는 3개의 필드가 있다.

요청 메서드 : GET, PUT, POST, PUSH, OPTIONS 등의 요청 방식이 온다.

요청 URI : 요청하는 자원의 위치를 명시한다.

HTTP 프로토콜 버전 : 웹 브라우저가 사용하는 프로토콜 버전이다.

그 외의 키값

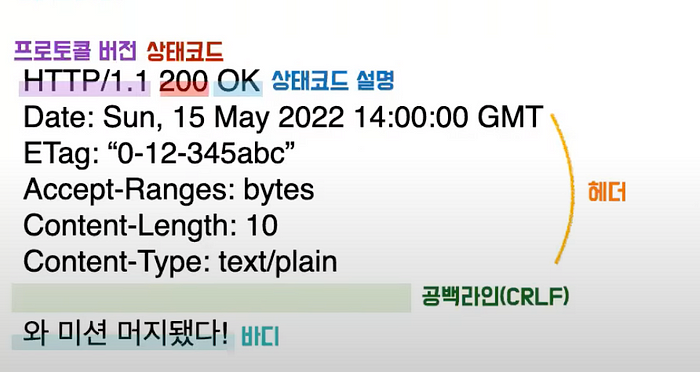
Host : 요청을 보내는 Host (예) www.google.co.kr

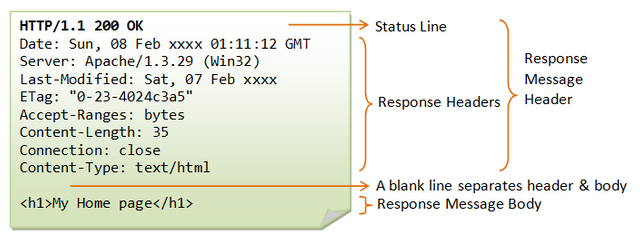
Content-Type : 요청에 바디가 있는 경우 그 파일 포맷 (예) Content-Type: application/json

Cookie : 웹 브라우저에 저장된 쿠키들

User-Agent : 클라이언트의 정보, 이를 통해 사용하는 브라우저 감지

**# Response ( 응답 메세지 ) : Server to Client**





응답은 프로토콜 버전, 상태코드, 상태코드 설명, 헤더, 바디로 구성되어 있습니다.

HTTP 프로토콜은 기본적으로 stateless 특성을 가지고 있습니다.

프로토콜과 응답코드 : 웹 브라우저가 사용하는 프로토콜, 서버의 응답 상태 (1xx~5xx), 응답 메시지를 보여준다

Set-Cookie : 웹 브라우저에게 쿠키 생성을 요청

(예) Set-Cookie: UserID=tester; Max-Age=3600; Version=1

Content-Type : 응답에 바디가 있는 경우 그 포맷

(예) Content-Type: text/html; charset=utf-8

**# Connectless & Stateless**

HTTP는 Connectless 방식으로 작동한다. 서버에 연결하고, 요청해서 응답을 받으면 연결을 끊어버린다. 기본적으로는 자원 하나에 대해서 하나의 연결을 만든다.

장점: 불특정 다수를 대상으로 하는 서비스에 적합한 방식이다. 수십만명이 웹 서비스를 사용하더라도 접속유지는 최소한으로 할 수 있기 때문에, 더 많은 유저의 요청을 처리할 수 있다.

단점: 연결을 끊어버리기 때문에, 클라이언트의 이전 상태를 알 수가 없다. 이러한 HTTP의 특징을 stateless라고 하는데, Connectless로 부터 파생되는 특징이라고 할 수 있다. 클라이언트의 이전 상태 정보를 알 수 없게 되면, 웹 서비스를 하는데 당장에 문제가 생긴다. 클라이언트가 과거에 로그인을 성공하더라도 로그 정보를 유지할 수가 없다. HTTP는 cookie를 이용해서 이 문제를 해결하고 있다.

**# HTTP Method**

메서드는 요청의 종류를 서버에게 알려주기 위해서 사용한다.

GET : 주어진 URL에서 자원을 요청

POST : 주어진 URL로 자원의 생성을 요청

PUT : 주어진 URL로 자원의 대체를 요청

DELETE : 주어진 URL로 자원의 삭제를 요청

HEAD : 주어진 URL에서 자원의 헤더만을 요청, 해당 자원이 존재하는지 혹은 서버에 문제가 없는지를 확인하기 위해서 사용한다.

OPTIONS : 주어진 URL에서 처리 가능한 메소드의 목록을 요청

**# HTTP Status Code**

**2xx (성공)**

200 (성공): 서버가 요청을 제대로 처리함

201 (생성됨): 요청이 성공했으며, 새로운 리소스가 생성됨

202 (허용됨): 요청을 받았으나 아직 처리하진 않음

204 (컨텐츠 없음): 요청을 처리했지만, 컨텐츠를 제공하지 않음

205 (컨텐츠 재설정): 요청을 처리했지만, 컨텐츠를 표시하지 않음. 그리고 문서를 재 설정할 것을 요구함

206 (일부 성공): 요청의 일부만 성공적으로 처리함

**4xx (요청 오류)**

400 (잘못된 요청): 서버가 요청의 구문을 인식하지 못함. 주로 헤더 포멧이 HTTP 규약에 맞지 않을 경우

401 (권한 없음): 인증을 필요로 하는 요청, 인증 실패

403 (Forbidden, 금지됨): 서버가 요청을 거부, 인가 실패

404 (Not Found, 찾을 수 없음): 서버가 요청한 리소스를 찾을 수 없음

405 (허용하지 않는 방법): 요청에 지정된 방법을 사용할 수 없음. 예를 들어 POST 방식으로 요청을 받는 서버에 GET 요청을 보내는 경우

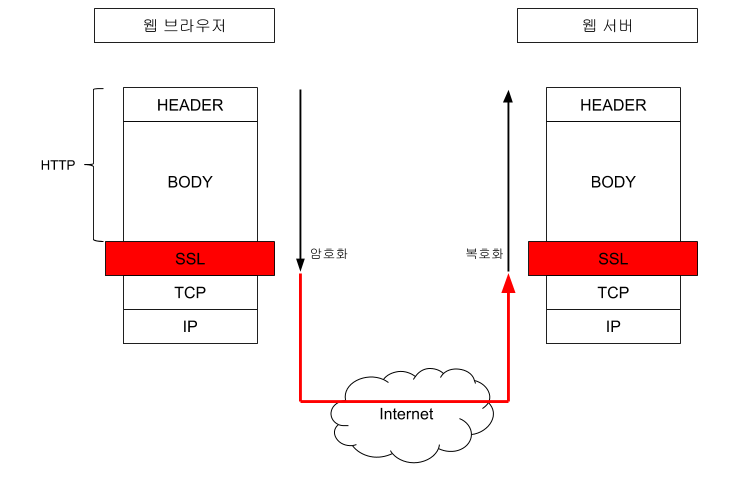
**5xx (서버 오류)**

500 (내부 서버 오류): 서버에 오류가 발생하여 요청을 수행할 수 없음

**HTTPS**

SSL은 전자상거래에서의 데이터 보안을 위해서 개발한 통신 레이어다. SSL은 표현계층의 프로토콜로 응용 계층 아래에 있기 때문에, 어떤 응용 계층의 데이터라도 암호화해서 보낼 수 있다.

HTTP는 기본적으로 평문 데이터 전송을 원칙으로 하기 때문에 개인의 프라이버시가 오가는 서비스들 (전자상거래, 전자메일, 사내문서)에 사용하기 힘들다. HTTPS는 SSL 레이어위에 HTTP를 통과 시키는 방식이다. 즉 평문의 HTTP 문서는 SSL 레이어를 통과하면서 암호화 돼서 목적지에 도착하고, 목적지에서는 SSL 레이어를 통과하면서 복호화 돼서 웹 브라우저에 전달된다.



HTTP와 다른점

HTTPS URL은 "https://" 로 시작한다. 기본 포트번호는 443이다. HTTP URL은 "http://" 로 시작한다. 기본 포트번호는 80.

HTTP는 평문 데이터를 기반으로 하기 때문에, 유저정보와 같은 민감한 정보가 인터넷 상에 그대로 노출된다. 이 정보는 수집되거나 변조될 수 있다. HTTPS는 이러한 공격을 견딜 수 있도록 설계돼 있다. HTTPS는 인증서를 이용해서, 접속 사이트를 신뢰할 수 있는지 평가할 수 있다.

일반적으로 HTTPS는 HTTP에 비해서 (매우 많이)느리다. 많은 양의 데이터를 처리할 경우 성능의 차이를 체감할 수 있다. 많은 웹 사이트들이 민감한 정보를 다루는 페이지 (로그인 혹은 유저정보) 페이지를 HTTPS로 전송하고, 기타 페이지는 HTTP로 전송하는 방법을 사용한다.

출처 :

<https://callmedevmomo.medium.com/http-%ED%94%84%EB%A1%9C%ED%86%A0%EC%BD%9C-%EC%95%8C%EA%B3%A0-%EC%93%B0%EC%9E%90-c0c8d14f18ed>

<https://velog.io/@dnjscksdn98/HTTP-%ED%94%84%EB%A1%9C%ED%86%A0%EC%BD%9C%EC%97%90-%EB%8C%80%ED%95%98%EC%97%AC>

<https://developer.mozilla.org/ko/docs/Web/HTTP/Overview>