Relatório

February 15, 2025

1 Introdução

A comunicação em redes de computadores é baseada em uma série de protocolos bem estabelecidos e conhecidos por cada nó da rede para que seja possível haver trocas de mensagens e informações entre esses diferentes nós. Com isso, na internet um dos principais protocolos de comunicação é o HyperText Transfer Protocol (HTTP) sendo hoje em dia a base de qualquer comunicação web. Porém o HTTP não surgiu com o intuito de prover segurança em suas comunicações e com o avançar da internet rapidamente percebeu-se a necessidade de criptografar essas comunicações HTTP. Assim, surgiu-se a ideia do HTTPS, uma extensão do HTTP com o intuito de fornecer segurança às comunicações. Primeiramente, o protocolo utilizado para fornecer essa segurança foi o SSL, mas devido algumas vulnerabilidades, hoje o mais recomendado é o HTTPS over TLS, outro protocolo que surgiu da evolução do SSL.

Este relatório tem como objetivo explorar os principais protocolos envolvidos na comunicação segura na web, abordando o funcionamento do HTTP, HTTPS, SSL e TLS. Além disso, será feita uma breve análise das versões de cada um desses protocolos, destacando também a transição do SSL para o TLS.

Por fim, ainda, esse relatório consta com uma parte prática, onde será implementado uma rede de comunicação do tipo cliente servidor utilizando o protocolo HTTPS over TLS. Esse experimento consiste em uma simulação de comunicação entre alguns clientes e um servidor, os clientes farão requisições sobre o protocolo HTTPS para o servidor solicitando alguns componentes HTMLs, também será feita uma pequena análise dessas comunicações, bem como a geração do certificado e chave privada do servidor. Vale ressaltar que esse experimento será implementado utilizando o módulo do python SSL para implementação do HTTPS over TLS, do módulo socket para comunicação na rede local e do openssl para criação do certificado digital e chave privada do servidor.

2 HyperText Transfer Protocol

O HyperText Transfer Protocol (HTTP) é um protocolo de comunicação da camada de aplicação que surgiu em 1991 com a crescente necessidade de transmissão de hiper texto de uma forma eficiente e com boa alta abstração. É amplamente utilizado no contexto web por navegadores e servidores que, ao interagirem, torna possível a troca de informações na web. Além da transmissão de hiper texto, rapidamente o protocolo evoluiu para transmissão de documentos, imagens, arquivos e outros tipos de dados por meio de uma estrutura simples baseada em requisições e respostas.

Porém os dados trafegados no protocolo HTTP são em formato plain text, assim, o protocolo não oferece nenhum grau de segurança por padrão, pois os dados trafegam de forma clara para qualquer usuário que tenha acesso a requisição. Dessa forma, com a crescente utilização da web para realizar

comunicações de grande importância, surgiu-se a necessidade de uma expansão do protocolo para comunicações que necessitem de segurança, o HTTPS.

Antes de partirmos para o HTTPS, primeiro vamos fazer uma breve análise das versões do https, iremos considerar apenas até o HTTP/2, já que sua versão 3 ainda não está totalmente suportada pela maioria das aplicações web.

2.1 Comparativo das Diferentes Versões

2.1.1 HTTP/0.9 (1991)

- Primeira versão do protocolo, extremamente simples.
- Suportava apenas o método `GET` e não possuía cabeçalhos nem suporte para envio de arquivos além de HTML.

2.1.2 HTTP/1.0 (1996)

- Introduziu cabeçalhos HTTP, permitindo informações adicionais nas requisições.
- Suporte para outros métodos além de `GET`, como `POST` e `HEAD`.
- Comunicação não persistente, ou seja, uma nova conexão TCP era aberta para cada requisição, tornando a comunicação lenta e ineficiente.

2.1.3 HTTP/1.1 (1997)

- Melhorias no desempenho com suporte a conexões persistentes (Keep-Alive).
- Introdução do pipelining, permitindo múltiplas requisições em uma única conexão TCP.
- Novos métodos adicionados, como `PUT`, `DELETE`, `OPTIONS`.
- Suporte a cache avançado e compressão de dados.
- Ainda é amplamente utilizado hoje em dia.

2.1.4 HTTP/2 (2015)

- Otimizações no desempenho, incluindo multiplexação, permitindo múltiplas requisições simultâneas em uma única conexão TCP.
- Redução da sobrecarga dos cabeçalhos HTTP, melhorando a eficiência da comunicação.
- Introdução do server push, permitindo que o servidor envie dados antecipadamente para o cliente sem esperar uma requisição explícita.
- Maior eficiência em conexões seguras (HTTPS é amplamente adotado junto ao HTTP/2).

3 Hyper Text Transfer Protocol Secure

Para a primeira implementação do Hyper Text Transfer Protocol Secure (HTTPS) foi utilizado o protocolo Secure Sockets Layer (SSL) o qual foi projetado pela Netscape Communications Corporation no início da década de 90 com o intuito de adicionar segurança a diversos protocolos, incluindo HTTP, SMTP e FTP. Assim, o SSH passou a permitir a transmissão criptografada de dados entre clientes e servidores, protegendo as comunicações de ataques como interceptações e manipulações de dados. Sua adoção foi ampla, sendo utilizado em bancos, lojas online e serviços de e-mail para garantir a segurança de informações sensíveis.

3.1 SSL

Agora partiremos para uma análise mais profunda do protocolo, analisando passo a passo como o SSL consegue transmitir privacidade, autenticação e integridade de dados em comunicações de internet. Na primeira tentativa de comunicação do cliente pelo servidor, o cliente e/ou servidor podem sinalizar que desejam utilizar o protocolo HTTPS, caso ambas aceitem, o protocolo do SSL começa a operar. O primeiro passo é chamado de handshake, o qual é uma etapa onde o servidor e o cliente estão acordando como a conexão irá acontecer. Por exemplo, o cliente transmite a lista de algoritmos de segurança que ele é capaz de operar e, assim, o servidor decide a lista de algoritmos que serão utilizados na comunicação.

Após essa etapa de acordo de algoritmos e funções, o servidor envia um certificado digital e sua chave pública, este certificado é previamente gerado por uma autoridade de certificação. Uma autoridade de certificação é uma instituição responsável por gerar certificados digitais para cada instituição que a solicite para tal, atuando como um terceiro confiável na comunicação entre dois agentes que ainda não possuem certeza de suas identidades.

O certificado digital é gerado pela AC a partir de informações de identificação do solicitante e uma chave pública que o solicitante queira cadastrar em seu certificado. Essa solicitação de certificado digital é um processo que é realizado em ambientes seguros e controlados, para se ter certeza que a pessoa que está se cadastrando realmente é quem diz ser. Assim, com informações de ID e a chave pública do solicitante a AC termina de gerar o certificado e por fim gera um hash desse certificado, o qual é assinado pela chave privada da AC e é concatenado ao certificado digital que deu origem ao hash. Vale-se ressaltar que essa concatenação do certificado com o hash assinado é armazenada pela AC a qual irá o disponibilizar para qualquer usuário que queira verificar qual é o certificado daquela instituição.

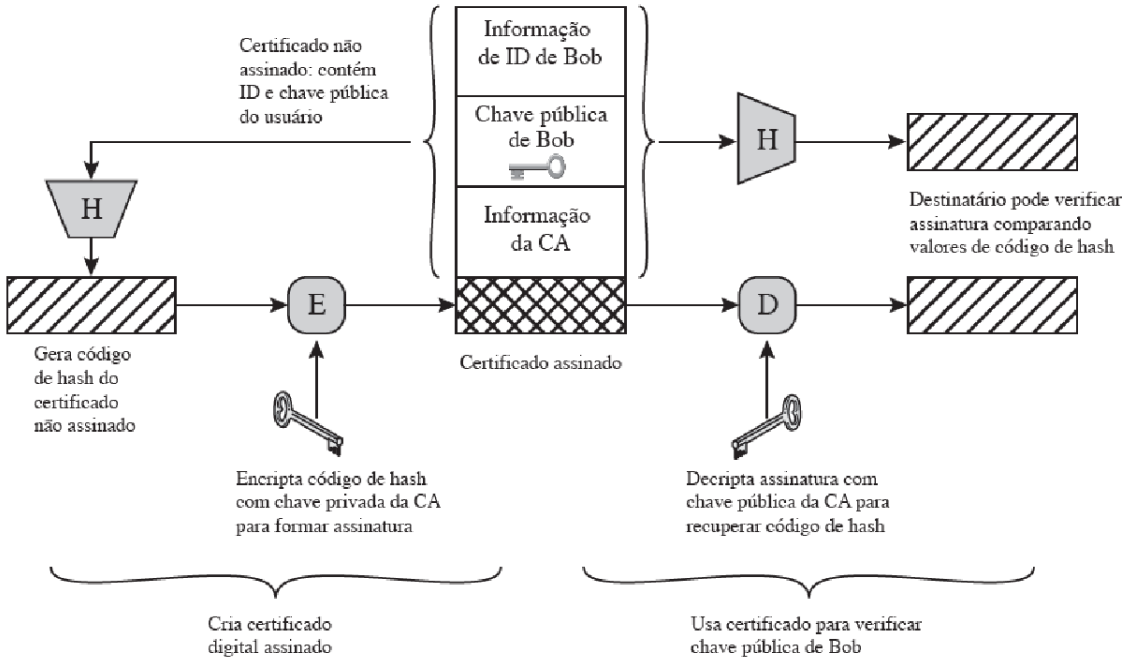
Com isso fica claro a estratégia do SSL. O que o SSL propoẽ é que quando um usuário receber o certificado digital de um emissor, ele irá fazer o hash deste certificado e irá solicitar o certificado dessa instituição na AC a qual o emissor diz ter gerado esse certificado (a AC é informada pelo servidor ao cliente durante a etapa de Handshake). Com o certificado recebido da AC, o cliente irá descriptografar o hash acoplado ao certificado, usando a chave pública da AC, caso o emissor diz ser quem é, esses dois hashs serão iguais garantindo a autenticidade do servidor, evitando-se assim, ataques de man in the middle.

Com a identidade confirmando, utilizando a chave pública fornecida pelo servidor, o cliente e o servidor vão traçar qual será a chave de criptografia simétrica utilizada para o restante da comunicação. Isso é feito ao cliente criptografar um número aleatório com a chave pública recebida do servidor e enviar o resultado ao servidor (o único capaz de descriptografar a mensagem com sua chave privada); ambas as partes então fazem uso do número aleatório para gerar uma chave de

sessão única para a criptografia subsequente dos dados durante a sessão.

Isso é feito, pois, normalmente, a criptografia simétrica é mais performática em relação a assimétrica. Para fins deste relatório, em relação a parte dois de experimentação, iremos sempre utilizar o RSA para nossa criptografia assimétrica e o AES para a criptografia simétrica. Assim, o processo de formação da chave simétrica é finalizado e inicia-se a conexão segura, que é criptografada e descriptografada com a chave de sessão até o fim da conexão. Se qualquer um dos passos acima falhar, o handshake também falha e a conexão não é criada.

Para tornar mais fácil a visuzalização do processo de validação do certificado digital, aqui está uma imageem didática que sintetiza o processo acima:



3.1.1 Vulnerabilidades do SSL

3.1.2 Análise de Versões

3.2 TLS

4 4. Explicação do TLS

4.1 4.1. O que é o TLS?

O **Transport Layer Security (TLS)** é um protocolo de segurança projetado para fornecer **criptografia, autenticação e integridade** na comunicação entre dispositivos conectados à internet. Ele é o sucessor do **Secure Sockets Layer (SSL)** e corrige diversas vulnerabilidades encontradas nas versões anteriores do SSL.

Atualmente, o **TLS é o padrão de segurança utilizado na web**, sendo essencial para **comunicações seguras em HTTPS**, conexões de email (SMTP, IMAP, POP3) e até mesmo em protocolos de comunicação em tempo real, como VoIP e VPNs.

4.2 4.2. Objetivos e Funcionalidades Gerais

O **TLS** tem três principais objetivos:

1. **Confidencialidade**
   - Garante que os dados transmitidos sejam protegidos contra interceptação e leitura por terceiros.

2. **Autenticação**
   - Verifica a identidade do servidor (e opcionalmente do cliente) por meio de certificados digitais.

3. **Integridade dos Dados**
   - Garante que os dados não sejam alterados durante a transmissão, utilizando assinaturas e funções de hash seguras.

4.2.1 Funcionalidades do TLS

- **Criptografia Forte** – Usa algoritmos modernos para cifrar os dados transmitidos.

- **Autenticação via Certificados Digitais** – Utiliza certificados emitidos por Autoridades Certificadoras (CAs).

- **Troca Segura de Chaves** – Usa **Diffie-Hellman** ou **Elliptic Curve Diffie-Hellman (ECDH)** para estabelecer uma chave compartilhada.

- **Verificação de Integridade** – Protege contra manipulação de dados com **HMAC (Hash-based Message Authentication Code)**.

- **Prevenção Contra Ataques Man-in-the-Middle (MITM)** – Evita que atacantes interceptem e modifiquem o tráfego.

4.3 4.3. Etapas de Segurança e Principais Algoritmos

O TLS usa um processo chamado **TLS Handshake** para estabelecer uma conexão segura entre cliente e servidor. Esse processo inclui **autenticação, troca de chaves e estabelecimento de uma sessão criptografada**.

4.3.1 4.3.1. TLS Handshake – Como o TLS Estabelece uma Conexão Segura

1. **Cliente Envia um "Hello"**
   - O cliente inicia a conexão enviando uma lista de versões do TLS e algoritmos suportados.
2. **Servidor Responde com um "Hello"**
   - O servidor escolhe a versão do TLS e os algoritmos a serem usados na conexão.

   - Envia seu **certificado digital**, que contém a chave pública e a identidade do servidor.
3. **Verificação do Certificado**
   - O cliente verifica a autenticidade do certificado com uma **Autoridade Certificadora (CA)**.

- Se o certificado for válido, a comunicação continua.

4. **Troca de Chaves Segura**
   - O cliente e o servidor negociam uma **chave compartilhada** para comunicação criptografada.
   - Métodos comuns incluem **Diffie-Hellman (DH)** e **Elliptic Curve Diffie-Hellman (ECDH)**.
5. **Criação da Sessão Segura**
   - O cliente e o servidor usam a chave compartilhada para **criptografar e autenticar** os dados transmitidos.

4.3.2 4.3.2. Principais Algoritmos Utilizados no TLS

Criptografia Assimétrica (Troca de Chaves)

- **RSA (Rivest-Shamir-Adleman)** – Utilizado para autenticação e troca de chaves.
- **Diffie-Hellman (DH)** – Algoritmo de troca de chaves segura sem a necessidade de criptografia prévia.
- **Elliptic Curve Diffie-Hellman (ECDH)** – Versão otimizada de DH, usada no TLS 1.2 e 1.3.

Criptografia Simétrica (Cifração de Dados)

- **AES (Advanced Encryption Standard)** – Algoritmo mais seguro e amplamente usado no TLS.
- **ChaCha20** – Alternativa ao AES, usada em dispositivos móveis para maior eficiência.

Funções de Hash (Integridade dos Dados)

- **SHA-256 (Secure Hash Algorithm 256-bit)** – Padrão atual para verificação de integridade.
- **HMAC (Hash-based Message Authentication Code)** – Método que combina hash e chave secreta para autenticação.

4.4 4.4. Resumo das Versões e Evolução do TLS

Versão	Ano	Características Principais
TLS 1.0	1999	Substituiu o SSL 3.0, mas herdou algumas vulnerabilidades.
TLS 1.1	2006	Melhorias na proteção contra ataques, mas ainda vulnerável.
TLS 1.2	2008	Adotado amplamente, suporta AES-GCM e SHA-256.
TLS 1.3	2018	Removidos algoritmos inseguros, handshake mais rápido e seguro.

4.4.1 4.4.1. TLS 1.0 e 1.1 – Obsoletos

- **TLS 1.0 (1999)** e **TLS 1.1 (2006)** foram **descontinuados em 2020** devido a vulnerabilidades.
- Principais falhas: uso de **MD5 e SHA-1**, algoritmos considerados inseguros.

4.4.2 4.4.2. TLS 1.2 – Padrão Atual

- **Lançado em 2008**, é amplamente utilizado até hoje.
- Introduziu o **AES-GCM**, um algoritmo eficiente para criptografia segura.
- Suporta **Elliptic Curve Diffie-Hellman (ECDH)** para troca de chaves mais rápida.
- Ainda compatível com métodos legados, mas deve ser atualizado para **TLS 1.3**.

4.4.3 4.4.3. TLS 1.3 – Mais Seguro e Rápido

- **Lançado em 2018**, é a versão mais segura e eficiente.
- **Handshake 40% mais rápido**, reduzindo a latência.
- Removeu **RSA key exchange**, preferindo **ECDHE** (Elliptic Curve Diffie-Hellman Ephemeral).
- Maior privacidade e resistência contra ataques de downgrade.
- Obrigatório para sites modernos e serviços de internet.