HTTPS

Lucas Alves Rodrigues 200022784

Jean Bueno Karia 21105529

GitHub: https://github.com/Jibabk/Trabalho2SEG

1 Introdução

Em um mundo conectado, proteger informações online é fundamental. Protocolos como o HTTPS garantem essa segurança, usando criptografia, autenticação e integridade de dados. Este trabalho explora como esse protocolo funciona, desde o handshake até os algoritmos usados para comunicações seguras.

2 SSL e TLS: Segurança na Comunicação Online

Os protocolos SSL (Secure Sockets Layer) e TLS (Transport Layer Security) são utilizados para proteger a comunicação na internet, garantindo confidencialidade, integridade e autenticidade dos dados transmitidos. Eles são a base do HTTPS, que criptografa a troca de informações entre o navegador do usuário e o servidor web.

2.1 Handshake: Estabelecendo a Conexão Segura

O processo começa com o handshake, em que o cliente e o servidor negociam a conexão segura:

- Troca de mensagens: O cliente envia uma mensagem "olá"para o servidor, informando a versão TLS e os conjuntos de cifras que ele suporta, junto com um "cliente aleatório"gerado de forma única.

- Resposta do Servidor: O servidor responde com sua própria mensagem "olá", incluindo seu certificado SSL (para autenticação), o conjunto de cifras escolhido, e um "servidor aleatório"gerado pelo servidor.

- Autenticação e Verificação: O cliente verifica o certificado SSL do servidor, confirmando sua identidade por meio da autoridade certificadora, garantindo que o servidor é legítimo.

- Geração do Segredo Pré-Mestre: O cliente gera um "segredo pré-mestre", criptografado com a chave pública do servidor (que ele obteve no certificado SSL) e o envia para o servidor.

- Descriptografia e Criação das Chaves de Sessão: O servidor usa sua chave privada para descriptografar o segredo pré-mestre e, com isso, tanto o cliente quanto o servidor geram as mesmas chaves de sessão, utilizando o "cliente aleatório", o "servidor aleatório"e o "segredo pré-mestre".

- Conclusão do Handshake: O cliente envia uma mensagem "finalizado"criptografada com a chave de sessão. O servidor responde com sua própria mensagem "finalizado"também criptografada, sinalizando que a comunicação segura foi estabelecida.

- Comunicação Segura: Com a troca de chaves de sessão, a comunicação agora é criptografada de forma simétrica, garantindo a segurança no restante da troca de dados.

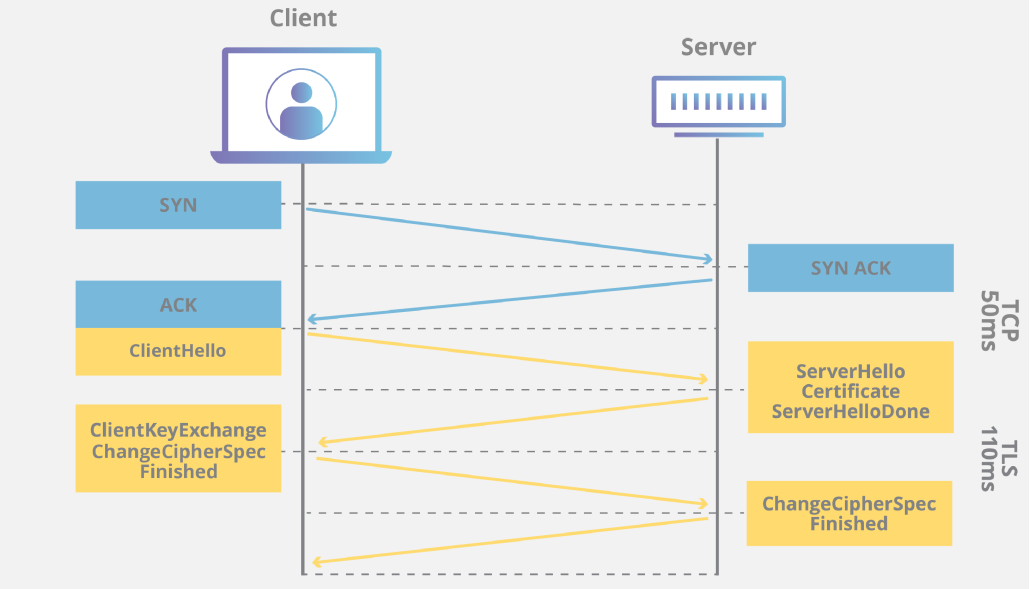

Figura 1: Etapas do handshake.

2.2 Criptografia e Integridade dos Dados

Após o handshake, os dados são protegidos de duas maneiras:

- Criptografia Simétrica: Usando a chave de sessão compartilhada, os dados são criptografados antes de serem enviados e descriptografados na recepção. Isso garante confidencialidade e alta performance.
- Integridade com HMAC: Cada mensagem inclui um código de autenticação de mensagem (HMAC) para garantir que os dados não foram alterados durante a transmissão. O HMAC é gerado usando a chave de sessão e um algoritmo de hash (como SHA-256).
- Suporte a algoritmos modernos como SHA-256 e HMAC na implementação com TLS.

2.3 Autenticidade com Certificados Digitais

Os certificados digitais garantem que o servidor é confiável e foi verificado por uma Autoridade Certificadora (CA). O certificado contém:

- Chave Pública: Usada para criptografar a chave de sessão durante o handshake.

- Assinatura Digital: Garantida pela CA, confirma a autenticidade do certificado. O cliente verifica a assinatura usando a chave pública da CA.

Figura 2: Detalhes do certificado de overleaf.com.

2.4 Fragilidade do SSL

Em 2014, foi descoberta uma grande falha no SSL 3.0, chamada POODLE. Essa falha permitia que hackers interceptassem e descriptografassem dados secretos trocados entre o usuário e o site, como senhas e cookies de sessão, mesmo com a criptografia ativada. Isso foi um grande problema porque muitos sites ainda usavam SSL 3.0, pensando que estava seguro. Após a descoberta, empresas como Google e Microsoft, detentoras de 81% do mercado de navegadores à época, desativaram o protocolo nos seus navegadores, forçando a migração para um protocolo mais seguro, o TLS.

2.5 Segurança Adicional no TLS 1.3

O TLS 1.3 introduziu melhorias significativas:

- Handshake Simplificado: Reduz o número de mensagens trocadas, melhorando a velocidade da conexão.
- 0-RTT (Zero Round Trip Time): Permite o envio de dados logo no início da conexão, reduzindo a latência em reconexões.
- Remoção de Algoritmos Inseguros: Abandonou o uso de RSA estático e algoritmos de hash fracos como MD5, utilizando apenas criptografia moderna como AES-GCM e ChaCha20-Poly1305.

3 HTTP

HTTP (HyperText Transfer Protocol) é o protocolo usado para a comunicação entre navegadores e servidores na web. Ele define como as mensagens são formatadas e transmitidas, além de como os navegadores respondem a comandos. No entanto, o HTTP não oferece criptografia, o que significa que as informações enviadas (como senhas e dados pessoais) podem ser interceptadas e lidas por terceiros.

4 TCP/IP e seu handshake

O HTTP funciona sobre o TCP/IP, um conjunto de protocolos que define como os dados são transmitidos na internet. O TCP (Transmission Control Protocol) é responsável por garantir que os pacotes de dados sejam entregues de forma confiável e na ordem correta. Para isso, ele utiliza um processo chamado 3-way handshake, que funciona assim:

- SYN: O cliente envia um pedido de conexão ao servidor.
- SYN-ACK: O servidor reconhece o pedido e responde.
- ACK: O cliente confirma a resposta, estabelecendo a conexão.

Esse handshake estabelece uma conexão confiável antes que os dados sejam transmitidos. No entanto, os dados enviados ainda estão em texto puro no HTTP, o que torna a comunicação vulnerável a ataques de interceptação.

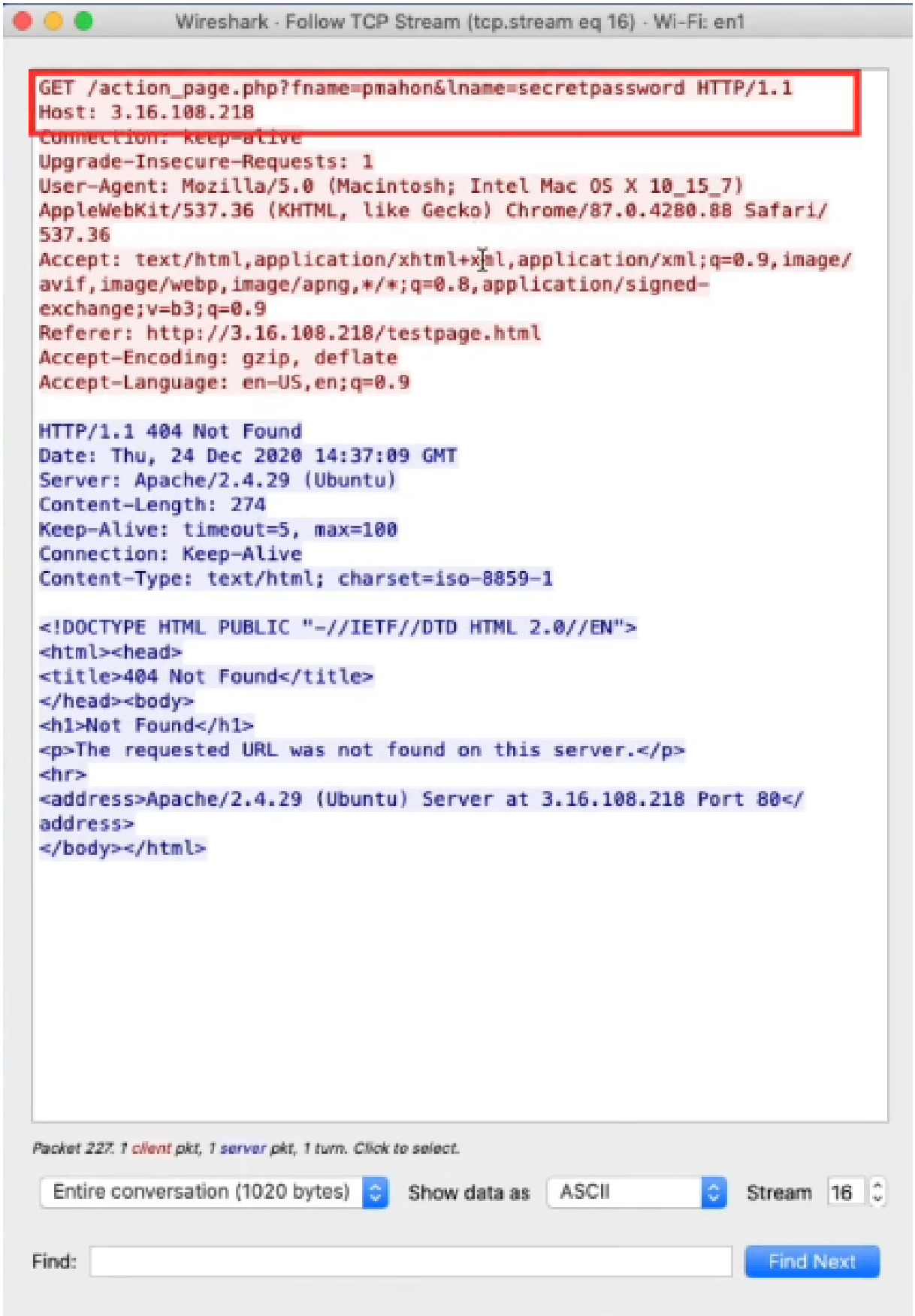

Figura 3: Exemplo de verificação de tráfico utilizando o Wireshark em um servidor HTTP.

5 HTTPS

HTTPS (HyperText Transfer Protocol Secure) é a versão segura do HTTP, o protocolo usado para acessar sites na internet. A diferença é que, com HTTPS, todas as informações trocadas entre o navegador e o servidor são criptografadas, garantindo privacidade e segurança.

5.1 Como Funciona?

- Criptografia: HTTPS usa SSL/TLS para criptografar os dados, impedindo que sejam lidos por terceiros. Isso protege informações como senhas e dados de pagamento.
- Certificado Digital: O site precisa de um certificado digital, emitido por uma Autoridade Certificadora (CA), que confirma sua identidade e permite a criptografia.
- Alerta na Barra de Endereço: O alerta na barra de endereços mostra de forma simples que o site não usa HTTPS, indicando que a conexão não é segura e os dados estão protegidos. Isso ajuda o usuário a identificar rapidamente sites confiáveis.

Figura 4: Alerta de segurança no Google Chrome.

5.2 Por que HTTPS é Importante?

- Segurança: Protege contra ataques de interceptação de dados (como o ataque "man-in-the-middle").
- Privacidade: Garante que ninguém possa espionar as informações trocadas entre você e o site.

- Confiança: Os usuários confiam mais em sites com HTTPS, pois sabem que suas informações estão protegidas.
- SEO (Posicionamento no Google): Em 2014, o Google anunciou que sites com HTTPS teriam um leve aumento no ranking de buscas, incentivando a adoção em massa.

Figura 5: Exemplo de verificação de tráfico utilizando o Wireshark em um servidor HTTPS.

6 Projeto de implementação

Como o objetivo de se verificar esses métodos estudados, foi realizado um projeto em Python3 implementando uma conexão HTTPS de mão dupla entre um cliente e o servidor.

6.1 Servidor

Primeiramente é definido uma porta e um host para definir a rota de conexão para esse servidor, no caso foi definido como localhost para o projeto ser executado localmente.

Em seguida, é inicializado um servidor HTTP fazendo uso da biblioteca http.server, tendo como rota o endereço previamente definido e uma página html simples como resposta do método GET

Com o servidor http já funcionando, agora é criado um contexto de criptografia para gerenciar os protocolos SSL/TLS. E "ssl.PROTOCOL_TLS_SERVER"indica que será utilizado um servidor TLS com a versão mais recente disponível. Em seguida, é carregado o certificado e a chave privada para o contexto de criptografia, para provar a identidade do servidor para os clientes. E para se realizar a conexão de mão dupla, requeremos que, para o acesso ao servidor, seja enviado um certificado do cliente. Por fim, aplicamos esse contexto ao socket do servidor HTTP, transformando-o em um socket seguro que utiliza TLS.

6.2 Cliente

Já do lado do cliente, as operações são mais simples. Primeiramente, é definida a URL do servidor alvo, e em seguida é enviada uma requisição ao servidor por meio da biblioteca requests, tendo como parâmetros a URL, o certificado composto pelo certificado assinado e a chave privada do cliente que comprove a posse do certificado, e a verificação da Autoridade Certificadora (CA) para validar o servidor. E o resultado será a resposta enviada pelo servidor, no caso uma página html.

6.3 Openssl

O OpenSSL é uma biblioteca de código aberto amplamente utilizada para implementar segurança em comunicações digitais por meio dos protocolos SSL (Secure Sockets Layer)

e TLS (Transport Layer Security). O OpenSSL trabalha com chaves criptográficas, certificados digitais e protocolos de segurança para proteger a comunicação entre sistemas.

E em nosso projeto A biblioteca OpenSSL foi utilizada para gerar e gerenciar certificados SSL/TLS, garantindo uma comunicação segura entre o cliente e o servidor. E através do terminal utilizamos os seguintes comandos:

6.3.1 Autoridade Certificadora

```
# Gerar uma nova chave privada da CA
openssl genrsa -out myCA.key 2048

# Gerar um certificado da CA
openssl req -x509 -new -nodes -key myCA.key -sha256 -days 365 -out myCA.pem
```

6.3.2 Certificado do Servidor

```
# Gerar uma nova chave privada para o servidor
openssl genrsa -out server.key 2048

# Criar uma nova solicitação de certificado
openssl req -new -key server.key -out server.csr -subj "/CN=localhost"

# Assinar o certificado com a CA
openssl x509 -req -in server.csr -CA myCA.pem -CAkey myCA.key -CAcreateserial -out
```

6.3.3 Certificado do Cliente

```
# Gerar uma nova chave privada para o cliente
openssl genrsa -out client.key 2048

# Criar uma nova solicitação de certificado
openssl req -new -key client.key -out client.csr -subj "/CN=ClienteHTTPS"

# Assinar o certificado com a CA
openssl x509 -req -in client.csr -CA myCA.pem -CAkey myCA.key -CAcreateserial -out
```

Listing 1: Código do servidor https

```
import http.server
import ssl

HOST = 'localhost'
PORT = 4443

class CustomHTTPRequestHandler(http.server.
    ↪ SimpleHTTPRequestHandler):
     def do_GET(self):
         if self.path == '/':
             self.path = 'index.html'
         return http.server.SimpleHTTPRequestHandler.do_GET(self)

server_address = (HOST, PORT)
httpd = http.server.HTTPServer(server_address,
    ↪ CustomHTTPRequestHandler)


context = ssl.SSLContext(ssl.PROTOCOL_TLS_SERVER)
context.load_cert_chain(certfile="certs/server.crt", keyfile="
    ↪ certs/server.key")

context.verify_mode = ssl.CERT_REQUIRED
context.load_verify_locations("certs/myCA.pem")

httpd.socket = context.wrap_socket(httpd.socket, server_side=True
    ↪ )
httpd.serve_forever()
```

Listing 2: Código do cliente https

```python
import requests # type: ignore

URL = "https://localhost:4443"

try:
    response = requests.get(URL,
                            cert=("certs/client.crt", "certs/
                                ↪ client.key"),
                            verify="certs/myCA.pem")
    print(f"Status Code: {response.status_code}")
    print(f"Resposta:\n{response.text}")
except requests.exceptions.RequestException as e:
    print(f"Erro na requisi  o: {e}")
```

7 Conclusão

Neste trabalho, exploramos o funcionamento do HTTPS e sua importância na segurança das comunicações online. Analisamos o papel do SSL e do TLS na proteção dos dados, detalhando o processo de handshake, a criptografia simétrica e a autenticação por certificados digitais. Também destacamos as melhorias do TLS 1.3 e a relevância do HTTPS para a privacidade e confiança dos usuários na internet. Compreender esses conceitos é essencial para garantir a segurança e a integridade das informações no ambiente digital.

Referências

- O que acontece em um handshake TLS? — Handshake SSL. Disponível em: ¡https://www.cloudf
  br/learning/ssl/what-happens-in-a-tls-handshake/¿. Acesso em: 14 fev. 2025.

- Pesquisa mostra que participação do Internet Explorer no mercado de navegadores
  cai - iMasters - We are Developers. Disponível em: ¡https://imasters.com.br/noticia/pesquisa-
  mostra-que-participacao-do-internet-explorer-no-mercado-de-navegadores-cai¿. Acesso

em: 14 fev. 2025.

- SAN. SSL e TLS: Saiba por que o SSL NÃO é mais usado — SAN. Disponível em: ¡https://blog.saninternet.com/diferenca-entre-ssl-e-tls¿. Acesso em: 14 fev. 2025.
- O que é HTTPS? Disponível em: ¡https://www.cloudflare.com/pt-br/learning/ssl/what-is-https/¿.