Documentação sobre Protocolos de Segurança: SSL, TLS e HTTPS

Seu Nome

February 4, 2025

1 Introdução

Os protocolos SSL (Secure Sockets Layer), TLS (Transport Layer Security) e HTTPS (HyperText Transfer Protocol Secure) são fundamentais para a segurança na comunicação digital. Eles garantem confidencialidade, integridade e autenticação dos dados transmitidos na internet, protegendo contra interceptações e ataques cibernéticos.

2 Versões dos Protocolos e Melhorias

2.1 SSL - Secure Sockets Layer

O SSL foi desenvolvido inicialmente pela Netscape nos anos 90 para garantir segurança nas comunicações pela internet. Ele passou por três versões principais antes de ser substituído pelo TLS:

- **SSL 1.0** - Nunca foi lançado publicamente devido a falhas de segurança graves.
- **SSL 2.0** - Lançado em 1995, apresentava melhorias, mas ainda vulnerável a vários ataques.
- **SSL 3.0** - Introduzido em 1996, corrigiu diversas falhas do SSL 2.0, mas ainda possuía vulnerabilidades como o ataque POODLE.

Devido às falhas de segurança do SSL, ele foi oficialmente descontinuado e substituído pelo TLS.

2.2 TLS - Transport Layer Security

O TLS foi desenvolvido como uma versão mais segura do SSL e passou por diversas atualizações:

- **TLS 1.0 (1999)** - Baseado no SSL 3.0, mas com melhorias na segurança e resistência a ataques.
- **TLS 1.1 (2006)** - Introduziu proteções contra ataques de criptografia, como CBC (Cipher Block Chaining).
- **TLS 1.2 (2008)** - Adicionou suporte para novos algoritmos de criptografia, como AES e SHA-2, tornando a comunicação mais segura.
- **TLS 1.3 (2018)** - Melhorou a eficiência da comunicação, reduzindo a complexidade do handshake e removendo algoritmos inseguros como RC4 e SHA-1.

O TLS 1.3 é atualmente a versão mais segura e recomendada para garantir uma comunicação protegida na internet.

2.3 HTTPS - HyperText Transfer Protocol Secure

O HTTPS é a implementação do HTTP sobre uma camada segura de TLS, garantindo que os dados transmitidos sejam protegidos contra ataques. Ele segue a evolução do TLS e atualmente suporta as versões mais seguras desse protocolo.

Os principais benefícios do HTTPS incluem:

- **Confidencialidade:** Os dados são criptografados, impedindo que terceiros os leiam.
- **Autenticação:** O uso de certificados digitais garante que o usuário está se comunicando com um servidor legítimo.
- **Integridade:** A tecnologia impede que os dados sejam alterados durante a transmissão.

O funcionamento do HTTPS ocorre da seguinte maneira:

1. O usuário acessa um site que utiliza HTTPS.
2. O servidor envia seu certificado digital, garantindo sua autenticidade.

3. O navegador verifica o certificado e estabelece uma conexão segura utilizando TLS.

4. Todos os dados transmitidos entre o usuário e o servidor são criptografados, evitando que sejam interceptados.

3 Etapas de Segurança e Algoritmos

3.1 Autenticação e Troca de Chaves

O processo de estabelecimento de uma conexão segura segue as seguintes etapas:

1. **Handshake SSL/TLS:**
   - Cliente e servidor negociam versões e algoritmos suportados.
   - O servidor envia seu certificado digital (emitido por uma Autoridade Certificadora).
   - O cliente verifica a validade do certificado e, caso válido, procede com a troca de chaves.
2. **Troca de Chaves:**
   - Algoritmos como RSA (Rivest-Shamir-Adleman) ou Diffie-Hellman são usados para gerar e trocar chaves seguras.
3. **Criptografia dos Dados:**
   - Os dados são protegidos com algoritmos simétricos como AES (Advanced Encryption Standard).
4. **Integridade dos Dados:**
   - Uso de funções hash (SHA-2, SHA-3) e códigos de autenticação de mensagem (HMAC) para verificar a integridade da comunicação.

4 Explicação do Código de Implementação em Python

O código a seguir demonstra a implementação de um servidor HTTPS e um cliente HTTPS em Python, utilizando a biblioteca *http.server* e *requests*.

O servidor é configurado para rodar localmente na porta 4443 e o cliente realiza uma requisição HTTPS para o servidor, verificando a autenticidade do certificado digital.

```
    import http.server
    import ssl
    import requests
    import socket
    from requests.exceptions import SSLError
    import threading
    import time
```

O código a seguir define um manipulador de requisições HTTPS que envia uma resposta simples ao cliente. Ele é responsável por tratar as requisições GET recebidas pelo servidor. A resposta enviada ao cliente é "Conexao segura estabelecida via HTTPS!".

```
    class HTTPSRequestHandler(http.server.
SimpleHTTPRequestHandler):
        def do_GET(self):
            self.send_response(200)
            self.send_header('Content-type', 'text/plain')
            self.end_headers()
            self.wfile.write(b'Conexao segura estabelecida
via HTTPS!')
```

O comando a seguir define a função *run_server()*, que configura o servidor HTTPS na porta 4443. O servidor é configurado para utilizar um certificado digital autoassinado (*server.pem*) e uma chave privada (*server.key*). O contexto SSL é definido com o certificado e a chave, e o modo de segurança é ativado para o servidor. Por fim, o servidor é iniciado e fica aguardando requisições.

```
    def run_server():
        # Configurando o servidor HTTP
        server_address = ('localhost', 4443)
        httpd = http.server.HTTPServer(server_address,
HTTPSRequestHandler)

        # Definindo o contexto SSL
        context = ssl.SSLContext(ssl.PROTOCOL_TLS_SERVER)
        context.load_cert_chain(certfile='server.pem',
keyfile='server.key')

        # Ativando o modo de seguranca
        httpd.socket = context.wrap_socket(httpd.socket,
server_side=True)
        print("Servidor HTTPS rodando em https://localhost
:4443")
        httpd.serve_forever()
```

O código a seguir define a função *run_client()*, que realiza uma requisição HTTPS para o servidor configurado. A URL do servidor é definida como "https://localhost:4443". A função *requests.get()* é utilizada para realizar a requisição, e o certificado do servidor (*server.pem*) é verificado. Caso o certificado seja inválido, uma exceção *SSLError* é lançada e a conexão é rejeitada.

```
    def run_client():
        # Configurando o cliente HTTPS
        url = "https://localhost:4443"
        try:
            #response = requests.get(url, verify=False)  # Desativa a verificacao SSL (somente para testes!)
            response = requests.get(url, verify="server.pem")
            print("Resposta do servidor:", response.text)
        # Tratamento de excecao para certificado invalido
        except SSLError:
            raise ValueError("Certificado invalido! Conexao rejeitada.")
            sys.exit(1)
```

O código a seguir inicia o servidor HTTPS em uma *thread* separada e aguarda 2 segundos para garantir que o servidor esteja rodando antes de iniciar o cliente. O servidor é iniciado chamando a função *run_server()* e o cliente é iniciado chamando a função *run_client()*.

```
    # Criando um certificado autoassinado
    server_thread = threading.Thread(target=run_server)
    server_thread.daemon = True
    server_thread.start()
    time.sleep(2)  # Espera o servidor iniciar
    run_client()
```

5 Conclusão

A adoção de protocolos como TLS e HTTPS é essencial para garantir segurança na internet. A evolução dessas tecnologias reforça a proteção contra ataques e vulnerabilidades, tornando a comunicação digital mais confiável e eficiente. O uso de certificados digitais e algoritmos de criptografia avançados assegura que as informações trocadas entre cliente e servidor permaneçam seguras e invioláveis.