

Uma avaliação do impacto da utilização do protocolos TLS/SSL em aplicações de tempo real

RAPHAEL LIMA SARAIVA

MARCOS VINICIUS DOS SANTOS SOMBRA

Resumo

O TLS é um protocolo de criptografia projetados para internet. Este Permite a comunicação segura entre os lados cliente e servidor de uma aplicação web. Portanto, este artigo apresenta o protocolo TLS e seu funcionamento, buscando demonstrar sua importância para proteger os dados enviados por usuários. Por fim é apresentada uma avaliação dos protocolos para verificar seu funcionamento e desempenho com diferentes linguagem (GO, Javascript e python) e quanto de acréscimo em tempo é adicionado por sua utilização junto ao protocolo http.

1 O QUE É SSL/TLS

Ao navegar em um site, seja para acessar o e-mail ou utilizar o internet banking, é preciso saber se que as informações fornecidas não serão interceptadas para a prática de ações fraudulentas. Independentemente do navegador, em seu lado esquerdo, é apresentado um cadeado indicado que está sendo utilizada uma tecnologia chamada SSL ou TLS por cima das requisições http do navegador formando assim o https. Isso garante que caso a requisição seja interceptada no meio do caminho, seu conteúdo não possa ser exposto.

O SSL significa **Secure Sockets Layer**, um tipo de protocolo que permite a comunicação criptografada entre um sistema cliente e um servidor. Atualmente a tecnologia se encontra depreciada e está sendo completamente substituída pelo TLS. TLS é uma sigla que representa a **Transport Layer Security** e certifica a proteção de dados semelhantemente ao SSL. Como o SSL não está mais de fato em uso, esse é o termo correto que deveria ser utilizado.

Assim é possível a criar e distribuir certificados com a tecnologia *TLS* para garantir a proteção das informações enviadas pela web.

2 IMPORTÂNCIA DO TLS

Em sites e e-commerces que utilizam um **certificado SSL/TLS**, podem garantir que informações como senhas, número do cartão de crédito ou algum tipo de documento, são codificadas e enviadas para o destinatário, onde apenas ele tem acesso à chave usada para sua decodificação. Através deste sistema, pode-se garantir que hackers e/ou softwares maliciosos não terão acesso aos dados, impedindo o uso indevido das informações.

Além disso, o certificado SSL/TLS, também transmite confiabilidade aos visitantes, pois desde o ano de 2018, sites de busca como o *google* passaram a sinalizar para os usuários sobre os sistemas que não possuem o certificado SSL/TLS. Isso faz com que os usuários pensem duas vezes antes de utilizarem ou até mesmo desistirem de acessar o sistema.

Os certificados SSL/TLS podem ser categorizados em:

- Self-signed — a empresa gera seu próprio certificado. Em geral, são de uso interno, não tendo interoperabilidade com os navegadores de internet;
- Domain Validated — a validação é realizada automaticamente pela Autoridade Certificadora, que confirma a existência do domínio. Para a validação, é levado em conta os dados cadastrados na entidade responsável pelo registro do domínio;
- Fully Authenticated — a Autoridade Certificadora valida se a solicitante do certificado SSL é a proprietária do site, se é legalmente constituída e se tem poder para efetuar a solicitação do certificado.

Sendo que o primeiro caso, um certificado auto-assinado é um SSL que não é assinado por uma autoridade de certificação. Estes certificados são fáceis de criar e não têm nenhum custo. No entanto, eles não fornecem todas as propriedades de segurança que os demais certificados assinados por uma CA.

3 FUNCIONAMENTO DO TLS

Como apresentado na Figura 1, o protocolo TLS em si é composto de duas camadas: o protocolo de Registro e os protocolos Handshaking TLS Handshaking Protocols.

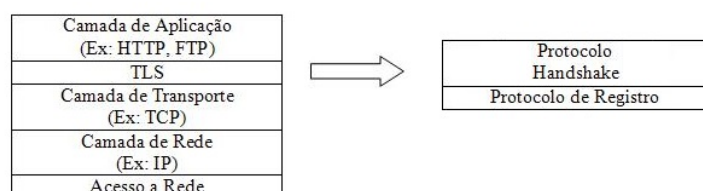


Figura 1

3.1. Protocolo de Registro

O TLS utiliza o protocolo de Registro para encapsular todas as mensagens dos demais protocolos das camadas superiores da Figura 1, explicando assim, sua independência com os protocolos de aplicação e facilitando o desenvolvimento de aplicações que necessitam de conexões seguras. O protocolo de Registro é o responsável por efetuar as operações necessárias para garantir a segurança da conexão. Pois este recebe as mensagens para serem transmitidas, fragmenta os dados em blocos e opcionalmente realiza a compressão dos dados. Além disso, aplica o MAC, encripta e por fim transmite o resultado. Logicamente, com os dados recebidos, o protocolo de Registro realiza a decriptação, verificação da integridade, realiza descompressão, reagrupa os blocos e os entrega para as camadas superiores.

3.2. Protocolos de Handshakings

Em relação ao protocolos de Handshakings este realizam a negociação de parâmetros de segurança na comunicação entre cliente e servidor. Os protocolos de Handshakings podem ser vistos como três subprotocolos (Handshake, Change Cipher Spec e Alerta), estes permitem que às partes cheguem a um acordo sobre os parâmetros de segurança que serão utilizados na camada de registro para autenticação, comunicação e para reportar condições de erro entre as partes.

O primeiro dos três (Handshake) é responsável por:

- Identificador da sessão: uma sequência de bytes escolhida pelo servidor para identificar uma sessão ativa ou uma sessão reiniciável.
- Método de compressão: qual o método de compressão que será utilizada antes da criptografia dos dados.
- Cipher Spec: especifica qual o algoritmo de criptografia vai ser utilizado, por exemplo, DES. Qual algoritmo de MAC (MD5, SHA) que vai ser utilizado. E define alguns atributos criptográficos como, por exemplo, o tamanho do hash
- Chave Mestre (master key): chave secreta de 48 bytes que será compartilhada entre o cliente e o servidor
- Is Resumable: flag que indica se a sessão pode ser utilizada para iniciar novas conexões.

O Change Cipher Spec existe para sinalizar mudanças nas estratégias de criptografia que estavam sendo utilizadas. Este protocolo é formado por apenas uma mensagem, a qual é encriptada e comprimida com os parâmetros que estavam previamente estabelecidos. A mensagem Change Cipher Spec é enviada por ambos, cliente e servidor, para que cada um deles passe a usar as novas regras de criptografia que foram negociadas.

Um dos tipos de conteúdo suportado pela camada de registros do TLS é o conteúdo de alerta. As mensagens de alerta transportam a importância do alerta e a descrição do alerta. Mensagens com

importância denominada fatal, resultam imediatamente no encerramento da conexão. Nesse caso outras conexões correspondendo à mesma sessão podem continuar, mas o identificador da sessão deve ser invalidado, prevenindo que essa sessão seja utilizada posteriormente para estabelecer novas conexões.

3.3. Estabelecimento da conexão

Os parâmetros criptográficos de uma sessão são determinados pelo TLS Handshake Protocol como já foi dito anteriormente. O TLS Handshake Protocol opera acima da Camada de Registro.

Quando um cliente e um servidor começam a se comunicar eles têm que entrar em acordo sobre qual versão será utilizada, qual algoritmo criptográfico será usado, opcionalmente autenticar um ao outro e usar criptografia assimétrica para compartilhar um segredo a ser usado pela criptografia simétrica, que é responsável pela maior parte da criptografia realizada pelo protocolo.

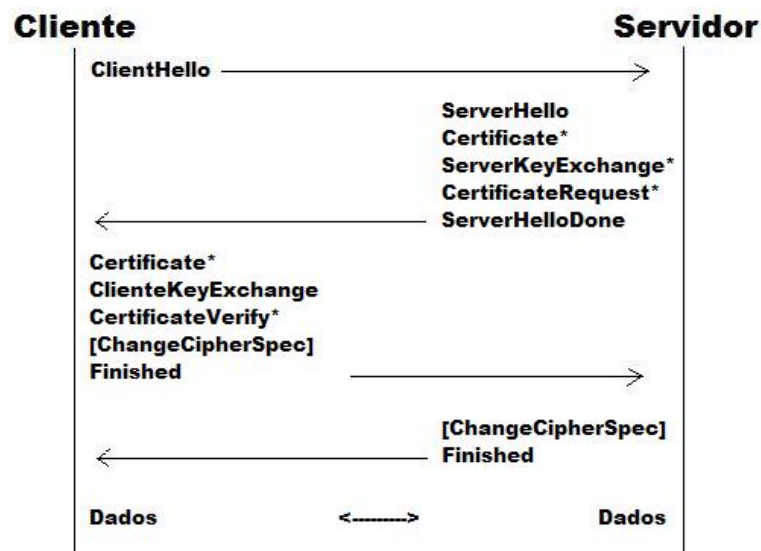


Figura 2

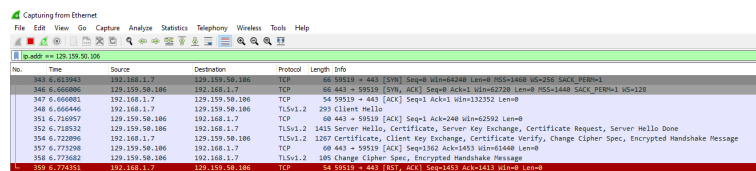
O processo de negociação mostrado acima pode ser explicado da seguinte forma, o cliente envia uma mensagem de ClientHello para o servidor, o servidor deve então enviar de volta uma mensagem ServerHello, caso contrário um erro fatal será caracterizado e a conexão encerrada. As mensagens de Hello servem para estabelecer quais as capacidades de segurança cada um dos lados possui. Essas mensagens estabelecem os seguintes parâmetros: versão do protocolo, ID da sessão, Suíte de Criptografia e método de compressão. Adicionalmente dois valores aleatórios são gerados e trocados, ClientHello.random e ServerHello.random. Após a troca das mensagens de Hello o servidor poderá enviar seu certificado, caso haja a necessidade de autenticação. Alternativamente o servidor pode enviar uma mensagem de ServerKeyExchange se não possuir certificado ou se o seu certificado for utilizado apenas para assinatura digital. Se o servidor for autenticado, ele pode requisitar ao cliente um certificado, caso a suíte de criptografia requiera tal característica do cliente. Depois dessa fase

o servidor envia uma mensagem de ServerHelloDone indicando que a fase de Hello da negociação acabou. O servidor então passa a esperar que o cliente responda. Se o servidor pediu um certificado do cliente, então ele ficará esperando até que o cliente envie o certificado. O cliente então envia uma mensagem de ClientKeyExchange e o conteúdo dessa mensagem dependerá do algoritmo de chave pública escolhido na fase de Hello. Caso o cliente tenha enviado um certificado com capacidade de assinatura, uma mensagem de digitally-signed certificate verify será enviada para verificar o certificado. Nesse ponto a mensagem ChangeCipherSpec será enviada pelo cliente e este copiará para a mensagem as informações pendentes sobre as Cipher Specs. Logo após essa mensagem o cliente envia uma mensagem Finished, já utilizando todos os parâmetros de segurança negociados. Em resposta o servidor manda uma mensagem de ChangeCipherSpec e logo após uma mensagem de Finished já com os parâmetros de segurança negociado sendo utilizados para enviar a mensagem. Agora os dois lados já podem começar a transmissão dos dados de forma segura.

4 AVALIAÇÃO DO PROTOCOLO TLS

4.1. Wireshark

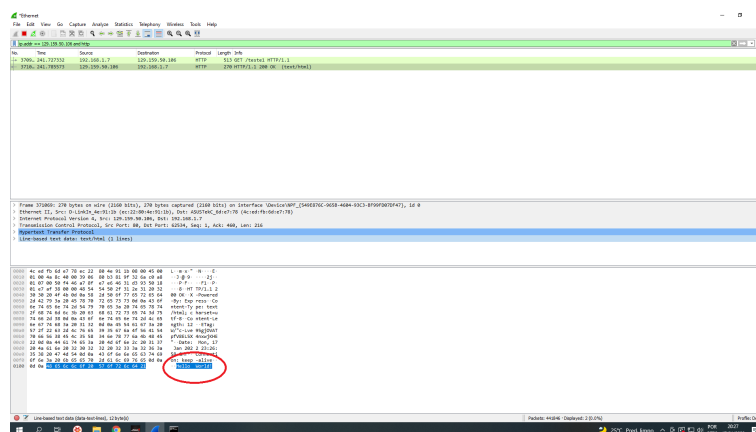
Para realizar a avaliação do protocolo TLS foi utilizado uma maquina virtual (VM) com o endereço de IP 129.159.50.106. Além disso utilizou-se também a ferramenta wireshark, onde filtrou-se os pacotes enviados para o endereço 129.159.50.106 por http e https.



The image shows a Wireshark packet capture window with the filter 'ip.addr == 129.159.50.106'. The packet list shows several packets related to a TLS handshake. The packet details pane shows the selected packet (No. 354) as a TLSv1.2 Client Hello.

No.	Time	Source	Destination	Protocol	Length	Info
347	6.655981	192.168.1.7	129.159.50.106	TCP	60	59519 → 443 [SYN] Seq=0 Win=0 Len=0
348	6.656006	129.159.50.106	192.168.1.7	TCP	60	443 → 59519 [ACK] Seq=62720 Len=0
349	6.656446	192.168.1.7	129.159.50.106	TLSv1.2	293	Client Hello
351	6.716957	129.159.50.106	192.168.1.7	TCP	60	443 → 59519 [ACK] Seq=1 Acks=240 Win=0 Len=0
352	6.720332	129.159.50.106	192.168.1.7	TLSv1.2	345	Server Hello, Certificate, Server Key Exchange, Certificate Request, Server Hello Done
354	6.722896	192.168.1.7	129.159.50.106	TLSv1.2	1207	Certificate, Client Key Exchange, Certificate Verify, Change Cipher Spec, Encrypted Handshake Message
357	6.773298	129.159.50.106	192.168.1.7	TCP	60	443 → 59519 [ACK] Seq=1302 Acks=143 Win=0 Len=0
358	6.773682	129.159.50.106	192.168.1.7	TLSv1.2	186	Change Cipher Spec, Encrypted Handshake Message
359	6.774351	192.168.1.7	129.159.50.106	TCP	60	59519 → 443 [EST. ACK] Seq=643 Acks=143 Win=0 Len=0

Figura 3



The image shows a Wireshark packet capture window with the filter 'ip.addr == 129.159.50.106'. The packet list shows several packets related to a TLS handshake. The packet details pane shows the selected packet (No. 354) as a TLSv1.2 Client Hello.

No.	Time	Source	Destination	Protocol	Length	Info
347	6.655981	192.168.1.7	129.159.50.106	TCP	60	59519 → 443 [SYN] Seq=0 Win=0 Len=0
348	6.656006	129.159.50.106	192.168.1.7	TCP	60	443 → 59519 [ACK] Seq=62720 Len=0
349	6.656446	192.168.1.7	129.159.50.106	TLSv1.2	293	Client Hello
351	6.716957	129.159.50.106	192.168.1.7	TCP	60	443 → 59519 [ACK] Seq=1 Acks=240 Win=0 Len=0
352	6.720332	129.159.50.106	192.168.1.7	TLSv1.2	345	Server Hello, Certificate, Server Key Exchange, Certificate Request, Server Hello Done
354	6.722896	192.168.1.7	129.159.50.106	TLSv1.2	1207	Certificate, Client Key Exchange, Certificate Verify, Change Cipher Spec, Encrypted Handshake Message
357	6.773298	129.159.50.106	192.168.1.7	TCP	60	443 → 59519 [ACK] Seq=1302 Acks=143 Win=0 Len=0
358	6.773682	129.159.50.106	192.168.1.7	TLSv1.2	186	Change Cipher Spec, Encrypted Handshake Message
359	6.774351	192.168.1.7	129.159.50.106	TCP	60	59519 → 443 [EST. ACK] Seq=643 Acks=143 Win=0 Len=0

Figura 4

É possível observar na Figura 3, todo o passo-a-passo do protocolo handshake para a comunicação do TLS, enquanto que na Figura 4 é apresentada a comunicação sem TLS utilizando apenas o http.

Por não estar utilizando o Tls na Figura, é possível observar no círculo vermelho, que todo o conteúdo do pacote está exposto.

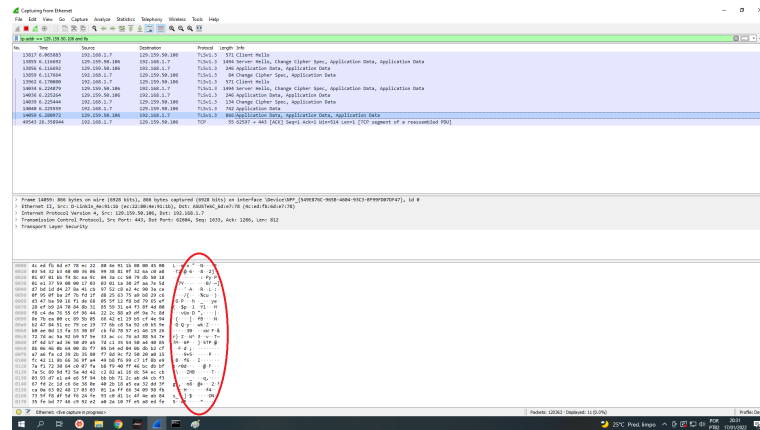


Figura 5

Por fim, o círculo vermelho na figura 5 demonstra que o conteúdo dos pacotes TLS utilizados nela estão protegidos.

4.2. avaliação de desempenho http x https

Uma avaliação de desempenho foi realizada usando a VM de IP 129.159.50.106, onde foi hospedando um servidor em javascript utilizando a biblioteca *express* e rodando nas portas 80 (http) e 443 (https). Para o experimento, foram realizados 10 rounds, onde cada round representa 10 execuções consecutivas de http e https para os servidor.

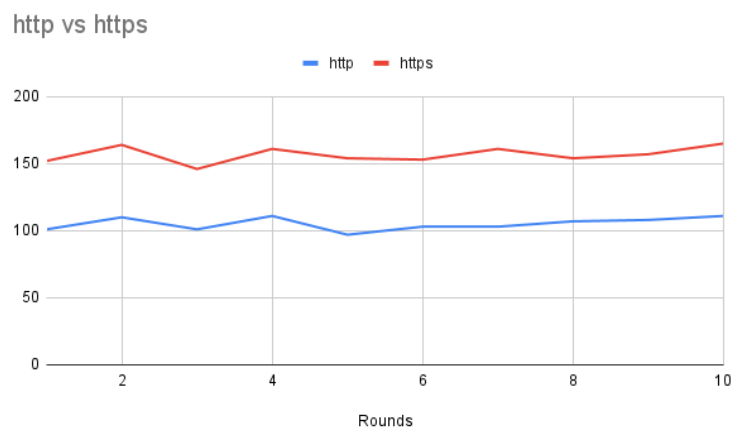


Figura 6

A figura 6 apresenta tempo em mili segundo necessários para fazer a primeira execução de cada round. é Possível observar que essa primeira execução e maior que as demais pois nela ocorre o *handshake* entre cliente e servidor. Porém as requisições utilizado https são em média bem maiores que as utilizado http, porém existe todo o processo de criptografia da mensagem.

A figura 7, apresenta a média das 9 demais execuções para http e https. Nela, é possível observar que praticamente não existe diferença entre a utilização deles, uma vez que após o processo de sincrônica, ambos gastas praticamente quase o mesmo tempo para enviam as requisições ao servidor.

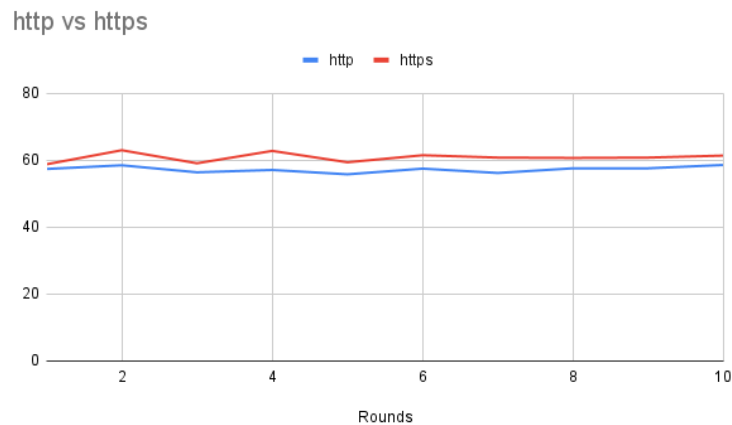


Figura 7

Por fim, foi feito o teste comparando as linguagens de programação (Go, javascript e Python) na hora da comunicação entre soquetes. O servidor utilizado foi um servidor em python hospedado na mesma VM da avaliação anterior. Este era encarregado apenas de fazer a conexão com três clientes utilizando as respectivas linguagem apresentadas.

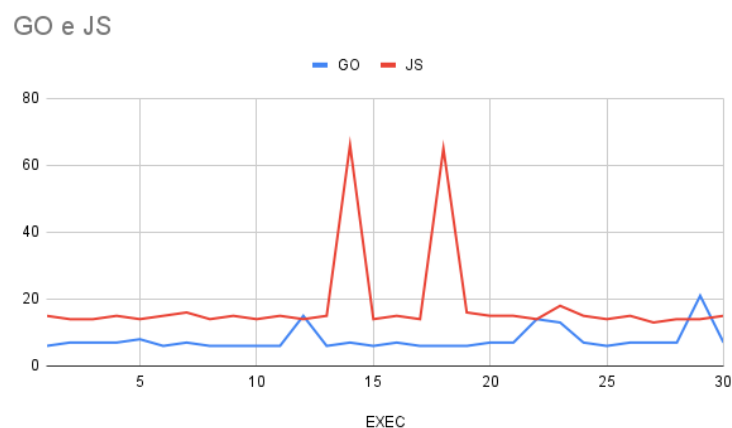


Figura 8

Os resultados mostrarm que a liguagem com pior desempenho foi python com uma media de 159 ms. Go foi a que teve o melhor desempenho levando cerca de 7.8 ms para realizar a conexão e A

javascript que teve um desempenho próximo do Go mas ainda perdendo por cerca de 10 ms

5 CONCLUSÃO

Por fim, pode-se concluir que https possui um preço pela garantia de segurança durante sua primeira conexão. Entretanto é um preço justo a se pagar pela segurança dos dados criptografados e além nas demais transmissões de pacotes quase não existe diferença entre o uso do tls dentro do protocolo http. No demais entre as três linguagem utilizada na avaliação, Go se mostrou a melhor em termos de velocidade para conexão do soquete tls.