Desenvolvimento de uma Aplicação Cliente de Envio de email através de Soquete de Rede

Clarel de Menezes Spies Filho, Luiza Daiane Rabuski

[1]Departamento de Computação – Universidade de Santa Cruz do Sul (UNISC)
Caixa Postal 96.815900 – Santa Cruz do Sul – RS – Brasil

{clarel,luizar}@mx2.unisc.br

Abstract. *This paper presents the tecnology behind sending email and how was developed a client application of email send wich commuticate through network socket using authentication with the sender sending server.*

Resumo. *Este artigo apresenta a tecnologia por trás do envio de email e como foi desenvolvido uma aplicação cliente de envio de email com a comunicação feita através de soquete de rede usando autenticação com o servidor de envio do remetente.*

1. Introdução

Para realizar o envio de email na internet é preciso realizar uma conexão através de um soquete de rede, com o endereço do servidor de email do destinatário e a porta pela qual o serviço de envio de mensagem é realizado. Devido a problemas com spam, vírus e mensagens sendo enviadas ao destinatário sem passar pelo servidor de um provedor de email e sem nenhuma autenticação, os provedores não podiam rastreá-las [Hoepers and Steding-Jessen 2009].

Para esse problema com mensagens indesejadas ser resolvido no Brasil, foi fechada a porta 25 pelos provedores e passou-se a usar a porta 465 e 587 para o envio de mensagens de email, estas portas necessitam de uma conexão segura com autenticação com o servidor de email do remetente, garantindo assim controle e rastreabilidade da origem das mensagens [Hoepers and Steding-Jessen 2009].

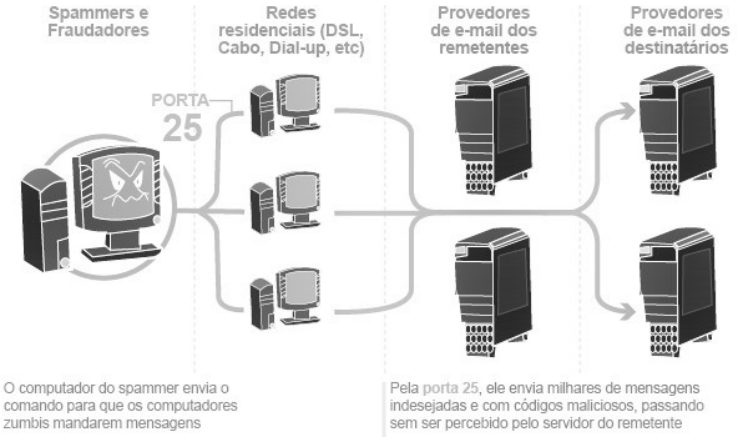

Figura 1. Funcionamento do envio de email através da porta 25

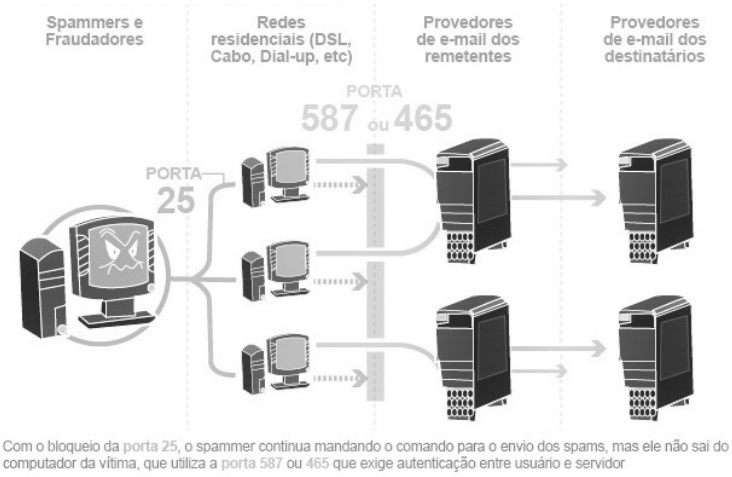

Figura 2. Funcionamento do envio de email através das portas 465 e 587

No artigo apresentaremos como foi desenvolvida nossa aplicação cliente de envio de email, que usa soquete de rede com o uso do protocolo SSL através da porta 465 para realizar o envio da mensagem, e abordaremos os assuntos relacionados as tecnologias que foram usadas.

2. Soquete de Rede

Basicamente um soquete de rede é uma conexão ponto-a-ponto entre dois processos através de uma rede de computadores, para fazer essa comunicação é necessário um endereço de soquete que é formado por um endereço de IP e um número de porta. Este endereço nos permite encontrar o servidor com o qual desejamos nos comunicar na rede, e identifica o processo deste servidor que queremos executar [Tanenbaum et al. 2011].

Uma interface de programação de aplicativo (API) de soquete TCP geralmente são fornecidos pelo sistema operacional para o uso de um ssoquete de rede, que é usado como um ponto final em um fluxo de comunicação entre dois aplicativos realizada através de uma rede. Uma API de soquete TCP geralmente é baseada no padrão de Berkeley [Tanenbaum et al. 2011].

As quatro primeiras primitivas da tabela são executadas do lado do servidor na mesma ordem descrita. Primeiramente a primitiva SOCKET aloca um espaço de tabela para ele na entidade de transporte e cria um novo ponto final. Um soquete recém criado não possui um endereço de rede e este endereço é atribuído através da chamada da primitiva BIND. No momento que o servidor informou um endereço para um soquete, este já pode receber uma conexão realizada por um cliente remoto. A primitiva LISTEN aloca espaço para uma fila de chamadas recebidas, caso vários clientes se conectem ao mesmo tempo. Para bloquear a espera por uma conexão de entrada é executada a primitiva ACCEPT [Tanenbaum et al. 2011].

Quando é solicitada uma conexão, a entidade de transporte cria um novo soquete com as mesmas propriedades do soquete que originou a requisição e retorna um descritor de arquivos normal, que pode ser usado para ler e gravar de maneira padrão. O servidor então desvia uma thread ou processo para tratar essa nova conexão [Tanenbaum et al. 2011].

Do lado do cliente também é necessário primeiramente criar um soquete através do uso da primitiva SOCKET, iniciando então o processo de conexão através da chamada da primitiva CONNECT. A conexão é concluída no momento em que recebemos do servidor uma chamada de segmento e o processo do cliente é desbloqueado para que a conexão seja estabelecida. Quando a conexão estiver estabelecida tanto o cliente como o servidor podem utilizar as primitivas SEND e RECEIVE para enviar e receber dados através de uma conexão full-duplex. O encerramento da conexão é simétrico, acontece quando ambos os lados executam a primitiva [Tanenbaum et al. 2011].

Tabela 1. AS PRIMITIVAS DE SOQUETES PARA TCP (TANENBAUM, 2011, p. 518)

Primitiva	Significado
SOCKET	Criar um novo ponto final de comunicação
BIND	Anexar um endereço local a um soquete
LISTEN	Anunciar a disposição para aceitar conexões; mostra o tamanho da fila
ACCEPT	Bloquear o responsável pela chamada até uma tentativa de conexão ser recebida
CONNECT	Tenta estabelecer uma conexão ativamente
SEND	Enviar alguns dados através da conexão
RECEIVE	Receber alguns dados da conexão
CLOSE	Encerrar a conexão

3. Secure Sockets Layer (SSL)

Secure Sockets Layer (SSL) é um protocolo de segurança que protege a comunicação via internet para serviços como e-mail (SMTP), navegação por páginas (HTTPS) e outros tipos de transferência de dados. Uma versão atualizada do protocolo SSL é chamado de Transport Layer Security (TLS), mas SSL e TLS são essencialmente o mesmo padrão [Yale 2016].

Muitos sites usam o protocolo SSL para proteger as informações confidenciais do usuário, como números de cartão de crédito. Por convenção, as URLs que requerem uma conexão SSL começam com ”HTTPS”em vez de ”HTTP”. Muitos clientes de e-mail também suportam SSL como um meio para criptografar mensagens para, assim, mantê-las seguras [Yale 2016].

SSL e TLS fornecem três medidas de segurança:

- A autenticação do cliente: Garante que o cliente possa identificar o servidor, e pode verificar se a transferência de dados será segura.
- Criptografia de dados: Os dados são misturados usando algoritmos de criptografia complexos, de modo que mesmo se for interceptado em rota não pode ser decifrado.
- Verifica a integridade dos dados: Verifica que não houve alteração dos dados durante o trânsito.

O protocolo SSL usa um sistema de criptografia que utiliza duas chaves para criptografar os dados, uma chave pública conhecida por todos e uma chave privada conhecida apenas pelo destinatário. A criptografia de mensagens/anexos com o protocolo SSL funcionará apenas entre servidores de email em que o protocolo SSL esteja habilitado [Yale 2016].

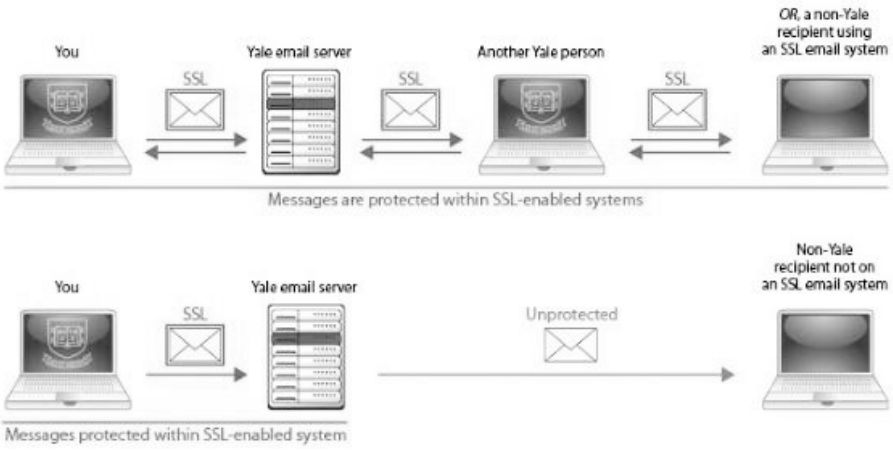

Figura 3. Comparação de envio de email entre sistemas SSL e não SSL

3.1. Arquitetura

O protocolo SSL foi criado como um protocolo separado para segurança, sendo assim, considerado como uma nova camada na arquitetura TCP/IP conforme ilustrado na figura abaixo:

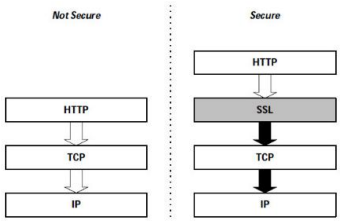

Figura 4. Comparação da arquitetura com e sem a utilização do protocolo SSL

Este método permite que o SSL seja utilizado para outras aplicações que não sejam o HTTP, como por exemplo, o FTP, POP3 e SMTP. [Helvio Junior, 2012]

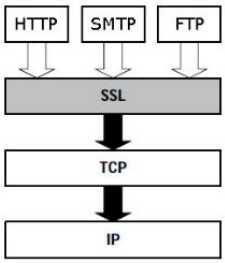

Figura 5. Ilustração da arquitetura para demais aplicações

4. SSL Socket em Java

O SSLSocket adiciona uma camada a mais na comunicação por soquete para proteger a comunicação e torná-la segura, o que os servidores de email como os do Gmail exigem. Para a nossa aplicação foi usada a linguagem de programação Java com o uso da biblioteca javax.net.ssl.SSLSocket, que faz a abstração para o desenvolvedor da comunicação por soquetes [Oracle 2016c].

Para instanciação da SSLSocket é necessário um endereço IP e a porta pela qual se conectará ao servidor e seus principais métodos são getInputStream, que representa os dados que serão lidos do arquivo no canal de entrada e getOutputStream, que representa os dados sendo escritos no canal de saída e que serão transportados [Oracle 2016c].

Para o email ser enviado foi usada a biblioteca java.io, que é usada para a entrada e a saída de dados. Para a entrada de dados é usado a classe java.io.BufferedReader, que representa um fluxo de entrada de caracteres e para a sua instanciação recebe o retorno de Socket.getInputStream [Oracle 2016a].

A classe java.io.DataOutputStream representa a saída e a escrita final dos dados e sua instanciação e recebe o retorno de Socket.getOutputStream. O método utilizado foi writeBytes, que recebe a string como uma sequência de bytes para então realizar a saída dos dados [Oracle 2016b].

5. Aplicação desenvolvida

O nosso cliente de envio de email foi desenvolvio em Java e possui um arquivo *Runnable Jar File*, o que permite ser executado apenas com dois cliques no arquivo se você possuir o Java JRE 1.7 ou mais recente instalado. Abaixo imagem da tela do programa.

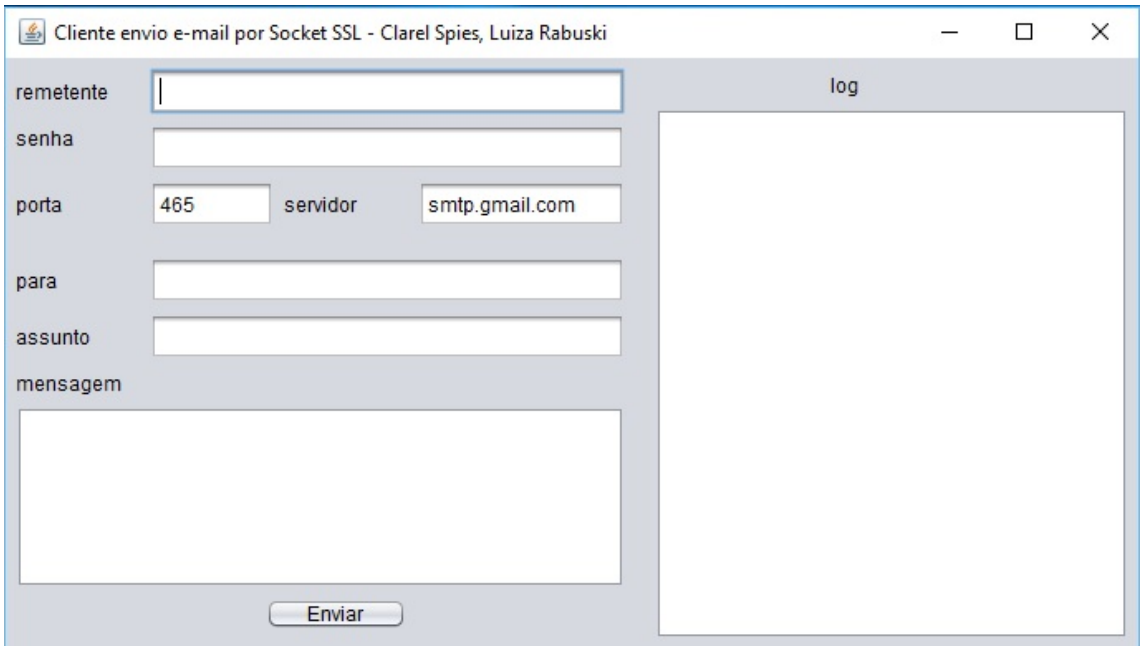

Figura 6. Imagem da aplicação desenvolvida

Para realizar a autenticação e se comunicar com o servidor de email, o campo remetente deve estar preenchido com seu email completo, o campo senha deve estar pre-

enchido com a senha do seu email e a porta e o servidor devem estar preenchidos com as informações do endereço do seu servidor de email. No campo para deve ser preenchido com o endereço de email do destinatário da mensagem, e no campo assunto e mensagem você digita o assunto e a mensagem do email, respectivamente. No campo ao lado de log, pode-se visualizar as trocas de mensagens realizadas entre o cliente e o servidor.

5.1. Autenticação

Assim que os dados do email forem preenchidos e o botão Enviar for pressionado, a aplicação iniciará o processo de envio do email. O primeiro passo é abrir a conexão com o soquete e realizar a autenticação.

Quando que a conexão com o servidor através do soquete é estabelecida com sucesso, recebemos a mensagem "*220 smtp.gmail.com ESMTP c196sm2087000ywa.43 - gsmtp*"informando que o processo aguarda por novas mensagens. Do contrário caso a conexão não seja estabelecida com o servidor, seja por processo ocupado, porta ou endereço errado, recebemos um erro na aplicação e a conexão é cancelada.

Após conexão estabelecida o Gmail exige que o cliente pergunte a ele quais extensões SMTP ele suporta, através de uma mensagem com cabeçalho "EHLO", mas a exigência deste comando é opcional em servidores. Então o cliente envia a mensagem "*EHLO smtp.gmail.com*". Na imagem abaixo podemos visualizar a resposta do servidor.

```
SERVER: 250-smtp.gmail.com at your service, [201.34.81.102]
SERVER: 250-SIZE 35882577
SERVER: 250-8BITMIME
SERVER: 250-AUTH LOGIN PLAIN XOAUTH2 PLAIN-CLIENTTOKEN OAUTHBEARER XOAUTH
SERVER: 250-ENHANCEDSTATUSCODES
SERVER: 250-PIPELINING
SERVER: 250-CHUNKING
SERVER: 250 SMTPUTF8
```

Figura 7. Resposta do SMTP Gmail a mensagem EHLO, com suas extensões suportadas

Após perguntarmos pelas extensões suportadas, o servidor nos permite enviar mensagens para executar serviços. No caso vamos enviar a mensagem "*AUTH LOGIN*"para iniciarmos o processo de autenticação. Após nossa requisição para autenticação ser enviada, o servidor responde com uma mensagem com código 334 e a data encriptada, então o cliente envia uma mensagem com a informação de usuário encriptada. Após receber o usuário, o servidor manda novamente uma mensagem com código 334 e a data encriptada, desta vez respondemos com a senha encriptada e aguardamos a resposta se nossa autenticação foi aceita ou rejeitada.

```
Cliente: AUTH LOGIN

SERVER: 334 VXNlcm5hbWU6
Cliente: Y2xhcmVsLmZpbGhvQGdtYWlsLmNvbQ==

SERVER: 334 UGFzc3dvcmQ6
Cliente: TmV3cGFzczE9

SERVER: 235 2.7.0 Accepted
```

Figura 8. Troca de mensagens com autenticação aceita

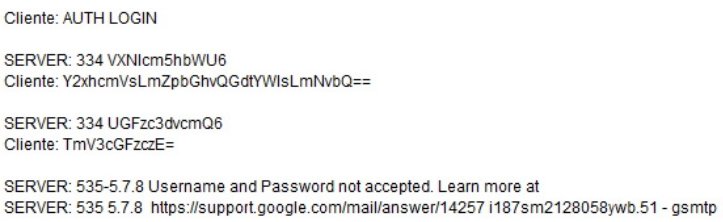

Figura 9. Troca dem ensagens com autenticação rejeitada

Se a senha e o usuário informados estiverem correto a autenticação será realizada com sucesso e a aplicação continuará para o envio da mensagem. Do contrário, ele informará que sua senha ou nome de usuário estão incorretos e finalizará a conexão soquete.

5.2. Envio da Mensagem

Após a conexão com autenticação ser estabelecida com sucesso, o sistema começa a enviar as informações da mensagem de email. Primeiramente é enviado a mensagem "*MAIL FROM:<clarel.filho@gmail.com>* "com a informação do remetente do email e então é esperada a mensagem de retorno do servidor "*250 2.1.0 OK t8sm2146558ywc.40 - gsmtp*".

Então é enviada a mensagem com a informação de quem será o destinatário da mensagem "*RCPT TO:<clarel.filho@gmail.com>*. Novamente é esperado uma mensagem de confirmação do servidor "*250 2.1.5 OK t8sm2146558ywc.40 - gsmtp*"caso o destinatário exista, ou então

Após o remetente e o destinatário da mensagem serem enviados com sucesso para o servidor, informamos que começaremos o envio da mensagem do email, que é formado pelo assunto e a mensagem. Primeiramente enviamos uma mensagem com "*DATA*"e o servidor nos responde confirmando que podemos continuar "*354 Go ahead t8sm2146558ywc.40 - gsmtp*".

Após a confirmação de que podemos continuar para a mensagem, primeiramente informamos o assunto do email "*Subject: teste assunto*", esta mensagem não possui retorno do servidor. Então é enviado uma mensagem com o que esta escrito no email "*teste linha 1*", que seria a mensagem de email que o destinatário vai receber. Após a mensagem de email for completamente enviada, enviamos a mensagem "."para informarmos o servidor que nosso email foi completamente montado e o servidor responde com "*250 2.0.0 OK 1466750370 e5sm2188027ywf.25 - gsmtp*"

5.3. Fechando conexão

Após receber a mensagem de confirmação do recebimento da mensagem pelo servidor, a aplicação envia a mensagem "*QUIT*"e aguarda a mensagem do servidor "*221 2.0.0 closing connection e5sm2188027ywf.25 - gsmtp*"confirmando que a conexão foi encerrada.

5.4. Resultados obtidos

Após completar o envio do email com sucesso ou não, a aplicação apresenta no campo log toda a comunicação realizada com o servidor. Abaixo segue imagem da comunicação realizada para o envio de um email de teste e o email recebido na caixa de entrada do destinatário.

```
SERVER: 220 smtp.gmail.com ESMTP h11sm2196644ywc.29 - gsmtp
CLIENTE: EHLO smtp.gmail.com

SERVER: 250-smtp.gmail.com at your service, [201.34.81.102]
SERVER: 250-SIZE 35882577
SERVER: 250-8BITMIME
SERVER: 250-AUTH LOGIN PLAIN XOAUTH2 PLAIN-CLIENTTOKEN OAUTHBEARER XOAUTH
SERVER: 250-ENHANCEDSTATUSCODES
SERVER: 250-PIPELINING
SERVER: 250-CHUNKING
SERVER: 250 SMTPUTF8
CLIENTE: AUTH LOGIN

SERVER: 334 VXNIcm5hbWU6
CLIENTE: Y2xhcmVsLmZpbGhvQGdtYWlsLmNvbQ==

SERVER: 334 UGFzc3dvcmQ6
CLIENTE: TmV3cGFzczE9

SERVER: 235 2.7.0 Accepted
CLIENTE: MAIL FROM:<clarel.filho@gmail.com>

SERVER: 250 2.1.0 OK h11sm2196644ywc.29 - gsmtp
CLIENTE: RCPT TO:<clarel@mx2.unisc.br>

SERVER: 250 2.1.5 OK h11sm2196644ywc.29 - gsmtp
CLIENTE: DATA

SERVER: 354  Go ahead h11sm2196644ywc.29 - gsmtp
CLIENTE: Subject: teste assunto

CLIENTE: teste mensagem linha 1
teste mensagem linha 2

CLIENTE: .

SERVER: 250 2.0.0 OK 1466750977 h11sm2196644ywc.29 - gsmtp
CLIENTE: QUIT

SERVER: 221 2.0.0 closing connection h11sm2196644ywc.29 - gsmtp
```

Figura 10. Resultados de log da aplicação ao enviar email de teste

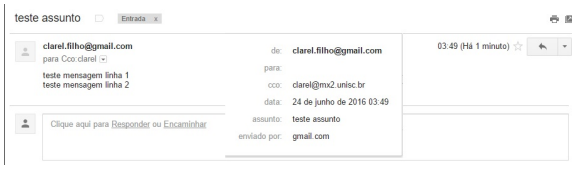

Figura 11. Resultados de log da aplicação ao enviar email de teste

6. Conclusão

Através de estudos realizados para o desenvolvimento da aplicação, pudemos entender como funciona os servidores de email e porque a porta 25 foi fechada no Brasil para a comunicação. Também foi possível verificar como funciona um serviço sob autenticação através da troca de mensagens por soquetes de rede.

Como ideia para trabalhos futuros pode ser implementado múltiplos destinatários, cópia oculta, mensagem HTML e arquivos em anexo para termos uma ainda mais ampla do serviço de SMTP e da troca de mensagens. O código fonte da aplicação esta disponível em *https://github.com/clarel/JavaSocketMail.git* para quem tiver interesse em estudar o funcionamento da aplicação ou até mesmo modifica-la.

Referências

Hoepers, C. and Steding-Jessen, K. (2009). Gerência de porta 25: Motivação, importância da adoção para o combate ao spam e discussões no brasil e no mundo. Disponível em: http://200.160.7.7/docs/ct-spam/ct-spam-gerencia-porta-25.pdf. Acesso em 24 junho 2016.

Oracle (2016a). Class bufferedreader. Disponível em: https://docs.oracle.com/javase/7/docs/api/java/io/BufferedReader.html. Acesso em 22 junho 2016.

Oracle (2016b). Class dataoutputstream. Disponível em: https://docs.oracle.com/javase/7/docs/api/java/io/DataOutputStream.html. Acesso em 22 junho 2016.

Oracle (2016c). Class sslsocket. Disponível em: http://download.java.net/jdk7/archive/b123/docs/api/javax/net/ssl/SSLSocket.html. Acesso em 21 junho 2016.

Tanenbaum, A. S. et al. (2011). *Redes de Computadores*. Pearson.

Yale (2016). Secure web and email: Ssl and tls. Disponível em: http://its.yale.edu/secure-computing/security-standards-and-guidance/device-security/safe-mobile-computing/secure-web-email-ssl-tls. Acesso em 22 junho 2016.