Autores

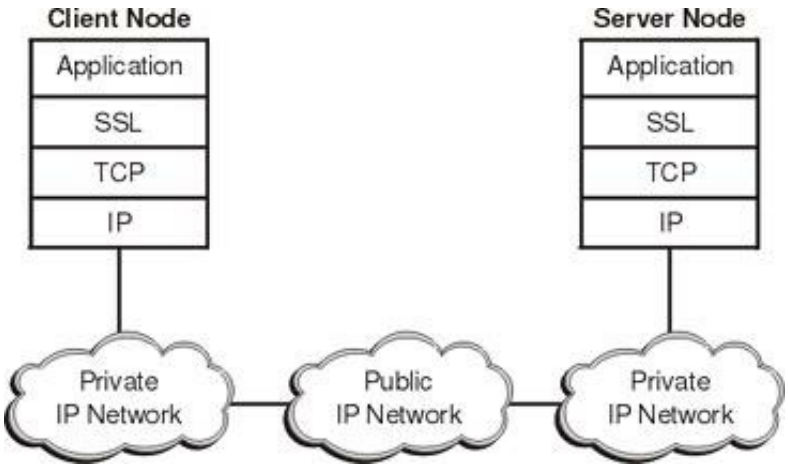
- **Thiago Senhorinha Rose** - 12100774 (https://github.com/thisenrose)
- **Igor Henrique Grajefe Feitosa** - 12106264

SSL detalhado

SSL e sua pilha de protocolos/serviços

O que é?

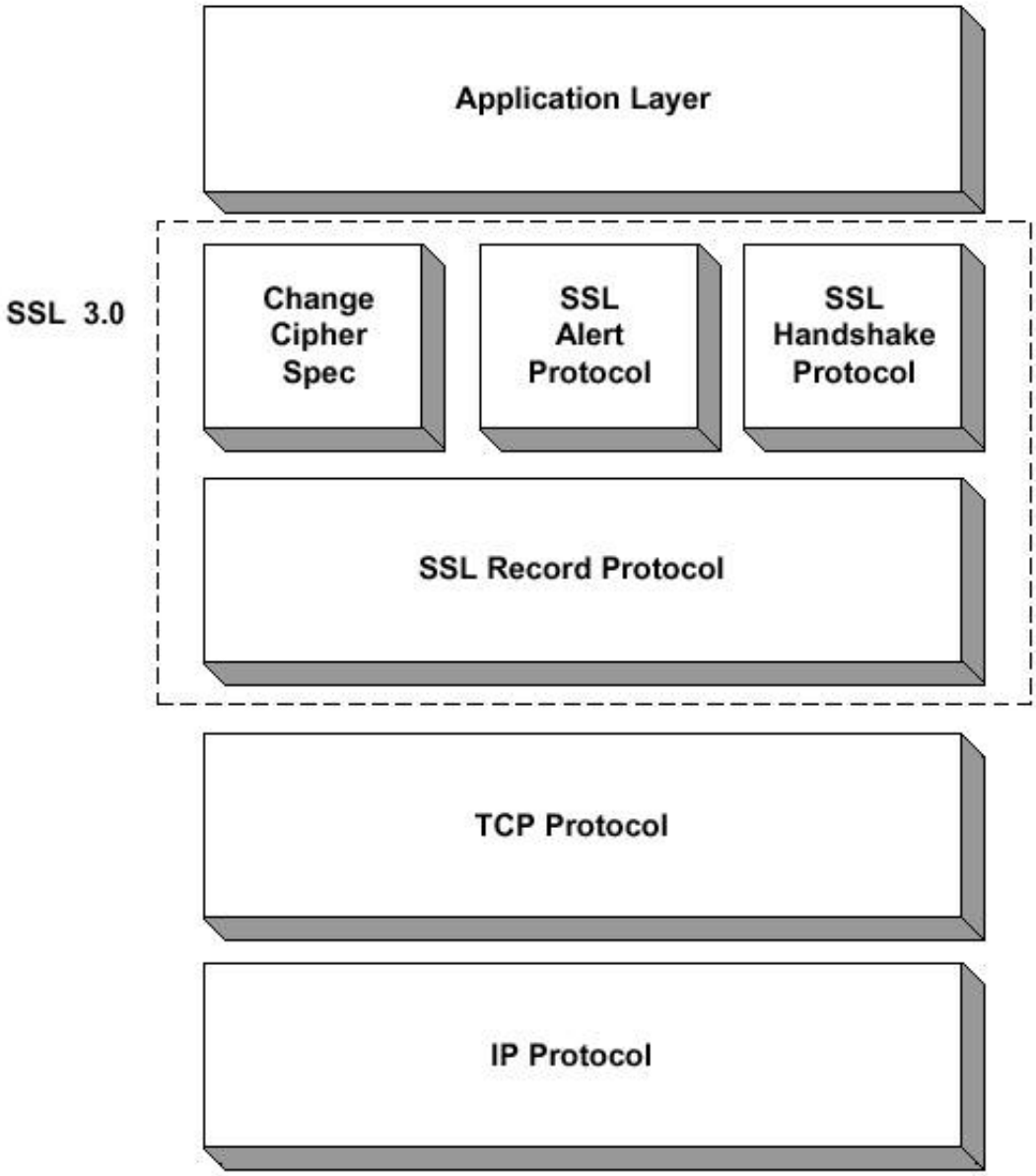
Secure Sockets Layer (SSL) é uma tecnologia de segurança padronizada para estabelecer uma conexão criptografada entre um server e um cliente. Geralmente usadas em web servers e navegadores, ou em e-mail server e e-mail cliente. Atualmente o SSL foi substituído pelo Transport Layer Security (TLS) como descrito no RFC 2246. No modelo OSI o protocolo SSL/TLS , embora se inicie na camada de sessão, atua na camada de apresentação, onde os dados são transformados para serem mandados a rede.



Conexão entre Cliente e Servidor

Pilha de protocolos e seus serviços

O SSL é composto de quatro protocolos **SSL Record Protocol**, **Change Chipher Spec**, **SSL Alert Protocol** e **SSL Handshake** que interagem com o protocolo **TCP** e com a camada de aplicação. A seguir explicaremos com mais detalhe cada um de seus protocolos.



Protocolos SSL

SSL Record Protocol

O Record Protocol é um protocolo de camadas. A cada camada, mensagens podem incluir campos para tamanho, descrição e conteúdo. Ele pega as mensagens que serão transmitidas, fragmenta os dados em blocos gerenciáveis, opcionalmente comprime o dado, aplica o MAC, cifra e transmite o resultado. Recebe o dado, decifra, verifica, descomprime, remonta os blocos e entrega para o cliente de maior nível (maior camada).

Esses protocolo oferece serviços básicos de segurança para vários protocolos da camada superior como Protocolo de Estabelecimento de Sessão SSL (Handshake Protocol), Protocolo de Mudança de Especificação de Cifra SSL (Change Cipher Spec Protocol), e o Protocolo de Alerta (Alert Protocol).

SSL Alert Protocol

É a principal função do Protocolo Alert no SSL informar a outra ponta sobre problemas (caso existam), na atual sessão A mensagem é composta de um campo que diz o nível crítico do alert e um campo de descrição Mensagens alert fatais resultam no fim de sessão SSL.

- Pode ser enviado a qualquer hora durante o handshake até o encerramento da sessão.
- Se usado como sinal de erro fatal, a sessão termina imediatamente.
- Se usado como aviso, o receptor pode decidir terminar a sessão se ele achar que a sessão não é mais confiável o suficiente para suas necessidades.

SSL Change Cipher Spec Protocol

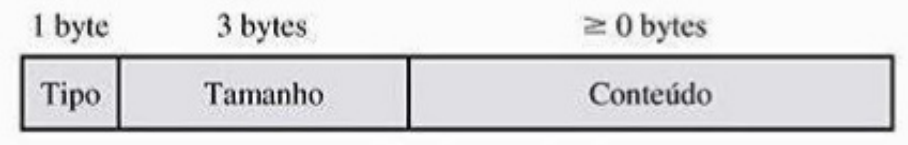
- É usado para mudar a criptografia usada pelo cliente e pelo server. Normalmente usado como parte do processo de handshake para mudar para criptografia simétrica. Este protocolo é uma única mensagem que diz ao receptor que o emissor quer mudar para um novo grupo de chaves, o qual é criado trocando informações pelo protocolo de handshake.

SSL Handshake Protocol

É considerada a parte mais complexa do SSL. Esse protocolo permite que o servidor e o cliente autentiquem um ao outro e negociem um algoritmo de criptografia e de MAC e chaves criptográficas a serem utilizadas para proteção dos dados enviados em um registro SSL. Esse protocolo é usado antes que quaisquer dados de aplicação sejam transmitidos.

- Usa uma combinação de chaves públicas cifradas e chaves simétricas cifradas.
- Uma sessão SSL sempre começa uma troca de mensagens com um SSL handshake. Este permitem o server se autenticar para o cliente usando técnicas de chaves públicas. Então ele permite o cliente e o server cooperar na criação de chaves simétricas para usar no ciframento, deciframento e "tamper detection" durante a sessão SSL. O SSL handshake também permite o cliente se autenticar para o server.

Esse protocolo é composto por uma série de mensagens trocadas entre cliente e servidor e possuem o formato especificado na figura abaixo.



Formato da mensagem para o Handshake Protocol

- **Tipo (1 byte):** Indica uma de 10 mensagens.

Tipo de mensagem
hello_request
client_hello
server_hello
certificate
server_key_exchange
certificate_request
server_done
certificate_verify
client_key_exchange
finished

Tipos de mensagem do protocolo de estabelecimento de Sessão SSL

- **Tamanho (3 bytes):** O tamanho da mensagem em bytes
- **Conteúdo (>= 0 bytes):** Os parâmetros associados a essa mensagem.

Parâmetros
Nulo
versão, aleatório, ID de sessão, conjunto de cifras, método de compactação
versão, aleatório, ID de sessão, conjunto de cifras, método de compactação
cadeia de certificados X.509v3
parâmetros, assinatura
tipo, autoridades
Nulo
Assinatura
parâmetros, assinatura
valor de hash

Tipos de parâmetros do protocolo de estabelecimento de Sessão SSL

Fases

O protocolo Handshake trabalha com 4 fases que serão detalhadas a seguir.

Estabelecer os atributos de segurança - Fase 1

Tem por finalidade estabelecer os atributos de segurança, incluindo versão de protocolo, ID de sessão, conjunto de cifras, métodos de compactação e números aleatórios iniciais.

A troca é iniciada pelo cliente que envia uma mensagem **client_hello" com os parâmetros:

- **Versão (version):** A última versão entendida pelo cliente.
- **Aleatório (random):** Uma estrutura aleatória gerada pelo cliente. Servem como nonces e são utilizados durante a troca de chaves para impedir ataques por repetição.
- **ID de sessão (session ID):** Possui tamanho variável. Os valores diferentes de zero indicam que o cliente deseja atualizar os parâmetros de uma conexão existente ou criar uma nova conexão nesta sessão. Para valores iguais a zero o servidor interpreta que o cliente deseja estabelecer uma nova conexão em uma nova sessão.
- **Conjunto de cifras (CipherSuite):** Lista com os algoritmos criptográficos admitidos pelo cliente em ordem decrescente de preferência.
- **Métodos de compactação (compression method):** Lista com os métodos de compactação admitidos pelo cliente.

Após o envio da mensagem **client_hello**, o cliente espera pela mensagem **server_hello** que possui os mesmos parâmetros da mensagem **client_hello**.

Autenticação do servidor e troca de chaves - Fase 2

O servidor inicia essa fase enviando seus certificados se for necessário o processo de autenticação. A mensagem de certificado (certificate message) é exigida para qualquer método e troca de chaves que tenham sido acordados, exceto Diffie-Hellman anônimo.

Em seguida, uma mensagem **server_key_exchange** pode ser enviada, se for necessário. Ela não é exigida em dois casos

1. O servidor enviou um certificado com os parâmetros Diffie-Hellman fixos
2. Troca de chaves RSA

A mensagem **server_key_exchange** é necessária para o seguinte:

1. Diffie-Hellman anônimo
2. Diffie-Hellman efêmero
3. Troca de chaves RSA
4. Fortezza

Em seguida, um servidor não anônimo pode solicitar um certificado do cliente. A mensagem **certificate_request** inclui dois parâmetros: certificate_type e certificate_authorities. A mensagem **certificate_type** indica o algoritmo de chave pública e seu uso:

- RSA, somente assinatura
- DSS, somente assinatura
- RSA para Diffie-Hellman fixo, assinatura somente utilizada para autenticação enviando certificado assinado com RSA
- DSS para Diffie-Hellman fixo, utilizado somente na autenticação.
- RSA para Diffie-Hellman efêmero.

- DSS para Diffie-Hellman efêmero
- Fortezza

O segundo parâmetro na mensagem **ceritficate_request** é uma lista dos nomes distintos de autoridades certificadoras aceitáveis.

A mensagem final na Fase 2 é a **server_done**. Ela é enviada pelo servidor com o objetivo de finalizar o *"hello"* do servidor e mensagens associadas. Após o envio, o servidor esperará pela resposta do cliente através de uma mensagem sem parâmetro.

Autenticação do cliente e troca de chaves - Fase 3

O cliente, ao receber a mensagem **server_done** verifica se o certificado fornecido é válido, caso necessário, e verifica se os parâmetros contidos no **server_hello* são aceitáveis. Se tudo ocorrer como esperado o cliente envia uma ou mais mensagens para o servidor.

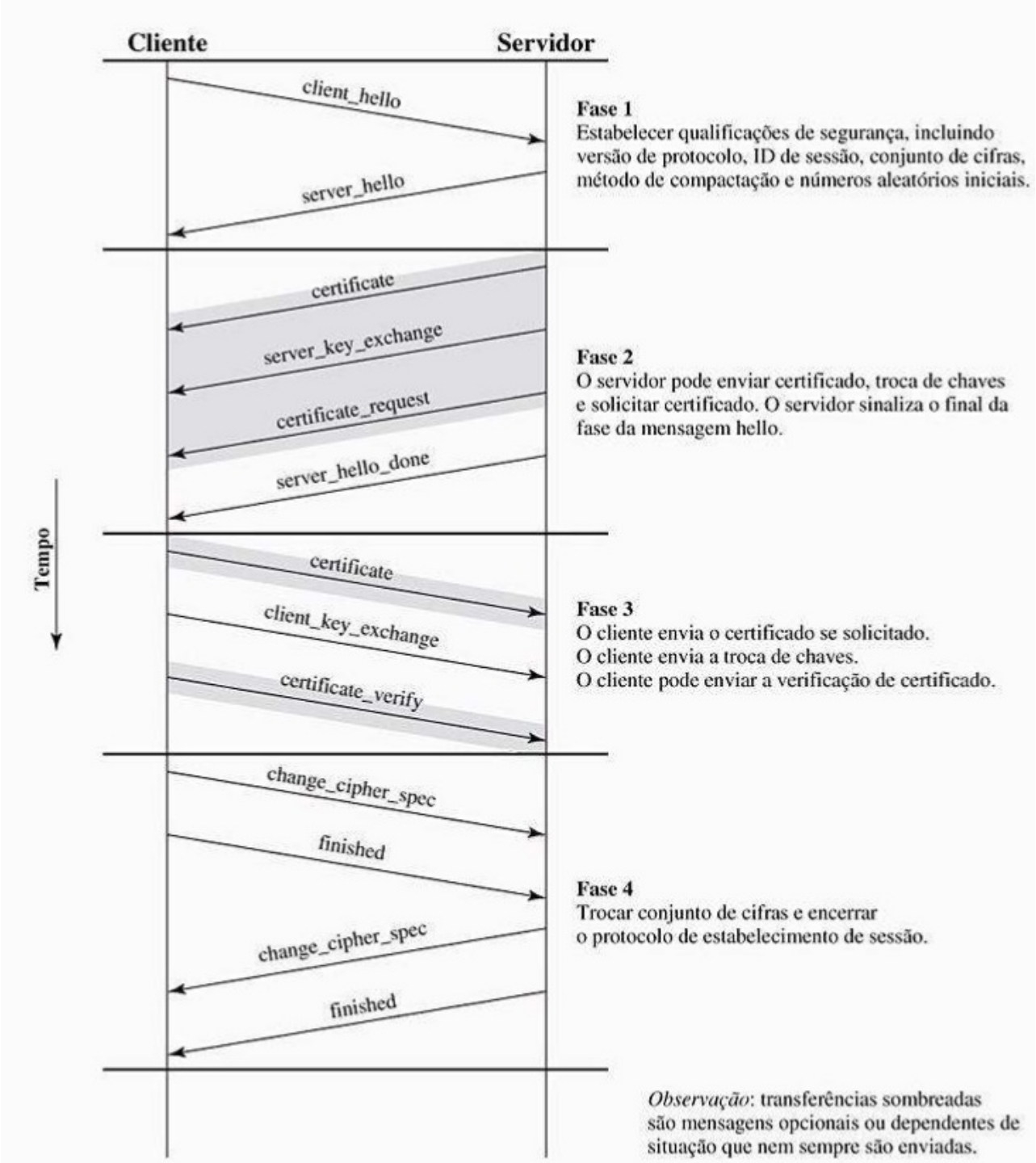
Se o servidor tiver solicitado um certificado, o cliente deverá envia-lo através da mensagem **certificate**. Caso o cliente não possuir um certificado este enviará um alerta **no_certificate**.Em seguida o cliente envia a mensagem **client_key_exchange**. O conteúdo da mensagem depende do algoritmo de criptografia acordado anteriormente.Por fim, o cliente envia uma mensagem **certificate_verify** para oferecer verificação explícita do seu certificado. Essa mensagem assina um código hash com base nas mensagens anteriores.

Término do Handshake - Fase 4

É nesta fase que se encerrá o processo de handshake. O cliente envia uma mensagem **change_cipher_spec** e copia o ChiperSpec pendente para CipherSpec atual. Após isso, envia uma mensagem de conclusão. A mensagem de conclusão verifica se os processos de troca de chaves e autenticação foram bem-sucedidos.

O cliente responde as duas mensagens enviando sua própria mensagem **change_cihpher_spec**, transfere o CipherSpec pendente para a atual e envia sua mensagem de conclusão e com isso o estabelecimento da conexão está completo tornando possível a troca de dados na camada de aplicação entre cliente e servidor.

Resumo



Resumo do funcionamento do SSL Handshake Protocol

Conexão SSL

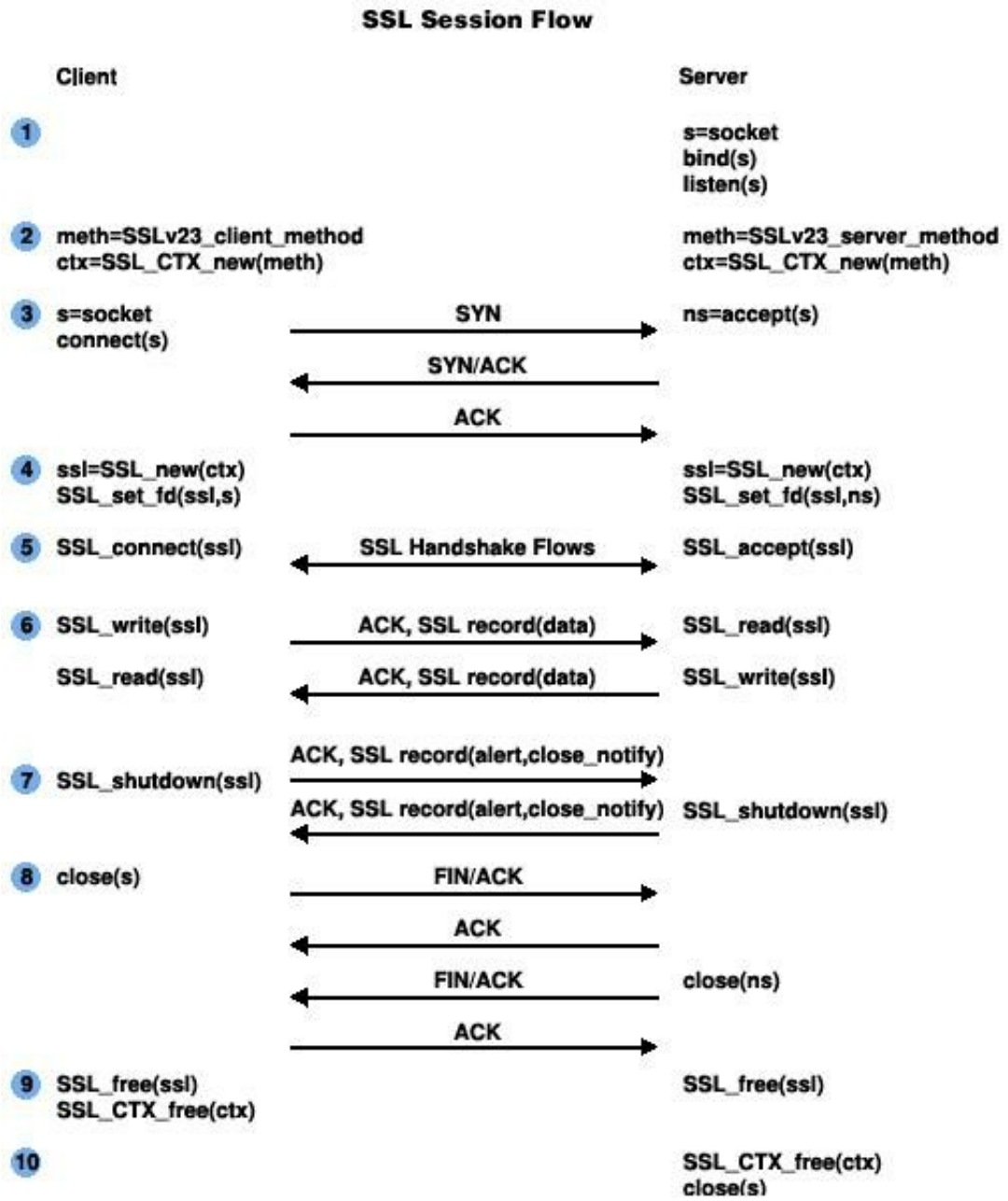
Uma conexão é um transporte que oferece um tipo adequado de serviço. No SSL as conexões são ponto a ponto (peer-to-peer). As conexões são transientes, ou seja, os dois pontos da conexão podem finaliza-la. Além disso, cada conexão está associada a apenas uma sessão.

Parâmetros que definem uma conexão SSL

- **Aleatório de servidor e aleatório de cliente (Server and client random):** É uma sequência de btyes preenchidas pelo servidor e cliente para cada conexão.
- **Segredo MAC de escrita do servidor (Server write MAC secret):** A chave secreta utilizada nas operações de MAC (Message Authentication Code) para dados enviados através do servidor.
- **Segredo MAC de escrita do cliente (Client write MAC secret):** A chave secreta utilizada nas operações de MAC para dados enviados pelo cliente.
- **Chave de escrita do servidor (Server write key):** A chave de criptografia convencional para dados criptografados pelo servidor e descriptografados pelo cliente.
- **Chave de escrita do cliente (Client write key):** A chave de criptografia convencional para dados criptografados pelo cliente e descriptografados pelo servidor.
- **Vetores de inicialização (Initialization vectors):** Se uma cifra de bloco em modo CBC (Cypher Block Chaining – Criptografia de Blocos Encadeados) for utilizado é necessário manter um vetor de inicialização (IV) para cada chave. Esse campo é inicializado primeiro pelo Protocolo de Estabelecimento de Sessão (Handshake Protocol) do SSL. Depois disso, o bloco de texto cifrado final de cada registro é preservado para uso como o IV com o registro seguinte.
- **Números de sequência (Sequence numbers):** Cada parte mantém números de sequência separados para mensagens transmitidas e recebidas para cada conexão. Quando uma parte envia ou recebe uma mensagem de mudança de especificação de cifra, o número de sequência apropriado é definido como zero. O número máximo permitido é `2^64 - 1`

Sessão SSL

Uma sessão SSL é uma conexão lógica entre um cliente ou uma aplicação de server sobre um "socket" do Protocolo de Controle de Transmissão (TCP) usando o protocolo SSL. Isso significa que uma aplicação SSL cria um "socket" TCP, inicia a conexão TCP e então inicia a sessão SSL sobre a conexão TCP. A sessão SSL é mapeada para o "socket" TCP, portanto, se o "socket" falhar, a sessão SSL irá falhar. É basicamente uma associação entre um cliente e um servidor estabelecida após o handshake (Handshake Protocol).



Estabelecimento de uma Sessão SSL

As sessões definem um conjunto de parâmetros de segurança criptográficos que são compartilhados com uma ou mais conexões. Esses parâmetros serão detalhados na próxima sub-sessão. A finalidade das sessões é evitar a negociação de novos parâmetros de segurança para cada conexão.

Parâmetros que definem uma sessão SSL

- **Identificador de sessão (session identifier):** É uma sequência de btyes preenchidas pelo servidor a fim de identificar um estado de sessão ativo ou que pode ser reativado.
- **Certificado da parte (peer certificate):** Um certificado X509.v3 (X.509 especifica, entre outras coisas, os formatos padrão para certificados de chaves públicas, listas de

certificados revogados, atribuir certificados, e um algoritmo de validação do caminho de certificação). Esse parâmetro pode ser nulo.

- **Método de compactação (compression method):** Estabelece o algoritmo utilizado para compactação os dados antes de criptografa-los
- **Especificação da cifra (cipher spec):** Estabelece o algoritmo de criptografia de dados podendo ser sem criptografia e um algoritmo de hash (MD5 ou SHA-1, por exemplo) utilizado para calcular o MAC (Message Authentication Code). Além disso, define as propriedades criptográficas, como hash size.
- **Segredo mestre (master secret):** Segredo (senha ou password) de 48 bytes conhecido pelo dois pontos da sessão (cliente e servidor).
- **É retomável (is resumable):** É um booleano que indica se a sessão poderá iniciar novas conexões

Diferenças entre SSL e TLS

Message Authentication Code (MAC)

Há duas diferenças entre os esquemas de MAC do SSLv3 e o TLS. Uma é o algoritmo real e a outra é o escopo do cálculo do MAC. O TLS utiliza o algoritmo HMAC definido na RFC 2104. Definição do HMAC:

$$HMACK(M) = H[(K^{+} \oplus opad)||H[(K^{+} \oplus ipad)]||M]$$

Em que:

- **H =** função de hash embutida (para TLS, MD5 ou SHA-1)
- **M =** entrada de mensagem para HMAC
- K^{+} **=** chave secreta complementada com zeros à esquerda, de modo que o resultado é igual ao tamanho do bloco de código de hash (para MD5 e SHA-1, tamanho de bloco = 512 bits)
- **ipad =** 00110110 (36 em hexadecimal) repetido 64 vezes (512 bits)
- **opad =** 01011100 (5C em hexadecimal) repetido 64 vezes (512 bits)

SSLV3 utiliza o mesmo algoritmo do TLS porém em vez de realizar o XOR dos bytes de complementação com a chave secreta complementada até o tamanho bloco ele concatena os bytes de complementação com a chave secreta. O nível de segurança é praticamente o mesmo para ambos.

O cálculo do MAC do TLS abrange todos os campos cobertos pelo cálculo do SSLv3, mais o campo TLSCompressed.version (versão do protocolo que está sendo utilizada):

$$HMAC\_hash(MAC\_write\_secret, seq\_num \parallel TLSCompressed.type \parallel TLSCompressed.version \parallel TLSCompressed.length \parallel TLSCompressed.fragment)$$

Código de alerta

No TLS é possível enviar uma mensagem de *"No Certificate"* enquanto no SSL não é necessário uma mensagem separada para tal. Outra diferença está no fato do TLS admitir todos os códigos de alerta presentes no SSLv3 exceto **no_certificate**. A seguir os códigos presentes tanto no SSL quanto TLS:

- **descryption_failed:** Um texto cifrado de maneira incorreta, seja por não ser múltiplo par do tamahn do bloco ou seus valores de complementação estão incorretos.
- **record_overflow:** Um registro do TLS recebeu algum payload (texto cifrado) com tamanho maior que \begin{math2^{14 + 2.048 bytes, ou o texto cifrado foi descriptografado para um tamanho maior que 2^{14 + 1.024 bytes.
- **unknown_ca:** Quando algum certificado válido ou alguma cadeia parcial não forem confiáveis pela CA.
- **access_denied:** Quando o emissor do certificado válido não prosseguir com a negociação.
- **decode_error:** Quando uma mensagem não foi decodificada com sucesso seja por algum campo fora do intervalo especificado ou o tamanho da mensagem estava incorreto.
- **export_restriction:** foi detectada uma negociação que não estava em conformidade com as restrições de exportação sobre o tamanho da chave.
- **protocol_version:** a versão do protocolo que o cliente tentou negociar é reconhecida, mas não aceita.
- **insufficient_security:** ocorre quando o servidor exige uma cifra mais segura do que as admitidas pelo cliente.
- **internal_erro:** algum erro interno que ocorreu que impede continuar.

A seguir os códigos presentes apenas no TLS:

- **decrypt_erro:** Quando uma operação criptográfica de estabelecimento de sessão falhou, o que inclui ser incapaz de verificar uma assinatura, descriptografar uma troca de chaves ou validar uma mensagem acabada.
- **user_canceled:** Ocorre quando por algum motivo não relacionado a falha no protocolo a conexão é cancelada.
- **no_renegotiation:** enviado por um cliente em resposta a uma solicitação "hello" ou pelo servidor em resposta a um "hello" do cliente após o estabelecimento inicial da sessão. Uma mensagem dessas normalmente resultaria em renegociação, mas esse alerta indica que o emissor não é capaz de renegociar. Essa mensagem é sempre uma advertência.

Troca de chaves

O TLS admite todas as técnicas de troca de chaves do SSLv3, com exceção do Fortezza.

Algoritmos de criptografia simétrica

O TLS inclui todos os algoritmos de criptografia simétrica encontrados no SSLV3, com exceção do Fortezza.

Verificação do certificado de mensagens

No TLS a verificação de mensagens já está contida na mensagem do handshake enquanto no SSL é necessário um procedimento para passar a verificação de certificado de mensagem.

Encerramento

NO TLS a mensagem de encerramento é criada através de um ***"output"*** do PRF com ajuda da mensagem de ***"client finished"*** ou *"server finished". Já no SSL a mensagem de encerramento é criada de maneira igual a criação de chaves, ou seja, utilizando cifra e parâmetros de informação.

Principais ataques

Abaixo listamos alguns ataques conhecidos do protocolo SSL:

- Ataque de Renegociação
- Ataque de Rollback de versões
- Ataque BEAST
- Ataque CRIME
- Ataque BREACH
- Ataque Padding
- Ataque POODLE
- Ataque RC4
- Ataque Truncamento
- Heartbleed Bug

Dos citados iremos abordar com maior detalhe o ataque de renegociação (SSL Renegotiation Attack) e o BEAST (Browser Exploit Against SSL/TLS Attack).

Solução

Uma maneira de corrigir a vulnerabilidade de renegociação para SSLv3 é desabilitar completamente a renegociação no lado do servidor.

Para o TLS existe uma proposta para incluir informações de handshakes anteriores durante o processo de renegociação.

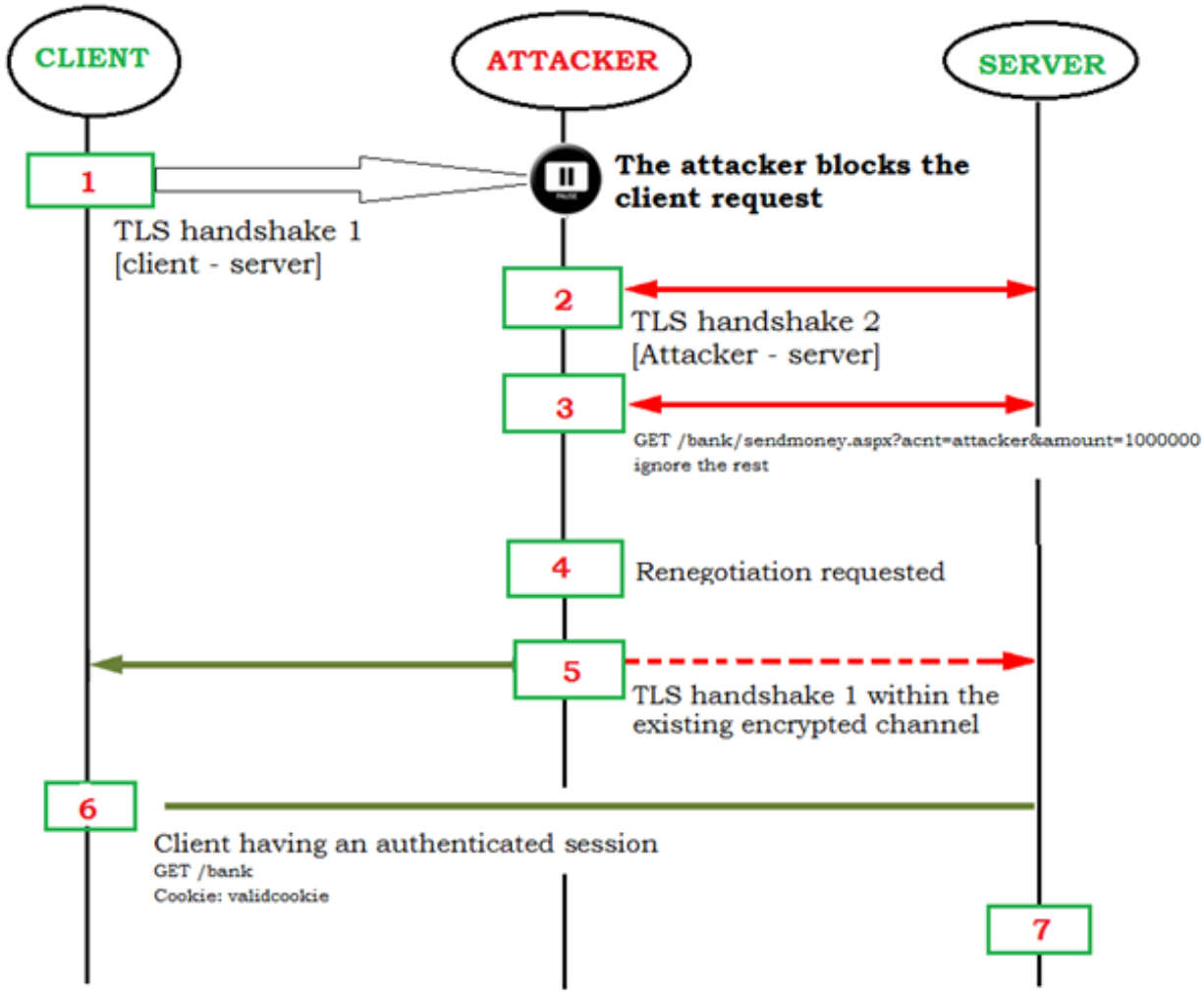
SSL Renegotiation Attack

Essa vulnerabilidade ocorre durante o processo de renegociação do SSL que permite ao invasor injetar texto simples em pedidos da vítima. Permitindo ao atacante injetar comandos em uma sessão HTTPS, realizar o dowgrade de uma conexão que utilize HTTPS para uma conexão HTTP, injetar respostas personalizadas, negar serviço, etc.

A renegociação SSL é útil para manter a conexão com o cliente já estabelecida para, por exemplo, não perder dados e criar uma nova sessão a fim de estabelecer, por exemplo, uma sessão com autenticação do cliente. Para facilitar a compreensão pense que você esteja utilizando um site de compras online que utilize o SSL, ou seja, HTTP. Inicialmente enquanto você procura por seus itens e os adiciona ao carrinha você está navegando de forma anônima mas quando você decidir comprar será necessário realizar a sua autenticação no site e com isso sua conexão SSL precisa ser ajustada para permitir a autenticação. Essa mudança deverá manter as informações as informações recolhidas antes da autenticação (por exemplo os itens adicionados ao carrinho) e com por isso é criado uma nova sessão utilizando a conexão já existente. Essa renegociação pode ser pedida pelo cliente ou pelo servidor a qualquer momento. Para o cliente solicitar a renegociação é necessário enviar uma mensagem "Client Hello" no canal criptografado já estabelecido e o servidor deve responder com um "Server Hello" e, em seguida, a negociação segue o processo de handshake normal. Já para o servidor solicitar é necessário enviar uma mensagem "Hello Request" e o cliente deve responder com a mensagem "Client Hello" e o processo de handshake prossegue da maneira padrão.

Exemplo na prática

Para explicar iremos utilizar o fluxo simulado de conexão SSL retratado na figura abaixo. Os números presentes na figura casam com os números presentes na lista ordenada abaixo.

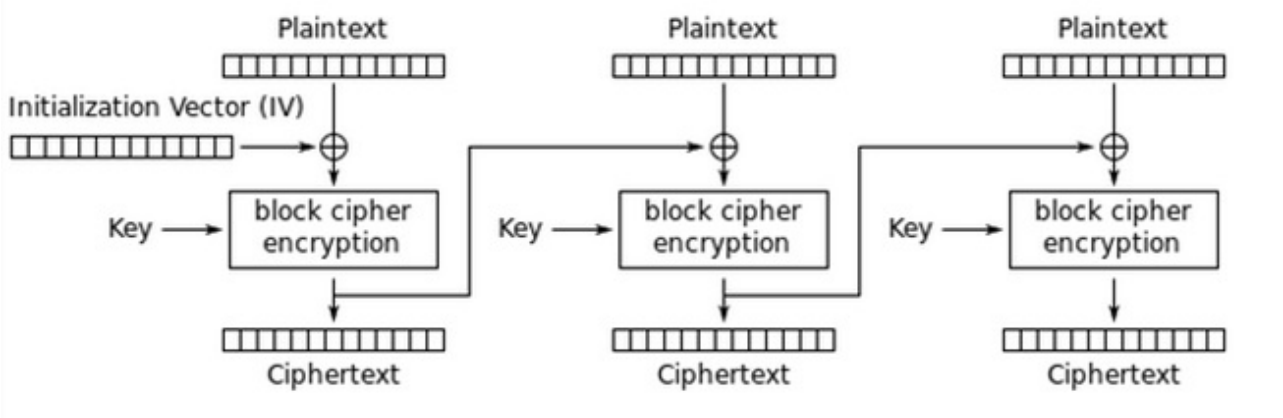


Fluxo simulado de um ataque de renegociação SSL

1. Assumimos que o cliente deseja se conectar ao site seguro de um banco por exemplo. Nesse passo é iniciada o processo de aperto de mão (Handshake)
2. O atacante bloqueia e guarda a requisição do cliente e cria um novo handshake até sua finalização com o servidor
3. O atacante realiza uma solicitação https do tipo GET para o servidor pedindo para transferir alguma quantia de dinheiro para sua conta.
4. O servidor pede por renegociação para entrar em modo autenticado
5. O atacante encaminha os pacotes guardados pelo passo 2 para o processo de renegociação e aguarda sua finalização.
6. O cliente agora está autenticado e possui um cookie válido.
7. Como o processo de autenticação foi concluído com sucesso o servidor assume que a solicitação de transferência solicitada na etapa 3 é valida, ou seja, foi realmente enviada pelo cliente prosseguindo com a realização da transferência.

Browser Exploit Against SSL/TLS Attack

Esse ataque foi revelado na Ekoparty Security Conference em 2011. BEAST é baseado num tipo de ataque de criptografia chamado de “chosen plain text attack.”



Cipher-block chaining (Modo CBC)

Foi percebido que no TLS 1.0 e corrigido no TLS 1.1, mas o TLS 1.1 não havia grande aceitação até a demonstração desse ataque, quando lidava com múltiplos pacotes, permitia que os pacotes seguintes usassem um vetor de inicialização que é o ultimo bloco de texto cifrado do pacote anterior. Em outras palavras, um atacante que conseguir capturar o tráfego cifrado pode notar o vetor de inicialização usado no cookie de sessão (Porque a localização dos cookies são previsíveis). Além disso, um atacante poderá coletar os vetores para cada "record" apenas monitorando a rede. Então se o atacante conseguir "adivinhar" o texto da mensagem, ele poderá adivinhar o cookie de sessão e ver se o texto cifrado é equivalente. Já que esse é um ataque de "homem do meio", o atacante pode misturar seu tráfego com o de sua vítima para ver os resultados.

RC4 como uma cifragem de stream é imune ao ataque BEAST. Portanto ele foi largamente utilizado para mitigar o ataque BEAST no lado do server. Em 2013 pesquisadores encontraram mais fraquezas no RC4 e, portanto, não era mais recomendado sua utilização.

Exemplo na prática

Considerando uma mensagem: FSDgfe254dsfsFDS=342SDFBbfd. Esse texto será "XORado" com o vetor de inicialização (que é o texto cifrado do bloco anterior). O atacante tem o valor do vetor de inicialização em suas mãos. Agora se ele conseguir "prever" o texto e seu XOR com o vetor, ele pode checar se ele corresponde ao valor do texto cifrado. Obviamente não é fácil prever tal valor, mas ele pode "chutar" um caractere por vez. Por exemplo: "FSDgfe254" pode ser adivinhado tentando diferentes caracteres. Uma vez que o primeiro é recuperado, ele troca o ataque para o próximo caractere. Dessa forma ele pode adivinhar um caractere por vez.

Limitações desse ataque:

- O atacante deve estar na mesma rede e agir como o "homem do meio".
- O atacante deve modificar o tráfego para ver os resultados de comparação; como resultado, múltiplos requisitos deverão ser enviados no processo.
- O atacante só pode adivinhar um bloco por vez.

Solução

Essa é uma vulnerabilidade nos blocos de cifragem que usam o modo de operação CBC (Cipher Block Chaining). Foi identificada no TLS 1.0. No entanto foi direcionado ao TLS 1.1 e TLS 1.2

pelo uso de "vetores de inicialização explícitos" para cada bloco. Consequentemente, TLS 1.1 e TLS 1.2 não são expostos a esse ataque.

Alguns navegadores tem tentado implementar uma solução para corrigir essa vulnerabilidade enquanto continuam compatíveis com o protocolo SSL 3.0/TLS 1.0.

- **Safari:** Mesmo lançando uma mitigação, escolheu manter esse protocolo desativado por padrão.
- **Google:** Updates no Chrome 16 ou posterior.
- **Microsoft:** Aplicado no patch MS12-006
- **Mozzilla:** Ipdate no Firefox 10 ou posterior.

SSL na prática

Fizemos uma aplicação web simples com JavaServerFaces (JSF) e utilizamos como servidor apache tomcat 8. Utilizamos essa combinação pelo domínio da equipe nas duas tecnologias.

Gerando os certificados

1. Criando a Autoridade de Certificação

```
openssl req -new -keyform PEM -outform PEM -passin env:SSLPASS -config /home/
fulano/trabalho_seguranca_chaves/CaServer/openssl.cnf -out /home/fulano/traba
lho_seguranca_chaves/CaServer/cacert.req -key /home/fulano/trabalho_seguranca
_chaves/CaServer/cacert.key -sha1 \end{minted
```

2. Criando um certificado para o servidor da aplicação

```
openssl req -new -keyform PEM -outform PEM -passin env:SSLPASS -config /home/
fulano/trabalho_seguranca_chaves/CaServer/openssl.cnf -out /home/fulano/traba
lho_seguranca_chaves/CaServer/req/server_request.pem -key /home/fulano/trabal
ho_seguranca_chaves/CaServer/keys/server_request_key.pem -sha1
```

3. Assinando com a Autoridade de Certificação o certificado da aplicação

```
openssl ca -batch -passin env:SSLPASS -notext -config /home/fulano/trabalho_s
eguranca_chaves/CaServer/openssl.cnf -name server_ca -in "/home/fulano/trabal
ho_seguranca_chaves/CaServer/req/server_request.pem" -days 365 -preserveDN -m
d sha1
```

4. Criando um certificado para o cliente da aplicação

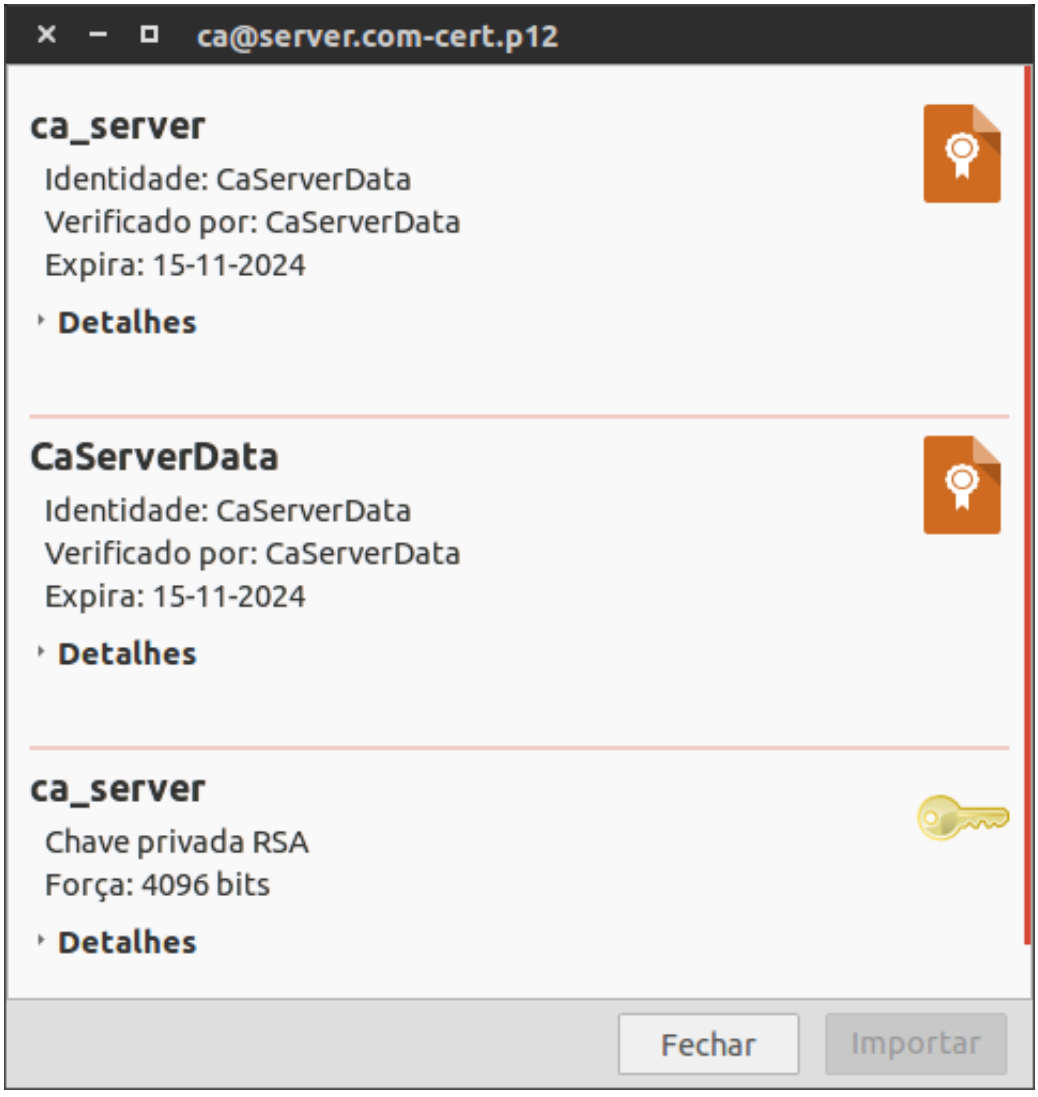
```
openssl req -new -keyform PEM -outform PEM -passin env:SSLPASS -config /home/
fulano/trabalho_seguranca_chaves/CaServer/openssl.cnf -out /home/fulano/traba
lho_seguranca_chaves/CaServer/req/client_request.pem -key /home/fulano/trabal
ho_seguranca_chaves/CaServer/keys/client_request_key.pem -sha1
```

5. Assinando com a Autoridade de Certificação o certificado do cliente

```
openssl ca -batch -passin env:SSLPASS -notext -config /home/fulano/trabalho_seguranca_chaves/CaServer/openssl.cnf -name client_ca -in "/home/fulano/trabalho_seguranca_chaves/CaServer/req/client_request.pem" -days 365 -preserveDN -md -sha1
```

6. Exportando o certificado da Autoridade Certificadora para utilizar no tomcat com formato pkcs12

```
openssl pkcs12 -export -out "/root/ca@server.com-cert.p12" -in "/home/fulano/trabalho_seguranca_chaves/CaServer/cacert.pem" -inkey "/home/fulano/trabalho_seguranca_chaves/CaServer/cacert.key" -certfile /home/fulano/trabalho_seguranca_chaves/CaServer/cacert.pem -name "ca_server"
```

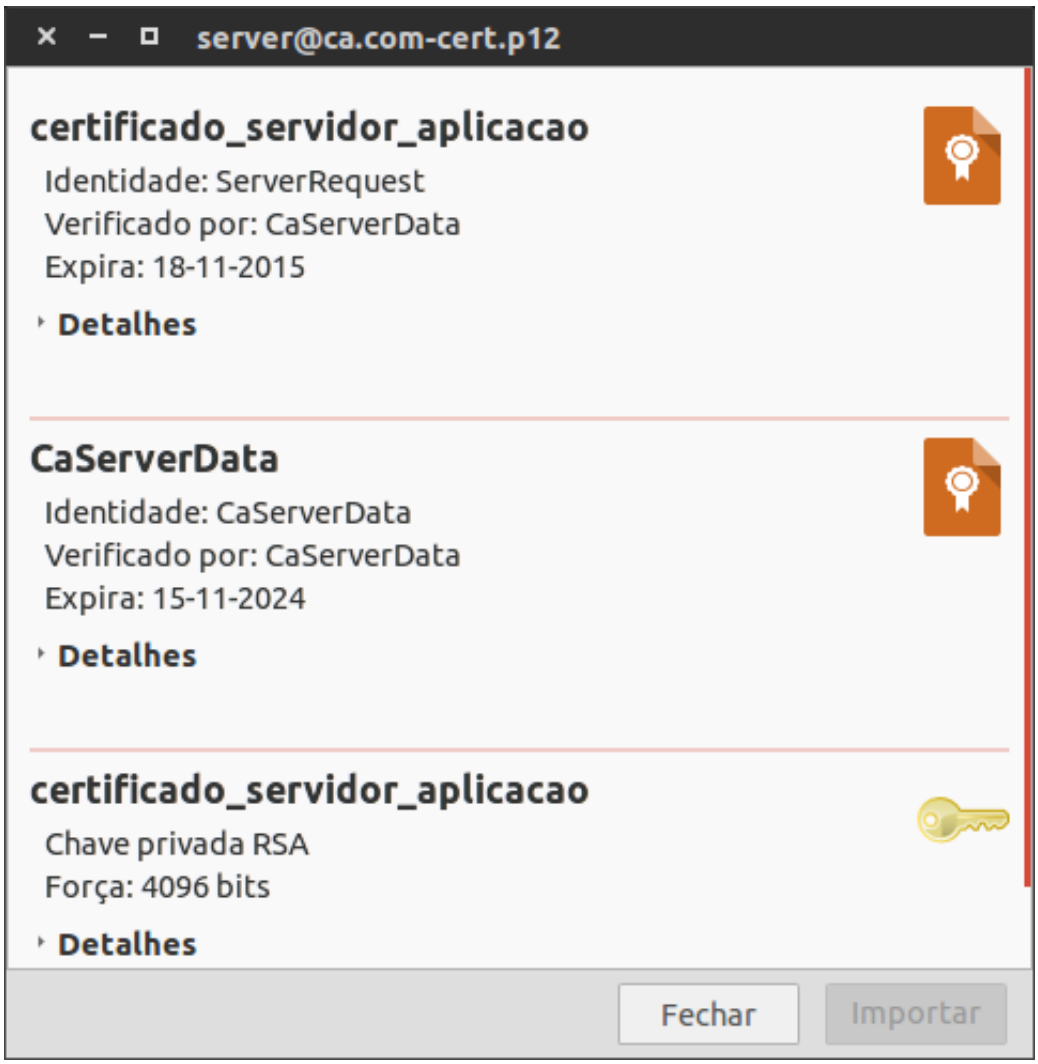


Certificado da Autoridade Certificado exportado.

7. Exportando o certificado do servidor da aplicação no formato pkcs12

```
openssl pkcs12 -export -out "/home/fulano/tomcat-8/keys/server@ca.com-cert.p12" -in "/home/fulano/trabalho_seguranca_chaves/CaServer/certs/server_request.
```

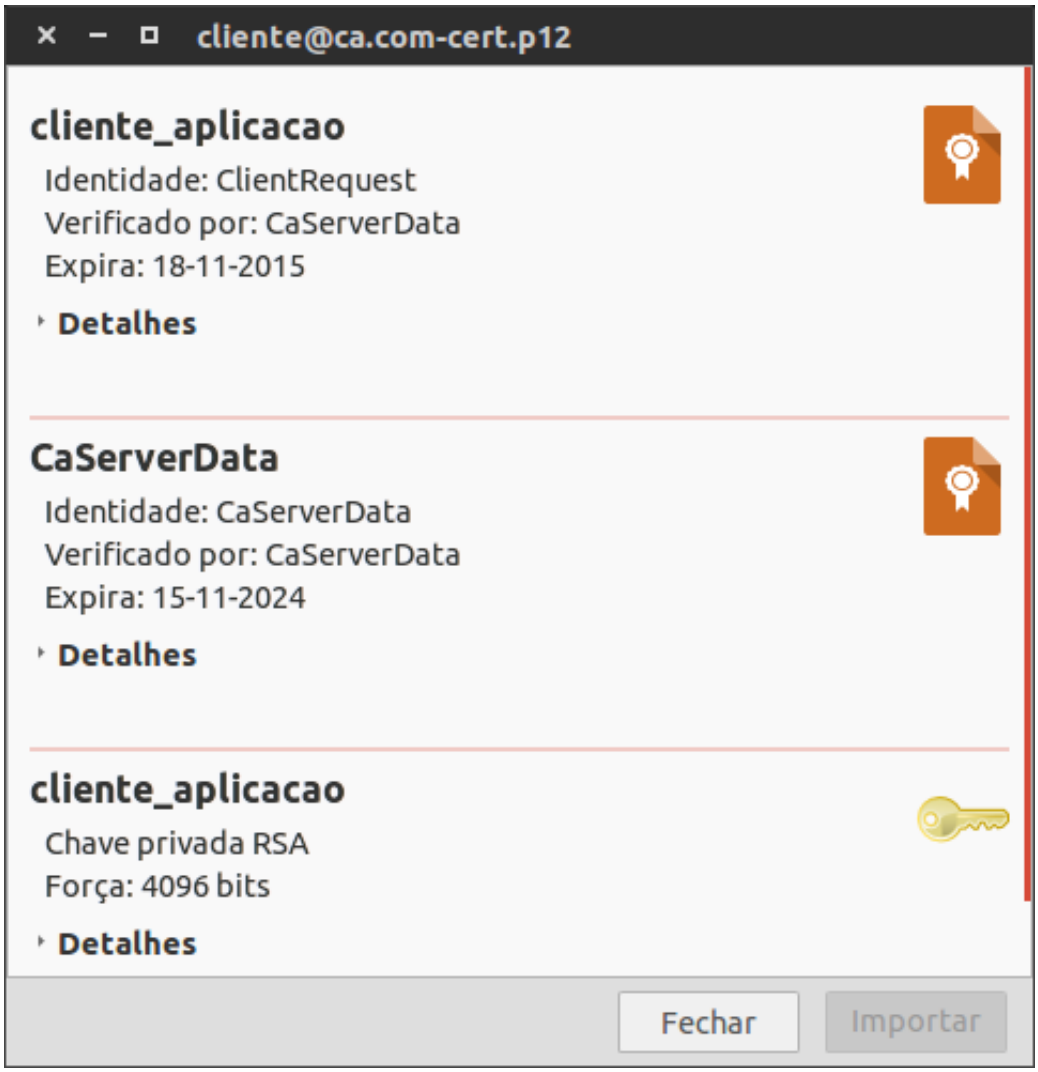
```
pem" -inkey "/home/fulano/trabalho_seguranca_chaves/CaServer/keys/server_request_key.pem" -passout env:P12PASS -passin env:SSLPASS -certfile /home/fulano/trabalho_seguranca_chaves/CaServer/cacert.pem -name "certificado_servidor_aplicacao"
```



Certificado do servidor exportado.

8. Exportando o certificado do cliente da aplicação no formato pkcs12

```
openssl pkcs12 -export-out "/home/fulano/keys/cliente@ca.com-cert.p12" -in "/home/fulano/trabalho_seguranca_chaves/CaServer/certs/client_request.pem" -inkey "/home/fulano/trabalho_seguranca_chaves/CaServer/keys/client_request_key.pem" -passout env:P12PASS -passin env:SSLPASS -certfile /home/fulano/trabalho_seguranca_chaves/CaServer/cacert.pem -name "cliente_aplicacao"
```



Certificado do cliente exportado.

Configurando o servidor tomcat 8

Baixamos o tomcat 8 do site oficial (http://tomcat.apache.org/download-80.cgi) e adicionamos ao arquivo de configuração */home/tomcat-8/conf/server.xml* a seguinte configuração:

```
<Connector port="8443" protocol="HTTP/1.1" SSLEnabled="true" SSLEngine="on" SSLVeri
fyDepth="2" sslProtocol="TLS" clientAuth="true"
    scheme="https" secure="true" keystoreType="PKCS12" keystoreFile="/home/fulano/t
omcat-8/keys/server@ca.com-cert.p12"
    keystorePass="scsc" truststoreType="PKCS12" truststoreFile="/home/fulano/tomcat
-8/keys/ca@server.com-cert.p12" truststorePass="scsc"/>
```

Explicação das tags:

- **port:** Porta para subir o serviço em modo seguro (https)
- **protocol:** Protocolo a ser utilizado
- **SSLEnabled:** Habilita o ssl
- **SSLVerifyDepth:** Especifica o número máximo de emissores de certificados intermediários.

- **sslProtocol:** Habilita o ssl
- **clientAuth:** Habilita a autenticação obrigatória via certificado pelo cliente.
- **schema:** Define o retorno da chamada request.getScheme()
- **secure:** Define o retorno da chamada request.isSecure()
- **keystoreType:** Especifica o tipo de certificado do certificado do servidor, no caso PKCS12 que unifica certificado e chave privada.
- **keystoreFile:** Local do certificado do servidor.
- **keystorePass:** Senha do certificado do servidor.
- **truststoreType:** Especifica o tipo de certificado de confiança (Autoridade Certificadora), no caso PKCS12 que unifica certificado e chave privada.
- **truststoreFile:** Local do certificado de confiança.
- **truststorePass:** Senha do certificado de confiança.

Execução da aplicação

Utilizando o java para simular o browser

Primeiramente configuramos o certificado do cliente utilizado no handshake com o servidor. Após isso mandamos uma solicitação get na página *https://localhost:8443/SiteSeguro/faces/verificar_autenticacao.xhtml* *(https://localhost:8443/SiteSeguro/faces/verificar_autenticacao.xhtml)* e se tudo ocorrer como esperado será retornado a página solicitada.

```
import org.apache.commons.httpclient.HttpClient;
import org.apache.commons.httpclient.URI;
import org.apache.commons.httpclient.methods.GetMethod;

public class App {

    public static void main(String[] args) {

        // Configurando certificado que será utilizado para simular cliente
        {
            System.setProperty("javax.net.ssl.keyStore",
                    "/home/thiago/Programas/apache-tomcat-8.0.15/keys/cliente@ca.co
m-cert.p12");
            System.setProperty("javax.net.ssl.keyStoreType", "pkcs12");
            System.setProperty("javax.net.ssl.keyStorePassword", "scsc");
        }

        try {
            HttpClient client = new HttpClient();
            GetMethod method = new GetMethod();
            method.setURI(new URI(
                    "https://localhost:8443/SiteSeguro/faces/verificar_autenticacao
.xhtml",
```

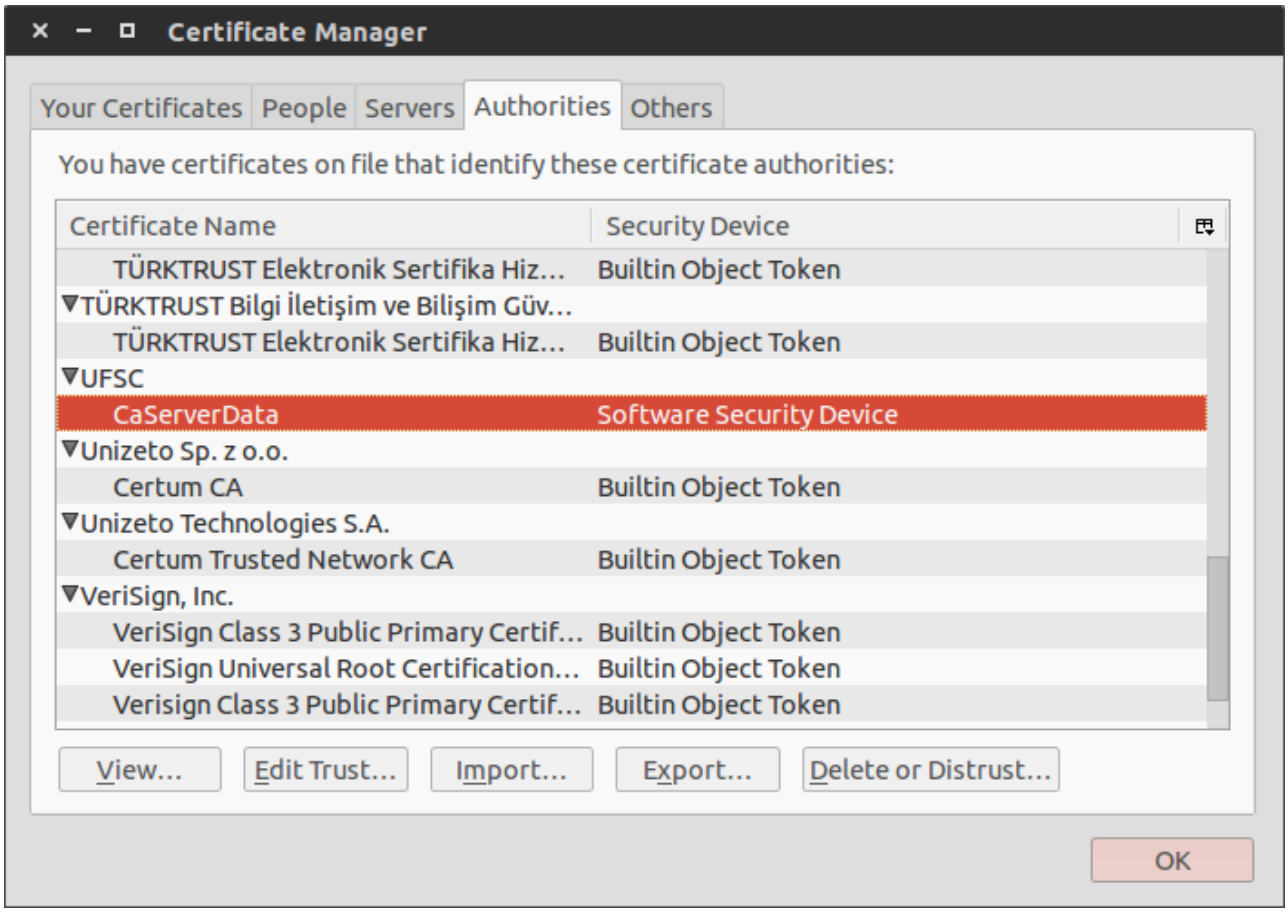
```
                    true));
            client.executeMethod(method);
            System.out.println(method.getResponseBodyAsString());
        }
         catch (Exception e) {
            System.out.println(e.getCause());
            e.printStackTrace();
        }
    }
}
```

Output:

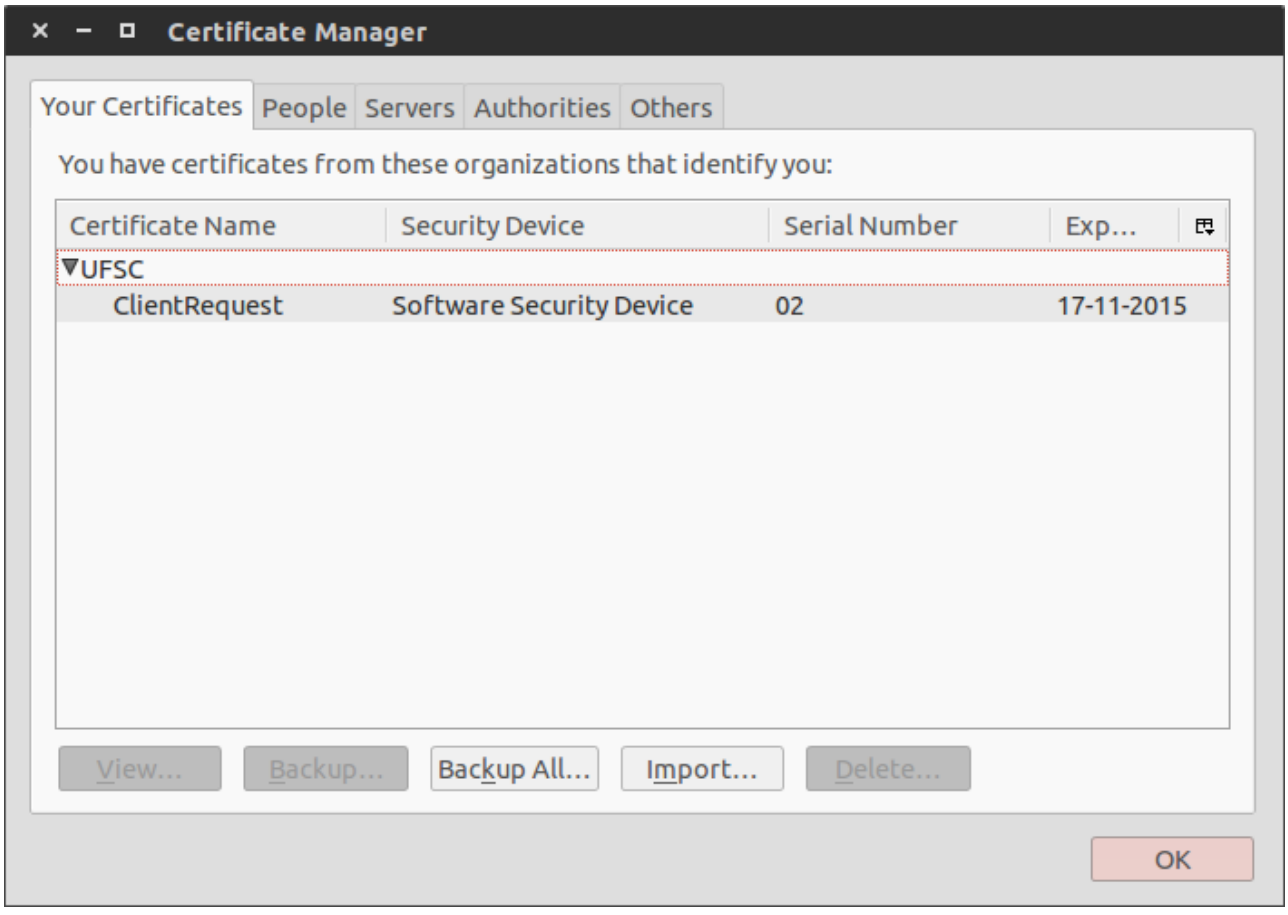
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://w
ww.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <title>Site Seguro</title>
    </head>
    <body>
        <h3>Cliente autenticado utilizando um certificado válido.</h3>
    </body>
</html>
```

Utilizando o browser

Primeiramente foi necessário configurar os certificados dentro do navegador Firefox.

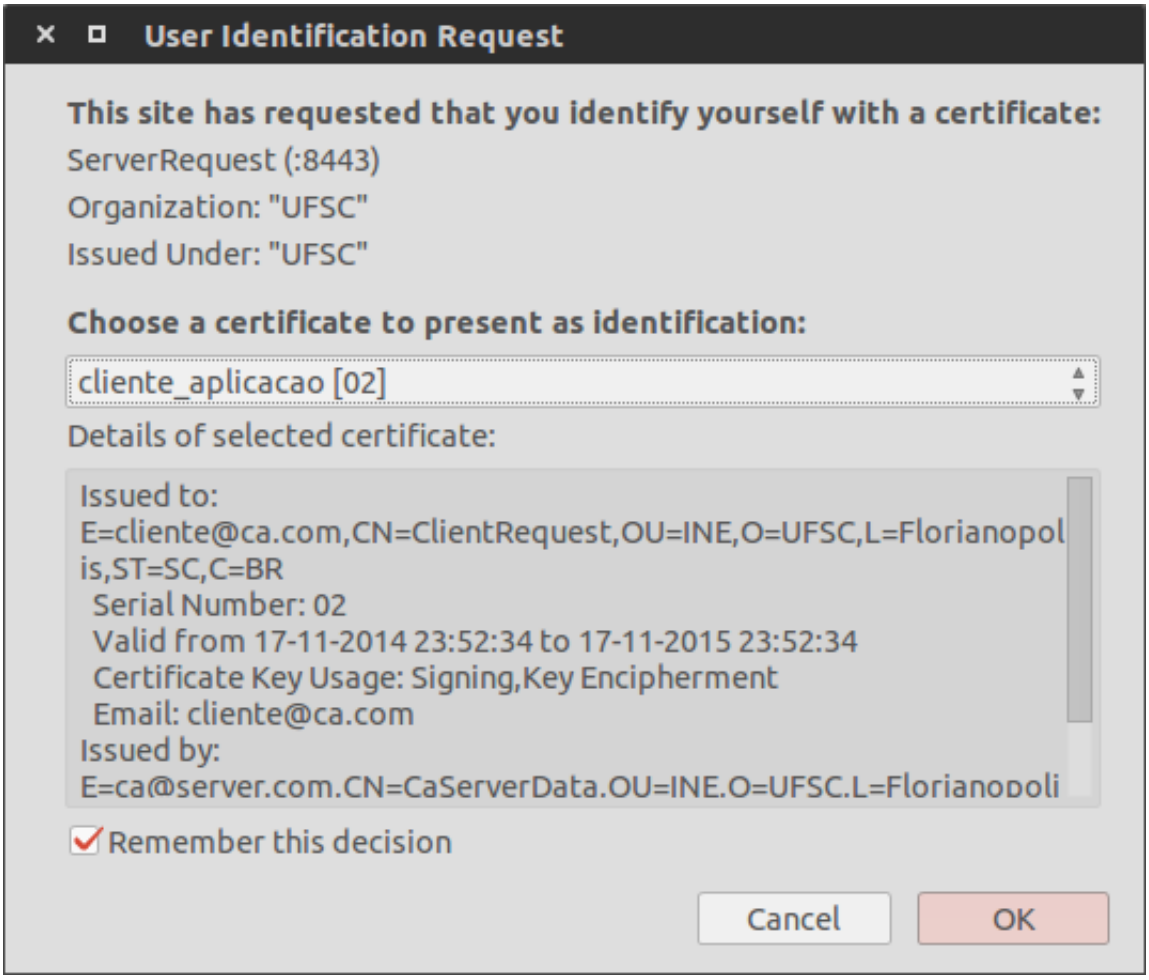


Configuração do certficado da autoridade certificado.



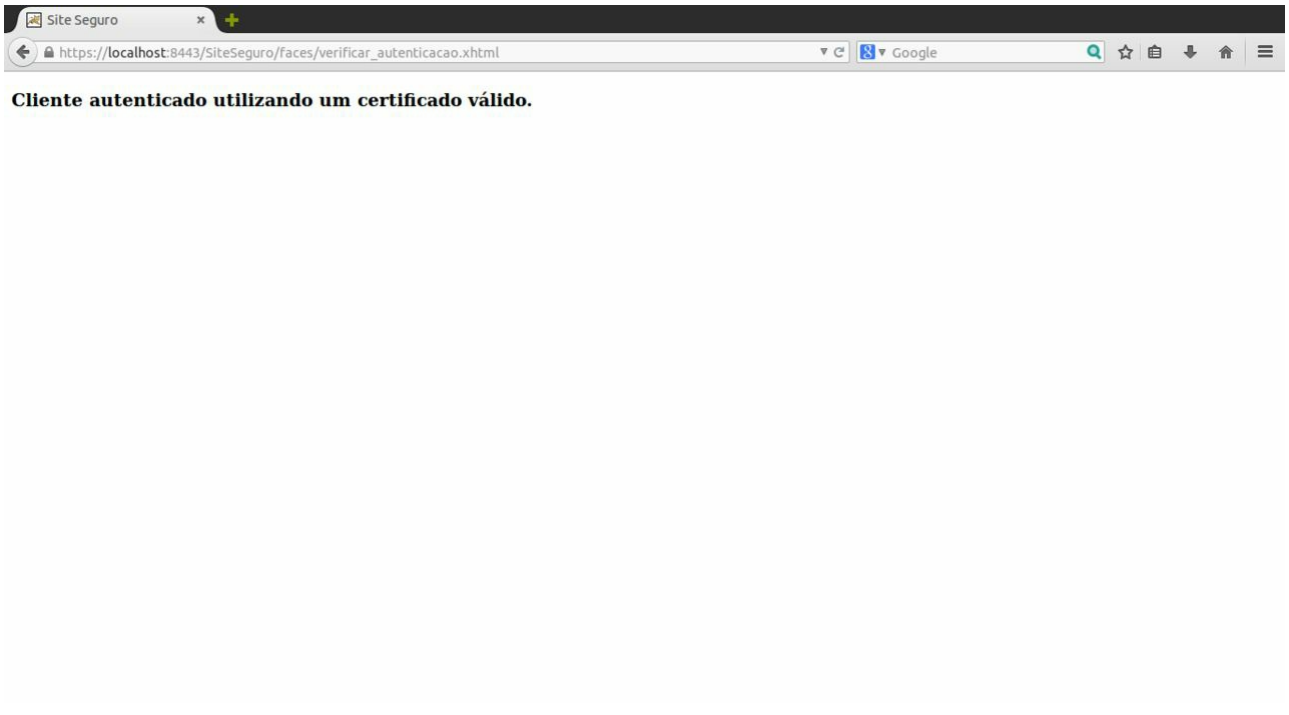
Configuração do certficado do cliente.

Ao abrirmos o browser na página é solicitado o ceritifcado do cliente o qual desejamos utilizar para autenticação.



Solicitação do certificado para autentição.

Após a escolha do certificado o servidor retornará a página solicitada se a autenticação for bem sucedida.



Página retornada pelo servidor em caso de autenticação bem sucedida.