Autenticação de entidades numa aplicação Cliente-Servidor

Segurança Informática nas Organizações

Tiago Melo 89005

João Nogueira 89262

Introdução

Este relatório visa explicar e descrever as decisões tomadas e mecanismos desenhados quando desenvolvemos uma aplicação cliente servidor com foco na autenticação de cada um dos participantes.

Iremos abordar a teoria por detrás da implementação, a máquina de estados que cada uma das entidades assume no seu ciclo de vida e como foram implementadas as noções teóricas.

Para além disso, é de salientar que, apesar do código do projeto anterior estar completamente funcional, foi utilizado o código fornecido pelos docentes, daí que nenhuma das mensagens esteja encriptada.

Acordo de Protocolos

Posto isto, a autenticação de cada uma das entidades pode ser feita da seguinte forma:

Autenticação	
Cliente	**Servidor**
Username e password	Certificados X.509
Cartão de Cidadão	

Isto pressupõe um acordo entre o cliente e o servidor sobre o método de authenticação usado para validar o cliente. Daí que a sua interação se inicie com uma troca de mensagens do tipo ‘HANDSHAKE’, onde irá ser acordado o método de autenticação do cliente.

O cliente envia o protocolo que irá seguir para se autenticar, o cliente verifica se o mesmo é suportado e, posteriormente, é desencadeado o processo de autenticação.

```
2019-12-04 11:06:13 melo root[13473] DEBUG Send: {'type': 'HANDSHAKE', 'method': 8}
2019-12-04 11:06:13 melo root[13473] DEBUG Received: b'{"type": "OK"}\r\n'
2019-12-04 11:06:13 melo root[13473] DEBUG Send: {'type': 'ACCESS_REQ', 'user': 'tiago'}
```

Screenshot 1: Acordo de Protocolo a seguir - ClientSide

Como a tabela indica, são suportados dois métodos para a autenticação do cliente:

- Username e Password, através do protocolo MS-CHAP.
- Cartão de Cidadão, através do uso da API PyKCS.

Autenticação do Cliente

Como já foi mencionado, o cliente possui duas formas de se autenticar: via Cartão de Cidadão com o uso de PyKCS ou via username e password, através do protocolo MS-CHAP. Ambos os protocolos assentam em aproximações Desafio Resposta.

Desafio Resposta – Conceito

As credenciais usadas para autenticar uma entidade não são constantes e dependem de um desafio gerado aleatoriamente, enviado pelo autenticador.

1. O sujeito a autenticar contacta o autenticador
2. O autenticador gera um desafio (NONCE)
3. O sujeito transforma o desafio usando algo que só ele sabe (como uma senha no MS-CHAP)
4. O resultado é enviado ao autenticador
5. O autenticador valida o resultado do desafio através da comparação do resultado enviado pelo sujeito com o seu cálculo feito localmente, usando o mesmo método.

Esta aproximação não só garante proteção contra MiTM ao não expor as credenciais num canal de comunicação, mas também permite ao autenticador escolher a transformação e a complexidade do desafio. Obriga contudo, à entidade autenticadora que faça uma boa gestão dos NONCEs gerados, visto que o contrário poderia criar uma vulnerabilidade a ataques por repetição. Uma forma comum e simples de resolver este problema é de concatenar um número gerado aleatoriamente com um *counter* ou um *timestamp*. Desta forma, mesmo que um número seja gerado aleatoriamente duas vezes, a outra parte do NONCE é, garantidamente, diferente.

MS-CHAP

O Microsoft Challenge-Handshake Authentication Protocol (ou *triple handshake*) oferece uma maior robustez que o seu predecessor PAP e que a sua versão original CHAP visto que não só não envia a palavra-passe do cliente que se deseja autenticar em *plain text* (ao contrário do PAP, que, quando surgiu, era uma solução viável, visto que todas as redes eram privadas e por fios, o que significa que havia menos chances de MiTM) como não a envia de todo; envia sim uma *fingerprint* da password+NONCE. Muito mais seguro.

O protocolo é desencadeado pelo envio do username do cliente, algo que, normalmente, é informação pública. O servidor, quando recebe esta mensagem, verifica que se o username está na sua base de dados/registos locais. Caso não esteja, pode cessar a comunicação, visto que este username nem sequer se encontra registado; caso contrário, gera um NONCE (número aleatório que é usado uma vez e, neste caso, o challenge) e envia-o ao cliente. O cliente, por sua vez, quando recebe este NONCE, concatena-o à sua password e calcula a *hash* do resultado. O servidor executa a mesma operação com a palavra-passe local do utilizador. Por fim, o cliente envia a sua resposta ao challenge e o servidor compara-a ao seu cálculo executado anteriormente: se ambos forem iguais, consideramos que o cliente é quem ele afirma ser. Caso contrário, não, e podemos cessar a comunicação.

Implementação

Como já foi mencionado, a comunicação entre cliente e servidor começa com uma troca de mensagens do tipo ‘HANDSHAKE’. O cliente assume o estado STATE_HANDSHAKE. No valor desta mensagem é também enviada a Macro (de valor igual tanto no cliente como no servidor, naturalmente) correspondente à autenticação por username e password (ver screenshot 1). O servidor verifica se a operação é suportada e responde com um ‘OK’.

Face a esta resposta, o cliente avalia o seu método de autenticação escolhido e atualiza o seu estado para STATE_ACCESS_REQ. Depois, envia uma mensagem do tipo ‘ACCESS_REQ’ que contém apenas o seu username.

Do outro lado, o servidor verifica se este username consta na sua “base de dados” que, no nosso projeto, é simulada através de um dicionário onde cada chave é o username e o valor é a síntese da palavra passe do mesmo, feita com SHA-512. Uma vez verificada a existência do username na base de dados, é gerado um NONCE: no nosso caso, este consiste na concatenação do timestamp com um valor gerado aleatoriamente. O uso de timestamp é feito para prevenir ataques por repetição, como já foi explicado. O servidor calcula a resposta correta ao challenge (hash(nonce || pwd)) e responde ao cliente com uma mensagem do tipo ‘CHALLENGE_REQ’, onde também serão enviados o NONCE e a hash function utilizada. O servidor atualiza o seu estado para STATE_AUTH.

Em *clientside,* o utilizador irá desempacotar o NONCE e calcular a síntese da concatenação do mesmo com a hash (também SHA-512, para coincidir com a hash presente em servidor) da sua palavra passe. Este cálculo é enviado ao servidor numa mensagem do tipo ‘CHALLENGE_REP’ e o estado é atualizado para STATE_CHALLENGE.

Por fim, o servidor analisa a resposta ao Desafio enviada pelo cliente. Caso coincida com a sua (calculada anteriormente) e feito o controlo de acesso (explicado abaixo), é enviada uma mensagem do tipo ‘OK’ para o cliente, o que irá desencadear a autenticação do servidor por parte do mesmo, como será explicado com mais detalhe à frente. Caso contrário, a ligação é terminada e o servidor mostra uma mensagem de erro sobre a autenticação.

```
2019-12-04 17:09:33 melo root[24497] DEBUG Send: {'type': 'ACCESS_REQ', 'user': 'tiago'}
2019-12-04 17:09:33 melo root[24497] DEBUG Received: b'{"type": "CHALLENGE_REQ", "value": "2019-12-04 17:09
:33.1132690.9795443536207493", "hash_func": "SHA-512"}\r\n'
2019-12-04 17:09:33 melo root[24497] DEBUG Send: {'type': 'CHALLENGE_REP', 'val': '34618cabd3a1ff31c022aecd
2815fbd4140d4b4a91f449d8f0cc7ca1e3c5b11e42c88adc8e252b3c28493cdce7a9b959feb41ad09da2a2d569fe94e6b9ef41bf'}
2019-12-04 17:09:33 melo root[24497] DEBUG Received: b'{"type": "OK"}\r\n'
```

Screenshot 2: Autenticação bem sucedida em MS-CHAP - Clientside

```
2019-12-04 17:09:33 melo root[24494] DEBUG Received: b'{"type": "ACCESS_REQ", "user": "tiago"}\r\n'
2019-12-04 17:09:33 melo root[24494] DEBUG Send: {'type': 'CHALLENGE_REQ', 'value': '2019-12-04 17:09:33.11
32690.9795443536207493', 'hash_func': 'SHA-512'}
2019-12-04 17:09:33 melo root[24494] DEBUG Received: b'{"type": "CHALLENGE_REP", "val": "34618cabd3a1ff31c0
22aecd2815fbd4140d4b4a91f449d8f0cc7ca1e3c5b11e42c88adc8e252b3c28493cdce7a9b959feb41ad09da2a2d569fe94e6b9ef4
1bf"}\r\n'
2019-12-04 17:09:33 melo root[24494] DEBUG Send: {'type': 'OK'}
```

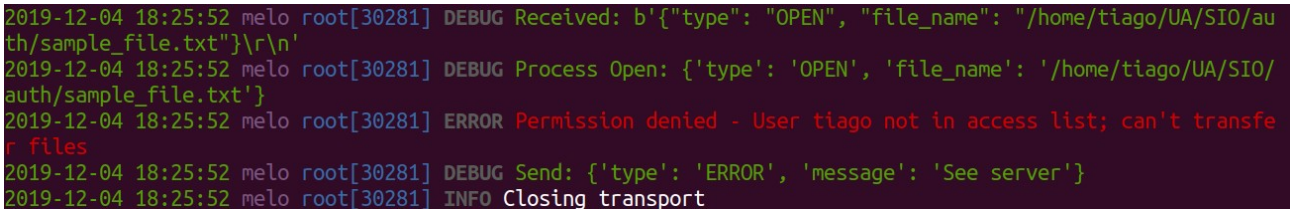
Screenshot 3: Autenticação bem sucedida em MS-CHAP - Serverside

```
2019-12-04 17:09:22 melo root[24466] DEBUG Received: b'{"type": "CHALLENGE_REP", "val": "63063cfd9369574ef6
248f0ce5a97892202275ccb6395c9e7ca3be96aa9b669813ca66ee0142b1618ded881f519e4f1a5486c1ab89c2bddea6c9a9b919772
bde"}\r\n'
2019-12-04 17:09:22 melo root[24466] ERROR Authentication failed
2019-12-04 17:09:22 melo root[24466] DEBUG Send: {'type': 'ERROR', 'message': 'See server'}
```

Screenshot 4: Falha da autenticação em Serverside

Controlo de Acesso

Quando o cliente desencadeia o processo de autenticação por MS-CHAP (através do envio do seu username), o cliente verifica se o nome enviado consta na sua "base de dados" como foi mencionado acima. Dado que a autenticação e a autorização são dois conceitos diferentes, a validação da autorização do cliente só é feita quando o mesmo tenta efetuar uma ação. Na nossa aplicação, visto que cada cliente só pode enviar ficheiros, esta ocorre quando o servidor recebe uma mensagem do tipo "OPEN". Neste caso, o servidor irá analisar se o username do cliente conectado está presente na lista de permissões, i. e., a lista que contém todos os utilizadores cujo envio de ficheiros é permitido. Caso esteja, a transferência é ocorre de forma normal; caso contrário, como a aplicação consiste **só** no envio de ficheiros, a ligação é terminada.



Screenshot 5: Negação de serviço ao utilizador "tiago" por falta de permissões

Autenticação com Cartão de Cidadão

De forma semelhante ao MS-CHAP, a autenticação de um cliente com Cartão de Cidadão baseia-se no conceito de Desafio-Resposta, explicado acima. Adicionalmente, é necessária a validação do CC. Para esta finalidade, o mesmo é carregado como um certificado X.509 e são aplicadas as medidas de validação do mesmo: validação temporal, validação da cadeia de certificação, validação das assinaturas presentes em cada certificado aprovado por uma CA e validação de cada CA presente na cadeia de certificação através de CRLs, isto é, verificar se a CA já foi, ou não, revogada.

Certificados em SmartCards permitem autenticar o portador do mesmo, isto é, o seu dono pode distribuir certificados que permitem que qualquer outra entidade verifique a sua entidade; possibilita que o dono autentique outras pessoas com cartões semelhantes graças à cadeia de certificação presente no SmartCard e, finalmente, possibilita que o cartão autentique clientes com certificados semelhantes.

A interação com o Cartão de Cidadão é feita através do uso de módulos *standard*, nomeadamente, a biblioteca PyKCS. Aplicações/APIs como esta permitem que a comunicação com SmartCards seja normalizada.

A autenticação de entidades com SmartCards pode ser feita da seguinte forma:

- A entidade que autentica gera um NONCE (um valor aleatório nunca antes usado)
- O smartcard do cliente cifra o NONCE gerado com a sua chave privada ou assina o mesmo, operações que necessitam o PIN do smartcard
- O autenticador decifra o resultado obtido com a chave pública do cliente ou valida a assinatura feita pelo mesmo. Se o resultado obtido for igual ao desafio ou a assinatura for validada com sucesso, o cliente foi autenticado.
- De seguida, irá ser validado o certificado associado ao cartão de cidadão, como foi explicado acima.

Na nossa implementação optámos pela assinatura do NONCE por parte do cliente. Adicionalmente, o controlo de acesso é feito também com uma Access List que contém os Números de Cartões de Cidadão permitidos, como iremos ver de seguida.

Implementação

O cliente é inicializado com um parâmetro self.auth_method, que pode tomar o valor CC, caso a autenticação seja feita por Cartão de Cidadão ou USERNAME_PWD, caso contrário. Quando ocorre o primeiro, o cliente irá procurar uma iniciar uma session com um slot que permite comunicação com um CC através da API PyKCS. Uma vez que o mesmo seja encontrado, é extraído o certificado associado ao CC e o mesmo em formato DER.

Tal como em MS-CHAP, a comunicação entre cliente e servidor é desencadeada por uma troca de mensagens do tipo ‘HANDSHAKE’. O cliente assume o estado STATE_HANDSHAKE. Desta vez, é enviada a Macro que diz respeito à autenticação por Cartão de Cidadão. Mais uma vez, o servidor verifica se a operação é suportada e responde com um ‘OK’.

Face a esta resposta, o cliente avalia o método de autenticação escolhido e atualiza o seu estado para STATE_ACCESS_REQ. Apesar do método de autenticação ser diferente, o estado assumido é o mesmo, por questões de simplicidade e reutilização de código. Depois, envia uma mensagem do tipo ‘CC’ que contém o Certificado associado ao Cartão de Cidadão lido, como foi explicado anteriormente, em formato DER.

Do outro lado, o servidor regista o certificado que acabou de receber e extrai a chave pública associada ao mesmo. De seguida, é gerado um NONCE tal como na autenticação com MS-CHAP: concatenação do timestamp atual com um valor gerado aleatoriamente. Este é enviado numa mensagem do tipo CHALLENGE_CC.

Em *clientside*, o utilizador irá desempacotar o NONCE, assinar o mesmo e enviar a assinatura numa mensagem do tipo SIGNATURE e é assumido o estado STATE_CHALLENGE.

Screenshot 6: Resposta a um Challenge request do tipo CC, assinatura de um NONCE e resposta de um OK após a assinatura e o cartão de cidadão terem sido validados

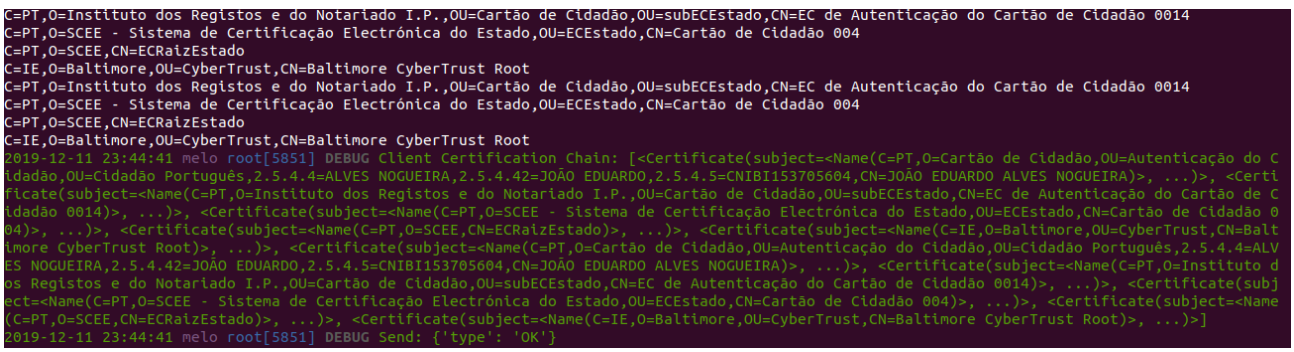
De seguida, o servidor verifica a assinatura feita pelo cliente da mesma forma que se valida uma assinatura em certificados X.509. Caso tal seja efetuado com sucesso, passamos à próxima etapa: validação do cartão de cidadão.

Como foi mencionado acima, a validação do CC é feita da mesma forma que é feita a validação de um certificado X.509: validação temporal, validação da cadeia de certificação, validação das assinaturas presentes em cada certificado aprovado por uma CA e validação de cada CA presente na cadeia de certificação. (isto é, ver se estão presentes em alguma CRL)

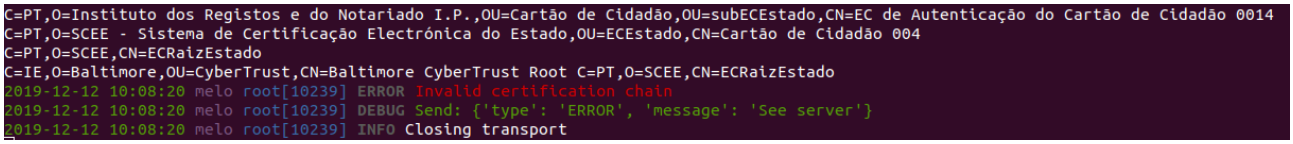
Quando o servidor é inicializado, ele lê todos os certificados presentes numa pasta fornecida pelos docentes (/PTEID/pem), donde irá registar para um dicionário do tipo { subject : certificate } os certificados que respeitem a validação temporal. É de salientar que foi adicionado um certificado (BaltimoreCyberTrustRoot.pem) de forma a finalizar a cadeia de certificação, transferido da DigiCert.

A validação temporal é trivial: extraímos a data de inicio e fim da validade do certificado, comparamos com a data atual e verificamos se está dentro do intervalo aceitável.

Posto isto, é executada a nossa função recursiva que constrói uma cadeia de certificação **apenas** composta por certificados válidos temporalmente. Para cada um destes é verificado se a assinatura do issuer coincide com a assinatura presente no certificado. Caso algum destes parâmetros falhe, a cadeia não é construída e a autenticação do cliente é abortada.



Screenshot 7: Cadeia de CAs associadas ao Cartão de Cidadão



Screenshot 8: Erro na construção da cadeia de certificação, induzido quando se remove o certificado de Baltimore do diretório que os contém

Por fim, deveriam ser analisadas as CRLs que dizem respeito a cada um dos certificados.

Caso sejam passados todos os testes, é feito, por fim, o controlo de acesso (explicado abaixo) e é enviada uma mensagem do tipo ‘OK’ para o cliente, o que irá desencadear a autenticação do servidor por parte do mesmo, como será explicado com mais detalhe à frente. Caso contrário, a ligação é cortada e o servidor mostra uma mensagem de erro sobre a autenticação.

Controlo de Acesso

Após o processo de autenticação e validação estarem concluídos, o cliente irá enviar uma mensagem do tipo OPEN, tal como em MS-CHAP. Ao receber a mesma, o servidor irá extrair o número de Cartão de Cidadão do certificado recebido anteriormente e irá verificar se o mesmo se encontra na lista de acesso. Caso o cliente em questão não tenha permissões de transferência de ficheiros, a ligação é terminada, tal como no protocolo de autenticação explicado anteriormente.

Screenshot 9: Negação de serviço ao cidadão "CNI..." por falta de permissões

Autenticação do Servidor

De forma a obtermos Transport Layer Security (TLS), é necessária a autenticação do servidor que, na nossa aplicação, é feita com certificados X.509.

TLS

A Transport Layer Security tem como principais objetivos promover a comunicação segura sobre a camada TCP/IP. Surgiu como uma evolução da norma SSLv3 e permite a confidencialidade e integridade da comunicação, isto é, as mensagens trocadas entre os intervenientes são cifradas de forma a que apenas os mesmos as compreendam e assinadas de forma a garantir que não foram adulteradas por MiTM. Isto pode ser feito através da distribuição de chaves simétricas, negociação de cifras através da troca de *ciphersuites*, abordadas no projeto anterior, entre outros.

Por outro lado, a autenticação das entidades intervenientes, tanto do servidor como do cliente, é assegurada por chaves assimétricas e certificados X.509.

Como podemos ver, para que a TLS funcione corretamente, é então, imperativa a autenticação de *both parties a priori* da troca de mensagens. A autenticação do servidor é feita através de certificados X.509.

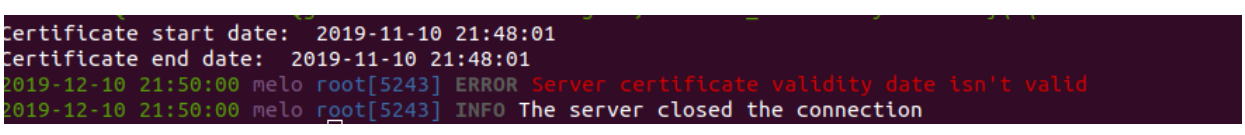
Certificados X.509

Em criptografia, X.509 é uma norma que define o formato de certificados que contém chaves públicas. Certificados X.509 são usados numa vasta gama de protocolos, nomeadamente, TLS/SSL, onde assenta HTTPS, o protocolo que permite *safe web browsing*. Um certificado X.509 contém uma chave pública e uma entidade (*hostname,* organização ou indivíduo) e é assinado ou por uma Certification Authority (CA) ou auto-assinado (caso seja a raiz da cadeia de certificação). Um certificado uma vez assinado por uma CA de confiança/válida, contém uma chave pública que pode ser usada para estabelecer comunicação seguras com uma outra entidade e validar documentos digitalmente assinados com a chave privada correspondente.

Implementação

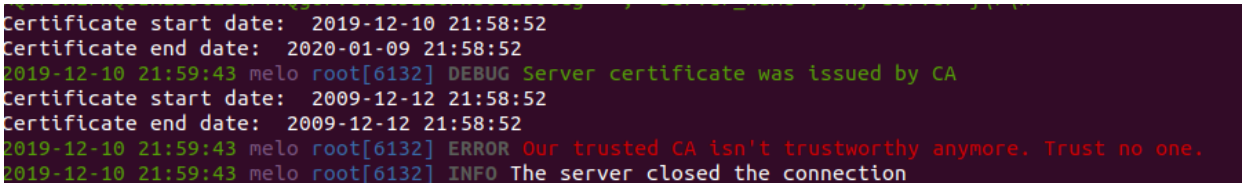
Na nossa implementação partimos do princípio que o cliente tem uma lista de trusted CA's pré-provisionadas. Isto é, quando o cliente é inicializado, é lido o certificado de um ficheiro onde foi armazenado o Certificado em formato PEM de uma CA por nós gerada. Adicionalmente, a partir da mesma foi também gerado o certificado que o servidor usa. Ambos estão armazenados em /certs. Posto isto, a autenticação de um servidor ocorre da seguinte forma:

- O cliente pede o certificado do servidor.
- O servidor envia o seu próprio certificado, a sua chave pública (redundante, mas facilita a implementação da validação) e o seu nome. (Common Name)
- O cliente tem agora tudo o que precisa para validar a autenticidade do servidor. Começamos por validar se a data atual se encontra dentro da janela temporal em que o certificado é válido. Caso não seja, o cliente devolve False e a ligação é cortada.



Screenshot 10: Checking if our current date is within the acceptable certificate validity date

- De seguida, o cliente irá validar a autenticidade do issuer do certificado do servidor a que se está a tentar conectar. Na nossa aplicação, caso o issuer não seja a CA confiável pre-provisionada, a ligação termina.
- Numa aplicação “real” seria necessário percorrer a cadeia de certificação até encontrarmos uma CA em que confiássemos ou percorrer a cadeia toda, caso não fosse encontrada nenhuma Trusted Root “pelo caminho”. Para além disso, seria também necessária a verificação se cada uma das CA’s está em alguma CRL, o que não foi implementado dado que não foi gerada nenhuma CRL, por questões de simplicidade e a cadeia tem de comprimento 2. (a CA Raiz encontra-se diretamente acima do certificado do servidor)
- Caso contrário, iremos proceder à validação do certificado em si, isto é, validamos se a assinatura do certificado do servidor pode ser validada com a CA em que confiamos. A assinatura de um certificado é feita gerando uma hash do documento, assinando essa mesma *hash* (como CA) e fazendo a *hash* do resultado novamente. Esta é a assinatura validada quando se segue este procedimento.
- Por fim, verificamos se a CA em que confiamos continua válida, i. e., verificamos se a sua validade ainda se encontra dentro da janela temporal, se CA continua True nas suas Basic Constraints e se ela mesma não está dentro da própria CRL. Se alguma destas verificações falhar, a CA é eliminada como sendo de confiança e a ligação é terminada. **(Ver /crls/CRL_Notes.txt)**



Screenshot 11: Checking if our CA is still acceptable (and failing, for date reasons). The first two prints refer to the server certificate we want to authenticate and the second to the issuing CA.

Screenshot 12: Clientside successful server authentication

Screenshot 13: Serverside successful server authentication