Relatório do Projeto 2 - Comunicações Seguras

LEI - Segurança Informática e nas Organizações (SIO)

Tiago Melo – 89005

João Nogueira – 89262

Índice

1. Introdução

Este projeto consiste no desenho, planeamento e implementação de um protocolo para estabelecer comunicações seguras entre dois interlocutores, exemplificados como cliente e servidor, de maneira a haver troca de mensagens confidenciais e íntegras entre os mesmos.

Neste relatório iremos abordar as principais características do nosso protocolo, nomeadamente:

- Negociação de algoritmos usados;
- Controlo da confidencialidade através do uso de cifras simétricas;
- Troca e rotação de chaves;
- Controlo de integridade das mensagens.

Serão também salientados detalhes de implementação quando os mesmos forem relevantes bem como capturas de ecrã que demonstram o bom funcionamento do protocolo.

As considerações tomadas ao longo do desenvolvimento deste projeto foram as mencionadas no enunciado do mesmo: tanto o cliente como o servidor suportam pelo menos duas cifras simétricas, dois modos de cifra e dois algoritmos de síntese. A negociação dos mesmos é concluída através do envio de uma *cipher suite* semelhante à mencionada.

2. Negociação de algoritmos

2.1 *Cipher suite*

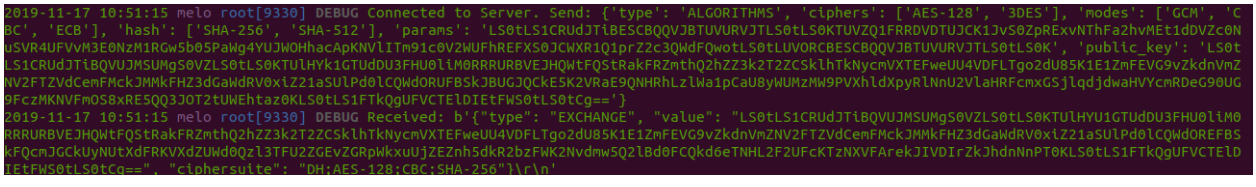
Após inicialização do cliente, a primeira mensagem enviada pelo mesmo é do tipo ‘ALGORITHMS’. Esta mensagem é enviada desencriptada e contém os vários algoritmos suportados pelo cliente, para cifra, modo de cifragem e síntese. Encontram-se também nesta mensagem os parâmetros para inicializar a troca de chaves com Diffie-Hellman, assunto discutido na segunda parte deste capítulo. É de salientar que, apesar de ser irrelevante no nosso exemplo, a decisão dos algoritmos usados é feita pelo servidor pelo facto de, em aplicações e sistemas reais, faz mais sentido que haja N clientes a adaptar-se a uma entidade central (servidor) e não uma entidade central a adaptar-se a N clientes.

Estes parâmetros encontram-se *hardcoded* no código do cliente. Contudo, poderiam ser lidos, por exemplo, de um ficheiro ou a partir do terminal de forma interativa. Quando a conexão é estabelecida são então enviados ao servidor.

O servidor, na nossa implementação, suporta:

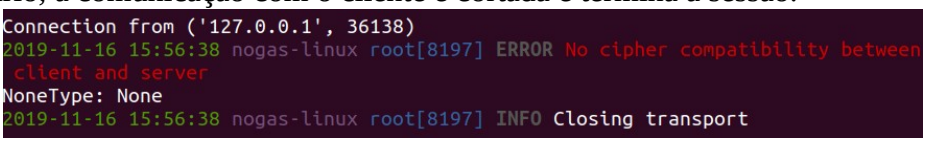
- 2 algoritmos de cifra, **AES-128** e **3DES**;
- 2 modos de cifragem para cada algoritmo, **GCM** para AES-128, **ECB** para 3DES e **CBC**, suportado por ambos, resultando num total de 3 modos diferentes;
- 2 algoritmos de síntese, **SHA-256** e **SHA-512**.

O servidor lê então a mensagem recebida, analisando os algoritmos suportados pelo cliente. É então calculada a interseção dos algoritmos suportados pelo cliente e pelo servidor. Caso seja encontrado pelo menos um algoritmo para cada caso (cifra, modo de cifragem e síntese) suportado por ambas as partes, é construída uma mensagem do tipo ‘EXCHANGE’ que irá conter a *cipher suite* calculada (e mais alguns campos que serão abordados na próxima secção), e enviada para o cliente sob a forma, por exemplo, “**DH;AES-128;CBC;SHA-256”**.



Screencapture 1 - Mensagens recebidas em clientside durante o Handshake

Caso contrário, a comunicação com o cliente é cortada e termina a sessão.



Screencapture 2 - Fim da sessão quando não há compatibilidade de algoritmos entre cliente e servidor

2.2 Diffie-Hellman

A troca de mensagens entre servidor e cliente, começa sempre com estabelecimento de um segredo partilhado através do algoritmo de Diffie-Hellman.

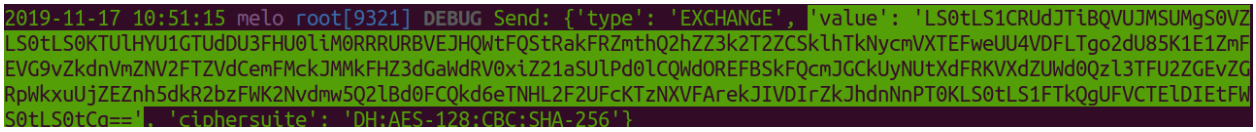
São para isso usados os métodos disponíveis na biblioteca *cryptography.hazmat* para gerar os parâmetros necessários à sua implementação. Esta troca de mensagens para estabelecimento de parâmetros e segredo partilhado pode ir desencriptada, visto que o algoritmo é resistente a ataques do tipo Man-in-the-middle. Isto acontece visto que, da forma como o algoritmo está implementado (e graças à rotação de chaves executada pontualmente), um atacante não consegue obter informação relevante para a descoberta das keys negociadas mesmo estando a observar o tráfego na rede.

Diffie-Hellman funciona através do acordo entre um número primo e um co-primo. A título de simplicidade, consideremos que estes são, nomeadamente, 3 e 17. A Alice escolhe um número aleatório (que será a sua chave privada) e, a partir desta e dos parâmetros acima, calcula a sua chave pública e envia ao Bob. Conisderemos que há uma Eve que também recebeu a chave pública. O Bob calcula a sua chave pública da forma análoga com os mesmos parâmetros e envia-a à Alice (e conisderemos que a Eve, novamente, também a recebe). Para obter o segredo partilhado, a Alice usa a sua chave **privada** e a chave **pública** do seu interlocutor, neste caso, Bob. O Bob faz a mesma coisa com a sua chave privada e a chave pública da Alice. Este cálculo do segredo partilhado é feito através dos mesmos parâmetros acima (3 e 17). Como a Eve não tem nenhuma das chaves privadas usadas para calcular as chaves públicas que ela conhece, é lhe **impossível** obter o segredo partilhado entre Alice e Bob. Algo que a Eve poderia fazer, contudo, seria agir como Bob para Alice e agir como Alice para Bob. Desta forma, cada um dos interlocutores pensaria que estava a ter uma conversa confidencial (e estavam, simplesmente não com a pessoa que gostariam) e autenticada (algo que o algoritmo DH não garante: apenas garante que a pessoa com quem inciamos a troca de mensagens se mantém ao longo de toda a "conversa").

Em termos de implementação, quando o cliente envia a mensagem que inicia a sessão, do tipo 'ALGORITHMS', é incluído um campo 'params' que contém os parâmetros a partir dos quais foi gerado o par de chaves para o cliente bem como um campo 'public_key' que contém a

chave pública do cliente que, como foi explicado acima, será crucial para obter um segredo partilhado entre cliente e servidor. É de salientar que os parâmetros foram importados da biblioteca `cryptography.hazmat.primitives.asymmetric` e não implementados de raiz. O servidor, a partir de 'params' irá gerar o seu próprio par de chaves e, a partir da chave pública presente em message['public_key'] irá obter um segredo partilhado que será derivado para obter uma chave partilhada. Após a mesma ter sido calculada, o servidor responde com uma mensagem do tipo 'EXCHANGE', que contém a sua chave pública e a *cipher suite*, como foi mencionado acima. A partir desta chave pública, o cliente irá conseguir replicar as operações efetuadas pelo servidor e, por conseguinte, obter a chave partilhada derivada a partir do segredo partilhado. **Esta será a chave usada para todas as encriptações por ambas as partes até ser *rotated out.*** (o que ocorre após 20 envios de mensagem por parte do cliente, que será explicado mais à frente.)

Screencapture 3 - Envio dos parâmetros para "cálculo" da chave pública e chave pública do cliente para o servidor



Screencapture 4 - Resposta do servidor com a sua pública key

Para além disso, para garantir que o cliente e servidor esperam pelo resultado das mensagens de construção do algoritmo DH foram usados estados para indicar quando é que a comunicação inicial (*handshake*) está concluída.

Em suma, no final de todo este processo obtemos um segredo partilhado que pode ser usado para derivar uma chave para encriptarmos as mensagens enviadas a partir deste momento, garantindo assim a sua confidencialidade.

Estamos assim prontos para iniciar uma comunicação segura entre as duas entidades.

3. Confidencialidade

A confidencialidade das mensagens trocadas entre cliente e servidor é assegurada através de algoritmos de cifra simétrica. Após as mensagens iniciais para o estabelecimento de algoritmos, todas as mensagens seguintes (à exceção das mensagens de rotação de chaves) respeitam o seguinte formato:

```
{
    type: SECURE_X
    payload: <Mensagem encriptada>
    iv: Initialization Vector usado no algoritmo de cifra
    hash: MAC/Síntese (discutido no capítulo 4. Integridade)
}
```

Para procedermos à encriptação, o cliente corre o método correspondente (presente no módulo *encrypt_decrypt_funcs.py)* tendo como argumento a mensagem que vai mandar, seja ela do tipo OPEN/DATA/CLOSE, convertida para binário, usando o algoritmo e respetivo modo previamente acordados, e uma shared key, obtida através da passagem do segredo partilhado do DH no algoritmo PBKDF2, como foi explicado anteriormente.

Esta função devolve a mensagem encriptada e o *initialization vector* usado na encriptação da mesma (gerado aleatoriamente), sendo estes dados colocados nos respetivos campos da mensagem do tipo ‘SECURE_X’. Esta mensagem é depois enviada ao servidor.

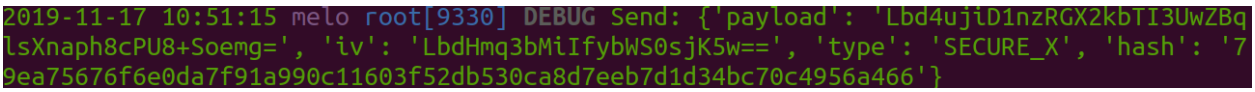
Do lado do servidor, é desencriptada a payload da mensagem SECURE_X, usando para tal o par algoritmo-modo negociados previamente, o IV presente na mensagem recebida, e a key proveniente do DH/PBKDF2.

Em termos de implementação, quando o cliente pretende enviar uma mensagem, se esta não for de configuração/negociação de algoritmos (ou seja, do tipo ‘ALGORITHMS’), irá ser cifrada e será construída uma nova mensagem com os outputs dessa mesma cifra, um novo tipo (‘SECURE_X’) e uma síntese da concatenação da mensagem **encriptada** com a **chave partilhada** (encrypt-then-MAC, explicado na próxima secção) para controlo de integridade. Esta nova mensagem será a que irá ser enviada para o servidor.

Por sua vez, o servidor quando recebe uma mensagem, verifica o seu tipo. Caso seja

‘SECURE_X’ trata imediatamente de decifrar o payload, com recurso ao IV e efetua o controlo de integridade. O processamento da mensagem recebida é agora feito sobre a o payload que acabou de ser decifrado. Tudo isto é feito dentro da função `on_frame` chamada por `data_received`.

Desta forma, garantimos que potenciais e*avesdroppers* não percebem que tipo de mensagens estão a ser enviadas, muito menos o conteúdo das mesmas. O controlo de integridade é explicado mais à frente.



Screencapture 4 - Mensagem do tipo SECURE_X, que contém a cifra da mensagem a enviar dentro de 'payload'

4. Integridade

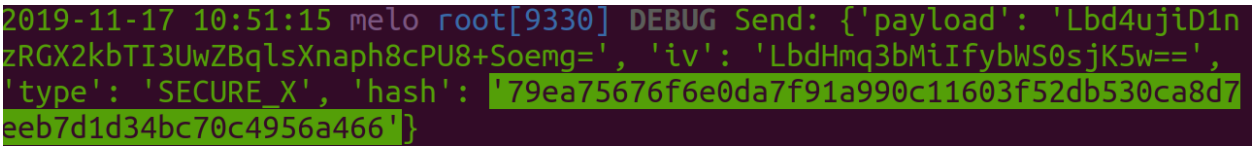
O controlo de integridade das mensagens é de extrema importância, visto que permite ao recetor saber se uma dada mensagem poderá ter sido adulterada através de mecanismos simples, mas eficazes: funções de síntese.

Uma função de síntese consiste na transformação de uma entrada de tamanho variável num *output* de tamanho fixo, indistinguível de ruído, isto é, caracteres aleatórios. Essencialmente, funciona como uma impressão digital de informação. Cada texto tem apenas associada uma *hash* e, numa função de síntese ideal, cada *hash* corresponde unicamente a um texto, isto é, a probabilidade de colisão para duas entradas diferentes é zero. É de salientar que, ao contrário do uso da impressão digital em humanos, em documentos/texto, é virtualmente impossível chegar ao texto original a partir da síntese sem ser através de força bruta. Naturalmente, nas funções de síntese usadas, soluções força bruta não são economica nem temporalmente viáveis. Por conseguinte, padrões de avaliação de funções de síntese normalmente consideram a sua resistência à descoberta de um texto (i. e. do texto original), resistência à descoberta de um segundo texto com a mesma síntese e resistência à colisão.

Posto isto, todas as mensagens do tipo SECURE_X contêm, dentro do campo ‘hash’, o MAC da mensagem enviada (payload) de forma a garantir a integridade e autenticidade da mesma.

Para calcular a hash aplicamos a função de síntese previamente estabelecida (SHA-256/SHA-512) à concatenação da mensagem, já *encriptada*, que queremos enviar, com a shared key proveniente do DH/PBKDF2.

Este modo de construção do MAC, *Encrypt-then-MAC*, traz duas importantes vantagens em termos de segurança: permite que a integridade da mensagem cifrada seja verificada antes do processo de decifra, e nega ao atacante a possibilidade de obter qualquer informação sobre o conteúdo da mensagem atraveś do MAC, visto que a síntese é criada através da private key (proveniente do segredo partilhado), que o atacante não conhece, e da mensagem cifrada (que mesmo podendo ser igual a uma mensagem anterior, irá produzir uma cifra a sua cifra diferente).



Screencapture 5 - MAC, calculado da forma síntese(mensagem encriptada + chave partilhada)

Do lado do servidor, é então calculado o MAC da payload (síntese de payload+key) e comparada com o campo ‘hash’ obtido na mensagem SECURE_X. Caso coincidam, a mensagem cifrada pode ser desencriptada seguramente. Garantimos através deste processo a integridade do conteúdo de todas as mensagens do tipo SECURE_X.

5. Rotação de chaves

Apesar do algoritmo Diffie-Hellman ser considerado bastante seguro, é boa prática não usar a mesma shared key durante muito tempo, não só porque pode ter sido descoberta por um atacante, mas principalmente para reduzir a quantidade de conteúdo encriptado com essa mesma chave, de forma a minimizar *leaks* de informação caso uma das chaves seja "quebrada".

Por conseguinte, a cada 20 mensagens enviadas pelo cliente, o mesmo pede uma nova negociação de algoritmos. Isto é, a troca de parâmetros para estabelecimento do DH (e também da *cipher suite*) é novamente executada entre o cliente e servidor, de forma semelhante a como foi feito incialmente.

Em termos de implementação, a leitura do ficheiro é interrompida e regista-se quanto do mesmo já foi lido através do uso da função `tell()`, nativa ao Python, que devolve o número de bytes já lidos de um ficheiro. Uma vez concluído o *handshake*, é usada a função `seek()` para retomar a leitura do ficheiro onde foi deixada. Mais uma vez são usadas as variáveis de state para garantir o funcionamento correto da máquina de estados, e são usadas mensagens cuja confidencialidade não é assegurada, visto que tal não é crítico para um bom funcionamento do algoritmo DH.

Screencapture 6 - Envio de uma mensagem encriptada, interrupção do envio de dados para re-negociação de algoritmos, continuação do envio de dados. (clientside logs)