eCommerce Communication Security

Part 1: Build it

Estudantes: 60400 Diogo Novo, fc60400@alunos.fc.ul.pt
60405 João Arcanjo, fc60405@alunos.fc.ul.pt
60493 João Lopes, fc60493@alunos.fc.ul.pt

Grupo 9

Professora: Ibéria Medeiros

Relatório realizado no âmbito de Tecnologias de Segurança

Mestrado em Engenharia Informática

Semestre de verão 2022/2023

7 de abril de 2023

Introdution

Este trabalho visa a desenvolver um sistema seguro de comunicação denominado por *eCommerce Communication Security* que visa a colocar em prática toda a matéria assimilada atá à data na cadeira de Tecnologias de Segurança, realizando a proteção devida dos dados na comunicação entre os programas da aplicação, ou seja, o cliente *(MBeC)*, a *store* e o *bank*, representados na Figura 1.

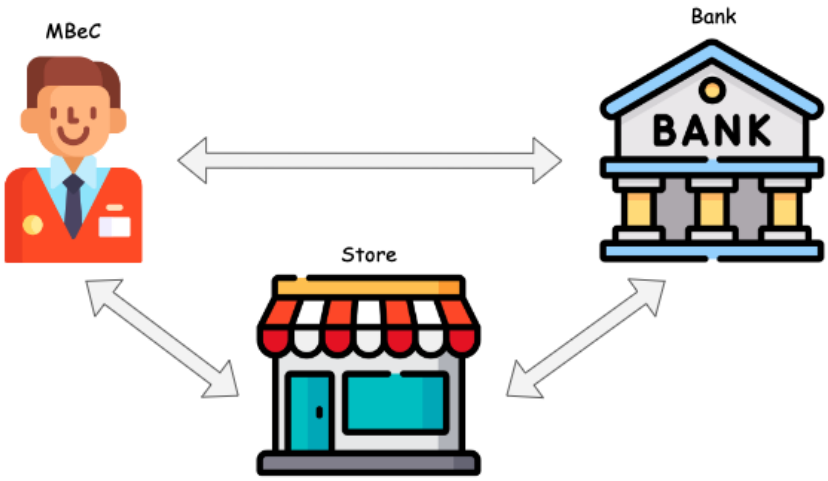

Figura 1: Arquitetura do sistema

System Design

O sistema foi construído de acordo com o que era requisito, sem a criação de ficheiros para além dos permitidos e sendo a única informação submetida a cada programa aquela que é passada pelos argumentos do programa quando este é inicializado, ou os valores definidos por *default*. A validação destes *inputs* será posteriormente explicada mais a diante.

A aplicação *MBeC*, é um programa que realiza apenas uma operação por cada execução, possuindo mecanismos que permitem aos utilizadores interagirem com as suas contas presentes no *Bank* e com a *Store*.

O *Bank*, por sua vez, necessita de estar operacional antes de cada execução do cliente e possui duas estruturas de dados que permite armazenar, durante o seu tempo de execução, a informação relativa à conta, bem como a informação do último *Virtual Credit Card* (VCC) ativado por cada utilizador. É também importante realçar que é na inicialização deste programa, tal como requerido no enunciado, que um dos ficheiros mais importantes do nosso sistema é criado, o *auth file*, cuja descrição encontra-se presente na secção Auth file.

Por último, a aplicação *Store* recebe apenas a operação de pagamento do cliente relativamente a uma compra e envia as informações associadas para o banco, que irá confirmar a

validade da operação.

O sistema apenas permite dois possíveis fluxos operacionais, ambos começam sempre na aplicação do cliente e encontram-se representados na Figura 2, na qual se distinguem pela cor vermelha e pela cor azul. Nesta mesma figura, para além dos possíveis fluxos, podemos verificar os canais utilizados entre as aplicações, representados a verde os canais seguros e a amarelo os canais inseguros.

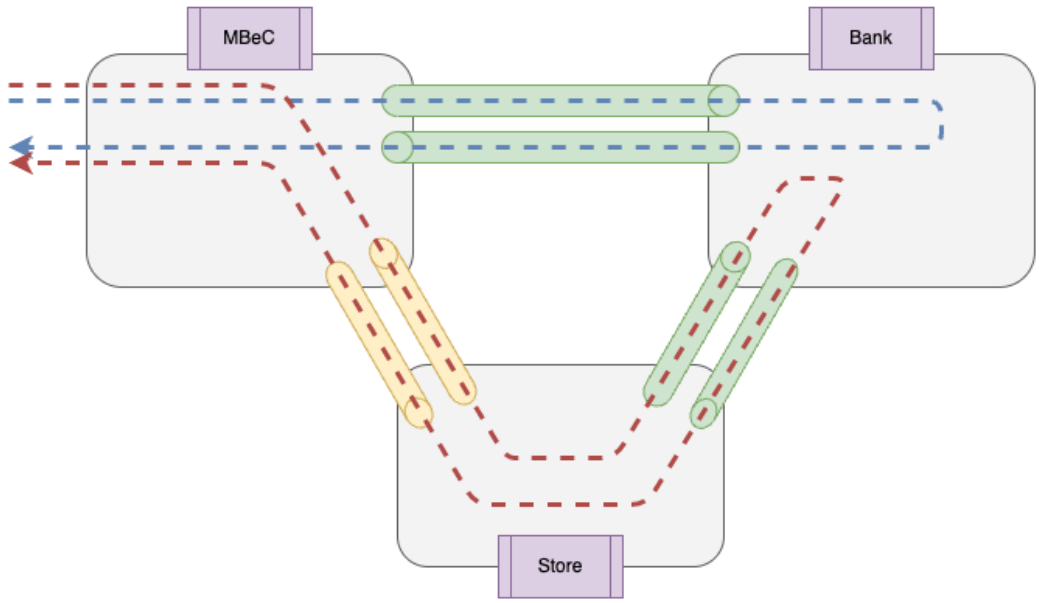

Figura 2: Possíveis fluxos operacionais do sistema

Todas as aplicações seguem um fluxo de dados muito idêntico, realizam o processamento do *inputs*, verificam se estão de acordo com alguma das possíveis operações oferecidas pelo sistema e realizam o pedido tendo em conta a operação identificada, tal como é possível observar na Figura 3.

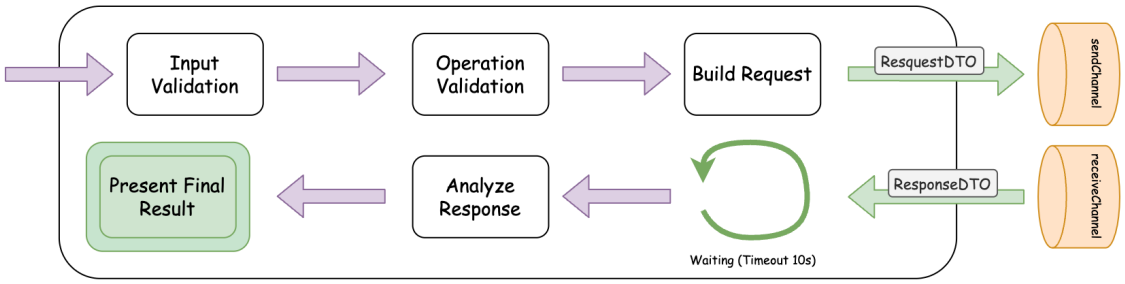

Figura 3: Fluxo de dados da aplicação *MBeC*

Files Content

Esta secção descreve o conteúdo de cada ficheiro e a justificação para a inclusão dessas informações nos mesmos.

Auth file

Este ficheiro é gerado quando o banco inicia a sua execução e é devidamente partilhado tanto com a aplicação *MBeC*, tal como com a *Store*, possuindo um papel fundamental na comunicação segura entre os respetivos componentes. Na Figura 4 está ilustrado um exemplo da constituição deste ficheiro, que possui uma estrutura em formato *JSON*, para que o seu mapeamento para um objeto interno seja realizado mais facilmente, e contém as chaves:

- ***key***, que corresponde a uma chave simétrica utilizada para estabelecer o protocolo de comunicação seguro e que será explicada na secção*Protocol description*;
- ***ip*** e ***port*** que permitem, maioritariamente, à *store* saber onde se encontra hospedado o *Bank* para que não seja necessário possuírem previamente essa informação *hardcoded*.

Figura 4: Exemplo do conteúdo de um *auth file*, denominado *bank.auth*

User file

Cada conta quando é criada, leva a que do lado da aplicação *MBeC* seja criado o *user file* que irá permitir ao dono da conta realizar operações sobre a mesma. Para tal, o ficheiros contém no formato *JSON* as seguintes chaves:

- ***name***: número da conta criada pelo utilizador;
- ***account_pin***: *pin* para utilizador conseguir aceder à sua conta;
- ***account_key***: chave simétrica que permite ao utilizador autenticar-se perante o banco na altura de realizar um pagamento com um VCC.

Figura 5: Exemplo do conteúdo de um *user file*, denominado *55555.user*

VCC file

O *VCC file* será gerado pelo MBeC quando o utilizador desejar criar um novo VCC e não tenha outra VCC ainda por utilizar. Este ficheiro é apenas utilizado na altura de realizar uma compra na *Store*, e como tal, ao ser enviado tem de conter as informações necessárias para que o banco associe a conta correta ao VCC utilizado. Como tal, contém em formato *JSON* as seguintes informações:

- ***user_file***: nome do *user file* do respetivo dono da conta e VCC;
- ***account***: número da conta associada ao VCC;
- ***vcc_number***: número de sequência associado ao VCC;
- ***vcc_amount***: montante máximo que pode ser utilizado para realizar uma compra.

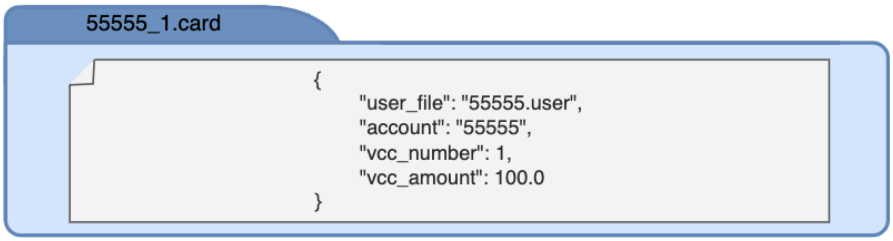

Figura 6: Exemplo do conteúdo de um *vcc file*, denominado *55555_1.card*

Communication Protocol Description

Para estabelecer uma comunicação segura entre as aplicações e o banco, tirou-se partido do facto do *Auth file* ser secreto e, como tal, nunca se encontrar exposto ao atacante, armazenando, assim, a chave simétrica para que depois fosse utilizada como forma de auxílio na geração de uma nova chave para cada sessão/conexão estabelecida.

O protocolo concebido baseia-se no uso da técnica de *Diffie-Hellman* para que seja possível ser gerado um segredo dos dois lados, sem que este nunca tenha sido exposto na rede. De forma a termos mais segurança, tempos de computação mais rápidos e dimensão de chaves mais eficientes, em vez do tradicional algoritmo, foi utilizado o protocolo de troca de

chaves *Diffie-Hellman* com Curvas Elípticas (ECDH). Contudo, não é suficiente para que consigamos oferecer uma comunicação segura, devido aos possíveis ataques que poderiam ser realizados para intercetar e manipular a comunicação, foram adicionadas mais algumas etapas ao protocolo.

Para se estabelecer o segredo, o protocolo recorre à chave simétrica contida no *auth file* para que se assine e encripte os valores que são enviados para a rede na execução do protocolo referido. A encriptação é realizada através de AES com o modo GCM, que garante não só confidencialidade, como também, integridade e autenticidade. Além disso, todas as mensagens são enviadas com uma *timestamp*, para que quando o recetor estiver a processá-las possa estipular se o tempo que demorou recebê-las foi ou não suspeito.

Finalmente quando todo o protocolo termina e é gerado o segredo comum a ambos os processos, o mesmo é utilizado para cifrar e assinar, com o mesmo algoritmo e com as mesmas garantias referidas anteriormente, as mensagens com o conteúdo sensível, garantindo, assim, que mesmo que algum atacante tente aprender algo sobre a encriptação, não consiga obter qualquer informação relevante, já que é usada uma chave diferente para cada conexão estabelecida, tal como não consegue, por exemplo, adulterar as mensagens devido ao algoritmo e modo utilizado.

Enquanto que para o canal seguro com o banco o formato das mensagens utilizado corresponde a: $Enc_k(data + timestamp) + IV$, em que *data* corresponde tanto à informação da operação que está a ser requerida tal como os dados para realizá-la. Para o canal inseguro, ou seja, a comunicação entre *MBeC* e *Store*, de forma a assegurar que é de facto o dono do VCC que está a realizar determinada compra e que nenhum adversário consegue alterar nem aprender nada acerca da mensagem enviada, garantindo, assim, autenticidade, confiabilidade e integridade, foi utilizado o formato: $operationID + Enc_{k_u}(purchasedata + timestamp) + IV$, sendo a chave utilizada para encriptar a chave simétrica presente no *user file* do dono do VCC. Esta chave é obtida a partir do *VCC file* que tem a informação do nome do ficheiro da respetiva conta de utilizador a que pertence. Desta forma, uma vez que é apenas o banco que possui a chave simétrica de cada utilizador para realizar a decifra, a *Store* limita-se a reencaminhar os conteúdos recebidos para o banco, que irá realizar as validações necessárias para concluir a respetiva operação. O conteúdo presente em *purchasedata*, corresponde ao conteúdo do VCC, que irá permitir ao banco identificar a respetiva conta, e o *amount* da compra efetuada.

Attacks Protection

Nesta secção explicamos como é que a nossa aplicação garante proteção suficiente contra os mais diversos ataques, garantindo que estes não colocam em causa a *correctness*, a *integrity* e a *confidentiality* dos dados que serão transmitidos na rede.

Importante salientar que a maioria das metodologias implementadas de forma a proteger os dados são extremamente semelhantes (ou até iguais) entre as diferentes aplicações, sendo, por isso, referenciado de forma sucinta, a localização de todos estes mecanismos dentro do código implementado.

Correctness Violations

O protocolo embora use valores aleatórios, todos estes estão de acordo com o esperado, ou seja, caso todo o processo não seja influenciado por terceiros, irá eventualmente terminar com a geração de um segredo que é comum a ambos os processos que iniciaram uma determinada comunicação. Mesmo que algum processo aborte, o protocolo continua a não ser comprometido visto que, caso alguma mensagem desejada não seja recebida ao fim de 10 segundos (ou 50 para o caso do *Challenge* que será referido mais em diante), o lado recetor irá abortar a operação e quem se encontra a enviar a mensagem será informado do ocorrido. Todos os valores aleatórios usados para determinada operação, sem considerar a chave simétrica que se encontra no ficheiro de autenticação, apenas eram conhecidos por aqueles dois processos e serão descartados, tornando-se inúteis após o término da comunicação.

Integrity Violations

Toda a integridade durante a troca de mensagens é garantida pelo o uso da cifra autenticada *AES-GCM*. Para a cifra *AES*, utilizou-se a versão que usa 256 *bits* durante a encriptação, tendo esta sido considerada como a mais difícil de quebrar. Além disso, ao utilizar o modo *Galois/Counter*, foi possível garantir a autenticidade da mesma, visto que neste modo, é gerado uma *authentication tag* automaticamente, com o intuito de garantir a integridade dos dados enviados. Através disto, o protocolo é capaz de tentar realizar a decifra dos dados recebidos e detetar automaticamente caso algum terceiro tenha manipulado os dados após o outro processo ter enviado a mensagem.

As funções utilizadas para a realização desta cifra e decifra podem ser observadas em todas as aplicações, dentro da diretoria **security**, com os nomes *encryptWithIv* e *decrypt*, respetivamente. Estas foram realizadas com o auxílio de outras funções fornecidas pelo *Bouncy Castle Provider*. Um exemplo da sua utilização poderá ser observado nas linhas 48 e 53 do ficheiro *SecureChannel.kt* da aplicação *MBeC*.

Confidentiality Violations/Packet Sniffing

A confidencialidade é garantida através não só do uso de uma cifra bastante poderosa, mas também pelo facto de antes de ser enviado qualquer tipo de mensagem por parte do cliente, é inicialmente realizado um *handshake*, através do *Elliptic Curve Diffie-Hellman*, onde se estipula uma nova chave a utilizar para aquela comunicação. Neste *handshake* são

sempre utilizados parâmetros aleatórios, garantindo assim que é computacionalmente difícil conseguir gerar os mesmos parâmetros e por sua vez gerar a mesma chave.

Para realizar esta geração do segredo, criou-se o método *dhProtocol*, dentro do ficheiro *SecureChannel* na diretoria *sockets*, que, dependendo de quem o invocar (*MBeC*, *Store* ou *Bank*), envia ou aguarda pelos parâmetros necessários à geração do novo segredo, invocando, no final, o método *doECDH* onde é calculado o segredo comum a ambos os processos que estabeleceram a conexão, para que possa ser usado como chave para realizar a cifra e decifra (em conjunto com um IV).

Replay Attacks

Para solucionar este tipo de ataques, foram implementadas múltiplas análises para garantir que a comunicação é a mais segura possível. A primeira, como já mencionada previamente, é o uso da geração de novas chaves sempre que é estabelecida uma nova comunicação/conexão, garantindo assim que é computacionalmente difícil, conseguir gerar os mesmos parâmetros e, consequentemente, gerar a mesma chave para forjar os valores transmitidos na rede. Além disso, todas as mensagens contêm um *timestamp* alusivo a quando a mensagem foi enviada, cabendo, então, ao recetor saber interpretá-la e, caso a mensagem recebida contenha um valor temporal muito antigo, excedendo um certo limite temporal ao qual seria esperado receber a mensagem, descartá-la.

A implementação do envio e validação do *timestamp*, pode ser observado dentro do ficheiro *SecureChannel*, no método *dhProtocol* (em todas aplicações). Pode também ser observado entre as linhas 64-74 no caso do projeto *Bank*, 44-56 no projeto *MBeC* e 49-64 na *Store*. Todo o processo acontece antes da mensagem ser encriptada e enviada, a qual é *wrapped* num objeto do tipo *TimestampDTO*, que contém o *payload* e a *timestamp* de quando foi gerado, sendo depois tudo enviado devidamente encriptado. Quando for recebida, é apenas necessário construir novamente o objeto (após a decifra ter sido realizada) e validar o valor que veio no campo do *timestamp* através da invocação do método *verifyTimestamp*, que compara o valor recebido com o valor temporal atual e verifica se está abaixo de um certo período.

Man-in-the-middle

Tendo em conta a possibilidade de um eventual atacante se colocar no meio da comunicação entre dois processos, foi necessário desenvolver medidas que pudessem impedir, pelo menos, que o mesmo conseguisse manipular o conteúdo dos dados a seu favor. Para isso, para além dos valores utilizados durante o protocolo ECDH serem sempre aleatórios, recorreu-se ao uso da chave simétrica entre os processos, para que todos os valores colocados na rede, fossem encriptados e acima de tudo, assinados garantindo assim que o atacante não consiga alterar os valores na rede. Além disso, recorreu-se ao uso do *timestamp*, que tal como já foi mencionado anteriormente, está contido nas mensagens enviadas. Através do seu uso, é

possível estipular um período de tempo aceitável para que a mensagem enviada seja recebida, baseando-se essencialmente na latência da rede e no tempo da computação da encriptação. Desta forma, é possível complicar eventuais tentativas de ataque, uma vez que para além dos valores serem aleatórios, todo o processo tem de ser extremamente rápido visto que apenas o facto do atacante se encontrar no meio da rede já irá introduzir alguma latência que, eventualmente, poderá ser suficiente para que o protocolo considere que o canal se encontre comprometido. Com os valores atuais, esta abordagem, provavelmente, iria gerar alguns problemas caso o tráfego estivesse mesmo a viajar na rede, devido a eventuais latências adicionais inerentes à rede, tendo, por isso, de se ajustar os valores dentro da função *verifyTimestamp*.

Brute force & DoS Attacks

Finalmente, embora se tenha assegurado que a comunicação com o banco é segura, todo o processo para garantir tal segurança é dispendioso e demora tempo a ser executado, podendo o atacante tirar proveito deste acontecimento, sobrecarregando o serviço. Como forma a solucionar esta vulnerabilidade, utilizou-se o conceito de *challenge*, onde caso o servidor detete que está a ser atacado com uma quantidade de pedidos que considere anormal, através da comparação dos *timestamps* dos últimos pedidos obtidos, envia primeiro um desafio aos clientes, antes de iniciar qualquer tipo de protocolo. Este terá de ser resolvido para que se possa prosseguir com todo o processo, sendo o desafio, obter um valor que possua um *hashcode* (MD5) igual ao que foi enviado no desafio. Esta operação acaba então por tornar a sobrecarga do servidor não tão trivial, impedindo também a obtenção de informações sobre possíveis chaves geradas após se estabelecer a comunicação segura, visto que as mesmas só são geradas depois do *challenge* ser cumprido pelo cliente. Este mecanismo pode ser encontrado mais uma vez dentro do ficheiro *SecureChannel*, na função *challenge*, a qual é acionada sempre que existe uma nova comunicação, sendo que atualiza um monitor (função *updateMonitor*), que determina se está ou não a haver um grande quantidade de acessos num curto período de tempo. Caso determine que tal está a acontecer, envia então o desafio.

Conclusion

Em suma, o desenvolvimento do protocolo de segurança referido acima visa a garantir a privacidade e a integridade dos dados transmitidos em ambientes cada vez mais vulneráveis a ameaças de segurança recorrendo a técnicas lecionadas em aula. A adoção das várias técnicas de segurança realizadas, tais como a encriptação com diferentes chaves por comunicação, a autenticação e a integração com o *challenge* permitiu-nos a implementação de um protocolo de segurança robusto que oferece o máximo de proteção possível, que minimiza os riscos de ataques e que melhora experiência de utilização do respetivo sistema.