Universidade do Minho
Escola de Engenharia

TITULO

Engenharia de Segurança

Mestrado Integrado em Engenharia Informática
Universidade do Minho

Perfil de Criptografia e Segurança de Informação

2º Semestre
2017-2018

Afonso Pontes
pg35389

Bruno Carvalho
a67847

Bruno Dantas
a74207

César Augusto
a72384

Daniel Rodrigues
a75655

João Miguel Carvalho
pg35392

José Cunha
a74702

Leandro Salgado
a70949

Luís Costa
a73434

Mariana Carvalho
a67635

Nunes Rafael

Paulina Suquina

Pedro Fonseca
a74166

Osvaldo Jamba

27 de Junho de 2018

Conteúdo

Capítulo 1

Introdução

O projeto consiste, de forma sucinta, em desenvolver um sistema de leilões de carta fechada, em que cada licitação efetuada é secreta e não pode ser conhecida por nenhuma das entidades participantes até que o leilão ao qual a licitação corresponde tenha terminado.

Um dos focos principais do projeto foi em garantir que o utilizador se mantém completamente anónimo na sua interação com o sistema, assegurando no entanto que a autenticidade e integridade das suas ações é sempre corretamente verificada.

Para conseguir atingir estes objetivos, o sistema faz uso de várias técnicas criptográficas que foram criadas com o objetivo de garantir, entre outras propriedades, confidencialidade, autenticidade e integridade de mensagens. Neste projeto, aplicamos este tipo de técnicas para garantir que a comunicação entre servidor e cliente, enquanto anónima, é fidedigna, e que a propriedade principal dos leilões de carta fechada, isto é, licitações secretas, se mantêm secretas.

Nos capítulos seguintes iremos descrever, de uma forma geral, a forma como o sistema funciona e explicar os seus componentes principais, a implementação, em que aprofundamos a descrição desses componentes, a base de dados criada e as características da aplicação segundo o RGPD e o *Software Assurance Maturity Model*.

Capítulo 2

Descrição do Sistema

Uma vez que o sistema implementado visa simular um sistema de leilões de carta fechada, é essencial que no seu desenho haja uma forte preocupação na capacidade de garantir certas propriedades de segurança, nomeadamente a confidencialidade, integridade e autenticidade.

Para que um leilão seja classificado como sendo de carta fechada, cada licitação efetuada deve obrigatoriamente ser secreta e inacessível até ao final do leilão, e em circunstância alguma poderá ser conhecido um valor de uma licitação antes do término do respetivo leilão, nem mesmo pelo próprio servidor. Esta característica é essencial para que o sistema faça sentido.

Para garantir que cada licitação é secreta, o grupo achou que fazia sentido utilizar técnicas de criptografia assimétrica. Assim, cada leilão possui um par de chaves **RSA**, em que a chave pública é usada por cada cliente para cifrar uma licitação nesse leilão e a chave privada para decifrar essa mesma licitação. Como é extremamente improvável que haja dois leilões distintos com a mesma chave pública, ficou decidido que a chave pública de um leilão serviria também como o seu identificador no sistema (tanto o servidor como o cliente identificam um leilão pela sua chave pública). No entanto, como neste sistema nenhuma entidade deve, por si só, ser capaz de decifrar o conteúdo das licitações antes do final do leilão, a chave privada do leilão não é guardada pelo servidor, sendo em vez disso dividida por várias entidades distintas através do esquema de divisão de *Shamir*, de forma a que seja necessário que essas entidades se reúnam para recuperar o segredo. Na prática, uma vez que as chaves **RSA** que estão a ser usadas pelo sistema têm 2048 *bits*, não é viável fazer a divisão de *Shamir* sobre a chave privada diretamente (devido ao tamanho, a divisão demoraria demasiado tempo), sendo esta primeiro cifrada com uma cifra simétrica (**AES**), e depois feita a divisão da chave do **AES** que, como tem muito menos *bits*, não é computacionalmente demasiado exigente. Portanto, cada leilão tem a sua chave privada cifrada e, portanto, inutilizável, até que a chave usada para a cifrar seja recuperada, que só acontece quando se reunirem suficientes entidades para a recuperar, algo que, por usa vez, só acontece depois desse leilão terminar. Assim, não se acredita ser possível que uma licitação seja decifrada antes do final do leilão.

O sistema possui, de uma forma geral, quatro entidades principais distintas:

- Servidor
- Cliente (utilizador)
- Leilão
- Entidade de Confiança

O servidor é a entidade que gere todo o sistema, funcionando como entidade que gere todo o sistema, comunicando com os utilizadores, gerindo os vários leilões criados e comunicando com as entidades de confiança.

O cliente representa o utilizador, e permite estabelecer comunicações com o servidor para que possa usufruir das funcionalidades do sistema, desde criar leilões, licitar ou simplesmente ver os leilões disponíveis.

O leilão é a entidade base do sistema que representa, como o próprio nome indica, um leilão. Este recebe licitações cifradas e, depois de terminar, decifra-as e determina o vencedor.

A entidade de confiança representa uma entidade externa ao sistema, cuja única função é guardar partes das chaves dos leilões depois de feita a divisão de *Shamir* e, quando se der o término de cada leilão, fornecer de volta a correspondente parte da chave.

A divisão da chave é feita sempre de forma a que as partes necessárias para recuperar a chave de cada leilão sejam sempre inferiores ao número total de partes. Assim, se por exemplo, forem criadas quatro partes distintas de uma chave, só serão necessárias três delas para a recuperar. Para minimizar a possibilidade de perda de licitações por incapacidade de recuperar a chave no fim de cada leilão, o grupo decidiu que o *quorum* da divisão deve ser sempre uma unidade a menos do que o número total de partes. Por exemplo, se houver cinco partes, terá de haver um *quorum* de quatro.

A divisão das partes pelas entidades que participam no sistema segue sempre a mesma regra: uma parte vai para o leilão (isto é, fica sob o controlo do servidor), outra vai para o vendedor (quem criou o leilão) e o resto é distribuído por entidades de confiança. Isso significa que o aumento do número de partes resulta apenas num número maior de entidades de confiança necessárias. Por questões de segurança, a parte de chave que é enviada para o utilizador dono do leilão é cifrada com **AES** (com uma chave que pertence ao servidor e nunca é enviada para ninguém), porque o utilizador não precisa que essa informação esteja não cifrada, já que a única utilidade que ela tem é no servidor, no qual pode ser decifrada de novo. Esta medida evita que haja informação privada guardada na aplicação do cliente sem necessidade.

A comunicação entre o servidor e o cliente, de forma a adicionar a segurança do sistema, é feita sempre com **SSL**. Regra geral, o **SSL** não seria necessário, uma vez que qualquer comunicação que é efetuada recorre a mecanismos de garantia de autenticidade e, por isso, um adversário ativo não consegue fazer-se passar por outro utilizador. Para além disso, cada licitação é cifrada recorrendo à chave pública do leilão e, sem a chave privada, nenhum adversário pode decifrar essa licitação. Poderia haver o problema de um adversário poder modificar a licitação que um utilizador fez, mas depois teria de produzir uma assinatura válida sobre esse novo pacote de dados. No entanto, um adversário passivo poderia conseguir obter uma licitação cifrada e, de seguida, comparar essa licitação cifrada com uma que ele próprio pudesse gerar. Como a cifra para um leilão é feita com a mesma chave, se o valor for o mesmo, o criptograma resultante é idêntico. Para evitar esse caso, cada utilizador, na altura do registo, envia um número aleatoriamente gerado que lhe fica permanentemente associado. Esse valor não é mais enviado numa comunicação entre o servidor e o cliente, mas é usado para todas as assinaturas geradas e para cifrar licitações, gerando criptogramas diferentes para os mesmos valores e chave. A comunicação **SSL**, no caso em que o registo é efetuado, ajuda assim a que o número possa ser mantido secreto.

De uma forma resumida, o sistema de leilões de carta fechada funciona, então, com base em duas aplicações: servidor e cliente. Cada cliente, quando é inicializado, faz o seu registo no servidor (o seu **ID** é a sua chave pública) e passa a ter acesso a todos os leilões que estão presentes no servidor. O cliente pode efetuar pedidos ao servidor que, se determinados como sendo válidos, i.e., se o pedido for validado pelas verificações de segurança, são efetuados e a resposta devolvida ao cliente. No caso em que o cliente quer licitar, assumindo que tudo está correto, o servidor, depois de guardar essa licitação, envia-lhe um recibo (um comprovativo assinado com a sua chave privada) que pode ser usado para comprovar essa licitação, caso esse cliente ganhe. Se um cliente quiser criar um novo leilão, depois de criado, o servidor envia-lhe os dados correspondentes a esse leilão, entre os quais está uma parte da chave necessária para decifrar as licitações. Cada cliente só pode licitar em leilões dos quais não é o vendedor. Todas as mensagens trocadas pelo servidor e cliente sofrem, entre outras, verificações de autenticidade e integridade, para garantir que a comunicação não é alterada de nenhuma forma.

No capítulo seguinte, iremos descrever a implementação mais detalhada de cada um dos componentes do sistema, referindo como são feitas as verificações que visam garantir a autenticidade e integridade das comunicações, bem como a confidencialidade das licitações e das chaves privadas dos leilões.

Capítulo 3

Implementação

Este capítulo visa explicar como foi implementado o sistema descrito brevemente no capítulo anterior, referindo algumas das decisões tomadas quanto à melhor forma de implementar as funcionalidades desejadas e justificando essas decisões. A implementação de todo o sistema (servidor e cliente) foi feita em **Java**. Contrariamente ao que poderia ter sido feito, que seria implementar um servidor *https*, o grupo optou por implementar um modelo servidor-cliente um pouco mais simples, utilizando *sockets TCP* para efetuar todas as comunicações entre aplicações. Uma das razões pelas quais essa decisão foi tomada tem a ver com a utilização de certificados no sistema, algo que será detalhado ainda neste capítulo. Outra razão surge simplesmente do facto de vários membros do grupo terem mais experiência a lidar com servidores *TCP* deste género.

A implementação da base de dados, embora faça parte do servidor, será detalhada noutro capítulo, sendo este mais focado na implementação das aplicações ao nível do *Business Layer*. O mesmo se aplica à descrição da interface de utilizador do cliente.

3.1 Servidor

O servidor, como foi referido acima, é uma aplicação **Java** com a capacidade de receber pedidos de conexão em ambiente *multithread*, de forma a possibilitar o atendimento de vários pedidos em simultâneo. Cada conexão gera uma nova *thread* que fica responsável por gerir toda a comunicação com o seu respetivo cliente.

O servidor possui um conjunto de estruturas de dados que servem para armazenar vários tipos de dados diferentes. Para além de armazenar as informações sobre os utilizadores (das quais nenhuma é informação pessoal) e as informações de cada leilão criado, armazena ainda vários *timers* (um por cada leilão existente) que servem para poder cancelar os leilões quando o tempo acabar.

A comunicação servidor-cliente é feita recorrendo a *ObjectInputStreams* e *ObjectOutputStreams*, de forma a que os dados transmitidos ou recebidos sejam sempre objetos **Java** com uma estrutura bem definida que é interpretada sempre da mesma forma em ambos os lados. Cada funcionalidade implementada no servidor é requisitada pelo cliente recorrendo a códigos que são conhecidos pelas duas aplicações e que identificam funções específicas. Por exemplo, o código **1** corresponde à funcionalidade de registo no sistema, enquanto que o código **2** corresponde à funcionalidade de *update* da lista de leilões, que é uma função que atualiza as informações do cliente sobre os leilões disponíveis no servidor. Esses códigos pertencem a um campo específico do *datapacket*, e identificam o seu tipo. Para além dos códigos habituais de funcionalidades do sistema, também há códigos, que são apenas enviados pelo servidor ao cliente, que identificam vários tipos de erros que podem ocorrer no sistema. Esses códigos são enviados ao cliente sempre como resposta a alguma ação ilegal que o cliente tente executar, como aceder a leilões que não lhe pertençam ou tentar fazer uma licitação num leilão inexistente.

Uma vez que a comunicação entre servidor e cliente recorre constantemente ao uso de técnicas de criptografia assimétrica, faz todo o sentido que o sistema precise de usar certificados. Uma vez que um dos objetivos mais importantes é o de manter o anonimato do utilizador, não faz sentido que este precise de um certificado para se autenticar no servidor, que resultaria numa quebra de anonimato

devido à necessidade da presença de informações pessoais nesse certificado. Assim, só o servidor é que precisa de ter um certificado para se autenticar para cada utilizador. A forma como essa funcionalidade foi implementada fez uso das *keystores* em **Java**. Para que a aplicação funcione do lado do servidor, está precisa de carregar uma *keystore* denominada *ServerKeyStore.jks*, que contém o par de chaves **DSA** do servidor (par que será usado para a comunicação com todos os clientes) e o correspondente certificado de chave pública. Sem essa *keystore* a aplicação não funciona, porque não é possível efetuar a conexão entre os *sockets TCP* do cliente e do servidor (o programa termina com uma exceção do tipo *SSLHandshakeException*). Esse problema ocorre porque, na verdade, as *sockets* são **SSL** (são do tipo *SSLSocket*).

Como já foi referido no capítulo anterior, o servidor tem, para além do par de chaves **DSA**, uma chave **AES** que usa para cifrar as partes de chave de cada leilão, obtidas pelo método de divisão de segredos de *Shamir*.

A única forma, pelo menos na implementação atual, de criar um leilão, excetuando para testes em que se pode criar diretamente no arranque do servidor, é se um utilizador fizer o pedido e este for aceite pelo servidor. A funcionalidade de criação de um leilão é implementada na função *createAuction*, em que várias ações são levadas a cabo. Em primeiro lugar, são geradas as chaves para o novo leilão (o par de chaves **RSA** e a chave **AES** para cifrar a chave privada) e a chave privada é imediatamente cifrada. De forma a tentar, dentro do possível, recorrer a boas práticas de segurança, as chaves usadas, logo que não sejam necessárias, são eliminadas (na verdade, uma vez que no *Java* a destruição de variáveis é limitada, nem sempre se consegue garantir a destruição imediata de chaves, pelo que esta fica dependente do *GarbageCollector*). De seguida, é criado o leilão (classe *Java*) com todos os parâmetros enviados pelo utilizador (parâmetros que foram já validados pela aplicação de cliente). No ato de criação do leilão, é criado também um objeto *Timer* que lhe fica associado e que, quando terminar, invoca a função *endAuction*. Por fim, são distribuídas as partes da chave *AES* que cifrou a chave privada do leilão pelas entidades já mencionadas.

A função *endAuction* só é invocada pelo servidor (nunca por pedido do cliente) quando o *Timer* criado para esse leilão chegar ao fim. É importante referir, antes de mais, que na altura da divisão da chave, as entidades de confiança inicializam elas também um *Timer* com o mesmo tempo do *Timer* do servidor. Só quando o tempo chegar ao fim é que a entidade de confiança aceita que a sua parte da chave seja devolvida ao servidor. Quando a função *endAuction* é invocada, a primeira coisa que é feita, a seguir a desativar o leilão (recorrendo a uma variável booleana que determina se o leilão está ativo ou não), é a recuperação da chave requisitando as partes da chave pertencentes a cada uma das entidades de confiança. No caso do sistema implementado, uma vez que as entidades de confiança são simplesmente objetos do servidor (para efeitos de simplificação), nunca existe um caso em que essas chaves não consigam ser recuperadas, logo nunca é necessário que o vendedor forneça a sua chave (embora a tenha recebido). Depois de recuperadas as partes, é chamado um método *getWinner* do leilão em que são fornecidas as partes da chave. Esse método recupera a chave e decifra a chave privada do leilão, que é usada para decifrar todas as licitações. Os utilizadores são então ordenados por ordem decrescente de valor de licitação até um máximo de 100 utilizadores e o vencedor é determinado como sendo o que tem maior valor de licitação. Caso seja determinado que tem de haver um novo vencedor (porque o primeiro não levantou o prémio, por exemplo), o segundo lugar será escolhido, e por aí fora (na verdade, embora os participantes sejam assim ordenados, a implementação de um limite de tempo para levantar o prémio ou de qualquer outra forma de escolher um novo vencedor não foi efetuada, havendo no entanto suporte para isso neste método).

3.2 Cliente

A aplicação de cliente é, de certa forma, mais simples do que a do servidor, simplesmente porque não envolve o uso de um ambiente *multithread* nem a gestão de várias entidades.

Uma vez que a comunicação com o servidor é sempre feita com **SSL**, e que por isso é preciso que o servidor tenha o seu certificado, é necessário que na aplicação do cliente esse certificado seja validado. Essa necessidade causou um problema na implementação, porque o certificado que foi gerado para o servidor não é obviamente assinado por uma autoridade de certificação, sendo por isso um certificado *self-signed*. Para que o cliente aceitasse o certificado, foi preciso adicioná-lo à *truststore* do **Java**, para que esse certificado fosse sempre aceite, mesmo não sendo possível de ser

validado pelos métodos normais. Para que a aplicação funcione, de forma análoga ao que foi feito no servidor, é preciso que esta carregue uma *keystore* denominada *ClientKeyStore.jks* que contém o certificado do servidor, com a correspondente chave pública. Essa chave é a que será usada para todas as verificações de assinaturas de dados enviados pelo servidor. A aplicação de cliente implementa várias funções que implicam que haja comunicação com o servidor. As principais são:

- *register*
- *getAuctionList*
- *bid*
- *createAuction*
- *checkAuction*
- *claimPrize*

A função *register* serve apenas para registar o utilizador no servidor, e implementa a geração do número aleatório necessário para estabelecer o resto das comunicações com o servidor. Esta função é invocada mal começa o programa. A função *getAuctionList* descarrega todos os leilões do servidor para a memória da aplicação. A função *bid* é responsável pela licitação num determinado leilão. Cada vez que um utilizador licita, esse valor é guardado localmente para poder ser consultado mais tarde. Para além disso, o recibo enviado pelo servidor é também guardado localmente, para comprovar a participação. A função *checkAuction* serve para verificar um leilão em particular, sem ter de descarregar todos os leilões do servidor. Finalmente, a função *claimPrize* serve para o utilizador poder dar-se como vencedor do leilão. Todas as funções que comunicam com o servidor têm algo em comum: todas utilizam assinaturas digitais para garantir que cada pacote de dados recebido é autêntico e não foi modificado.

Sempre que um pacote de dados é recebido, é verificado o valor que está no seu campo *type*. Se o valor for igual ao que foi enviado, então a resposta do servidor é a esperada (o servidor validou tudo o que havia para validar), e se o valor for diferente é chamada a função *errorHandle* que tenta determinar se o valor corresponde a um tipo de erro enviado pelo servidor, pois ambas as aplicações têm conhecimento dos valores de erros e do que estes significam.

3.3 Propriedades de segurança

Quer na implementação do servidor, quer do cliente, é importante tentar garantir que haja confidencialidade, autenticidade e integridade de comunicações e de dados que devem ser secretos. As medidas de segurança implementadas resumem-se, sucintamente, à utilização de cifras (simétricas e assimétricas), assinaturas digitais e funções de *hash*.

Todas as funções relacionadas com segurança são encapsuladas numa classe designada por *Crypto*, que visa simplificar o uso dos diferentes algoritmos criptográficos, bem como de serialização e desserialização de chaves e objetos, aplicações de funções de *hash* e ainda da divisão de *Shamir* (a divisão de *Shamir* é implementada nas classes *SecretShare* e *Shamir*, que não foram implementadas pelo grupo). A ideia desta classe é poder abstrair o uso destas funções da sua implementação, que por vezes se torna extensa e pouco legível. Assim, se quisermos, por exemplo, cifrar uma mensagem, basta invocar o método *encryptText*, que cifra um *plaintext String* num *ciphertext String*. Existem também métodos que transformam *arrays* de *bytes* em *Strings* e vice-versa, ou mesmo funções invertíveis que transformam objetos *PublicKey, PrivateKey e SecretKey* em *Strings*. Essas funções são muito usadas para várias operações no sistema, já que as chaves públicas também servem de identificadores. A divisão de *Shamir*, por exemplo, é um dos casos em que as funções de *devide* e *combine* são encapsuladas na classe *Crypto*, sendo muito simples dividir uma chave e recuperar essa mesma chave apenas usando dois métodos desta classe (*getShamirShares* e *reverseShamir*.

3.3.1 Cifras

O sistema implementado só usa duas cifras: **RSA** e **AES**. A primeira só é usada para cifrar as licitações de cada leilão e, por isso, embora cifrar dados com criptografia assimétrica seja ordens de grandeza mais lento (daí muitas vezes se usar apenas como envelope digital) do que com criptografia simétrica, achamos que isso não influencia o sistema de nenhuma forma visível, pois as licitações não são estruturas de dados muito grandes. O **AES** é usado para cifrar chaves privadas **RSA** e partes de chaves quando é feita a distribuição da chave **AES** de um leilão.

Cada licitação é implementada como um objeto *Bid*, que tem o valor e o número aleatório do utilizador que está a licitar. Antes de ser enviada para o servidor, essa licitação é cifrada recorrendo à classe *SealedObject* do **Java**. Um objeto *SealedObject* representa um objeto genérico cifrado com uma cifra genérica. Ao usar o *SealedObject*, foi possível cifrar a licitação inicializando-o com o objeto *Bid* e um objeto *Cipher* inicializado com a chave **RSA** do leilão. Portanto, o objeto que é enviado para o servidor é um *SealedObject*, e não um *Bid*. A funcionalidade de cifrar e decifrar licitações foi também encapsulada na classe *Crypto*, com os métodos *encryptBid* e *decryptBid*.

3.3.2 Assinaturas e funções de Hash

O único algoritmo de assinaturas digitais implementado foi o **DSA**, e as chaves usadas são sempre ou as chaves do servidor, no caso em que o servidor assina e o utilizador verifica, ou as do utilizador, no caso em que este assina e o servidor verifica. As assinaturas são sempre usadas da mesma forma, independentemente do tipo de comunicação específico que está a ser feito, i.e., é irrelevante se se está a executar a funcionalidade de *register* ou outra qualquer. As assinaturas são sempre feitas da mesma forma, e sobre o mesmo tipo de dados. Os dados enviados são sempre representados pela classe *Data* que contêm todas as informações necessárias para serem recebidas pelo servidor, incluindo a assinatura. A assinatura é feita sobre o objeto *Data* já com todas as informações a enviar exceto, obviamente, a assinatura. Esta é efetuada sobre o objeto serializado (em formato *byte array*) e, de seguida, colocada no objeto *Data* para ser enviado. Do lado do servidor, este valida a assinatura retirando-a do objeto *Data* e validando o *byte array* desse objeto.

As funções de *hash* são usadas em inúmeros casos devido às suas propriedades de rapidez de execução e de encurtamento da infomação. Assim, são usadas muitas vezes para representar leilões e utilizadores no sistema (*hashes* das chaves públicas). São também usados por vezes para fazer assinaturas, sendo as assinaturas aplicadas sobre o *hash* de um objeto e não sobre o objeto em si. O algoritmo usado para todos os *hashes* calculados fooi o **SHA256**.

3.4 Interface gráfica

Inicialmente, dado que o projeto se trata de Leilões Online, decidiu-se construir a Interface Gráfica em Web. Uma vez que o Backend foi implementado em Java, a forma mais simples de transformar em Web seria recorrendo à Framework Java Server Faces (JSF).

Java Server Faces é uma especialização Java que permite a construção de interfaces Web através de componentes, que podem ser nativos do JSF (como uma simples caixa de texto) ou bibliotecas como PrimeFaces (que permite usar eventos AJAX, para além de ter mais funcionalidades que os componentes nativos). JSF, permite no fundo, criar páginas Web com estrutura XHtml que comunica com as Classes, Métodos e Variáveis normais de um programa em Java.

Contudo, durante a implementação, verificou-se que havia um problema na parte Web com o Certificado e Keystores que estavam a ser usados na parte de segurança. Após várias tentativas falhadas de adicioná-los às Keystores do Java, do GlassFish (O servidor do Netbeans que estava a ser usado para exibir as páginas XHtml) e até mesmo adicionar o Certificado ao Browser, decidiu-se passar para a implementação em Swing, que permite criar também interfaces gráficas apartir do Java, mas sem a inclusão de páginas Web.

3.4.1 Estrutura

A aplicação está dividida em 5 páginas, **Home, Create Auction, Check all Auctions, Bid e Claim**. O acesso a essas páginas é feito num painel lateral com botões para o efeito. No canto inferior direito, é mostrada ainda, a chave pública do utilizador.

A aplicação inicia com o registo automático do utilizador, como mostra a figura abaixo.

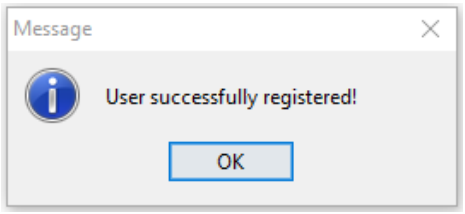

Figura 3.1: Mensagem de registo

Home

Depois disso, a aplicação abre na **Homepage**. Esta página apenas contém uma label de boas vindas, como se verifica abaixo.

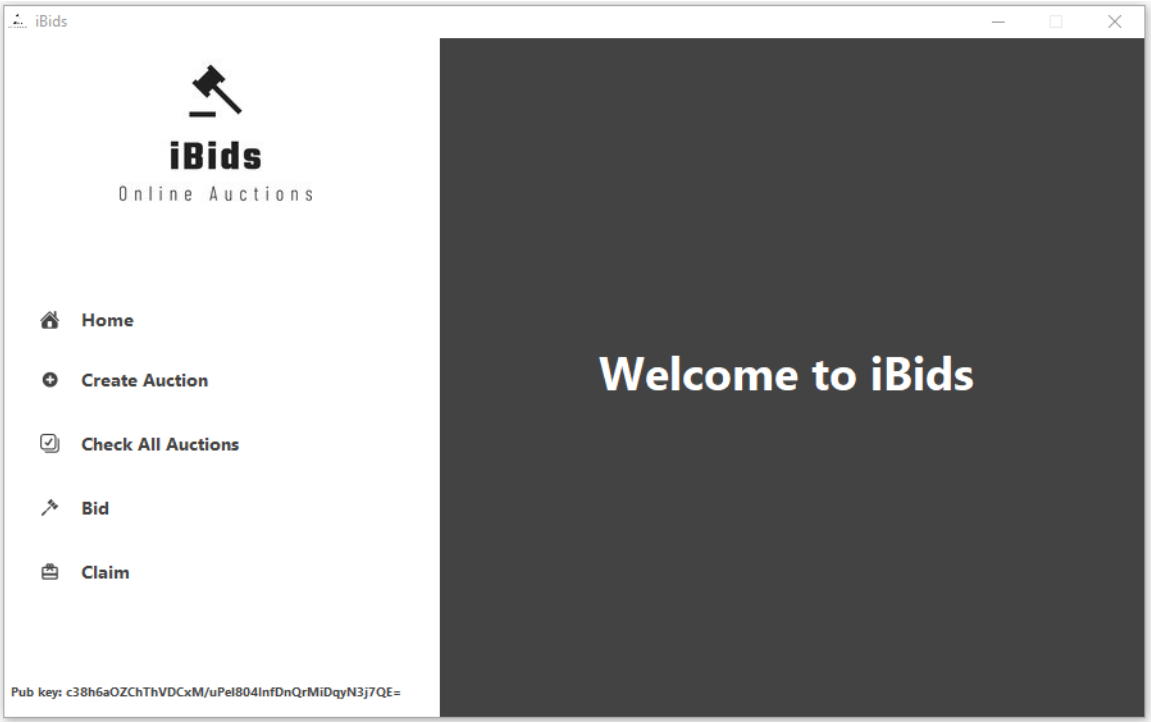

Figura 3.2: Home

Create Auction

Esta página é composta por 3 TextFields (Caixas de texto), uma onde é inserida a descrição do produto, na outra onde é inserido o preço base e uma terceira onde é inserido por quanto tempo o leilão daquele produto estará ativo (Este tempo é em milisegundos). Esta página é apresentada abaixo.

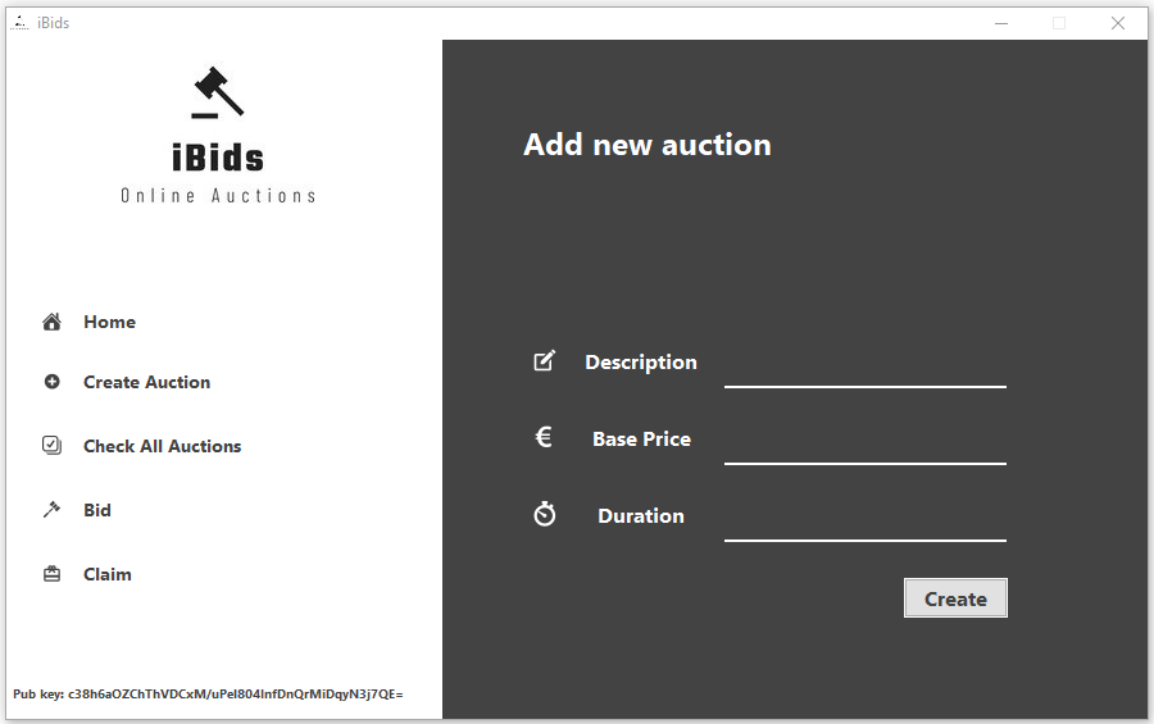

Figura 3.3: Create Auction

Assim que o utilizador cria o um leilão, uma mensagem de sucesso é apresentada, como se pode ver abaixo.

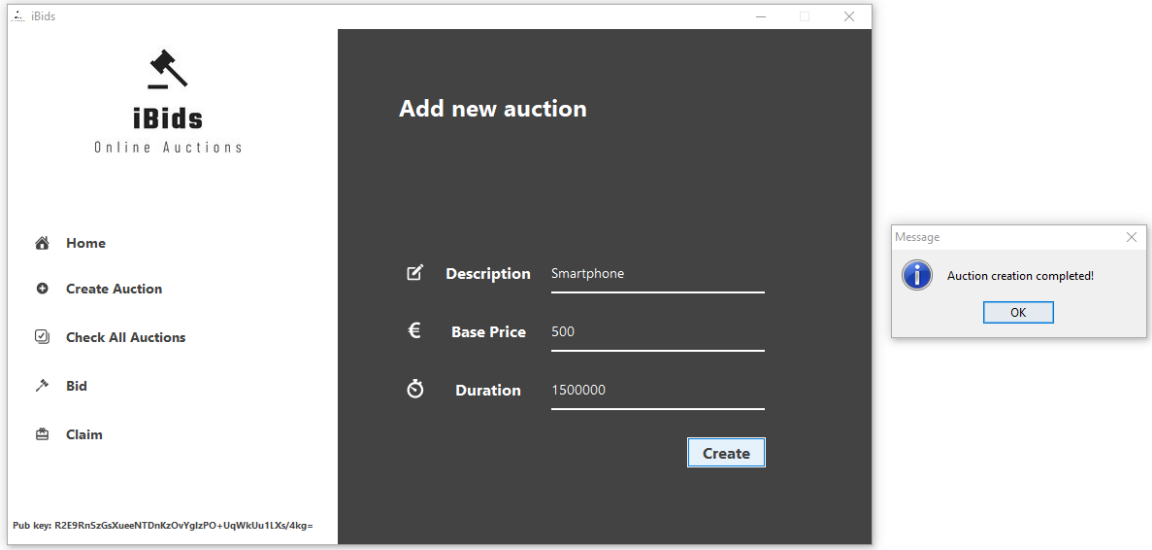

Figura 3.4: Criação de um leilão

Caso o utilizador não preencha pelo menos um dos campos, ou insira valores de um tipo errado, mensagens aparecem para o notificar.

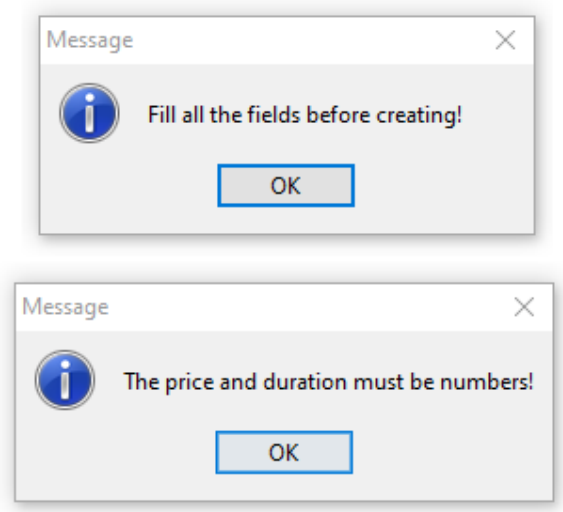

Figura 3.5: Mensagens ao desrespeitar o tipo de dados ou deixar campos em branco

Check All Auctions

A terceira página, permite ao utilizador, verificar os seus leilões criados, os leilões que estão ativos de momento e verificar ainda os leilões que ganhou.

A página possui 3 botões, um para cada uma das ações cima referidas, uma TextArea que apresenta toda essa informação, e um botão que atualiza a informação, sem escrever na TextArea.

Os leilões criados, aparecem automaticamente ao abrir essa página.

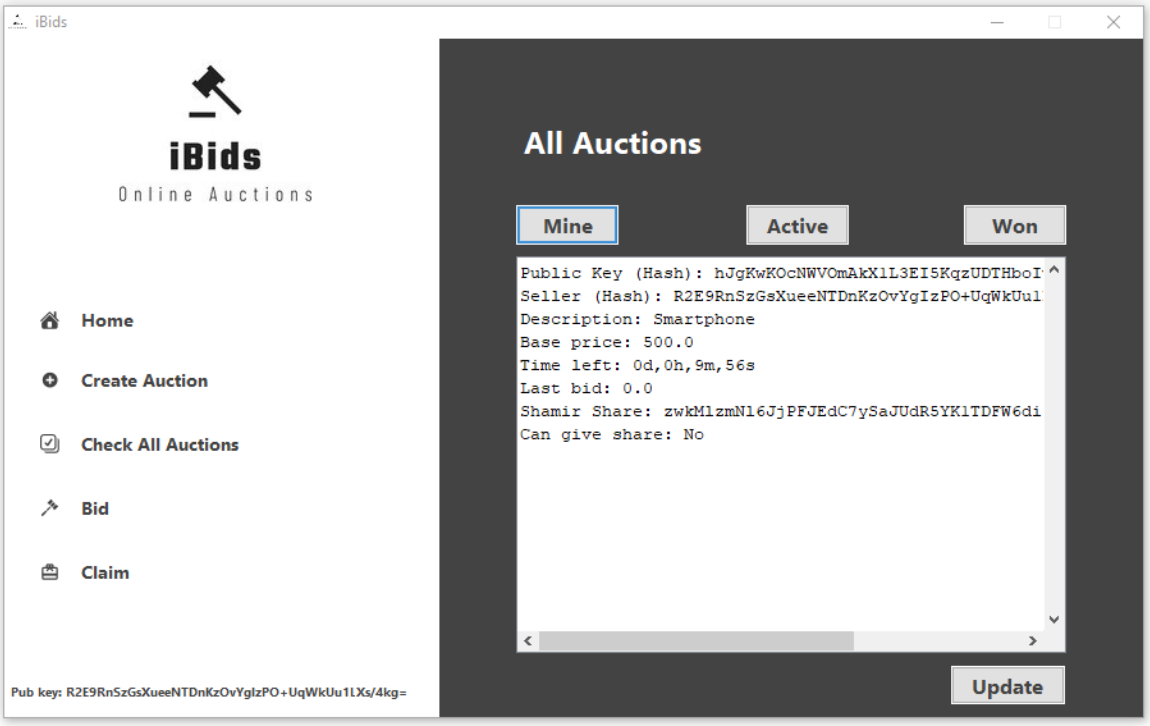

Figura 3.6: Leilões criados pelo utilizador

O botão Active, mostra os leilões ativos de momento, como referido anteriormente. Aqui, só aparecem leilões de terceiros. O exemplo abaixo, é apenas um leilão de teste já guardado no servidor

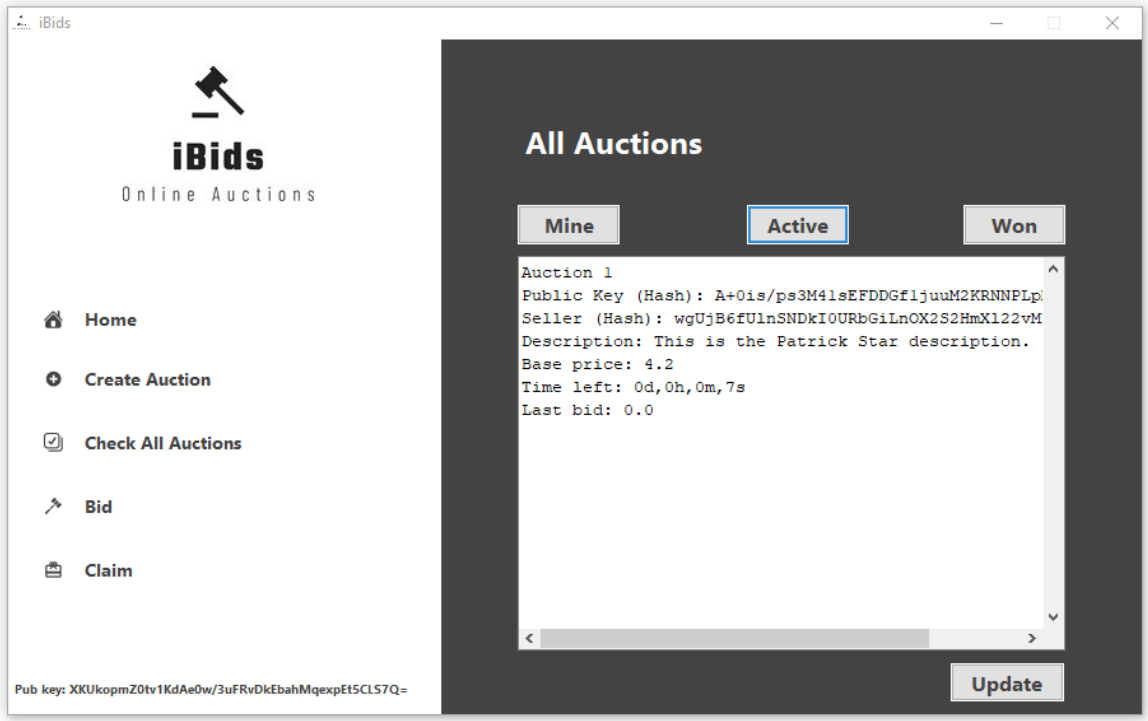

Figura 3.7: Leilões em curso no momento

Se um utilizador ganhar um leilão, pode vê-los clicando no botão **Won**. O exemplo que se segue foi conseguido, licitando no leilão que já está guardado no servidor. Este leilão tem uma duração de apenas 15 segundos. A licitação, será explicada de seguida.

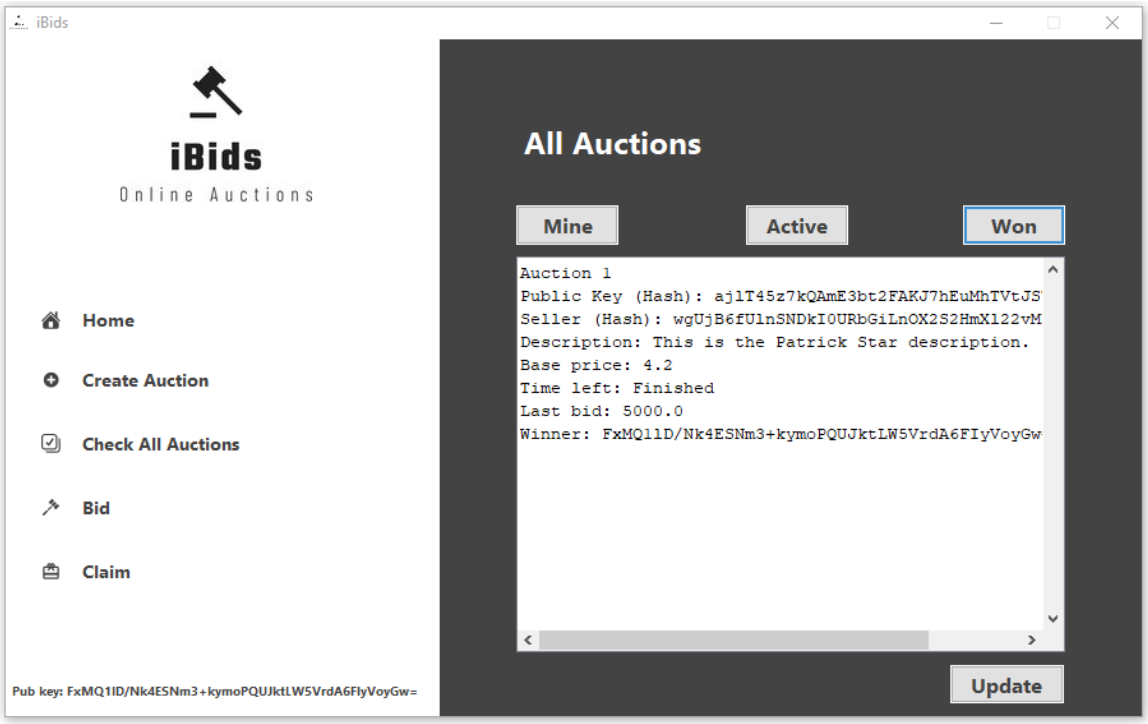

Figura 3.8: Leilões que o utilizador ganhou

O botão **Update**, apenas atualiza a informação que é adquirida para ser mostrada.

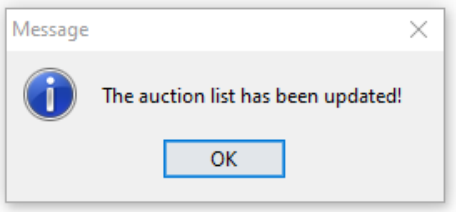

Figura 3.9: Atualização da informação

Bid

A página **Bid** permite ao utilizador licitar. Aqui são mostrados todos os leilões, independentemente de terem acabado, terem vencedor ou não. Naturalmente, os leilões criados apenas aparecem a outro utilizador, já que não faria qualquer sentido licitar nos próprios leilões.

Esta página contém uma TextArea onde são apresentados os leilões (A informação é mostrada automaticamente ao abrir esta página), um botão que permite atualizar a informação dos mesmos, uma ComboBox para escolher em qual dos leilões se quer licitar, uma TextField para se poder colocar o Valor a licitar, e um botão que efetua essa mesma licitação. Abaixo, está ilustrada essa mesma página.

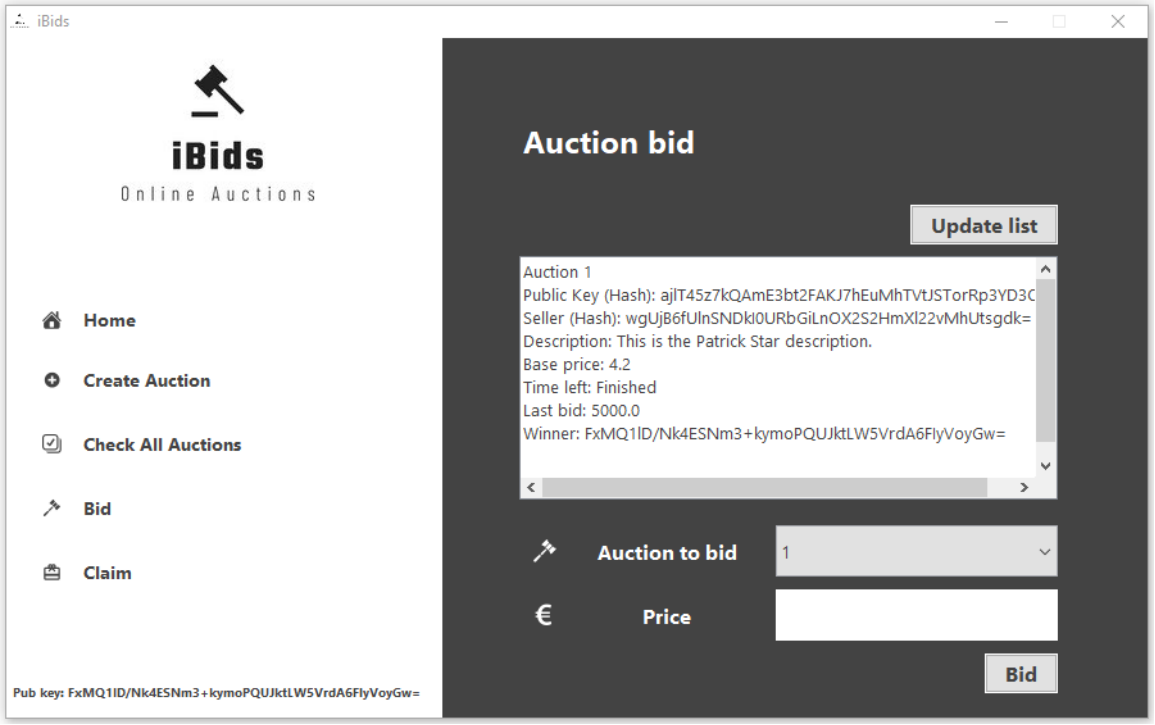

Figura 3.10: Bid

Na imagem anterior pode ver-se o leilão que foi usado para mostrar a página de Leilões Ganhos pelo utilizador. É possível ver que o vencedor foi o próprio utilizador (Através da sua chave), a licitação vencedora, o estado do leilão, a descrição do produto, o preço base, e as chaves do leilão e do vendedor.

A figura abaixo mostra o mesmo leilão em curso. Neste caso, será o mesmo que o apresentado anteriormente para explicar os leilões ativos na janela Check All Auctions.

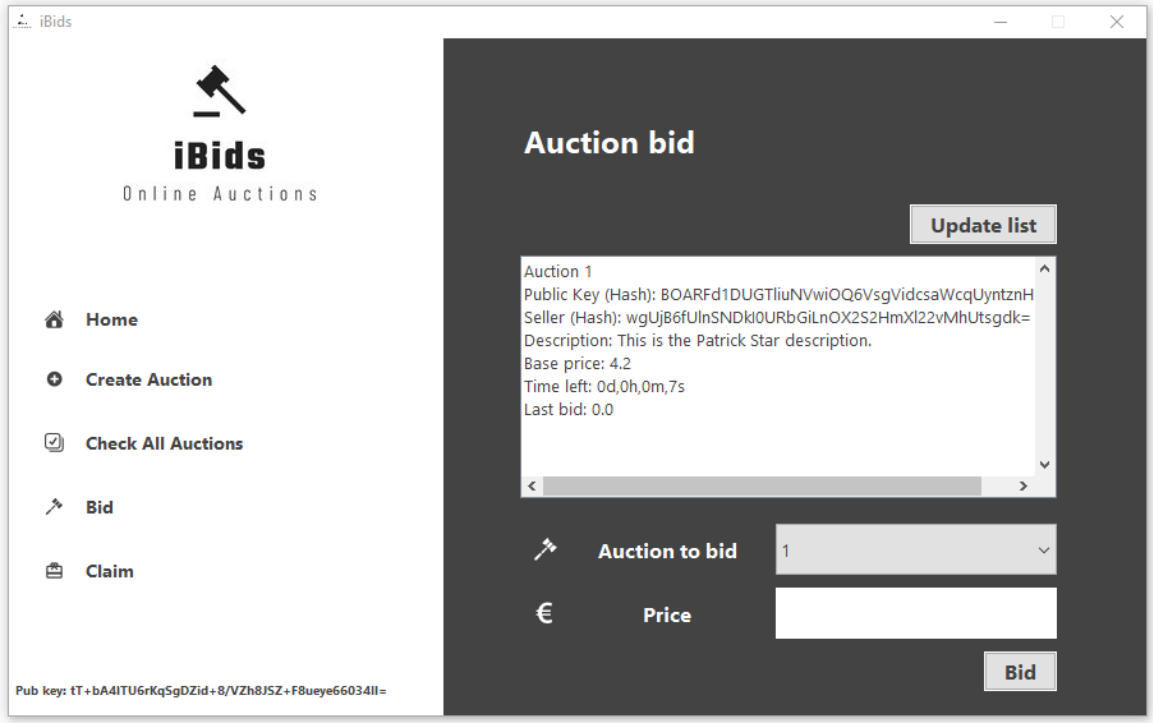

Figura 3.11: Leilão em que pode ser efetuada uma licitação

Ao efetuar uma licitação, uma mensagem de sucesso aparece. Caso o utilizador tente licitar sem colocar um valor ou inserir um tipo de dados errado, ou tentar licitar num leilão que já terminou, mensagens aparecem para o notificar. Todas essas mensagens são ilustradas abaixo.

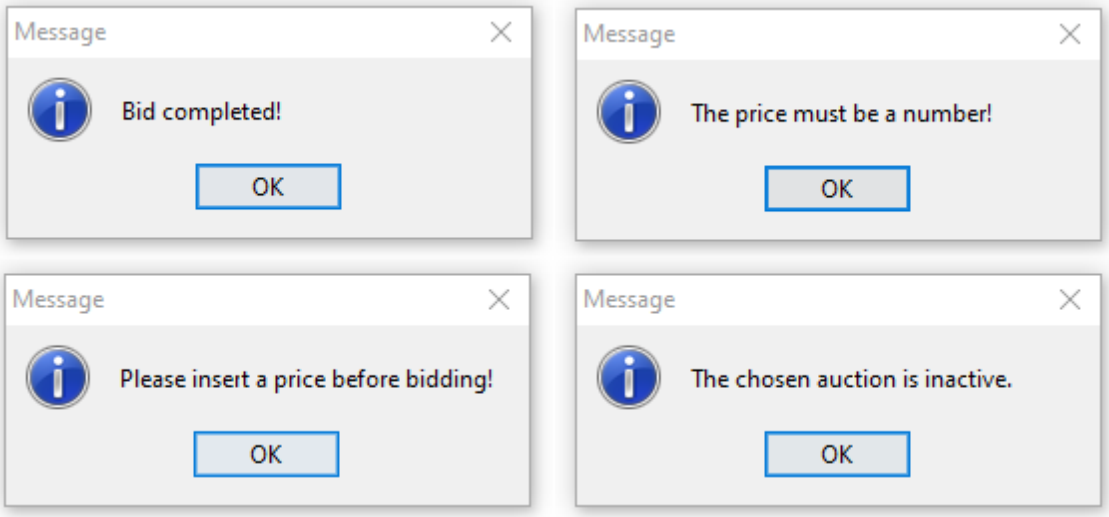

Figura 3.12: Mensagens possíveis ao efetuar um leilão

Claim

Nesta página, o utilizador pode reivindicar "o prémio". Aqui, apenas aparecerão os leilões que o utilizador ganhou, assim como toda a informação (Essa informação aparece na TextArea e aparece assim que a página é carregada). Caso exista mais que um leilão para reivindicar, o utilizador pode escolher qual deles através da ComboBox.

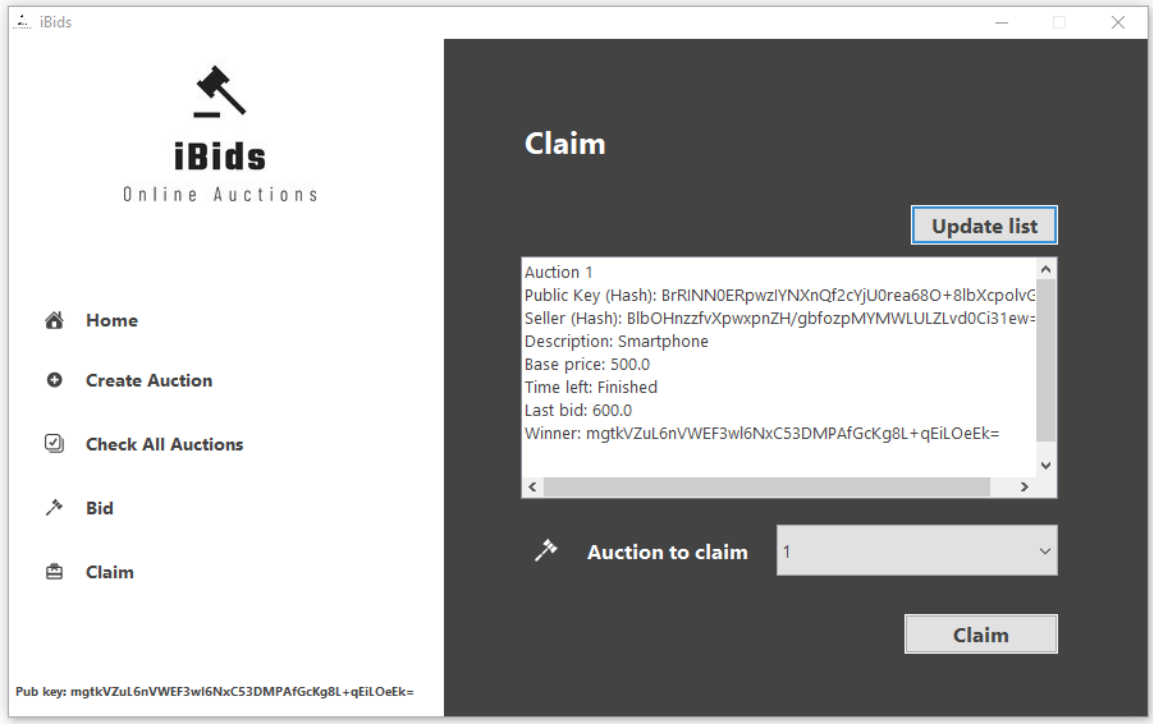

Figura 3.13: Leilões que podem ser reivindicados

Pressionando o botão de **Claim**, o utilizador recebe um recibo do servidor (Neste caso é apresentado ao utilizador como forma de mensagem), em como esse leilão está validado, e foi realmente ganho por ele.

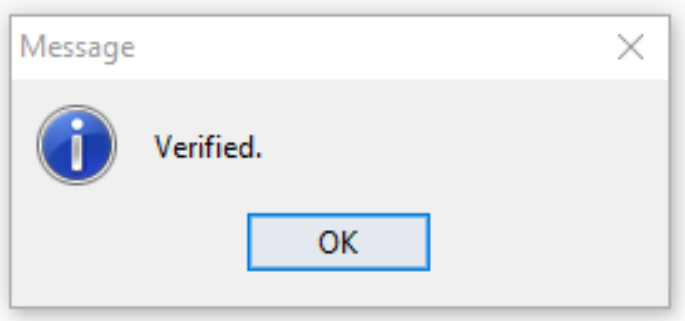

Figura 3.14: Recibo

Se o utilizador atualizar a informação, em frente à chave do vencedor do leilão, aparecerá **"Verified by Server"**.

A mesma informação é apresentada na página **Check All Auctions**, ao pressionar o botão **Won**. Abaixo são apresentados ambos os casos, na página **Claim** e **Check All Auctions**, respetivamente.

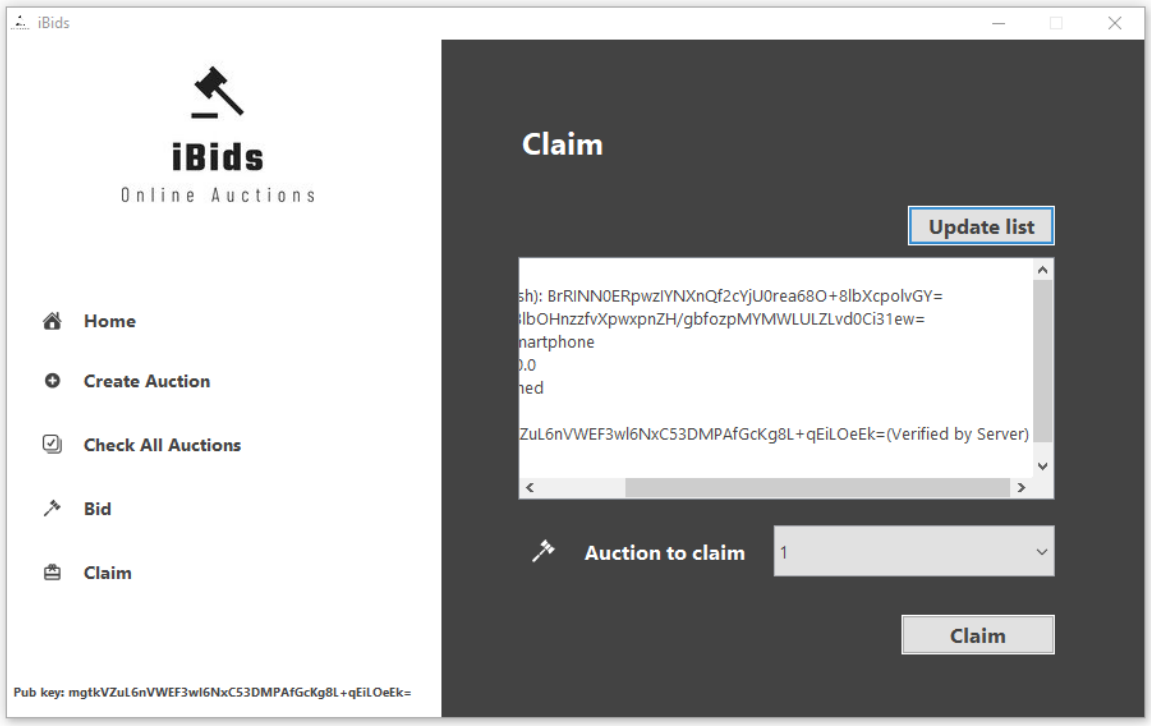

Figura 3.15: Leilão verificado na página Claim

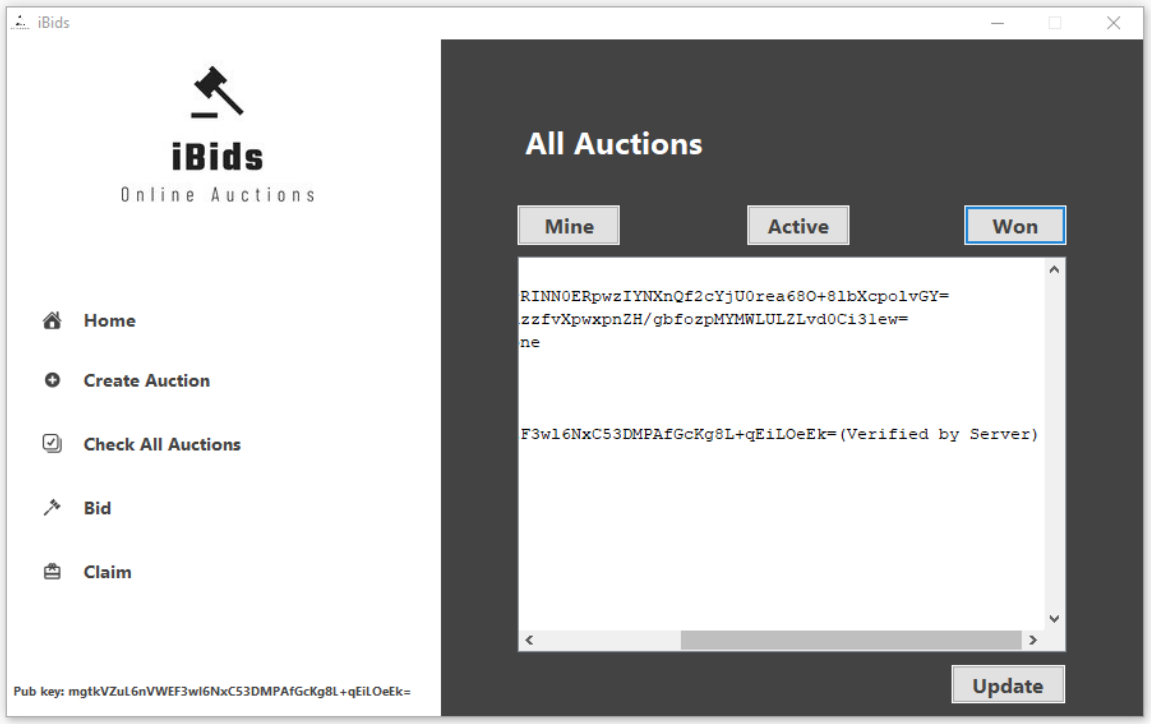

Figura 3.16: Leilão verificado na página Check All Auctions

Capítulo 4

Base de Dados

Para cada uma das funcionalidades implementadas no servidor, é realizada uma conexão à base de dados por intermédio do driver JDBC, permitindo o envio de queries para guardar informação, e da mesma forma recolher informação da base de dados para preencher as estruturas do servidor. Foram criadas tabelas para guardar dados relativos aos utilizadores (tabela *user*), aos leilões (tabela *auction*) e às licitações (tabela *bid*). Adicionalmente, existe também a tabela *shamir* para guardar os componentes da divisão de Shamir de uma chave de um leilão.

Modelo Relacional

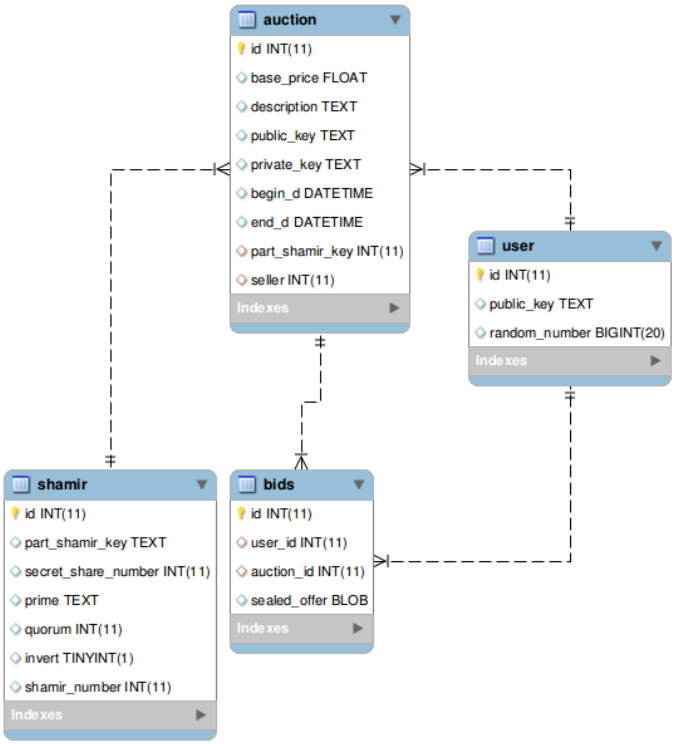

Figura 4.1: Modelo relacional da base de dados.

Visto a base de dados não ser uma componente vital para o funcionamento da aplicação e por manifesta falta de tempo, não foi totalmente implementada.

Capítulo 5

Regulamento Geral de Proteção dos Dados

O **RGPD** só é aplicável quando um dado sistema guarda e/ou processa informações pessoais dos seus utilizadores (clientes). No caso deste sistema, uma vez que um dos principais focos é o de manter o anonimato dos utilizadores, cada utilizador só é identificado pela sua chave pública, que não constitui um dado pessoal. Assim, o **RGPD** não é aplicável neste sistema.

Capítulo 6

Software Assurance Maturity Model

Este capítulo é inteiramente dedicado à aplicação do modelo de maturidade *SAMM* (*Software Assurance Maturity Model*). O objetivo do *SAMM* passa por melhorar a segurança no desenvolvimento de software através da classificação de algumas práticas de segurança (definidas pelo modelo) em níveis de maturidade. Esses níveis vão de um a três (sucessivamente mais seguros e formais) e as empresas devem definir em que níveis se encontram. Partindo de uma primeira classificação, procuram-se atingir níveis superiores de maturidade nas práticas que se acharem convenientes. O nível de integração deste modelo depende muito dos casos em que está a ser aplicado, não só por motivos económicos mas também pelo rigor com que é aplicado. No nosso caso - por ser um serviço web que (tenta) garantir as propriedades de segurança já referidas - é importante perceber quais as práticas que pretendemos garantir a curto prazo (urgentes) e a médio/longo prazo.

Para a aplicação deste modelo, consideramos apenas três das seis fases definidas pelo mesmo (*Assessment*, *Set the Target* e *Define the Plan*). As três fases correspondem a uma primeira classificação, à escolha dos níveis que se pretendem atingir e ao processo como serão atingidos, respetivamente. Para cada uma das fases está atribuída uma secção onde são explicadas todas as decisões efetuadas.

6.1 Aplicação do modelo

O *OWASP* (*Open Web Application Security Project*) disponibiliza uma *toolbox* em *Excel* que acompanha estas três fases do *SAMM*. Contudo, utilizaram-se alguns pressupostos na aplicação do modelo. O primeiro é que não existem várias equipas de projeto na empresa (existe apenas uma), pelo que os conjuntos de respostas que se referem à quantidade de equipas é respondida apenas com *No* ou *Yes, the majority of them are/do*.

Outro aspeto importante lida com o facto de não se pretender atribuir o máximo nível de maturidade em todas as atividades. Em primeiro lugar porque algumas atividades não são aplicáveis às características e/ou práticas da empresa. Em segundo porque o custo de evoluir de um nível para outro pode não ser vantajoso por diversos fatores.

Finalmente, a integração do modelo foi efetuada ainda antes da disponibilização do primeiro produto de software, pelo que na última função de negócio (*Business function*) - *Operations* - todas as respostas foram negativas.

6.1.1 Assessment

Para esta fase é disponibilizado uma *sheet* no ficheiro da *toolbox* que permite responder (quatro opções) a um conjunto de questões para os vários níveis de maturidade de cada prática de segurança. À medida que as respostas são escolhidas, vai sendo calculado um *rating* para cada nível de maturidade (entre 0 e 1).

As respostas foram efetuadas por três entidades diferentes da empresa. As respostas não idênticas entre estes elementos foram levadas para discussão entre outros elementos da empresa. Esta abordagem possibilitou aumentar a qualidade da primeira avaliação (não se recorreu a uma auditoria devido à reduzida dimensão da empresa). Os resultados desta fase encontram-se em anexo.

6.1.2 Set the target

Esta fase consiste em definir que atividades se pretendem melhorar dentro de cada prática de segurança. Dada a sua importância, cada prática de segurança é abordada individualmente nas próximas sub-secções.

Nesta fase realiza-se um conjunto de atividades que é distribuída nos três níveis de maturidade. Cada uma possuí diferentes objetivos, aumentando sucessivamente de dificuldade. O primeiro nível tem como objetivo estabelecer guiões estratégicos para melhorar a segurança no desenvolvimento de software dentro da organização. O segundo consiste em medir o valor/importância dos dados, a criticidade do software e a tolerância da aplicação relativamente aos riscos ao qual está sujeita. O último nível tem como objetivo estimar despesas de segurança no desenvolvimento de novas camadas de negócio.

Esta prática tem uma elevada utilidade no que concerne às aplicações web. Consiste em efetuar pesquisas e identificar normas/*standards* com os quais é exigida a conformidade. Está orientada fundamentalmente em torno do pessoal que direta ou indiretamente afetam a maneira pela qual a organização constrói ou usa software. Além disso, permite instruir as entidades internas ou o pessoal a seguir normas definidas e assim estarem em conformidade.

O nível de maturidade obtido é 1, mas é necessário melhorar este nível. Deste modo, as pesquisas/normas realizadas ou estabelecidas pela organização permitem identificar possíveis mudanças futuras que possam estar sempre em conformidade com os requisitos mais relevantes. Contudo, com esta prática é importante fornecer tempo suficiente para que a auditoria aconteça.

Esta prática foca-se na formação do pessoal envolvido no desenvolvimento do software. Os conhecimentos das entidades da empresa acompanham o ciclo de vida do software, e são importantes para projetar, desenvolver e implementar software seguro. Com a melhoria no acesso à informação, as equipas de projeto possuem outras condições para identificar e mitigar pro-ativamente os riscos específicos de segurança aplicáveis à organização. Logo, um dos principais objetivos é assegurar a formação de todos os funcionários.

O nível de maturidade encontra-se no segundo nível. Visto que já existe pessoal formado anualmente, não existe necessidade de amadurecer esta prática de segurança. Evitam-se, desta forma, custos de recursos humanos e despesas gerais na implementação do serviço de certificação de funcionários.

Esta prática assume um papel preponderante no caso de aplicações web, e procura identificar os riscos existentes no desenvolvimento do tipo de software e do ambiente no qual este se executa. Note-se que nestas condições a suscetibilidade da aplicação a ataques aumenta significativamente porque existem mais pontos de contacto (*browser*, rede, interoperabilidade, etc). Uma vez que a classificação desta prática na fase anterior (*Assessment*) indica um nível de maturidade de nível 0, considerou-se prosseguir com o amadurecimento na fase *Define the Plan*.

Relativamente a esta prática de segurança, o principal objetivo é especificar o comportamento do software com base em possíveis riscos. A conclusão da primeira fase determinou que esta se encontra no primeiro nível de maturidade. Assim como a prática anterior, é importante introduzir conceitos sobre segurança o mais cedo possível. O principal objetivo desta prática passa por desenvolver atividades que formalizem/documentem esses conceitos para utilização na especificação de requisitos. Apesar de, durante o desenvolvimento deste projeto vários desses conceitos terem sido abordados, nunca houve a preocupação de os formalizar/documentar.

A prática *Secure Architecture* tenta oferecer um conjunto de medidas que permitem desenhar e construir software seguro por defeito. Dadas as características desta prática e tendo em conta o nível de maturidade calculado na fase anterior (0) considerou-se a possibilidade de aumentar o nível de maturidade para o primeiro nível. Uma vez que é a primeira aplicação desenvolvida pela empresa, os esforços para cumprir com níveis de maturidade superiores ao nível 1 podem não ser compensáveis. Por exemplo, se futuramente a empresa desenvolver software com características muito diferentes da atual, não interessa ter estabelecido regras *standard* se estas não forem aplicadas à maioria dos casos.

Esta prática foca-se essencialmente na avaliação do *design* e arquitetura para problemas relacionados com a segurança. Isto permite detetar problemas de arquitetura de forma ainda precoce no desenvolvimento do software e evita que posteriormente seja necessário fazer *refactoring* devido a problemas de segurança que implica custos altos para a empresa. Neste momento o nosso software encontra-se no nível de maturidade 1 perto de já estar no 2. É necessário então fazer uma revisão formal segura do *design*.

Esta prática foca-se na inspeção do código fonte e ao nível da configuração do *software* de forma a encontrar vulnerabilidades de segurança. Este tipo de vulnerabilidades é por norma fácil de entender conceptualmente, mas por vezes até os melhores programadores cometem alguns erros que deixam o seu código aberto a ataques. Neste momento ainda não alcançamos o primeiro nível de maturidade pois não estamos ainda a fazer revisões à implementação.

Esta prática foca-se em inspecionar o *software* em *runtime* de forma a encontrar vulnerabilidades na segurança. Isto permite identificar erros que poderiam não ter sido previstos na lógica de negócio anteriormente produzida. Neste momento também ainda não alcançamos o primeiro nível de maturidade pois não realizamos ainda nenhum tipo de teste em *runtime* que permita verificar se existem de facto vulnerabilidades em certos protocolos utilizados.

Nesta prática de segurança são avaliados os mecanismos de suporte oferecidos às equipas em questões relacionadas com problemas de segurança e o seu desempenho na resolução e divulgação desses problemas, estes mecanismos consistem numa equipa especializada na resolução de problemas de segurança e nos processos de identificação, divulgação e documentação dos mesmos. O nível de maturidade é 0, uma vez que não existe nenhum grupo de suporte ou processos de tratamento de falhas de segurança.

Considerando as dimensões da organização e considerando que todos os elementos da organização têm um conhecimento intermédio em questões de segurança, não é necessário formar uma equipa especializada na resolução de problemas de segurança. No entanto é importante definir processos de tratamento das falhas de segurança, portanto esta é uma prática que a organização vai considerar numa fase mais avançada. Não será adotada na próxima fase uma vez que o projeto está numa fase inicial de desenvolvimento.

Esta prática consiste na avaliação dos processos de manutenção do projeto, incluindo a documentação dos requisitos de segurança dos ambientes de execução, a verificação de atualizações de componentes de *software* de outras organizações, sistema de *patches*, e *software* adicional para promover a segurança do projeto. O nível de maturidade é 0, pois de momento não forma definidos mecanismos de atualização do *software*, como também não foram analisados os requisitos de segurança dos ambientes de execução.

Destas medidas a mais importante é a definição de processo de *patches* para tratar situações críticas. Esta prática é necessária para possibilitar a correta atualização de possíveis falhas de segurança. Considerando os custos associados a uma manutenção não eficiente, estes sobrepõem-se aos custos associados a implementação desta prática. No entanto esta prática não é relevante para a próxima fase, uma vez que nos encontramos numa fase inicial de desenvolvimento.

Nesta prática é avaliada a documentação de configurações de segurança e os guias de operação referentes às questões de segurança do projeto que são entregues ao utilizador, a verificação e correção dos mesmos e a possibilidade de verificar a autenticidade do *software* disponibilizado pela organização, através da assinatura digital. O nível de maturidade é 0, dado que estes casos ainda não foram considerados.

Destas medidas as mais importantes são a assinatura do *software* disponibilizado e a documentação das configurações de segurança, estas medidas são relevantes para a correta utilização do *software* e para a segurança do utilizador. Como ambas as medidas não requerem grandes investimentos de recursos da organização, estas medidas serão adotadas na próxima fase.

6.1.3 Define the plan

A última fase consiste no estabelecimento dos objetivos a atingir no final de cada período. Foram considerados quatro períodos de três meses cada. As atividades que se consideraram fáceis e rápidas de atingir os principais objetivos no final do primeiro período, assim como atividades que se verificaram estar num estado mais crítico. Para os restantes períodos é tido em atenção a dificuldade em aplicar a atividade, a utilidade desta dentro da empresa e a relação custo/resultados.

Nesta fase o nível de maturidade inicial é praticamente nulo mas é possível melhorar de forma a se construir uma lista com os riscos mais críticos ao nível da camada de negócios causado pelo software. É importante definir um guião personalizado que aborda a segurança em função das necessidades da organização. Estabeleceu-se que o programa de garantia deveria crescer com o tempo, atingindo-se dessa forma o nível de maturidade 1,08 no final dos primeiros 4 períodos.

Nesta fase, achou-se necessário proceder com o amadurecimento da prática. Estabeleceu-se que a melhoria desse nível deveria ser gradual, desde o primeiro até ao último período, onde se estabeleceu como limite o último nível de maturidade. Apesar dos custos a empreender neste nível, tais como - i) garantir as ferramentas de desenvolvimento ou licenças para automatizar auditoria contra padrões internos;ii) manutenção contínua de auditoria portões e processo de exceção e a construção ou licença de padrões internos em conformidade com padrões internos e auditoria; - neste nível podemos alcançar resultados satisfatórios, tais como: i) a visibilidade no nível da organização de riscos aceites devido a não conformidade; ii) garantia concreta para conformidade no nível do projeto; acompanhamento preciso do passado histórico de conformidade do projeto; e iii) processo eficiente de auditoria alavancando ferramentas para cortar o esforço manual. Portanto, independentemente dos custos, é imperial que se atinja o último nível de maturidade.

Tendo em conta a importância da prática, decidiu-se que após o final do primeiro período a maturidade se deve encontrar no primeiro nível. Ambas as atividades descritas no documento oficial (desenho de protótipos/modelos de ameaças e simulação de perfis de atacantes relacionadas com a aplicação) são importantes para a aplicação a desenvolver. Ao dominar estas atividades adquire-se a capacidade de perceber que fatores podem afetar negativamente a empresa durante o processo de construção, ou seja, ponderar entre melhorias de segurança ou desenvolvimento de *features*.

O *trade-off* de resultados/custos no segundo nível de maturidade parece positivo, pelo que decidimos incluí-lo a médio prazo (período 4). Não apenas pelas vantagens/resultados de aplicar as atividades, como também pelas dependências existentes com amadurecimentos de outras práticas (nível dois na prática *Secure Architecture*). Não se pretende, a longo prazo, atingir o último nível de maturidade ou melhorar alguma das atividades, uma vez que a gestão, manutenção e atualização dos detalhes de ataques mais comuns (ou possíveis) é custosa, assim como descobrir todas as dependências no software de terceiros.

Como já foi referido na fase anterior, decidiu-se melhorar a prática *Security Requirements*. Não só pela importância das suas atividades, como também pela dependência existente entre as atividades desta prática e da *Thread Assessment* (aumentar a maturidade da prática atual para o segundo nível está relacionado com o aumento da anterior para o primeiro nível). Dito de outro modo, ao aumentar uma das práticas ajuda diretamente a aumentar a outra (mais precisamente na segunda atividade que visa especificar requisitos de segurança com base nos riscos existentes). A nova atividade - que não depende de outras referidas até agora - é a construção de uma matriz em que se são os diferentes utilizadores do sistema e as capacidades que devem possuir sobre os recursos do sistema. Por estes motivos, decidiu-se aumentar o nível de maturidade para o segundo nível na primeira fase.

Pelo menos a última atividade do terceiro nível de maturidade pode/deve ser alcançada, uma vez que parte grande do trabalho de a incluir já é feito na prática *Policy and Compliance*. É de notar que esta prática atinge o último nível de maturidade (em princípio) no último período (após 12 meses). Este facto é aproveitado através da inclusão da segunda atividade, que permite a realização de auditorias sobre os requisitos de segurança especificados pelas equipas.

A primeira atividade do primeiro nível de maturidade permanece inalterado. A empresa possuí um repositório com ferramentas recomendadas e seguras, mas não restringe o desenvolvimento de software a essas ferramentas. As ferramentas recomendadas podem variar consoante o tipo de aplicação a desenvolver e por esse motivo não é integrado um conjunto restrito de ferramentas.

A segunda atividade é bastante importante e consiste em manter *checklist* para o pessoal da empresa (principalmente de cargos superiores) terem uma noção do nível de conhecimento em que determinadas entidades se encontram no que diz respeito à segurança. Deste modo, é possível atribuir atividades mais críticas a entidades com maiores conhecimentos e prevenir a introdução de vulnerabilidades por erros humanos. Dada a pequena dimensão da empresa, não se colocou a hipótese de as atividades do segundo nível de maturidade serem integradas antes do quatro períodos.

Dado que já estamos perto de atingir o nível 2 de maturidade nesta prática, o objetivo será no mínimo alcançá-lo e se possível chegar mesmo ao nível 3. Para tal, necessitamos de fazer uma verificação formal do *design* da aplicação, isto é, dos requisitos de segurança explícitos e para conseguirmos chegar ao terceiro nível, teremos de desenvolver diagramas de como os dados serão processados e tratados de forma a verificar se existem falhas e, caso tudo esteja mediante os parâmetros essenciais, pode-se passar finalmente à fase seguinte.

Tendo em conta a importância desta prática e o baixo nível de maturidade, o mínimo exigível será atingir o primeiro nível de maturidade. Para tal, é necessário que sejam feitas verificações dos requisitos mais comuns de segurança mas também, uma verificação de pedaços de código que sejam considerados de maior risco.

Esta prática também tem muita importância na fase de verificação e procura de vulnerabilidades no *software* e dado que ainda não foram realizados quaisquer testes, o nível mínimo a alcançar nesta próxima fase será o primeiro, e se possível tentar chegar já ao segundo. Para atingir o objetivo é necessário começar a derivar casos de teste de requisitos de segurança já conhecidos assim como realizar testes de penetração em novos modelos de *software*.

Na fase inicial está prática tem o valor de maturidade 0, pretende-se aumentar o nível de maturidade mas não é necessário, para já, chegar ao nível 1. Para atingir este objetivo é necessário definir e implementar os mecanismos de verificação da autenticidade do *software* disponibilizado e proceder a documentação das configurações de segurança relevantes para o utilizador final.

Em suma, o que denotamos através da análise das várias fases e práticas de segurança que o *SAMM* explicita, é que devido a estarmos ainda numa fase inicial da produção do *software* não nos encontramos nos melhores níveis de maturidade, e como tal é necessário fazer planos para que no fim do ciclo seja possível que o nosso software já se encontre preferencialmente no nível 3 em todas, ou quase todas, práticas que o *SAMM* enuncia.

Capítulo 7

Conclusão

Segundo o que foi apresentado ao logo deste relatório, pode ser afirmado que, embora nem todas as características do sistema tenham sido implementadas segundo o que era previsto de início, como acontece com a base de dados, por exemplo, a maior parte das funcionalidades propostas desde o início foram de facto implementadas com sucesso. Uma das maiores dificuldades foi a implementação das técnicas criptográficas em conjunto com o modelo servidor-cliente, já que a transmissão de dados implica sempre que sejam serializados e desserializados os objetos que são enviados pelas *sockets*. É relevante ainda salientar que a implementação do método de divisão de *Shamir* causou algumas dificuldades, principalmente na gestão das diferentes partes na altura da recuperação da chave. Podemos no entanto, afirmar que a implementação foi bem sucedida, a divisão de *Shamir* é sempre efetuada quando um leilão é terminado e um vencedor determinado. (Isto é sobre a interface)

É importante referir novamente que não foi criada uma interface gráfica em *Web* devido aos problemas encontrados com os *certificados/keystores* usados na parte de segurança e implementação. O conceito está implementado e funciona.