

SSI

(Computer Systems Security)

Threat model

João Azevedo & Paulo Araújo

University of Minho, Braga, PT
{a85227,a85729}@alunos.uminho.pt

Abstract. Este documento contém um modelo de ameaças detalhado para um Sistema de Identificação Móvel e Digital cujo objectivo passa por substituir os habituais documentos de identificação pessoal num processo similar, digital, que pode ocorrer sob várias condições e das quais precisamos de implementar determinadas garantias. Esta tecnologia e, de forma geral, este sistema aplicacional pretende que entidades qualificadas possam fazer a verificação e prova de identidade de indivíduos garantindo uma série de propriedades de segurança que passam pelas diferentes camadas do modelo *STRIDE* desenvolvido pela *Microsoft*.

Keywords: Threats · STRIDE · Vulnerabilities · Security · Mobile and Digital Identification

1 Caso de estudo

Um serviço de identificação móvel e digital trata uma área do desenvolvimento de *software* que exige uma análise criteriosa no que toca aos componentes que o definem e avaliam se, no afim de contas, um indivíduo é realmente quem diz ser que é.

Esta questão de prova de identidade pressupõe uma interação entre partes (entidades) que atuam em sinergia com vista a tomar decisões associadas, essencialmente, a questões relacionadas com a lei imposta num determinado país.

Em particular, podemos definir três entidades principais: os portadores (cidadãos) que possuem um determinado *software* que visa provar a sua identidade, os verificadores (indivíduos qualificados, tipicamente agentes da polícia) que possuem um outro produto que comunica com o portador e com uma terceira entidade confiável, a chamada entidade emissora de documentos autênticos, íntegros e com informações, geralmente, fornecidas por entidades governamentais.

1.1 Breve descrição do funcionamento

O sistema apresenta uma forma de substituir um documento de identificação pessoal comum, por um digital (*mID*), que se encontrará armazenado num *smart-*

phone e, escusado será dizer que estes dados sensíveis devem ser preservados no que toca à sua integridade e autenticidade.

De todas as vezes que é necessária a verificação de identidade de um dado indivíduo, o verificador procede à transferência de uma série de atributos do portador, atributos esses permitidos pelo indivíduo, ou então de uma autorização dada implicitamente pelo portador para que o verificador possa contactar directamente a entidade emissora. A decisão de fazer a comunicação *online* ou *offline* cabe inteiramente ao agente verificador.

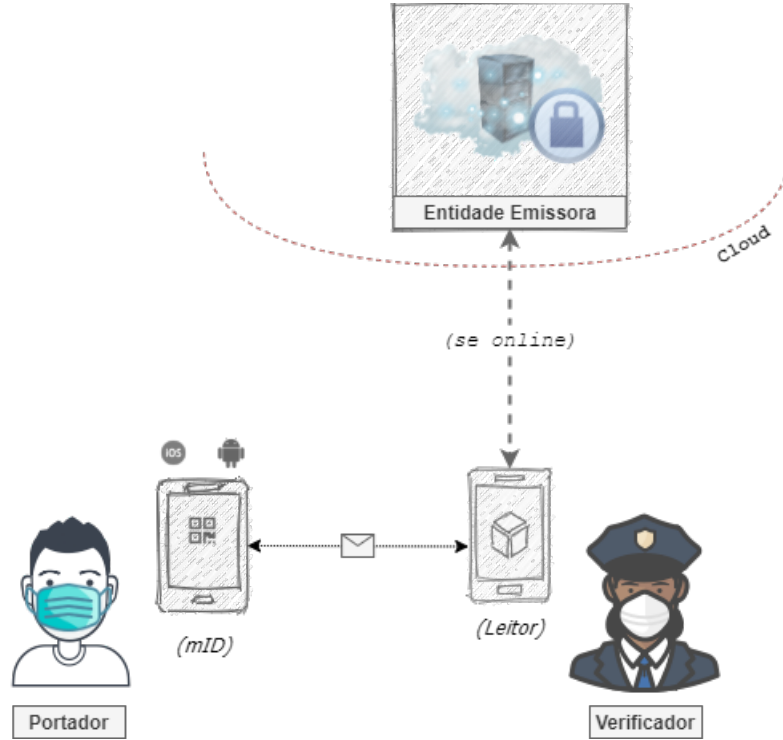


Fig. 1. Esquema geral da interação entre as entidades.

Será importante agora perceber as diferentes partes do sistema no que toca a formas de comunicação, fluxo de dados e protocolos especificados, estando a próxima secção inteiramente dedicada a isso mesmo.

2 Modelação do sistema

Como já foi referido anteriormente, o sistema em questão é composto por três entidades principais: uma aplicação do portador (*mID*), uma aplicação do verificador (*leitor*) e, por fim, o *backend* (entidade emissora).

A interação é feita segundo um conjunto de protocolos de rede para comunicações *online* e *offline*, sendo esta a essência do sistema: providenciar a prova de identidade sem que seja necessário comunicação com a *Cloud*, no entanto, será esta a porção do *software* mais complicada de implementar mecanismos de segurança que minimizem os riscos de exposição e acessos ilegítimos aos dados.

2.1 Aplicações do portador e do verificador

App. *mID*: Esta aplicação está inserida em qualquer *smartphone* que suporte sistemas operativos como o *Android* e o *IOS*, e possui três modos de funcionamento.

Numa primeira utilização, será necessário a transferência dos dados associados ao documento de identificação do indivíduo. O *download* em si pressupõe uma comunicação entre a *mID* e o *backend* utilizando um conjunto de protocolos de rede englobados na pilha *TCP/IP*, como por exemplo *HTTP/HTTPS* (camada de aplicação) que podem (ou não) estar a ser aplicados. A conexão com o *backend* exige uma autenticação completa, numa primeira instância, e utiliza o formato popular *JSON* para envio e receção de dados.

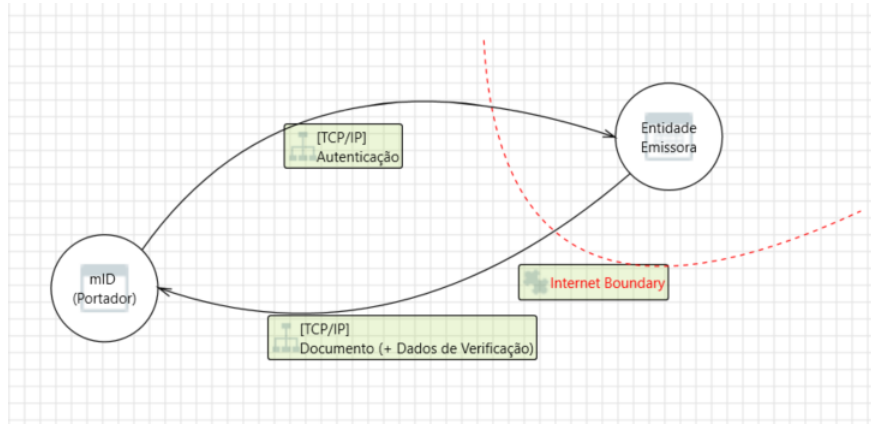


Fig. 2. Interação inicial entre a *mID* e o *backend*.

Nas utilizações posteriores já não é necessária uma autenticação explícita ao sistema, existindo um outro tipo de validação dos pedidos por um mecanismo que tenha por base a autenticidade do utilizador portador da aplicação.

Aqui é importante ressaltar que o fluxo de dados deve garantir a integridade e a confidencialidade do conteúdo transacionado e também dos dados armazenados no telemóvel do portador, pois só assim é que será possível implementar uma identificação móvel e digital.

App. leitora: Esta aplicação é idêntica à *mID*, funcionando portanto também em sistemas *Android* e *IOS*, não sendo isto estritamente necessário, podendo correr num outro ambiente operacional, desde que suporte os protocolos referidos em cima.

É portanto necessário a autenticação de um dado verificador que utilize o seu leitor. Tal permitirá auditar, ou seja, produzir registos das comunicações, pedidos, entre outros, feitos com o indivíduo portador.

A aplicação leitora pode também proceder a dois modos de funcionamento, *online* e *offline*, cujo processo será explicado de seguida.

Interação *mID* e leitor: A interação destes sistemas visa funcionar em ambientes com rede (acesso à *internet*) e, também, em condições adversas, onde apenas as tecnologias de comunicação à curta distância podem atuar. Em ambas as situações a aplicação leitora inicia a conexão com um *QR Code* fornecido pela *mID* que contém todas as informações para poder iniciar a conexão, que pode usar tecnologias como **BLE** - *Bluetooth Low Energy*, **NFC** - *Near Field Communication* ou **Wifi-Aware**.

- **Modo *offline*:** Aqui dá-se a transferência, por parte do leitor, de todos os dados (atributos) de identificação, assim como outros para verificar a sua integridade e autenticidade.

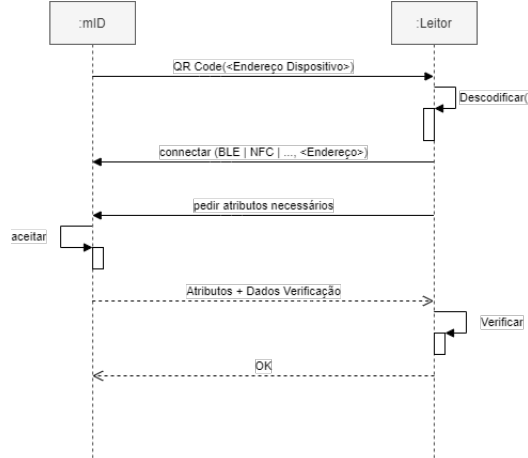


Fig. 3. Diagrama exemplo da comunicação *offline*.

- **Modo *online*:** Já neste modo, não são fornecidos nenhuns atributos por parte do portador, sendo apenas necessário obter um *token* de autorização, em que este sim será dado pela aplicação *mID* do portador. Este *token* serve como cabeçalho de um pedido à entidade emissora para obter um

subconjunto de dados que o portador permite que sejam partilhados, por isso é prioritário que estes não se alterem por parte do leitor.

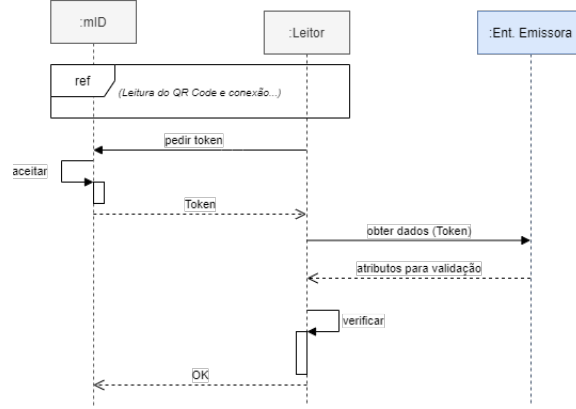


Fig. 4. Diagrama exemplo da comunicação *online*.

Relembra-se também que todo o sistema de *logs* se encontra presente de forma implícita, isto é, cada vez que um dado agente requer um subconjunto de atributos ao sistema, este pedido fica catalogado associando um dado verificador a um portador.

2.2 Entidade emissora (*Backend*)

Esta entidade representa a infraestrutura que comunica com ambas as aplicações anteriores fazendo chegar os dados dos documentos requisitados, assim como comunicar diretamente com a base de dados.

É extremamente importante que este sistema não tenha problemas relacionados com a disponibilidade dos seus serviços, apesar desta ser uma tarefa difícil visto que a comunicação é feita via rede pública. Por outro lado, deve ser também crucial garantir que os documentos por ela fornecidos são íntegros e autênticos tendo de, para isso, prover de mecanismos que garantam isso mesmo.

3 Modelação de ameaças

Deste modo, após uma extensiva modelação do sistema de identificação móvel e digital, temos em mão a questão perceber o que pode correr mal na interação entre as diferentes entidades e o que é crucial proteger para manter uma análise bem estruturada, seguindo, por exemplo, o modelo de ameaças *STRIDE*.

A análise deve, ao nosso ver, de seguir um padrão estruturado, pelo que isso também se reflete na definição do sistema feita até agora. Por outro lado, focar

apenas no *software* provocaria uma idealização centralizada de um tema tão importante como este pois, para além de todas as camadas lógicas da aplicação, temos de perceber quem poderia querer atacar certos *layers* e porquê. Assim, o nosso modelo de ameaças será baseado numa visão mista de ambas as estratégias de modelação.

Daremos também início à deteção de ameaças e vulnerabilidades através da divisão do problema em várias partes, correspondentes a várias secções do sistema. Para isso, vamos igualmente partir da divisão idealizada até agora pela análise de requisitos fornecida, aplicando depois a categorização *STRIDE* de modo a catalogar possíveis problemas deste sistema.

3.1 Aplicação *mID*

Primeira questão: Que interações existem com a aplicação? Ora existem várias, sendo a primeira delas a **transferência de um documento através da rede pública**, seguindo protocolos de comunicação do modelo *TCP/IP*.

Aqui poderão existir uma série de problemas inesperados:

1. (***Spoofing***) Será que a aplicação *mID* está a comunicar realmente com a Entidade Emissora? E vice-versa?

Muitos protocolos do modelo *TCP/IP* não providenciam mecanismos de autenticação da origem e do destino de uma mensagem, tornando-os vulneráveis a ataques de falsificação de dados no envio e receção a partir de um dado *host*. Usar apenas, por exemplo, o *source/destination IP* para autenticação é uma das pontes principais para este tipo de ataques.

2. (***Tampering***) A integridade do endereço para conexão é garantida?

Certamente que o acesso ao servidor principal recorre a uma API *REST*, através de um endereço de uma entidade conhecida. Este problema pode estar aliado ao anterior, no entanto é mais difícil de acontecer, visto que envolve alterar a própria configuração da aplicação, podendo até ser resolvido com algum tipo de autenticação feita entre os *hosts*.

3. (***Spoofing e Information Disclosure***) A informação partilhada está devidamente encriptada? Poderão existir interceções *MITM - Man in the middle*?

Problemas de encriptação: Em primeiro lugar, nada nos é dito acerca da camada *TCP/IP* exata que cobre a comunicação com o *backend*, estarão *http*, *https* ou *tls/ssl* presentes na camada de aplicação? (por exemplo). Como é garantida a confidencialidade do conteúdo? Em segundo lugar, a autenticação das mensagens enviadas é também uma ameaça neste comunicação pois recorrer a encriptação não garante que as mensagens estão autenticadas.

Interceções das comunicações: Sem excluir os problemas anteriores, queremos introduzir esta possível ameaça que surge como consequência da não

resolução outras. É uma forma de ataque em que o intrusor interceta toda a comunicação entre, neste caso, o *mID* e o *backend*, através de uma conexão nova em ambas as partes. É claro que o risco associado a esta vulnerabilidade é altíssimo, isto é, ter acesso, por exemplo, às credenciais e/ou informações do documento de identificação afeta a confidencialidade e dá um chave de entrada em outros serviços governamentais que utilizem as mesmas credenciais.

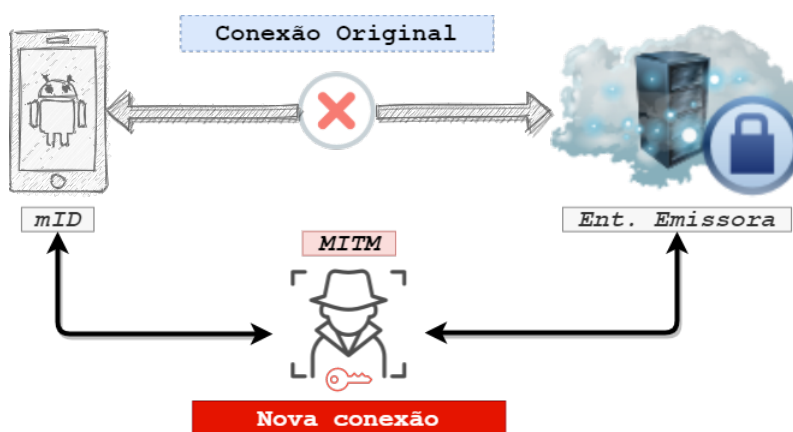


Fig. 5. Exemplo de ataque - Man in the middle.

Caso estes fatores sejam comprometidos podemos ter uma série de fraquezas típicas associadas a este modelo e que podem representar um ataque de elevado risco expondo geralmente muitas informações confidenciais, como as credenciais de autenticação, os atributos transferidos, entre outros. As categorias de ameaças mais comuns para este problema em questão envolvem *Spoofing* e *Information Disclosure*.

Segunda questão: Temos de **analisar o documento em si**, visto que este será crucial na comunicação *offline* e, sem ele ou sem a garantia de integridade e autenticidade do mesmo, não será possível fazer uma prova de identidade de um portador. Este é, inicialmente, guardado no dispositivo do portador e atualizado, periodicamente, com os atributos mais recentes relativos a um indivíduo. Mas e se a comunicação for *offline*? Existirá problemas em utilizar *JSON - JavaScript Object Notation* para transferir os dados? Os dados de verificação são suficientemente seguros? São estas as questões que nos levaram a identificar as seguintes ameaças:

1. (**Data Tampering**) Armazenar um documento (*bytes*) em *Android/IOS* garante um acesso exclusivo à aplicação? Conseguimos alterar os atributos?

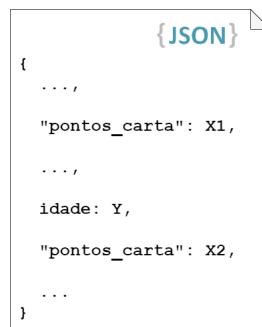
Quando se opta por utilizar um qualquer dispositivo (*smartphone*) para ser a ponte entre os dados e o verificador, é prioritário perceber se o sistema operativo (e a sua versão) apresentam algum tipo de vulnerabilidade com risco associado à manutenção da integridade do documento de identificação. Neste caso, só sabemos que pode ser *Android* ou *IOS*, tanto para o *mID* como para a aplicação *leitora*. Ora, segundo o “*The Open Web Application Security Project (OWASP)*” (1), o armazenamento de dados é um dos *top 10* problemas de segurança, havendo ainda assim alternativas para evitar que *malware* possa aceder a esses dados, estratégia que será revelada mais à frente.

2. (**Data Tampering**) Ainda sobre o mesmo tópico: Os dados armazenados estarão íntegros? Mesmo armazenando uma estrutura extra de verificação da integridade dos dados, será essa estrutura suficiente a ataques de *brute-force* ou algo similar na partilha da mesma e dos dados de verificação?

A verdade é que para armazenar estes dados e uma estrutura espelho do documento principal armazenado em memória não garante, inteiramente, que não se detetem padrões por ataques de tentativa-erro. A quantidade de *bits* usados será um fator extremamente importante, ou até uma dependência direta entre atributos numa relação verificada, por exemplo, em estruturas *blockchain*, assunto que será melhor tratado na secção de tratamento dos ameaças.

3. (**Data Tampering**) Por fim, mas menos comum, será que as estruturas *JSON* enviadas/recebidas são verificadas? Pode a aplicação estar sujeita a *JSON injections*(3)?

Embora menos comum, isto é uma realidade para algumas aplicações. A estruturação do *JSON* é muitas vezes assumida como correta, isto é, contendo os campos que se planeou no envio da estrutura pelo servidor, ou na receção da mesma pela aplicação. Considere-se o seguinte exemplo:



```

{
  ...,
  "pontos_carta": X1,
  ...,
  idade: Y,
  "pontos_carta": X2,
  ...
}

```

Fig. 6. Exemplo de uma estrutura *JSON* mal formada.

A estrutura apresentada claramente contém uma pequena falácia, mas que pode ser ignorada pela aplicação. Os *parsers JSON* assumem que o *input* é propriamente higienizado, isto é, sem atributos repetidos, utilizando o último atributo definido como aquele que fica guardado. Neste caso, um atacante poderia alterar o atributo “pontos_carta”, por exemplo, para um valor que lhe convenha, ainda na transferência do documento pela rede pública, por exemplo.¹

Não apresenta um elevado risco na medida em que este tipo de ameaças se resolve facilmente com boas práticas de desenvolvimento, referenciadas mais à frente.

3.2 Aplicação leitora

Numa fase inicial temos o trabalho facilitado na medida em que os problemas relacionados com os Sistemas Operativos ou do armazenamento de dados já estão identificados.

Relembrando que, apesar dos exemplos de problemas serem diferentes, o foco mantém-se na memória do dispositivo, como por exemplo: garantir que o verificador não altera o conjunto de atributos que pode aceder; garantir que o mesmo faz a devida autenticação com a entidade emissora, entre outros.

O que será mais crítico é poder garantir um correto registo de *logs* para o sistema final: É imperativo que se registre que atributos foram acedidos, a que horas e por quem, identificando devidamente os intervenientes, portador e verificador.

Fundamental: Tal necessidade permite tratar problemas de não-repúdio nas ações feitas pelos verificadores, pois questões de lei do país e proteção de dados estão sempre envolvidas. Por isso, visto que não temos informação de como são feitos os registos, definimos um conjunto simplificado de fraquezas que podem estar adjacentes ao sistema de *logs*:

1. (**Repudiation**) Podemos estar a incorrer num problema de *Insufficient Logging*, como referido no *CWE(4)*? Ou o contrário, *Logging of Excessive Data*?

Um *log* individual deve mostrar, sem introduzir qualquer tipo de dúvida ou redundância, o evento que ocorreu, isto é, sem omitir detalhes importantes. Por outro lado, introduzir demasiada informação ou muitos registos de *logs* para, por exemplo, operações mais simples, faz com que estes ficheiros sejam muito difíceis de processar. No modo *offline*, por vezes, os dispositivos estão sujeitos à acumulação contínua de *logs*, apesar de ser por um curto espaço de tempo, podemos tornar o processo distribuído de processar esses *logs* muito complexo, ou até provocar um *Denial-of-Service* não intencional ao servidor que gere este processo.

¹ O processo de *JSON injection* pode ser consultado neste url: <https://www.acunetix.com/blog/web-security-zone/what-are-json-injections/>.

2. (**Repudiation**) O Regulamento Geral de Proteção de Dados pode estar a não ser cumprido, i. e., *Insertion of Sensitive Information into Log File*, como referido no *CWE*?

Isto não é necessariamente um problema existente na aplicação, no entanto, é nos fornecida pouca informação sobre este processo. A catalogação desta fraqueza surge como medida preventiva que se resolve facilmente.

3.3 Interação *mID* e leitor

Todas as operações necessárias a uma prova de identidade, seja pela transferência de toda a informação de um documento armazenado ou de apenas um subconjunto deste, ocorrem entre um dispositivo leitor e um dispositivo do portador. Estas operações podem assumir dois modos: *offline* e *online*.

Modo *offline*: Neste modo, a principal possível ameaça prende-se em algo já enunciado anteriormente, é necessário garantir a integridade e autenticação dos dados armazenados no dispositivo móvel do portador, visto que na leitura dos dados o portador envia os atributos de identificação e os dados necessários para a verificação dos mesmos.

Modo *online*: Aqui surge uma nova componente denominada *token* que irá servir de autorização para a leitura dos dados autorizados por parte do leitor, enviando em seguida o mesmo para entidade emissora de modo ter acesso aos dados autorizados pelo portador.

A questão que se prende é a seguinte: Sendo o *token* um chave de acesso ao *backend* e aos atributos de um dado portador, este é seguro?, longo/forte? aleatório o suficiente?

É necessário garantir que quando se partilha o *token* do portador para o leitor não existem alterações do mesmo, ou seja, que este se mantém íntegro, nem que depois de o leitor armazenar o *token* em sua posse o vá alterar. Querendo com isto garantir que o leitor não possa ler mais do que aquilo que o portador lhe deu autorização, nem utilizar de forma indevida este *token*.

Analisando agora a comunicação em si, em ambos os modos a esta é estabelecida com uma das ferramentas descritas de seguida:

1. (**Spoofing**) QR Code comunicação inicial e validade

A comunicação é sempre iniciada pelo dispositivo do portador através de um QR Code contendo toda a informação necessária para que o dispositivo leitor o encontre e inicie a conexão. Como é possível garantir que os dados de identificação presentes no QR Code para iniciar a comunicação são referentes ao próprio dispositivo móvel do portador alvo da identificação e não de um outro dispositivo próximo?

Visto que nos são um pouco desconhecidas estas capacidades do QR Code deduzimos não ser possível existir uma relação intrínseca entre o código apresentado e o dispositivo móvel que o apresenta. Deste modo é possível

que os atributos de identificação que o leitor se encontra a ler após a conexão do QR Code apresentado possam ser referentes a uma identidade próxima e não do alvo de identificação.

Por outro lado, este código pode ser utilizado para acessos ilegítimos se o deixarmos disponível, ou seja sem data de expiração definida, ou mesmo se este persistir após uma primeira utilização.

2. (*Spoofing e Denial of Service*) BLE - Bluetooth Low Energy

Os ataques mais comuns referentes ao BLE residem no facto do mecanismo de *broadcast* BLE poder causar negação de serviço e condições Man-in-the-Middle.

Falsificação da identificação: Os dispositivos periféricos BLE são descobertos através uma comunicação *broadcast*, no qual o periférico transmite continuamente pacotes para os dispositivos centrais de escuta. Pacotes esses que normalmente incluem o endereço do dispositivo, o nome do dispositivo e informações sobre a conectividade do mesmo. Geralmente, assume-se que os dispositivos têm endereços diferentes. E se um dispositivo anuncia que tem o mesmo endereço de um outro? Como é que os dispositivos centrais decidem que informação está correta?

Aplicado ao nosso problema em concreto como é que o dispositivo móvel do leitor identifica verdadeiramente o dispositivo do portador através da conexão BLE se tem vários dispositivos a fazerem-se passar pelo dispositivo portador?

Possível negação de serviço: Um outro problema reside no facto de como os dispositivos periféricos são descobertos através de *broadcast*, sendo possível que um dispositivo invasor esteja sempre a enviar pacotes para a rede com os seus dados impedindo que o dispositivo central encontre outros dispositivos. Aplicado ao nosso caso o leitor pode não conseguir se conectar ao dispositivo portador alvo, visto que existe um invasor a congestionar a rede de observação do leitor.

3. (*Information disclosure*) NFC - Near Field Communication

Como o NFC é uma interface de comunicação sem fios, é óbvio que a espionagem é uma questão importante. Quando os dois dispositivos estão a comunicar via NFC, eles usam ondas *RF* - *Radio frequency* para falar um com o outro e um invasor pode usar uma antena para também receber estes sinais. Depois por meio de pesquisa ou da utilização da força bruta este invasor pode extrair os dados do sinal RF transmitido. A verdadeira questão aqui reside no facto de quão perto um invasor precisa de estar para conseguir recuperar um RF utilizável, infelizmente ao que parece não existe um número exato e tudo depende de um grande número de variáveis.

4. (*Information disclosure*) WiFi-Aware

Um pouco idêntico ao NFC, o Wifi-Aware permite que um dispositivo transmite *pings* curtos para que outros dispositivos possam saber um pouco sobre

ele sem estabelecer uma conexão bidireccional. Podendo também existir invasores à escuta de informação que irá ser transmitida entre o leitor e o portador, bastando para tal os dispositivos concordarem todos que têm uma aplicação comum que optam por usar.

No fundo, em comunicações à curta distância, podemos incorrer em ameaças relacionadas com negação de serviço ou espionagem do conteúdo partilhado entre os dispositivos, levando-nos à nossa última questão:

1. A utilização de um formato de serialização de dados, em particular, o formato *CBOR* — *Concise Binary Object Representation*, pode ser uma ameaça?
 - A questão vai mais longe que isso, a utilização do *CBOR* por si só apenas permite ter uma estrutura idêntica ao *JSON*, mas em formato binário. Deste modo, garantimos uma representação concisa dos dados e uma transferência mais rápida, com o custo de não o conseguir interpretar como *plain-text* (*human-readable*).
O problema surge quando a aplicação, assim como no caso do *JSON* apresentado, não garante que o que descodifica é a estrutura que tinha sido planeada como protocolo de comunicação entre a *origem-destino*, podendo estar sujeita a injeção de código que possa expor dados da mesma.
2. E quanto à confidencialidade da comunicação?
 - Não foram referidas nenhuma aplicação de encriptação na comunicação entre os dispositivos pelo que isto deve ser descrito como uma ameaça. Em comunicações com tecnologias limitadas, que não dispõem camadas de segurança viáveis de encriptação fim-a-fim, podemos ter atacantes passivos a visualizar o conteúdo partilhado entre as partes.

3.4 Entidade Emissora (*backend*)

Esta entidade é responsável por autenticar utilizadores, fornecer informações sobre os mesmos (documentos) e também persistir um conjunto de registos sobre eventos que ocorreram no sistema. De ressaltar que o esquema aplicacional mostrado na figura 7 é apenas uma representação possível, certamente, com omissão de algumas comunicações (não sendo isso o foco principal deste modelo).

Fundamental: Será importante então perceber se o conjunto de componentes (e respetivas versões) apresentam um esquema vulnerável no contexto deste sistema.

Nesta secção em particular, a análise será feita, maioritariamente, por um estudo geral de risco e de frequência com que certas ameaças representam vulnerabilidades nas componentes do *backend*.

– Infraestrutura do *backend*

Na figura seguinte apresentamos um possível esquema da arquitetura aplicacional que fica exposta para os utilizadores, ou seja, o *web server* principal e de toda a componente de gestão, não acessível nem ao portador nem ao verificador.

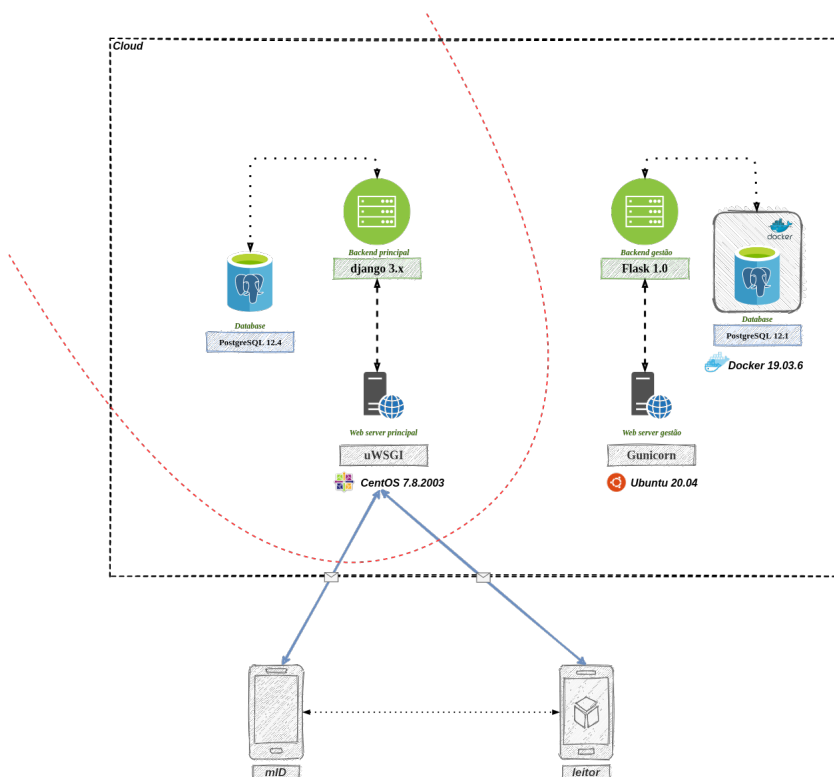


Fig. 7. Backend (possível esquema) do sistema de identificação móvel.

A primeira questão a colocar será: em termos gerais, quais são os principais problemas associados ao *backend* de uma aplicação comum? Poderá esta informação identificar ameaças importantes?

Segundo a *OWASP*(1), referida anteriormente, o *top 10* de vulnerabilidades para *APIs*, estabelecido em 2019, indica que, entre outros, problemas rela-

cionados com *Data Exposure*, *Injections*, *Lack of resources* e *Insufficient logging and monitoring* representam alguns dos pontos mais críticos verificados em várias aplicações que têm por base uma comunicação via *web api*:

OWASP API Security Top 10 Vulnerabilities 2019	
API1:2019	Broken object level authorization
API2:2019	Broken authentication
API3:2019	Excessive data exposure
API4:2019	Lack of resources and rate limiting
API5:2019	Broken function level authorization
API6:2019	Mass assignment
API7:2019	Security misconfiguration
API8:2019	Injection
API9:2019	Improper assets management
API10:2019	Insufficient logging and monitoring

Fig. 8. “OWASP API Security Top 10 Vulnerabilities 2019”

Isto permite-nos dividir a análise em diferentes componentes:

1. (***Tampering***) Validação de estruturas e *Inputs*:

Mais uma vez, assim como na comunicação entre dispositivos, aqui temos como um dos principais problemas no desenvolvimento de *APIs* a validação dos dados nos vários *endpoints*, não se tratando propriamente de uma vulnerabilidade mas sim de uma ameaça ao bom funcionamento do *backend*. Exemplos disso são injeções em bases de dados *SQL*, *NoSQL*, estruturas *XML* e *JSON* (exemplo já referido noutras secções).

2. (***Information Disclosure***) Excessiva exposição de dados:

Também muito comum, este problema trata o caso em que objectos/dados são expostos ao *client-side* com informação a mais, por vezes filtrada apenas no cliente e, muitas vezes, com a mesma estrutura com a qual são lidos da base de dados original.

3. (***Denial of Service***) Recursos insuficientes, *logging* e monitorização:

Uma estrutura desta natureza vai, com certeza, ter muitos utilizadores, ao mesmo e de forma distribuída a aceder aos recursos do *backend* e também muitos possíveis atacantes, com vetores de ataque relacionados com diminuição da disponibilidade do sistema, factor que será crucial na fase *online* do processo de prova de identidade, fazendo com que o verificador tenha um acesso com muita latência ao sistema.

É através disto que conseguimos identificar a maior ameaça ao *backend*, i.e. uma possível negação de serviço, seja por muitos acessos à API, seja

pelos problemas de *auditing/logging* descrito na secção 3.2, seja pela própria falta de monitorização automática (e manual) ao sistema que pode detetar se diferentes componentes estão a ter mais acessos e por isso devem ser melhor distribuídos.

Segundo a *CVE - Common Vulnerabilities and Exposures*(5), e em particular para o sistema de bases de dados *Postgresql*, podemos verificar a seguinte análise do histórico de vulnerabilidades detetadas desde 1999 até ao ano passado:

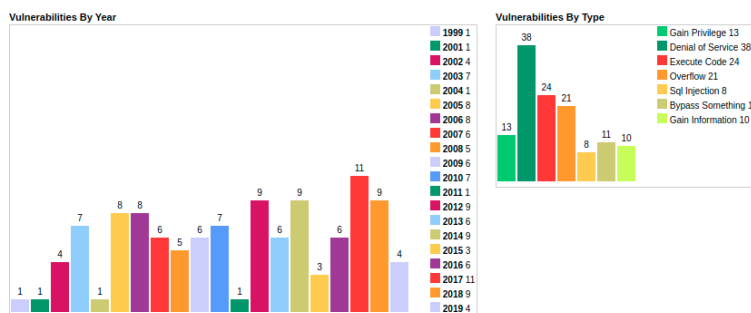


Fig. 9. Vulnerabilidades detetadas no sistema de bases de dados *Postgresql*.

Cerca de 30% das ocorrências estão associadas a problemas de negação de serviço e, por exemplo, na *framework django* temos, da mesma forma, 30% de vulnerabilidades, na mesma janela temporal, relacionadas com *Denial of Service* e uma fatia de 24% relacionados com *XSS - Cross Site Scripting*.

4 Medidas para combater/mitigar as ameaças

Depois de uma análise detalhada de quais as possíveis ameaças para a descrição do sistema fornecida chega a fase de, para cada uma, tentar arranjar uma solução que, mediante o risco associado, evite problemas futuros, geralmente, caros de implementar, ou então que aceite certas vulnerabilidades de forma controlada.

Isto pode ser feito, claramente, com fundamentos de padrões de desenvolvimento de *software* dedicados a cada caso específico, partindo novamente de uma divisão do problema em partes.

4.1 Aplicação *mID* - Autenticação e transferência do documento

Nesta fase foram identificadas ameaças que se incluem nas camadas do modelo *STRIDE: Spoofing, Information Disclosure e Tampering*, afetando a integridade,

autenticidade e confidencialidade, tanto dos dados como da identificação do portador e do *backend*.

Soluções possíveis para as ameaças identificadas:

1. Autenticação das mensagens na receção e no envio nos diferentes *hosts*:
 - (a) Ataques de *spoofing* tendo por base o modelo *TCP/IP* podem ser mitigados introduzindo *firewalls* capazes de analisar profundamente os pacotes ou tomar medidas que verifiquem a identidade dos intervenientes da comunicação;
 - (b) Assumir que o *IP* da conexão é completamente ilegítimo, visto que o mesmo é volátil em dispositivos móveis. Temos de nos concentrar num tipo de autenticação criptográfica, onde cada *host* partilha uma chave e quando se pretende conectar obtém uma chave de sessão encriptada com a chave pública do destino e concatenada nas mensagens para o destino, podendo deste modo autenticar as mensagens trocadas;
2. Autenticação inicial à aplicação:
 - (a) Uma outra sugestão, para a autenticação segura de um portador ou verificador à entidade emissora, seria a utilização de uma autenticação de dois fatores, tão comum hoje em dia. Mesmo que um atacante possa obter as credenciais de um utilizador, acrescentamos um novo *layer* à segurança já implementada pela entidade emissora.
3. Encriptação da informação partilhada:
 - (a) Esta medida aplica-se às várias comunicações existentes entre as aplicações e a entidade emissora e a solução pode passar por introduzir certificados *SSL/TLS* para estaabelecer uma camada de aplicação segura como *HTTPS*.

4.2 Aplicação *mID* - Armazenamento de dados

Nesta fase foram identificadas ameaças que se incluem na camada do modelo *STRIDE*: *Tampering*: afetando, essencialmente, a integridade das estruturas armazenadas.

Soluções possíveis para as ameaças identificadas:

1. Integridade dos documentos armazenados no dispositivo:

- (a) No caso do *Android*, a própria plataforma aproveita a proteção baseada no utilizador *Linux* para identificar e isolar recursos de aplicações, não deixando que diferentes aplicações, possivelmente maliciosas, atuem no mesmo espaço de memória. Essa *sandbox* atua ao nível do *kernel* do OS e, portanto, usando a mesma, temos o problema resolvido. Da mesma forma, no *IOS*, temos a possibilidade de ativar esse ambiente de desenvolvimento de aplicações.
2. Dados de verificação suficientes? bem estruturados?:
- (a) Apesar de haver proteção de acesso aos dados, por processo, durante a execução das aplicações é necessário estabelecer uma estrutura que os verifica quando são transferidos para o dispositivo leitor, garantindo que estes não foram alterados. Como sugerido, a estrutura funciona como um espelho e ao mesmo tempo um *hash-mapping* entre cada atributo armazenado e o seu valor utilizando uma função de *hash* (*integrity check*).
- (b) Aqui deve-se ter em atenção o poder dessa função, recomendando a utilização de muitos bits para o tamanho da codificação, possivelmente *XORs* e adicionando um *salt* (dado aleatório) evitando *dictionary attacks* ou situações semelhantes.
- (c) Uma outra abordagem bastante interessante seria a utilização do cálculo dos valores de *hash* fazendo depender um atributo do outro como se verifica na abordagem descentralizada da tecnologia *Blockchain*. A alteração de um simples **byte** num dos blocos elimina toda a integridade da estrutura para trás, ideal para este sistema.
- Deste modo, cada *payload* de dados do bloco aqui representado contém o atributo de identificação do portador, juntamente com a *hash* do bloco anterior(2).

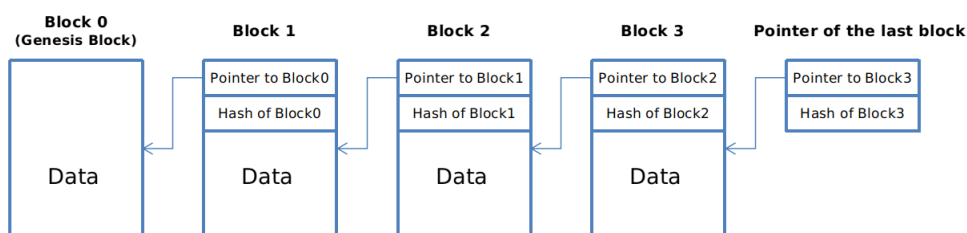


Fig. 10. Exemplo de utilização da tecnologia *Blockchain*.

3. *JSON injections*:

- (a) Não há muito que falar nesta secção, é importante como boa prática sempre tratar os dados que recebemos ou enviamos percebendo a estrutura de dados que estamos a trabalhar, ou seja, *JSON* aqui não será o problema mas sim avaliar todo o *input* deste tipo que pode ser recebido tanto nas aplicações como na entidade emissora.

4.3 Aplicação *leitora*

Nesta fase foram identificadas ameaças que se incluem na camada do modelo *STRIDE: Repudiation* e, possivelmente, *Denial of Service*: afetando, essencialmente, a disponibilidade dos servidores e o não-repúdio dos intervenientes nos eventos.

Soluções possíveis para as ameaças identificadas:

1. Correto processo de *Logging* (nota: não se aplica apenas ao leitor):
 - (a) Aqui é simples, um *log*, neste sistema, deve representar o essencial de um evento, seja ele de requisição de um dado conjunto de atributos a um indivíduo por um verificador, seja ele um registo de acesso indesejado à infraestrutura.
 - (b) Se por um lado queremos assegurar e registar todos os acessos não confiáveis à rede da entidade emissora podemos ter, por exemplo, uma componente/ferramenta de *IAM - Identity and management* como um serviço à parte dos existentes.
 - (c) Se por outro lado queremos garantir que o sistema de *logs* não provoca nenhum tipo de latência no acesso ao sistema para autenticação, *download* de documentos, entre outros, poderia ser viável optar por fazer essa gestão num outro serviço aplicacional assíncrono.
 - (d) A análise passa por um balanceamento entre a disponibilidade da infraestrutura principal para provas de identidade *online* e *offline* e, também, pelo registo de todos os eventos e eventual disponibilização, por questões legais, aos intervenientes, do que ocorreu num dado momento, sem violar a proteção dos dados.

4.4 Interação *mID* e leitor

Nesta fase foram identificadas ameaças que se incluem na camada do modelo *STRIDE: Spoofing* e, possivelmente, *Denial of Service*: afetando, essencialmente, a disponibilidade do leitor e conexões indesejadas.

Soluções possíveis para as ameaças identificadas:

1. Garantir a correta utilização do *token* para o leitor:

- (a) O *Token* deve ser, impreterivelmente, de utilização temporária. Com isto, não permitimos tanta margem de manobra a um possível atacante. O leitor é portanto autorizado a fazer o pedido ao servidor por um determinado período de tempo, invalidando o *token* após esse *time-to-live*.
- (b) O *Token* também deve ser de utilização única. Quer isto dizer que o mesmo se torna inválido num acesso concorrente aos dados, ou que tenha validação nas mãos de um outro utilizador para além do primeiro, o verificador.
- (c) A integridade do *Token* é importante que seja verificada e, visto que o mesmo mapeia uma *hash*, possivelmente, para um conjunto de atributos e não podemos correr riscos de ataques como os referidos anteriormente, *brute-force* por padrões de *hash*, situação que se resolveria facilmente com *hashs* maiores e com *salt*, por exemplo, impedindo que, em tempo útil, o padrão seja decifrado.

2. Utilização de *QR codes*:

- (a) A não definição dos dados que são decodificados nos códigos partilhados não nos permite, com certeza absoluta, assumir na totalidade o que poderá ter sido partilhado. A verdade é que, o problema de identificar o dispositivo pode ir para além do endereço lido no código. Este *Spoofing attack* pode ser resolvido com o seguinte processo, representado no seguinte diagrama:

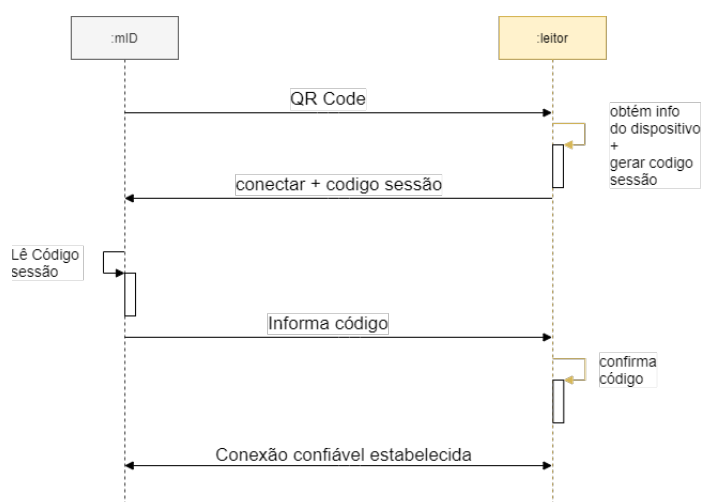


Fig. 11. Exemplo de uma sequência de conexão complementar.

Onde um código de confirmação extra é trocado entre o verificador e o leitor, sem haver intervenção das aplicações, isto é, o portador comunica o código ao leitor e este verifica manualmente.

- (b) O código QR deve seguir a mesma implementação que foi adotada para o *Token*, na medida em que queremos impedir que o verificador, por exemplo, possa guardar o QR Code e utilizá-lo mais tarde de forma ilegítima. Quer isto dizer que este código deve ser temporário e de utilização única, confiando claramente na aleatoriedade garantida pela geração de códigos providenciada pela serialização de conteúdo.
- (c) Por último, para estabelecer uma comunicação segura entre ambas as partes e impedir que outros dispositivos leiam o que está a ser transmitido poderá fazer parte do processo de envio e receção de dados uma troca inicial de chaves, utilizando criptografia de chave pública, e uma chave de sessão (chave simétrica) para encriptar as mensagens. Por exemplo, o seguinte esquema:

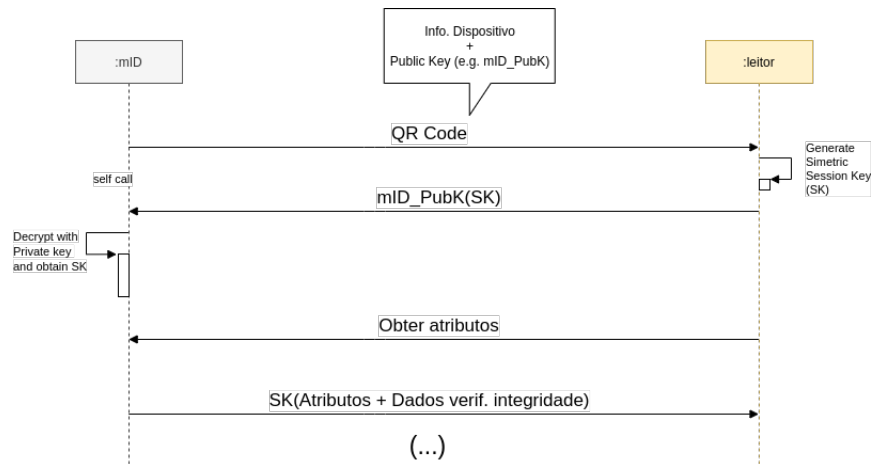


Fig. 12. Exemplo de um possível mecanismo de comunicação seguro.

4.5 Entidade emissora

Nesta fase foram identificadas ameaças que se incluem na camada do modelo *STRIDE*: *Denial of Service* e, possivelmente, *Tampering* e *Information Disclosure*: afetando, essencialmente, a disponibilidade do leitor e conexões indesejadas.

Aqui não vale a pena estar a descrever soluções que surgem da aplicação de outras já referidas até então, por isso, será importante focar o desenvolvimento final do sistema *backend* nos seguintes pontos:

- Validação de dados: Toda a informação que depender de serialização de dados deve garantir que as estruturas lidas seguem um padrão previamente estabelecido, evitando fazer injeção de dados não tratados em métodos da API sem um tratamento prévio;
- Robustez da estrutura: É sem dúvida um passo muito importante e por vezes esquecido no desenvolvimento de *software* de grande escala. Neste caso, devemos concentrar o processo em: combater as ameaças detetadas, passando por intensivos testes de carga e de intrusão e só assim é que conseguimos perceber o que pode correr mal na análise distribuída dos componentes.
- Arquitetura: Propõem-se a introdução de sistemas de deteção de intrusão (IDS) e de monitorização do sistema, assim como uma gestão do processo de *logging* num novo servidor para que o tráfego normal de autenticação não interfira com o registo dos eventos.

5 Análise final e validação do modelo

Ao longo do desenvolvimento deste documento percebemos que grande parte das ameaças surgem da comunicação entre as diversas entidades e a dificuldade, por vezes, concentra-se na manutenção de um canal seguro vs. segurança por mensagem, visto que nem sempre temos a possibilidade de garantir um ou outro tentamos solucionar o problema introduzindo uma camada adicional e mecanismos extra para garantir a confidencialidade e autenticidade das informações partilhadas.

O foco deste sistema é, no entanto, fazer provas de identidade *online* e *offline*, em diversas condições e, por isso, com diferentes exigências de segurança da informação. Por outro lado, numa infraestrutura aplicacional desta natureza e com as aplicações reais em informações sensíveis de indivíduos/cidadãos e na necessidade de as preservar e assegurar a prevenção de acessos ilegítimos, temos um risco muito elevado associado à maior parte das ameaças encontradas.

Para concluir, é crucial assegurar confidencialidade das informações partilhadas em tecnologias vulneráveis como o *BLE*, que não garantem um túnel seguro, assegurar a integridade dos documentos e respetivos atributos em dispositivos móveis e, de modo a garantir a disponibilidade do sistema, distribuir o mesmo consoante a necessidade descrita pela sua constante monitorização, seja por deteção de intrusões, balanciamento de carga, entre outros, não deixando o processamento de *logs* no servidor principal, dedicando uma componente extra para isso mesmo.

Assim, como desenvolvedores de *software* temos de concentrar as fases iniciais de desenvolvimento nos requisitos não nos desligando da modelação de ameaças que nos consegue mostrar muitos problemas do sistema antes de escrever uma única linha de código.

References

- [1] Owasp.org. 2020. OWASP Foundation — Open Source Foundation For Application Security. [online] Available at: <https://owasp.org/>
- [2] Medium. 2020. Hash Pointers And Data Structures. [online] Available at: <https://medium.com/@zhaohuabing/hash-pointers-and-data-structures-f85d5fe91659>.
- [3] JSON Injections. [online] Available at: <https://www.acunetix.com/blog/web-security-zone/what-are-json-injections/>.
- [4] Cwe.mitre.org. 2020. CWE - Common Weakness Enumeration. [online] Available at: <https://cwe.mitre.org/>.
- [5] Cvedetails.com. 2020. CVE Security Vulnerability Database. Security Vulnerabilities, Exploits, References And More. Available at: <https://www.cvedetails.com/>.
- [6] Cwe.mitre.org. 2020. CWE - Common Weakness Scoring System (CWSS). [online] Available at: https://cwe.mitre.org/cwss/cwss_v1.0.1.html.