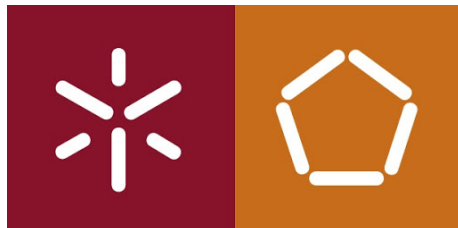


# Segurança de Sistemas Informáticos

## Trabalho Prático 1 Desmaterialização de Documentos de Identificação Pessoal

a85954 Luís Ribeiro



Mestrado em Engenharia Informática  
Universidade do Minho

## Conteúdo

<b>1</b>	<b>Desmaterialização de Documentos de Identificação Pessoal (Aplicação mID)</b>	<b>3</b>
1.1	Aplicação do Utilizador (Aplicação mID)	3
1.2	Aplicação do Responsável de verificação (Aplicação Leitor)	4
1.3	Entidade Emissora ( <i>Back-end</i> )	4
<b>2</b>	<b>Modelação do Sistema</b>	<b>6</b>
2.1	Threat Model	6
2.2	Modelo do Sistema	6
<b>3</b>	<b>Ameaças e Vulnerabilidades do Sistema</b>	<b>7</b>
3.1	Definição das Métricas do CVSS	8
3.1.1	Exploitability metrics	8
3.1.2	Scope	8
3.1.3	Impact Metrics	9
3.2	STRIDE	10
3.3	Application Layer	11
3.3.1	Comunicação Aplicação-Entidade	11
3.3.2	Comunicação Aplicação-Aplicação	15
3.3.3	Vulnerabilidades nas Aplicações - Android/iOS	20
3.4	Service Layer	23
<b>4</b>	<b>Observações finais</b>	<b>33</b>
4.1	Resumo e Análise de Risco	33
4.2	Possíveis melhorias	35
<b>5</b>	<b>Conclusão</b>	<b>37</b>

# 1 Desmaterialização de Documentos de Identificação Pessoal (Aplicação mID)

O objetivo deste trabalho é o estudo de um sistema de **Desmaterialização de Documentos de Identificação Pessoal**, a um nível da **Segurança** do Sistema, onde devemos levantar os requisitos de segurança e identificar e explorar os principais riscos a este Sistema.

Este sistema consiste na digitalização de documentos de identificação pessoal, de forma a usar em oposição ao uso de documentos físicos. Surge na era digital, onde tudo é desmaterializado para facilitar o quotidiano da população. Neste caso, está em causa a possível validação de identificação através de uma aplicação telemóvel, havendo assim uma alternativa à validação física do documento.

O sistema em estudo, é constituído por 3 principais componentes:

- **Aplicação do Utilizador:** Aplicação do utilizador, onde os dados e os documentos associados a ele estão digitalizados.
- **Aplicação do Responsável de verificação:** Aplicação que lê e valida os dados recebidos da aplicação do utilizador.
- **Entidade Emissora:** Entidade onde os dados são confirmados pelo Verificador, e são transferidos para a aplicação do Utilizador.

## 1.1 Aplicação do Utilizador (Aplicação mID)

Corresponde a uma aplicação móvel para sistemas operativos *Android* e *iOS*. Esta aplicação armazena os dados de um documento de identificação e os elementos usados pela aplicação leitor para verificar a sua integridade e autenticidade.

Na primeira vez que o portador usa a aplicação, esta conecta-se com a infraestrutura de uma entidade emissora de documentos (usando **comunicação TCP/IP**) para o *download* de todos os dados associados a este documento. Para isso, o cidadão autentica-se ao respectivo serviço, por forma a iniciar a transferência do seu documento. Esta operação repete-se periodicamente para eventuais atualização de dados, neste caso, sem recorrer a uma autenticação explícita ao sistema.

O processo de prova de identidade, isto é, a transferência de informação relativos a um documento de identificação, ocorre entre um dispositivo leitor e o dispositivo do utilizador. Este processo assume 2 modos:

- **Offline:** O dispositivo do utilizador transfere diretamente os seus atributos de identificação para o leitor, assim como os dados necessários para a sua verificação.
- **Online:** A aplicação leitor transfere um *token* de autorização para que responsável de validação consulte diretamente a entidade emissora do do-

cumento, garantindo assim, os dados mais recentes sobre o respetivo cidadão.

Em ambas as situações, o estabelecimento da comunicação é, sempre, iniciada pelo dispositivo do utilizador através de um *QR Code* contendo toda a informação necessária para que o dispositivo leitor o encontre e inicie a conexão. Tecnologias usadas:

- BLE - *Bluetooth Low Energy*
- NFC - *Near Field Communication*
- *WiFi-Aware*

A informação transferida para o dispositivo leitor é suportada por mensagens codificadas no formato **CBOR** (*Concise Binary Object Representation*).

## 1.2 Aplicação do Responsável de verificação (Aplicação Leitor)

Assim como a aplicação mID, a aplicação leitor corresponde a uma aplicação móvel para sistemas operativos *Android* e *iOS*, ou qualquer outro dispositivo que suporte os 3 protocolos de comunicações e operações aqui definidos. Através desta aplicação, um verificador estabelece comunicação com o portador e solicita atributos suficientes para o identificar no âmbito de um serviço em particular.

Cabe ao verificador decidir se a operação será em modo *offline* ou em modo *online*. No modo *online* de operação, é preciso garantir que a aplicação leitor não é capaz de alterar a lista de atributos autorizados pelo portador antes da consulta à infra-estrutura da entidade emissora.

## 1.3 Entidade Emissora (*Back-end*)

Entidade com o poder de emitir e conferir autenticidade a um documento de identificação pessoal.

Como descrito nas secções anteriores, esta entidade comunica-se com a aplicação mID e com aplicação leitor via rede pública recorrendo a tecnologias suportadas pela arquitectura TCP/IP. Apesar de não intervir nas operações em modo *offline*, é imperativo a garantia da disponibilidade dos serviços por ela oferecidos. Abaixo estão identificados os detalhes da infra-estrutura.

- Sistema operativo do servidor web: CentOS 7.8.2003
- Backend principal: Django v3.0
- Servidor web: UWSGI
- Base de dados principal: PostgreSQL 12.4
- Sistema operativo do serviço de gestão do sistema: Ubuntu 20.04

- Backend de gestão: Flask 1.0
- Servidor web: Gunicorn
- Base de dados de gestão: PostgreSQL 12.1 (a correr em um container Docker versão 19.03.6)

## 2 Modelação do Sistema

Nesta secção é apresentado uma modelação do sistema para facilitar a compreensão e análise das possíveis ameaças que podem surgir em cada componente do Sistema.

### 2.1 Threat Model

*Threat modeling* é um processo onde ameaças potenciais, tais como vulnerabilidades estruturais ou a ausência de segurança, podem ser identificadas e enumeradas. O objetivo da modelação é fornecer defesas que precisam de ser implementadas, dado a natureza do sistema, o tipo de possíveis invasores e os que mais os motiva a atacar o sistema.

Podemos agora identificar as entidades externas, os processos do sistema e entidade responsável pelo armazenamento de dados:

- **Entidades externas:** Utilizador e Verificador.
- **Processos:** Aplicação leitor, aplicação do utilizador/portador e entidade emissora.
- **Base de dados:**
  - *Online:* PostgreSQL
  - *Offline:* Dispositivo leitor

### 2.2 Modelo do Sistema

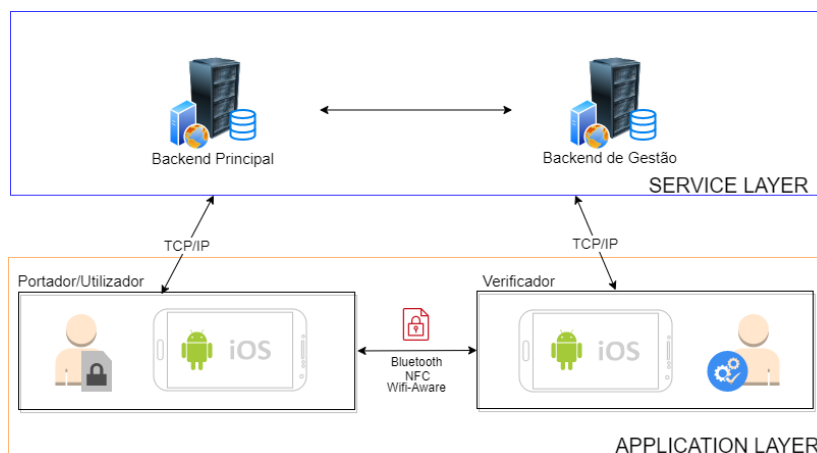


Figura 1: Modelação

### 3 Ameaças e Vulnerabilidades do Sistema

Esta secção dedica-se à identificação de possíveis ameaças ao Sistema utilizando o modelo **STRIDE**. Para isso, vamos ter em conta a modelação do Sistema (Figura 1) e também a exploração de certas vulnerabilidades encontradas dentro do Sistema. Ou seja, se houver uma vulnerabilidade associada a uma ameaça, será apresentada também.

***Application Layer*** refere-se às comunicações dentro do Sistema, comunicações entre aplicações e comunicações entre aplicação e entidade emissora. ***Service Layer*** refere-se ao processamento e às componentes dentro da entidade emissora.

As primeiras 2 secções descrevem o modelo *STRIDE* e as métricas de catalogação de vulnerabilidades (métricas *CVSS*).

As seguintes vão se dedicar ao estudo dessas ameaças dentro de cada *Layer* do Sistema, separadamente (*Application Layer* e *Service Layer*).

### 3.1 Definição das Métricas do CVSS

Antes de apresentar as vulnerabilidades encontradas no sistema, apresenta-se nesta secção, a definição de cada métrica do CVSS, para facilitar a compreensão dos vetores de cada vulnerabilidade.

O *CVSS score* é dado pela soma do *Impact Subscore*, do *Exploitability Subscore* e mais alguns fatores temporais.

#### 3.1.1 Exploitability metrics

- **Attack Vector (AV):**

Esta métrica reflete o contexto pelo qual a exploração da vulnerabilidade é possível. O valor da métrica será maior quanto mais remoto um atacante pode estar, de modo a explorar a componente vulnerável.

O **Attack Vector** tem 4 possíveis valores: *Network*, *Adjacent Network*, *Local* e *Physical*.

- **Attack Complexity (AC):**

Esta métrica descreve as condições, além da capacidade ou controlo do invasor, que devem existir para explorar a vulnerabilidade. Estas condições requerem acumulação de informação sobre o alvo em causa, ou a presença de certas configurações de sistema, ou até exceções computacionais.

O **Attack Complexity** tem 2 possíveis valores: *High* e *Low*.

- **Privileges required (PR):**

Descreve o nível de privilégios que um atacante deve ter acesso antes de explorar uma vulnerabilidade, com sucesso.

O **Privileges Required** tem 2 possíveis valores: *None*, *Low* e *High*.

- **User Interaction (UI)**

O UI reflete a necessidade da participação do utilizador, para além do atacante, na exploração da vulnerabilidade. Descreve se a vulnerabilidade pode ser explorada apenas pelo atacante, ou se algum utilizador também precisa de "participar" de alguma maneira.

O **User Interaction** tem 2 possíveis valores: *None* e *Required*

#### 3.1.2 Scope

- **Scope (S):**

Uma propriedade capturada no CVSS v3.0 é a capacidade de uma vulnerabilidade num componente *software* ter impacto nos recursos, além dos seus privilégios. Verifica se a componente de vulnerabilidade é a mesma que a componente de impacto.

O **Scope** tem 2 possíveis valores: *Unchanged* e *Changed*.



### 3.1.3 Impact Metrics

- **Confidentiality Impact (C):**

Esta métrica tem em conta o impacto na confidencialidade dos recursos de informação que são administrados pela componente *software* dado a exploração de uma vulnerabilidade. Confidencialidade refere-se a limitar o acesso à informação apenas a utilizadores autorizados, bem como impedir o acesso aos não autorizados.

O **Confidentiality Impact** tem 3 possíveis valores: *None*, *Low* e *High*.

- **Integrity Impact (I):**

A Integridade reflete a veracidade da informação e dos dados. Então, esta métrica refere-se ao impacto da integridade na exploração da vulnerabilidade.

O **Integrity Impact** tem 3 possíveis valores: *None*, *Low* e *High*.

- **Availability Impact (A):**

Por último, esta métrica mede o impacto da disponibilidade da componente resultante da vulnerabilidade explorada. Enquanto que as outras 2 métricas se referem à perda de confidencialidade e de integridade da informação usada pela componente impactada, esta métrica refere-se à falta de disponibilidade da componente em si. *Availability* descreve a disponibilidade dos recursos de informação. Ataques que consomem a banda de *network*, os ciclos de processador ou o espaço em disco, são exemplos de ataques que comprometem a disponibilidade.

O **Availability Impact** tem 3 possíveis valores: *None*, *Low* e *High*.

## 3.2 STRIDE

Nesta secção vou apresentar o conceito de STRIDE, um acrónimo que representa, de modo geral, as possíveis ameaças do nosso sistema. O objetivo desta identificação de ameaças serve para tentar garantir a segurança.

- **Spoofing:**  
O ato de fazer-se passar por alguém ou por algo, que não o próprio. *Impersonating* de um sistema ou pessoa. A propriedade violada é a Autenticação.
- **Tampering:**  
O ato de modificar dados num disco, memória ou numa *network*. A propriedade violada é a Integridade.
- **Repudiation:**  
Rejeita a autoria de algo que aconteceu, violando o Não-Repúdio.
- **Information Disclosure:**  
Divulgação de informação privada a uma entidade não autorizada a aceder a esses dados, quebrando assim a confidencialidade da informação.
- **Denial of Service:**  
Atacar um sistema, de modo a absolver os seus recursos necessários para providenciar um serviço. A disponibilidade do sistema é comprometida.
- **Elevation of Privilege - EoP:**  
Permitir uma entidade a fazer algo que não deveria poder fazer. Compromete a autorização do sistema.

### 3.3 Application Layer

Como descrito em cima, esta camada do sistema descreve as comunicações de todo o sistema, também como a informação geral das aplicações leitor e do utilizador.

Portanto, temos que ter em conta todos os processos de troca de dados entre Aplicação-Aplicação e Aplicação-Entidade.

#### 3.3.1 Comunicação Aplicação-Entidade

Ambas as aplicações comunicam com a entidade emissora (de forma bi-direcional) via rede pública recorrendo a tecnologias suportadas pela arquitetura TCP/IP. Os dados transferidos da entidade emissora para a aplicação do portador estão em formato *JSON*.

O *Three Way Handshake* dentro do TPC assegura que ambas as partes da comunicação confirmam a conexão e que estão preparadas para a transferência de dados.

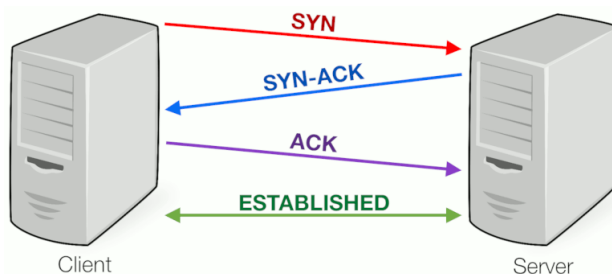


Figura 2: Three Way Handshake

Os *SYN* e os *ACKS* para além de contribuir para o estabelecimento da ligação, também servem de "mensagens" de controlo. Existem possíveis ataques que podem explorar este controlo.

De relembrar que existem várias possíveis explorações dentro dos protocolos que implementam o TCP/IP. Por exemplo, uma sessão *HTTP* é estabelecida através de uma conexão TCP/IP, e dentro do *HTTP* existe vulnerabilidades nas *cookies* do *browser* (*Cookie Poisoning*), acesso a dados na cache, até interceção de pacotes *HTTP*.

- **Spoofing**

- Como descrito no problema, a comunicação entre aplicação e entidade emissora é feita através da comunicação TCP/IP. Muitos dos protocolos dentro do TCP/IP não disponibilizam mecanismos de autenticação, tanto de origem como destino.

Assim, os ataques de *Spoofing* são mais frequentes neste tipo de protocolos.

**Mitigação:** Garantir a autenticação dentro da comunicação.

- **TCP Blind Spoofing:** O atacante poderá conseguir obter a sequência de números e o número da porta da comunicação.

Assim, pode se fazer passar por uma das partes da comunicação. Neste sistema, poderá fazer-se passar por o portador e recebe a informação relativo a este.

**Mitigação:** A proteção contra este tipo de ataques passa pela filtragem dos pacotes, autenticação dos endereços IP e utilização de uma *firewall*.

- O *JSON* não garante a autenticação de ambas as partes da comunicação. Assim, um atacante poderá enviar informação para uma aplicação, fazendo-se passar pela Entidade Emissora.

**Mitigação:** O uso do *JSON Web Token* garante a identificação digital de ambas as partes, isto é, a autenticação é assegurada.

- **Tampering**

- Ataques de *Spoofing* pode desencadear outros ataques, como o ataque *Man in the middle*, que permite o atacante fazer-se passar por outro, acedendo aos seus dados, podendo modifica-los se quiser, comprometendo a sua integridade.

**Mitigação:** A melhor maneira de prevenir ataques do tipo *Man in the middle* é usar um mecanismo de encriptação forte.

- O *JSON* não garante a integridade dos dados, portanto um atacante poderá alterar a informação transferida entre a entidade emissora e a aplicação.

**Mitigação:** O uso do *JSON Web Token* garante a identificação digital de ambas as partes. Como a assinatura é calculada através do *payload* e os *headers*, conseguimos verificar que os dados não foram alterados.

- **Repudiation**

- O *JSON* não garante o não repúdio. Então, um atacante poderá negar um pedido que foi realizado por este.

**Mitigação:** O *JSON Web Signature* garante o não repúdio através de uma assinatura nos dados transferidos.

- **Information Disclosure**

- O *Man in the middle* permite o atacante interferir e colocar-se no "meio" da comunicação entre a aplicação e a entidade emissora. Para além do atacante poder modificar a informação que percorre na comunicação (como descrito no *Tampering* acima), poderá também visualizar essa informação, expondo a possíveis *leaks*.

**Mitigação:** A melhor maneira de prevenir ataques do tipo *Man in the middle* é usar um mecanismo de encriptação forte.

- O protocolo TCP/IP não dispõe um mecanismo de cifragem de informação, portanto um atacante poderá intercetar pacotes através de um *sniffer*, e a informação trocada em *plain-text* estará exposta para o atacante.

**Mitigação:** Cifrar a informação transmitida na comunicação.

- **Denial of Service**

- ***SYN Flooding***: Os *SYN* e os *ACK*, que estão implícitos no *Three Way Handshake*, descrito em cima, servem para manter a comunicação entre servidor-cliente contínua. O *SYN Flooding* usa esta propriedade, onde o atacante envia pacotes *SYN* para o servidor/entidade emissora de um endereço falso. Assim, o servidor enviará continuamente pacotes *SYN+ACK* para o endereço que não existe, comprometendo a disponibilidade a portadores legítimos da aplicação.

**Mitigação:** As técnicas de prevenção mais comuns para este tipo de ataque são: ***Micro blocks*** e ***SYN cookies***. Os administradores podem alocar um micro registo (até 16 *bytes*) na memória do servidor para cada solicitação *SYN* de entrada, em vez de uma conexão completa (*Micro blocks*). O *SYN cookies* consiste no uso da técnica de *hash* criptográfico e do endereço IP do cliente, para a comunicação entre o servidor e o cliente.

- O *Ping Flooding* consiste num atacante enviar pacotes *ICMP* (*Internet Control Message Protocol*) a um servidor continuamente, causando um ataque de negação de serviço (*DoS*). Também afeta diretamente os utilizadores da aplicação, pois ao enviar pacotes *ICMP*, o servidor terá que lhe responder com o mesmo número de pacotes.

**Mitigação:** Configurar a *firewall* para impedir *pings* externos e filtrar os *pings* com intuito malicioso, usando um histórico para não perder a capacidade de diagnosticar problemas no servidor.

- **Elevation of Privileges**

- Os dados transferidos da entidade emissora para o portador da aplicação são em formato *JSON*.

Se o atacante tiver acesso a esses dados em *JSON* (*Tampering*), usando a função *parseLookup* do *JSON*, poderá inserir comandos arbitrários dentro deste, sem permissão.

Este tipo de ataque foi identificado como uma vulnerabilidade.

**Vulnerabilidade:** CVE-2020-7712

**Base Score:** 7.2 HIGH

**Descrição:** A vulnerabilidade em causa afeta as versões dos pacotes *JSON* inferiores à 10.0.0, e consiste na injeção de comandos arbitrários e externos usando uma função implementada dentro do *JSON*.

**Exploração:** O atacante poderia ter acesso a permissões de execução de comandos usando a função *parseLookup* do *JSON*.

CVSS v3.1 Severity and Metrics:  
Base Score: 7.2 HIGH  
Vector: AV:N/AC:L/PR:H/UI:N/S:U/C:H/I:H/A:H  
Impact Score: 5.9  
Exploitability Score: 1.2

---

Attack Vector (AV): Network  
Attack Complexity (AC): Low  
Privileges Required (PR): High  
User Interaction (UI): None  
Scope (S): Unchanged  
Confidentiality (C): High  
Integrity (I): High  
Availability (A): High

Figura 3: CVSS Version 3.x Metrics

**Mitigação:** Uma das técnicas consiste na **parametrização** dos dados. A Parametrização consiste na separação automática de dados e código, facilitando na validação deste.

### 3.3.2 Comunicação Aplicação-Aplicação

Como descrito anteriormente, as 2 aplicações precisam de estabelecer comunicação de uma certa forma, para que o dispositivo leitor verifique os dados do portador (Prova de Identidade). Estas operações de transferência de dados tem 2 modos: *offline* e *online*.

Para a análise de potenciais ameaças temos que ter em conta as tecnologias usadas neste processo. Sendo estas,

- **QR Code**: Usado para o dispositivo leitor "encontrar" o dispositivo do portador.
- **BLE** (*Bluetooth Low Energy*); **NFC** (*Near Field Communication*): Usado para estabelecer comunicação entre os dois dispositivos.
- **CBOR** - *Concise Binary Object Representation*: Toda a comunicação é suportada por mensagens codificadas.

O processo de conexão usando *Bluetooth Low Energy* usa a técnica de geração de chaves entre os dois dispositivos.

O BLE, inicialmente, consiste na troca de **Temporary Keys** (TK), onde é gerada um valor aleatório pequeno (até 6 dígitos), e um dos dispositivos lê, ou insere, a *Temporary Key* do outro.

Após este processo, os 2 dispositivos em causa vão gerar uma **Short Term Key** (STK), que consiste na "soma" da TK mais um valor gerado em cada dispositivo. Depois são comparados ambas as STKs, e se forem correspondentes então as STKs serão usadas para encriptar a conexão.

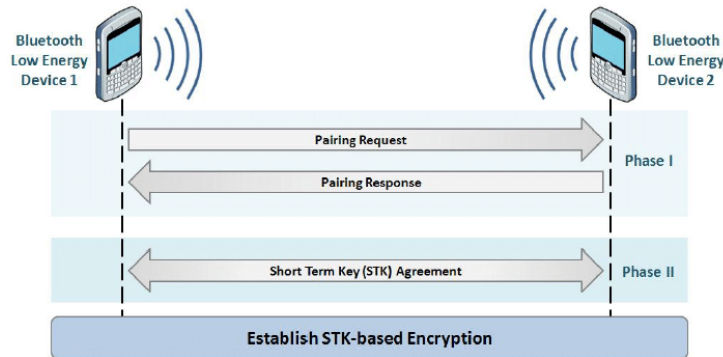


Figura 4: Estabelecimento de comunicação BLE

- **Spoofing**

- Existe *Spoofing* no *Near Field Communication*, quando um atacante falsifica uma *tag* e um utilizador lê essa *tag*, podendo ter conteúdo malicioso.

Porém neste caso, trata-se de 2 dispositivos a tentar comunicar entre si, ou seja, não há essa possibilidade de falsificação de *tags*.

- **Tampering**

- Apesar de ser necessário a proximidade na comunicação *NFC*, o ataque *Man in the middle* é possível. Um atacante poderá intercetar essa informação, podendo até modificá-la. Temos que assegurar que o CBOR codifique as mensagens.

**Mitigação:** A melhor maneira de prevenir ataques do tipo *Man in the middle* é usar um mecanismo de encriptação forte.

- Em modo *offline*, é o portador que envia a informação, tal como os dados necessários para a sua verificação para o verificador. O portador poderá alterar a informação relativa a si próprio na prova de identidade.

**Mitigação:** Como os dados transferidos são em formato *JSON*, podemos usar a mesma técnica em cima para a comunicação Aplicação-Entidade Emissora, o *JSON Web Token*, que garante a identificação digital de ambas as partes. Como a assinatura é calculada através do *payload* e os *headers*, conseguimos verificar que os dados não foram alterados.

- Se houver um atacante com acesso à informação codificada pelo CBOR, poderá explorar uma vulnerabilidade dentro deste mecanismo de codificação.

Existe uma em que é possível alterar dados através de uma inserção no descodificador.

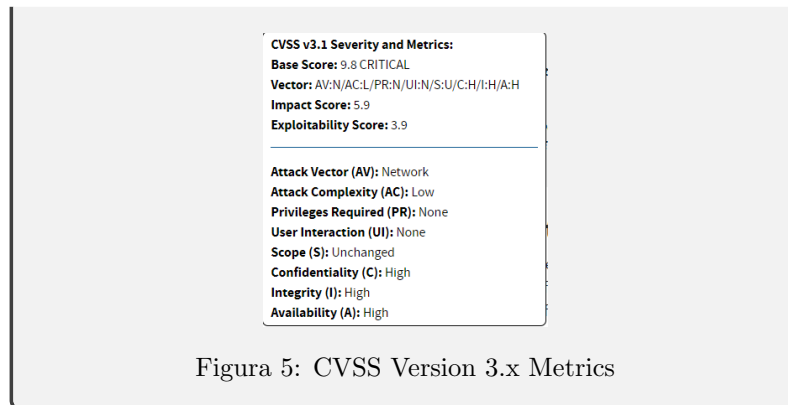
**Vulnerabilidade:** CVE-2020-24753

**Base score:** 9.8 CRITICAL

**Descrição:** Esta vulnerabilidade ocorre sobre a memória, onde há uma corrupção de memória, nas versões anteriores à lançada em Agosto de 2020.

**Exploração:** Um atacante podia executar código através de um *input* de CBOR para o descodificador *cbor2json*. Um erro não detetado ao descodificar *strings* resultava ao uso de um valor na *stack* não inicializado controlado pelo atacante. O *Tampering* está implícito nesta vulnerabilidade, pois é possível modificar a memória, causando falhas potencialmente exploráveis.





**Mitigação:** O mecanismo mais eficaz para este tipo de ataques é a **Validação de *input***. A validação de *input* considera-se tudo "malicioso", e é utilizado uma lista de *inputs* aceitáveis e esperados.

- **Repudiation**

- Para a prova de identidade, o portador autoriza uma lista de atributos para que o dispositivo leitor consulte a entidade emissora. A aplicação leitor poderá alterar esses atributos e negar a sua alteração, se não houver algum mecanismo que a negue.

**Mitigação:** O *JSON Web Signature* garante o não repúdio através de uma assinatura nos dados transferidos.

- A alteração do *QR code* por parte do portador, de modo a que o verificador leia e seja afetado pela injeção de código (ver *Elevation of Privileges*), no entanto, não é possível confirmar que foi o portador que afetou o dispositivo do leitor. Assim, compromete-se o Não Repúdio.

**Mitigação:** Implementar um sistema de verificação de *QR code*, para não permitir este tipo de ataques.

- **Information Disclosure**

- Para além do atacante poder modificar a informação que percorre na comunicação (como descrito no *Tampering* acima), poderá também visualizar essa informação, expondo a possíveis *leaks* (*Man in the middle*).

**Mitigação:** A melhor maneira de prevenir ataques do tipo *Man in the middle* é usar um mecanismo de encriptação forte.

- Existe uma vulnerabilidade dentro do BLE, na criação da chave de encriptação, ou da *Short Term Key*, o comprimento da *Temporary Key* nessa chave é demasiado pequeno, aumentando a possibilidade de um atacante "descobrir" a STK na ligação.

Tendo isto, é possível que um atacante intercete a ligação BLE entre o portador e o verificador, descobrindo a STK. Como em modo *offline*, os dados são transferidos diretamente do portador para o verificador, o atacante poderá ter acesso à informação do portador, comprometendo a sua privacidade.

**Mitigação:** O aumento do tamanho da *Temporary Key* permite uma maior segurança, dificultando a descoberta da *Short Term Key*.

- Dentro do BLE também é possível um ataque do tipo *Man in the middle*. Explora uma vulnerabilidade de como os dispositivos administram os *disconnects/reconnections*. Assim, um atacante poderia interceptar a comunicação entre portador e verificador, simular uma desconexão entre eles, para tentar se conectar a um deles, enquanto ambos os dispositivos tentam a reconexão, e ter acesso à informação que corre dentro da conexão.

**Mitigação:** A melhor maneira de prevenir ataques do tipo *Man in the middle* é usar um mecanismo de encriptação forte.

- **Denial of Service**

- É possível, dentro de uma comunicação NFC, que um atacante comprometa a transmissão de dados entre dispositivos (leitor e portador), através da transmissão de sinais rádio, para que o leitor não consiga decifrar a informação.

**Mitigação:** Verificar sinais RF durante a transmissão.

- Se a comunicação entre portador e verificador for do tipo BLE, é possível que um atacante envie pacotes continuamente, onde o *header* desse pacote inclui o **LAP** (ou *Lower Address Part*, que é *header* de 3 *bytes* enviado em cada pacote) correto do dispositivo que ele queira afetar.

Neste caso, um atacante poderá querer atrasar a transferência de informação entre o portador e o verificador.

**Mitigação:** Não há uma solução concreta para este tipo de ataque, mas penso que uma ideia seria passar pela filtragem ou cifragem de pacotes.

- **Elevation of Privileges**

- Em modo *online*, o dispositivo leitor poderá pedir atributos não autorizados pelo portador, e este não tem privilégios para o fazer.

**Mitigação:** O *JSON Web Signature* garante o não repúdio através de uma assinatura digital. Assim, visto que os dados são dados pelo portador, o verificador não conseguirá alterá-los.

- O portador poderá querer alterar o *QR code*, de modo a que, quando o dispositivo leitor lê-se o código QR alterado. Este *QR code* alterado poderá ser malicioso, contendo código arbitrário que será executado

aquando a leitura do mesmo. Ou seja, é possível que o portador ganhe permissões no dispositivo leitor através de injeção de código arbitrário no *QR code*.

**Mitigação:** Implementar um sistema de verificação de *QR code*, para não permitir este tipo de ataques.

### 3.3.3 Vulnerabilidades nas Aplicações - Android/iOS

Após a análise de risco na parte das comunicações, falta explorar possíveis ameaças nos sistemas operativos que as aplicações usam, sendo estes o sistema *Android* e o sistema *iOS*.

As ameaças mais comuns neste sistemas são as do tipo *Elevation of Privileges*, onde um atacante poderá ter acesso a um dispositivo ou executar código arbitrário, sem a permissão para o fazer.

O **Android** está exposto às aplicações instaladas, isto é, as vulnerabilidades mais exploradas encontram-se nas aplicações instaladas e a sua segurança. Aplicações maliciosas podem comprometer o sistema e o utilizador de todas as maneiras, portanto o cuidado a ter na instalação de aplicações é obrigatório.

O sistema **iOS** protege mais a sua *App store*, de modo a assegurar que este tipo de ataques às aplicações sejam drasticamente reduzidos. No entanto, existem vários ataques explorados através email do *iOS*, onde um atacante podia ganhar controlo total do email. Isto é, um atacante poderia ter acesso remoto a um dispositivo *iOS* através do envio de emails que consumiam uma grande quantidade de memória (causando *Denial of Service* também). Outra vulnerabilidade garantia privilégios de execução de código a atacantes, e estes podiam alterar e expor informação dentro do email (*Tampering* e *Information Disclosure* implícitos).

Isto serve para explicar que apesar de haver *exploits* dentro dos sistemas operativos *iOS* e *Android*, o *User interaction* está implícito na maior parte dos ataques ao sistema.

Abaixo refere-se 1 exemplo de vulnerabilidades para cada sistema apresentado, um para o *Android*, outro para o *iOS*.

- **Elevation of Privileges**

- Em algumas versões mais recentes do Android era possível que um atacante compromete-se o sistema através de uma técnica de nome *tapjacking*. Em baixo, encontra-se a vulnerabilidade explorada.

**Vulnerabilidade:** CVE-2020-0416

**Base Score:** 8.8 HIGH

**Descrição:** Nas definições de vários *Screens*, havia a possibilidade de ataques **tapjacking** através de um valor padrão não protegido. O *tapjacking* consiste na captura dos toques que o utilizador dá sobre o ecrã ao mostrar atividades/aplicações sobrepostas (*overlay* de aplicações). Dessa forma, o utilizador acredita que "toca" sobre a atividade que está a ver, no entanto, os "toques" estão a ser aplicados numa atividade escondida/subjacente.

**Exploração:** Como qualquer ataque *tapjacking*, uma aplicação pode levar à instalação de várias aplicações que tentam "ganhar" permissões. Neste caso, poderia levar a um

aumento local de privilégios e permissões sem privilégios de execução. Esta vulnerabilidade é suportada pelo *User Interaction*, visto que precisamos do desconhecimento da aplicação por parte do utilizador.

<b>CVSS v3.1 Severity and Metrics:</b>	
<b>Base Score:</b>	8.8 HIGH
<b>Vector:</b>	AV:N/AC:L/PR:N/UI:R/S:U/C:H/I:H/A:H
<b>Impact Score:</b>	5.9
<b>Exploitability Score:</b>	2.8
<hr/>	
<b>Attack Vector (AV):</b>	Network
<b>Attack Complexity (AC):</b>	Low
<b>Privileges Required (PR):</b>	None
<b>User Interaction (UI):</b>	Required
<b>Scope (S):</b>	Unchanged
<b>Confidentiality (C):</b>	High
<b>Integrity (I):</b>	High
<b>Availability (A):</b>	High

Figura 6: CVSS Version 3.x Metrics

**Mitigação:** Para utilizadores comuns, apenas se deve esperar que estes não transfiram aplicações não fiáveis, para que este tipo de ataques não aconteça. Para além disso, devem estar atentos aos pedidos de instalação de outras aplicações, ou pedidos de permissão.

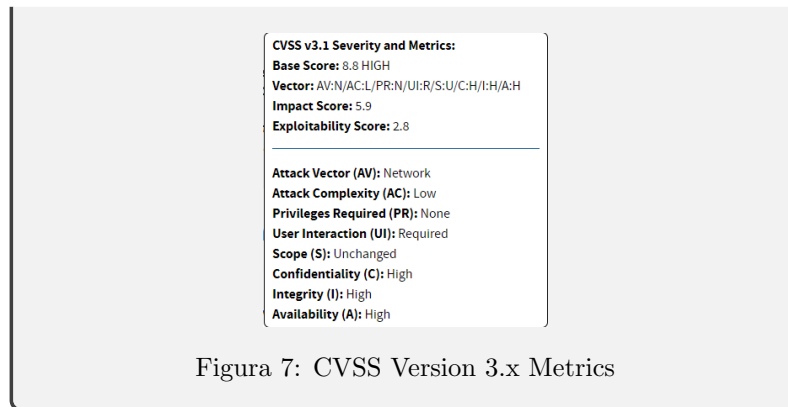
- O processamento de certas páginas *Web* maliciosas podiam resultar na execução de código arbitrário. Em baixo, apresenta-se o CVE e o CVSS correspondentes.

**Vulnerabilidade:** CVE-2020-9932

**Base Score:** 8.8 HIGH

**Descrição:** Havia uma vulnerabilidade de corrupção de memória dentro das versões abaixo da 13.1. Esta vulnerabilidade foi corrigida após o lançamento desta versão.

**Exploração:** Dentro da vulnerabilidade dentro da memória, era possível executar código externo através do processamento de conteúdo *web* malicioso.



**Mitigação:** Executar ou compilar o *software* usando extensões que fornecem automaticamente um mecanismo de proteção que atenua ou elimina *buffer overflows*.

### 3.4 Service Layer

Esta camada, denominada como *Service Layer*, corresponde à entidade emissora e ao seu sistema. Tal como referido na descrição do Sistema, a infraestrutura da entidade emissora ou *Back-end* do Sistema pode estar exposta a vulnerabilidades. Abaixo estão apresentados os detalhes sobre a infra-estrutura.

1. **CentOS 7.8.2003:** Sistema Operativo do servidor Web.
2. **Django v3.0:** Backend principal.
3. **UWSGI:** Servidor Web.
4. **PostgreSQL 12.4:** Base de dados principal.
5. **Ubuntu 20.04:** Sistema operativo do serviço de gestão do sistema.
6. **Flask 1.0:** Backend de gestão.
7. **Gunicorn:** Servidor web.
8. **PostgreSQL 12.1:** Base de dados de gestão (a correr em um container Docker versão 19.03.6).

De notar, que vários serviços se interligam, ou servidores são baseados, ou são derivações de outros servidores, portanto ameaças que surgem num destes podem afetar outros relacionados, indiretamente.

Como já tratamos da comunicação Aplicação-Entidade Emissora, falta-nos apresentar as ameaças e vulnerabilidades no sistema físico.

- **Spoofing**

- Um atacante poderia configurar um servidor falso, fazendo-se passar pela entidade emissora. E todas as ligações Entidade-Aplicação seriam afetadas.

**Mitigação:** Este tipo de comunicação deve ser sempre autenticada. Se quisermos assegurar um maior nível de segurança poderá ser utilizada autenticação multi-fatores, baseada em parâmetros pré-definidos, ou num historial de comunicação ou implementar um sistema de assinaturas digitais.

- Como esta *layer* é constituída por 2 servidores, o principal e o de gestão de dados, um atacante poderá configurar um servidor falso, que se faz passar pelo servidor de gestão de dados, recebendo toda a informação proveniente do servidor principal.

**Mitigação:** Este tipo de comunicação deve ser sempre autenticada. Se quisermos assegurar um maior nível de segurança poderá ser utilizada autenticação multi-fatores, baseada em parâmetros pré-definidos, ou num historial de comunicação ou implementar um sistema de assinaturas digitais.

- ARP Spoofing: *ARP spoofing* é uma técnica pela qual um atacante envia mensagens ARP para uma rede local, com o intuito de associar o endereço MAC do atacante ao endereço IP de um utilizador, como *gateway* padrão, fazendo com que o tráfego destinado ao utilizador seja enviado ao atacante.

Relativamente ao sistema em questão, trata-se de um problema típico do **Docker**. Quando o *Docker* cria um novo *container*, também é estabelecido uma interface virtual *Ethernet*, que se conecta à *bridge*. Como a interface também está ligado à interface *eth0* do *container*, este pode enviar pacotes para a *bridge*. A *bridge* encaminha todos os pacotes que recebe, sem filtragem deles, portanto o *Docker* está exposto a ataques de *ARP spoofing*.

**Mitigação:** As técnicas mais usadas para combater este tipo de ataques são, o **ARP authentication** e **Static ARP mappings**. O *ARP authentication* será alterado para não processar pacotes ARP, a menos que tenha um código criptográfico predefinido. O *Static ARP mappings* é a maneira mais eficaz de evitar estes ataques, configurando as entradas ARP manualmente, e desta forma o atacante não terá forma de falsificar a cache ARP do *host*.

- **Tampering**

- O ataque *SQL injection* permite o atacante interferir com as *queries* feitas à base de dados, conseguindo aceder e até modificar informação lá dentro.

**Mitigação:** Na validação de *input* considera-se tudo "malicioso", e é utilizado uma lista de *inputs* aceitáveis e esperados.



- Um dos problemas do *Docker* passa pela fiabilidade das imagens do *Docker*, que passa pela análise do *Docker daemon* e da integridade dos seus *containers*. Um atacante poderá usar uma forma de corromper imagens *Docker* para comprometer os utilizadores do *Docker*.

**Mitigação:** Garantir a integridade das imagens *Docker*, ou usar imagens fiáveis.

#### • Information Disclosure

- Relativamente à base de dados, o atacante poderá tentar aceder diretamente aos dados que estão dentro da base de dados, através por exemplo, de *SQL injections*.

**Mitigação:** Usar um mecanismo de encriptação de dados suficientemente forte.

- Existe um painel de controlo que vem nos pacotes de instalação do CentOS (qualquer versão deste). Este painel de controlo denomina-se de *CentOS Web Panel (CWP)*, e tem como objetivo facilitar a administração de servidores virtuais.

No último ano de 2020, foram encontradas várias vulnerabilidades relativos ao CWP, onde havia a possibilidade de *leaks* de informação. Apesar de ser uma vulnerabilidade relatada a um pacote de instalação, a segurança do sistema poderá ser comprometida no futuro com ataques semelhantes a estes.

**Vulnerabilidade:** CVE-2020-15627

**Base Score:** 7.5 HIGH

**Descrição:** No CWP existe uma vulnerabilidade dentro do *ajax\_mail\_autoreply.php*, onde não há um processo de validação quando um parâmetro é transformado/analísado (*parsed*).

**Exploração:** Sendo que não há qualquer validação ou autenticação, um atacante explora esta vulnerabilidade levando à exposição de informação privada.

<b>CVSS v3.1 Severity and Metrics:</b>	
<b>Base Score:</b> 7.5 HIGH	
<b>Vector:</b> AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N	
<b>Impact Score:</b> 3.6	
<b>Exploitability Score:</b> 3.9	
<hr/>	
<b>Attack Vector (AV):</b> Network	
<b>Attack Complexity (AC):</b> Low	
<b>Privileges Required (PR):</b> None	
<b>User Interaction (UI):</b> None	
<b>Scope (S):</b> Unchanged	
<b>Confidentiality (C):</b> High	
<b>Integrity (I):</b> None	
<b>Availability (A):</b> None	

Figura 8: CVSS Version 3.x Metrics

**Mitigação:** Este tipo de vulnerabilidades estão relacionados com ataques do tipo *SQL injections*. As estratégias mais recorrentes são as de **parametrização** de dados e a **validação de *input***. A Parametrização consiste na separação automática de dados e código, facilitando na validação deste. Na validação de *input* considera-se tudo "malicioso", e é utilizado uma lista de *inputs* aceitáveis e esperados.

- O Django v3.0 representa o *Back-end* principal do sistema. Dentro das vulnerabilidades encontradas das últimas versões do Django, podemos destacar as de *Information Disclosure*.

Portanto, um atacante poderá atacar o *Back-end* do sistema de modo a expor informações relativas aos utilizadores da aplicação/sistema.

Em baixo, encontra-se uma vulnerabilidade encontrada relativa a este tipo de ataques de *Information Disclosure*.

**Vulnerabilidade:** CVE-2020-7471

**Base Score:** 9.8 HIGH

**Descrição:** Em algumas versões do Django v3.0, havia a possibilidade da ocorrência de uma *SQL injection*, se houvesse dados não autenticados (sem confiança) usados como um delimitador *StringAgg*, como por exemplo em aplicações de Django que oferecem *downloads* de informação com um delimitador de coluna especificado pelo utilizador.

**Exploração:** Uma *SQL injection* é uma técnica de injeção de código que pode destruir a base de dados do sistema. Neste caso, ao passar um valor delimitador específico para a instância *StringAgg*, podia levar a uma injeção SQL maliciosa.

CVSS v3.1 Severity and Metrics:	
Base Score:	9.8 CRITICAL
Vector:	AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H
Impact Score:	5.9
Exploitability Score:	3.9
<hr/>	
Attack Vector (AV):	Network
Attack Complexity (AC):	Low
Privileges Required (PR):	None
User Interaction (UI):	None
Scope (S):	Unchanged
Confidentiality (C):	High
Integrity (I):	High
Availability (A):	High

Figura 9: CVSS Version 3.x Metrics

**Mitigação:** As maneiras mais eficazes de prevenir este tipo de ataques é a **validação de *input***, a implementação de uma ***firewall*** e a **parametrização** dos dados.

- Um atacante poderá atacar o Servidor WEB do sistema, de modo

a explorar os dados que correm no servidor, levando aos *leaks* de informação.

uWSGI é um servidor de container de aplicação desenvolvido em C, rápido e auto-ajustável a desenvolvedores e administrados de um sistema. O uWSGI visa desenvolver uma *stack* completa para a construção de serviços de hospedagem (*hosting services*).

**Vulnerabilidade:** CVE-2018-7490

**Base Score:** 7.5 HIGH

**Descrição:** As versões de uWSGI abaixo da 2.0.17 estão afetadas por esta vulnerabilidade, que consiste na má administração/manipulação de verificação do *DOCUMENT\_ROOT* durante o uso da opção *-php-docroot*, permitindo um ataque *Directory Traversal*.

**Exploração:** Um ataque deste tipo foca-se em aceder a ficheiros e diretorias que estão armazenados fora da diretoria pretendida. Manipulando certos ficheiros (do tipo *../*), há a possibilidade de acesso a ficheiros arbitrários e diretorias armazenados dentro do sistema de ficheiros, incluindo código fonte, configurações e outros ficheiros críticos. Permite ao atacante ganhar acesso a informação sobre a estrutura de dados, permitindo também que o atacante consiga ler ficheiros com conteúdo privado (*Information Disclosure*).

<b>CVSS v3.0 Severity and Metrics:</b>	
<b>Base Score:</b>	7.5 HIGH
<b>Vector:</b>	AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N
<b>Impact Score:</b>	3.6
<b>Exploitability Score:</b>	3.9
<hr/>	
<b>Attack Vector (AV):</b>	Network
<b>Attack Complexity (AC):</b>	Low
<b>Privileges Required (PR):</b>	None
<b>User Interaction (UI):</b>	None
<b>Scope (S):</b>	Unchanged
<b>Confidentiality (C):</b>	High
<b>Integrity (I):</b>	None
<b>Availability (A):</b>	None

Figura 10: CVSS Version 3.x Metrics

Relativamente a esta vulnerabilidade **CVE-2018-7490**, foi publicado um *exploit* na *EXPLOIT DATABASE*.




2018-03-02    uWSGI < 2.0.17 - Directory Traversal WebApps PHP Marios Nicolaidis

Figura 11: Exploit encontrada

**Mitigação:** O uso de uma *firewall* é a maneira mais eficaz de controlar este tipo de ataques.

- **Repudiation**

- Os ficheiros *logs* permitem reconhecer quem acedeu e alterou tudo dentro da Base de Dados, e o atacante poderá alterar/remover esses ficheiros *logs*, permitindo o repúdio das suas ações.

**Mitigação:** Utilizar um sistema de controlo de integridade seguro, ou um sistema de autenticação e guardar todas as operações efetuadas num ficheiro de *log* seguro.

- **Denial of Service**

- O *Back-end* de um sistema normalmente está alojada num servidor de rede. Se o atacante conhecer essa rede, poderá sobrecarregar e torná-la indisponível e inacessível, por se encontrar saturada de *requests* ao sistema.

**Mitigação:** Uma solução fiável seria usar uma *firewall* que reconheça e filtre pedidos possivelmente nocivos para o sistema.

- O atacante poderá comprometer a disponibilidade dos servidores web do sistema, através de pedidos *HTTP*, em que o servidor envia cabeçalhos HTTP derivado da função *process.headers*.

Em baixo encontra-se uma vulnerabilidade explorada no *Gunicorn*, um servidor web utilizado pelo sistema em questão.

**Vulnerabilidade:** CVE-2018-1000164

**Base Score:** 7.5 HIGH

**Descrição:** Algumas versões do Gunicorn contém um pacote de sequência vulnerável à divisão de resposta HTTP na função *process.headers*.

**Exploração:** Um atacante pode fazer com que o servidor *Gunicorn* afetado, retorne cabeçalhos HTTP arbitrários.

<b>CVSS v3.0 Severity and Metrics:</b>	
<b>Base Score:</b> 7.5 HIGH	
<b>Vector:</b> AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:H/A:N	
<b>Impact Score:</b> 3.6	
<b>Exploitability Score:</b> 3.9	
<hr/>	
<b>Attack Vector (AV):</b> Network	
<b>Attack Complexity (AC):</b> Low	
<b>Privileges Required (PR):</b> None	
<b>User Interaction (UI):</b> None	
<b>Scope (S):</b> Unchanged	
<b>Confidentiality (C):</b> None	
<b>Integrity (I):</b> High	
<b>Availability (A):</b> None	

Figura 12: CVSS Version 3.x Metrics

**Mitigação:** Uso de *firewall* ou outro mecanismo para a filtragem de pacotes.

- É muito comum também o atacante tentar comprometer a disponibilidade da Base de Dados, podendo levar à impossibilidade de acesso.

**Mitigação:** De forma a precaver que o sistema sofra um ataque deste tipo, deveremos ter os dados guardados em diferentes bases de dados (**Redundância de Dados**).

- É possível que um atacante explore vulnerabilidades dentro dos pacotes ou módulos de instalação do sistema operativo. Os ataques mais comuns ao sistema operativo CentOS 7.8.2003 é recorrer a esses pacotes para comprometer a disponibilidade do Sistema.

Em baixo, apresenta-se uma vulnerabilidade explorada neste sistema operativo, relativa a um pacote de instalação, de nome **BIND** (*Berkeley Internet Name Domain*). Neste caso, a vulnerabilidade foi explorada pela *Red Hat*, uma empresa que desenvolve sistemas operativos baseados no Linux, e também soluções de *software*. *Red Hat Errata* ajudam os utilizadores do sistema a escolher e a perceber os *updates* disponíveis. Estas atualizações podem resolver problemas de segurança, corrigir *bugs* ou fornecer novos recursos para pacotes individuais.

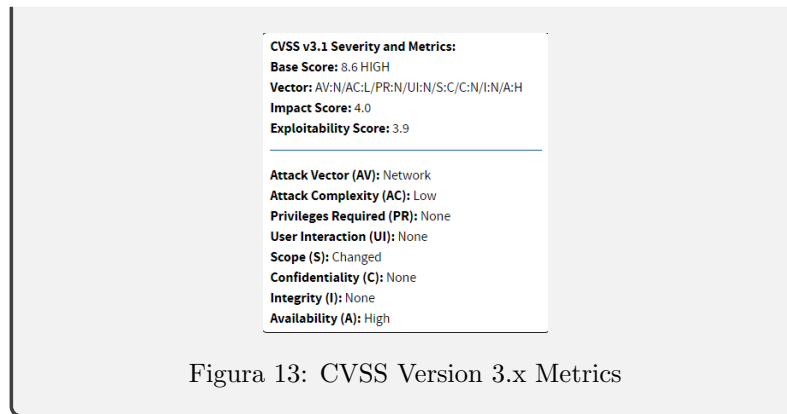
O BIND é o servidor para o protocolo **DNS** mais utilizado dentro da Internet, especialmente em sistemas do tipo **Unix**. O BIND inclui um servidor DNS, uma biblioteca de resolução e ferramentas para verificar se o servidor DNS está a funcionar corretamente.

**Vulnerabilidade:** RHSA-2020:2344 : CVE-2020-8616

**Base Score:** 8.6 HIGH

**Descrição:** O BIND não limita o número de procuras realizadas durante o processamento de referências. Um erro lógico no código que verifica a validade do TSIG pode ser usado para acionar uma falha de declaração. (CVE-2020-8616)

**Exploração:** Um atacante que explora intencionalmente esta falta de limitação no número de procuras durante o processamento de referências pode, por meio do uso de referências especialmente criadas, fazer com que um servidor recursivo emita um grande número de procuras na tentativa de processar a referência, comprometendo a disponibilidade e performance do servidor.



**Mitigação:** Uso de *firewall* ou outro mecanismo para a filtragem de pacotes.

- Sendo o *flask* uma *framework*, a maioria dos ataques são relativos às bibliotecas que este implementa, incluindo, por exemplo, o *JSON*. Os ataques mais esperados dentro do *flask* em si, são do tipo de *Denial of Service*.

**Mitigação:** O *framework flask* disponibiliza uma série de atualizações, que permitem corrigir este tipo de vulnerabilidades.

#### • Elevation of Privileges

- Um atacante poderá executar código arbitrário, sem permissão, através da exploração de pacotes de instalação do sistemas operativos, tanto no CentOS 7.8.2003 (Sistema operativo do servidor principal) como no Ubuntu 20.04 (Sistema operativo do serviço de gestão do sistema). Segue-se em baixo, uma vulnerabilidade relativa ao pacote de instalação do *curl* no CentOS 7.8.2003 e uma vulnerabilidade relativa ao pacote de instalação *libvirt* no Ubuntu 20.04.

Os pacotes do *curl* disponibilizam *librarys* e ferramentas para descarregar ficheiros de servidores, podendo usar vários protocolos como o HTTP, FTP e LDAP.

**Vulnerabilidade:** RHSA-2020:3916 : CVE-2019-5482

**Base Score:** 9.8 CRITICAL

**Descrição:** Num dos controladores de protocolos do curl, o controlador do protocolo **TFTP**, ocorria um *buffer overflow* na *Heap* da memória.

**Exploração:** Um *buffer overflow* pode esultar na sobreposição de dados/funções em memória, e possivelmente, execução de código sem permissão.

**CVSS v3.1 Severity and Metrics:**  
**Base Score:** 9.8 CRITICAL  
**Vector:** AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H  
**Impact Score:** 5.9  
**Exploitability Score:** 3.9

---

**Attack Vector (AV):** Network  
**Attack Complexity (AC):** Low  
**Privileges Required (PR):** None  
**User Interaction (UI):** None  
**Scope (S):** Unchanged  
**Confidentiality (C):** High  
**Integrity (I):** High  
**Availability (A):** High

Figura 14: CVSS Version 3.x Metrics

**Vulnerabilidade:** CVE-2020-15708

**Base Score:** 9.3 CRITICAL

**Descrição:** O pacote *libvirt* do Ubuntu 20.04 criava um *socket* de controlo, com permissões globais de leitura e escrita.

**Exploração:** Um atacante podia usar este *socket* para escrever sobre ficheiros arbitrários (*overwrite*), ou para executar código externo.

**CVSS v3.1 Severity and Metrics:**  
**Base Score:** 9.3 CRITICAL  
**Vector:** AV:L/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H  
**Impact Score:** 6.0  
**Exploitability Score:** 2.5

---

**Attack Vector (AV):** Local  
**Attack Complexity (AC):** Low  
**Privileges Required (PR):** None  
**User Interaction (UI):** None  
**Scope (S):** Changed  
**Confidentiality (C):** High  
**Integrity (I):** High  
**Availability (A):** High

Figura 15: CVSS Version 3.x Metrics

**Mitigação:** Definir permissões apropriadas durante a instalação dos pacotes/programas. Isso impedirá a passagem de permissões não seguras para qualquer utilizador que instale o pacote. Garantir que todos os executáveis, ficheiros de configuração e bibliotecas só possam ser lidos e escritos pelo administrador do *software*.

- Os ataques à base de dados podem ser feitos através da exploração de extensões de instalação.

Os ataques mais comuns ao **PostgreSQL**, são relativos ao *search\_path*. O *search\_path* permite controlar os *Schemas* que são procurados. Um atacante poderá utilizar o *search\_path* para comprometer o sistema de várias formas (principalmente através de acesso privilegiado de execução de código), daí a segurança deste ser obrigatório.

Em baixo, encontra-se uma vulnerabilidade explorada anteriormente à versão 12.4 do PostgreSQL, relativo ao *Elevation of Privileges*.

**Vulnerabilidade:** CVE-2020-14350

**Base Score:** 7.3 HIGH

**Descrição:** Foi descoberto que algumas extensões do PostgreSQL não usam *search\_path* com segurança em seu *script* de instalação. Afeta todas as versões abaixo da 12.4 do PostgreSQL, incluindo a 12.1.

**Exploração:** Um invasor com privilégios suficientes, pode usar esta falha para induzir um administrador a executar um *script*, especialmente criado durante a instalação ou atualização de tal extensão.

CVSS v3.1 Severity and Metrics:  
Base Score: 7.3 HIGH  
Vector: AV:L/AC:L/PR:L/UI:R/S:U/C:H/I:H/A:H  
Impact Score: 5.9  
Exploitability Score: 1.3

---

Attack Vector (AV): Local  
Attack Complexity (AC): Low  
Privileges Required (PR): Low  
User Interaction (UI): Required  
Scope (S): Unchanged  
Confidentiality (C): High  
Integrity (I): High  
Availability (A): High

Figura 16: CVSS Version 3.x Metrics

**Mitigação:** A versão 12.4 do PostgreSQL resolveu todas as vulnerabilidades encontradas relativamente ao *search\_path*, no entanto deve-se ter sempre em conta a sua segurança e possíveis falhas.



## 4 Observações finais

Esta secção final divide-se em 2 subsecções. A primeira, **Resumo e Análise de Risco**, dedica-se ao resumo da análise de ameaças, indicando as ameaças mais significantes para cada subsecção explorada. Este tipo de análise final permite aos desenvolvedores do sistema definirem prioridades de segurança, definindo dentro de as ameaças possíveis, quais podem afetar ou comprometer significativamente o sistema. Na segunda, **Possíveis melhorias**, apresento possíveis implementações dentro do sistema, principalmente, na parte da Entidade Emissora.

### 4.1 Resumo e Análise de Risco

Threat	Aplicação-Entidade	Aplicação-Aplicação	Service Layer
<i>Spoofing</i>	<b>X</b>		<b>X</b>
<i>Tampering</i>	<b>X</b>	<b>X</b>	<b>X</b>
<i>Repudiation</i>	<b>X</b>	<b>X</b>	<b>X</b>
<i>Information Disclosure</i>	<b>X</b>	<b>X</b>	<b>X</b>
<i>Denial of Service</i>	<b>X</b>	<b>X</b>	<b>X</b>
<i>Elevation of Privileges</i>	<b>X</b>	<b>X</b>	<b>X</b>

Em cima apresenta-se uma tabela que nos indica as ameaças mais significativas, assinaladas a **vermelho**, dentro de todos as ameaças, para cada camada do sistema. Como podemos ver, a tabela tem 3 colunas associadas a cada ameaça do modelo **STRIDE**, sendo estas:

- **Aplicação-Entidade:** Refere-se à comunicação Aplicação-Entidade, estudada em cima, dentro da *Application Layer*.
- **Aplicação-Aplicação:** Refere-se à comunicação Aplicação-Aplicação, estudada em cima, dentro da *Application Layer*.
- **Service Layer:** Refere-se à camada da infra-estrutura, estudada na subsecção *Service Layer*.

De notar que nesta tabela não está referido a subsecção **Vulnerabilidades nas Aplicações**, que estuda os possíveis ataques e vulnerabilidades dentro do sistema *iOS* e *Android*, pois, como os ataques mais comuns são provenientes de fatores externos ao sistema, como aplicações maliciosas ou *spam* de email, o sistema não tem controlo sobre essas possíveis ameaças.

Para a comunicação **Aplicação-Entidade**, considerei o *Spoofing* e a *Information Disclosure* as ameaças mais significantes. *Spoofing* porque, como esta comunicação é suportada pelo TCP/IP, estamos sujeitos a vários ataques dado a falta de autenticação deste protocolo. *Information Disclosure* porque, como nesta comunicação está implícita a troca de dados entre a aplicação e

a entidade emissora, e estes dados são relativos ao portador e são privados, é essencial a privacidade desta informação.

A comunicação **Aplicação-Aplicação** indica a troca de dados entre o portador e o verificador, numa situação de prova de identidade. Estes dados transferidos são remetentes ao portador, e estes devem ser privados e protegidos de forma a que nenhum atacante tenha acesso a eles, daí a relevância da ameaça **Information Disclosure**. Para além disso, a integridade dos dados deve ser mantida, isto é, tem que ser garantido que os dados não são alterados, pois pode afetar a prova de identidade. De referir também que o verificador não pode alterar os dados autorizados pelo portador, assim o sistema deve-se focar em ameaças do tipo **Tampering**.

Relativamente à Entidade Emissora, as partes mais significativas a garantir é a disponibilidade de serviço, a segurança dos dados e de privilégios. Os ataques de **Denial of Service** são muito comuns, de modo a comprometer a disponibilidade do sistema, portanto é importante manter a entidade emissora disponível para que não haja *delays* ou até indisponibilidade total no acesso por parte das aplicações. A informação, que é guardada na base de dados da entidade, tem que ser preservada e protegida, por se tratar de dados privados dos portadores, assim o sistema deve garantir a máxima prevenção a ataques de **Information Disclosure**. Sendo a entidade emissora a "peça" fundamental de todo o sistema, que serve de suporte para o resto, os privilégios de acesso e de execução devem ser restritos a administradores, portanto o sistema deve ter em conta ataques de **Elevation of Privileges**.

Para concluir esta secção, sinto que devo acabar com uma opinião de carácter pessoal. Penso que, este tipo de análise de ameaças deve ser contínuo, isto é, apesar de ser imprescindível na fase de implementação, este processo deve ser repetido para tentar estar "1 passo à frente" dos atacantes. Claro que este é impossível, visto que existem atacantes a tentar explorar qualquer tipo de vulnerabilidade e, posteriormente, atacar o sistema, no entanto, este processo de estudo contínuo aumenta o controlo e prevenção aos ataques que surgem.

## 4.2 Possíveis melhorias

Como dito anteriormente, esta secção será apresentado possíveis implementações de modo a aumentar o nível de segurança do sistema, especificamente, na infraestrutura da entidade emissora.

Entre as várias possíveis implementações, destaquei 3, que considero ser as mais importantes, sendo estas:

- Implementação de um servidor *Proxy*
- Implementação de uma *Firewall*
- Implementar uma **Criptografia de Chave Pública** na comunicação Aplicação-Entidade.

Tipicamente o uso de um **servidor *Proxy*** está associado à parte do cliente, isto é, um cliente conecta-se ao *proxy* para solicitar recursos de outros servidores, normalmente dentro da *Web* e assim, conseguem manter o anonimato. Neste caso, a implementação de um servidor *proxy* é do lado da entidade emissora, funcionando como um ***reverse proxy***. Assim, quando um portador quer fazer a transferência dos seus dados, o seu *request* é feito ao *proxy* e não ao servidor principal. Esta implementação garante mais uma camada de segurança nos pedidos e também permite alguma filtragem dos pedidos ao servidor.

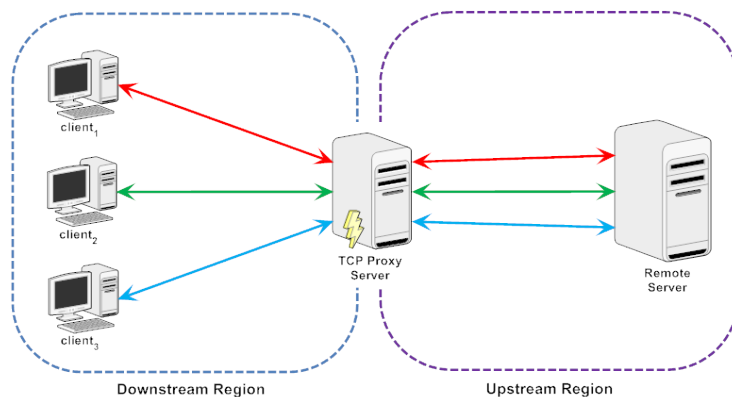


Figura 17: *TCP Reverse Proxy*

O conceito de ***Firewall*** foi referido várias vezes na mitigação de possíveis ameaças. A *firewall* tem como objectivo aplicar uma política de segurança a um determinado ponto da rede, geralmente associados a redes TCP/IP, permitindo uma melhor filtragem de *request* e pacotes. Apesar do *reverse proxy* permitir alguma filtragem de pedidos à entidade emissora, a *firewall* é muito mais eficaz neste processo. Portanto, a implementação de uma *firewall* nas comunicações à entidade emissora, permite reduzir drasticamente os ataques de *Denial of Service*, *Tampering* e *Information Disclosure*.



Figura 18: *Firewall*

A política de **Criptografia de Chave Pública**, também conhecida como criptografia assimétrica, é uma criptografia que usa um par de chaves: Uma **chave pública**, que é conhecida por todos, e uma **chave privada**, apenas conhecida pelo proprietário. Esta técnica permite a confidencialidade dos dados, pois só consegue decifrar quem possui a chave privada associada à chave pública usada para cifrar. Tipicamente nesta técnica também são implementados mecanismos de autenticação, como a **Assinatura digital** e mecanismo de integridade, como o uso de uma **hash criptográfica**. Assim, é possível, dentro da comunicação Aplicação-Entidade, garantir a autenticação de ambas as partes, a integridade dos dados e a confidencialidade dos dados. A entidade emissora poderia funcionar como uma **Autoridade de Certificação (CA)** que associa a chave pública a uma determinada entidade, isto é, a um utilizador da aplicação.

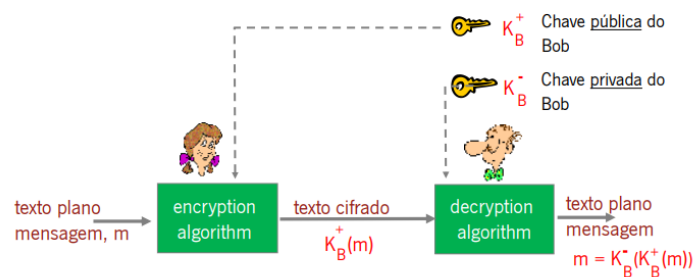


Figura 19: Criptografia de Chave Pública

## 5 Conclusão

Depois de terminado este primeiro trabalho prático da Unidade Curricular de Segurança de Sistemas Informáticos, concluí que a realização deste permitiu consolidar os conhecimentos estudados nas aulas da disciplina. Deu para perceber que a análise de segurança é fundamental no desenvolvimento de um sistema. A modelação do sistema, com o recurso a um *Threat Model*, ajuda na identificação de possíveis ameaças, pois conseguimos analisar, separadamente, cada camada do sistema.

Em suma, apesar das dificuldades encontradas ao longo do trabalho, a realização deste ajudou bastante na compreensão destes conceitos envolvidos na Segurança.