Monitorização do Espectro WiFi

Mestrado em Cibersegurança e Informática Forense

Joel Francisco Figueiredo

Leiria, setembro de 2022

Monitorização do Espectro WiFi

Mestrado em Cibersegurança e Informática Forense

Joel Francisco Figueiredo

Trabalho de Projeto realizado sob orientação do Professor Doutor Carlos Antunes

Leiria, setembro de 2022

Originalidade e Direitos de Autor

O presente relatório de projeto é original, elaborado unicamente para este fim, tendo sido devidamente citados todos os autores cujos estudos e publicações contribuíram para o elaborar.

Reproduções parciais deste documento serão autorizadas na condição de que seja mencionado o Autor e feita referência ao ciclo de estudos no âmbito do qual o mesmo foi realizado, a saber, Curso de Mestrado em Cibersegurança e Informática Forense, no ano letivo 2021/2022, da Escola Superior de Tecnologia e Gestão do Politécnico de Leiria, Portugal, e, bem assim, à data das provas públicas que visaram a avaliação destes trabalhos.

Dedicatória

À minha filha Laura Figueiredo por eu não ter estado presente todo o tempo que ela merecia.

À minha esposa Elisabete Pereira, por todo o apoio incondicional que me prestou ao longo deste curso. Sem ela, não teria sido possível realizá-lo.

Agradecimentos

Gostaria de agradecer a todos os que me acompanharam durantes este percurso académico.

Ao meu orientador, Professor Especialista Carlos Antunes, muito obrigado pelo apoio, disponibilidade e compreensão demonstrada ao longo do projeto.

Aos meus amigos e colegas de curso um muitíssimo obrigado por toda a ajuda e apoio que sempre me disponibilizaram.

Por último, um agradecimento àqueles que me permitiram chegar onde cheguei, obrigado aos meus familiares pelo seu apoio.

Resumo

As redes WiFi são uma tecnologia cada vez mais usada, a nível empresarial e doméstico. É normal uma organização ter mais do que uma rede WiFi no seu interior, no entanto, estas redes têm várias problemáticas associadas.

Apesar de já terem sido feitas várias atualizações nos protocolos de rede WiFi, existem algumas problemáticas que ainda não foram resolvidas, bem como muitos dispositivos de rede antigos que não utilizam os protocolos mais recentes. A rede WiFi é então suscetível a vários tipos de ataques que, sem o devido software de análise, são muito difíceis de identificar. É na sequência desta necessidade que foi desenvolvido este projeto, para que seja possível captar e identificar potenciais ataques à rede WiFi e mapear todos os dispositivos WiFi, sejam eles novos routers ou novos dispositivos clientes, mesmo sem que estes estejam ligados a uma rede WiFi.

Neste trabalho foi proposta uma abordagem e desenvolvido um software recorrendo à plataforma "Kismet", que permite efetuar uma captura de diversos pacotes gerados durante as comunicações WiFi numa determinada área, sendo que o "Kismet" permite a execução de capturas remotas (não tendo o servidor de se encontrar no mesmo espaço físico que o recetor) e ainda vários pontos de captura em simultâneo.

A plataforma desenvolvida encontre-se divida em 3 módulos, onde se encontra a captura, tratamento e análise dos dados, e ainda uma componente de alarmística configurável que permite receber alarmes, despoletados por anomalias detetadas na rede, que poderão ser consultadas através da *interface web* ou rececionadas por SMS ou e-mail.

Palavras-chave: WiFi, Kismet, Segurança de redes, Rougue AP, De-Authentication, Captura de rede.

Abstract

WiFi networks are an increasingly used technology at the business and home envoriement. It's normal for an organization to have more than one WiFi network inside, however, these networks have several associated problems.

Although several updates have been made to the WiFi network protocols, some issues have not yet been resolved, as well as many older network devices that do not use the latest protocols. The WiFi network is then susceptible to various types of attacks which, without the proper analysis software, are very difficult to identify. It's because of that necessity for security that this project was developed, to give the possibility to capture and identify potential attacks on the WiFi network and map all WiFi devices, whether they are new routers or new client devices, even if they are not connected to a WiFi network.

In this project, it was proposed an approach and developed a software using the platform “Kismet” that allows the capture of several packets in the WiFi communications, especially using a "Kismet" feature that allows the execution of remote captures (not having the server to be in the same physical space as the receiver) and even multiple captures simultaneously.

The developed platform is divided into 3 modules, the capture of data, processing and analysis of that data, and also a configurable alarming component that allows users to receiving alarms, triggered by anomalies detected in the network, which can be consulted through the interface web or received by SMS or email.

Keywords: WiFi, Kismet, Network Security, Rougue AP, De-Authentication, Network Capture.

Índice

Lista de Figuras

Lista de Tabelas

Lista de siglas e acrónimos

ESTG	Escola Superior de Tecnologia e Gestão
AP	Acess Point
API	Application Programming Interface
BSSID	Basic Service Set Identifier
BYOD	Bring your Own Device
CVE	Common Vulnerabilities and Exposures
CSV	Comma-Separated Values
DB	Database
GPS	Global Positioning System
GUI	Graphical User Interface
JPA	Java Persistence API
JSON	JavaScript Object Notation
LLC	Link Layer Protocol
MAC	MAC address
ORM	Object-relational mapping
PDF	Portable Document Format
PNG	Portable Network Graphics
SMS	Short Message Service
SO	Operating System
VPN	Virtual Private Network
WEP	Wired Equivalent Privacy
WPA	WiFi Protected Access
WPA2	WiFi Protected Access 2
WPS	WiFi Protected Setup

1. Introdução

Com a revolução tecnológica da última década, as empresas têm aderido cada vez mais à utilização de redes WiFi nas suas instalações, quer sejam redes para convidados ou para uso da própria organização. No entanto, este tipo de redes sem fios pode trazer alguns problemas a nível de segurança, problemas esses não encontrados nas redes com ligação por cabo onde, por exemplo, é possível delimitar na perfeição a área de acesso. Esse controlo é difícil de replicar nas redes WiFi, o que poderá causar potenciais problemas.

Existem inúmeras organizações em que a política de BYOD (*Bring Your Own Device*) se apresenta definida com fracos conceitos de segurança, o que faz com que cada pessoa possa ter vários dispositivos ligados à rede, ou mesmo que alguém com um smartphone crie um AP *(Acess Point)* para fins não fidedignos. Todas estas problemáticas são, no geral, bem conhecidas, no entanto, as organizações carecem de um software que faça uma monitorização de redes WiFi e disponibilize uma componente de alarmística que lhes permita ter alguma capacidade de vigilância e controlo de segurança relativamente às redes WiFi utilizadas.

Atualmente, é possível, estando fora das instalações da organização, na rua ou até mesmo noutro edifício, ter acesso à rede WiFi de uma organização, utilizando hardware específico de longo alcance, e posteriormente utilizar esses acessos para desenvolver atividades ilegais.

Para além desta problemática, existe também a questão dos protocolos e técnicas utilizadas para acesso a redes WiFi que, na maior parte das empresas, é descurada por falta de conhecimento, investimento ou simplesmente por comodidade.

Foi com base em todas estas problemáticas das redes WiFi que o presente projeto foi desenvolvido.

1.1. Objetivos

O presente projeto tem como principal objetivo colmatar a lacuna que existe na monitorização de utilização das redes WiFi, pretendendo-se desenvolver uma solução que disponibilize as seguintes capacidades:

- Recolha local e remota de dados de redes WiFi, podendo fazer a recolha em vários pontos simultaneamente de forma remota;
- Filtragem e armazenamento de dados através do desenvolvimento de uma aplicação entre a captura de dados e parte de alarmística que executa uma filtragem e, posteriormente, o armazenamento de dados;
- Análise de dados e componente de alarmística, implementando uma aplicação que consiga ler e analisar todos os dados, podendo gerar alarmes a partir dos mesmos;
- Desenvolvimento de uma plataforma de gestão que permita a gestão do próprio software.

1.2. Planeamento

Na Tabela 1 - Cronograma das tarefas macro deste projeto, são apresentadas as tarefas macro realizadas neste projeto.

Em setembro 2021 foi iniciado o planeamento do projeto, definidos os objetivos e delineadas as suas primeiras etapas.

Durante os meses de setembro e outubro foi efetuada uma investigação de projetos na área com o objetivo de perceber a viabilidade do mesmo. Após esta investigação, foi efetuado um estudo aprofundado do comportamento das redes WiFi e os seus tipos de comunicações (*frames)* associados, durante a parte final do mês de novembro e no mês de dezembro.

Após a investigação da tecnologia WiFi e dos softwares da área deste projeto, foi necessário aprofundar o conhecimento sobre o software Kismet, que serviu de base para todo o projeto, pelo que foram utilizados os meses de dezembro e janeiro para perceber o seu funcionamento, bem como a sua API (Application Programming Interface).

Depois de conhecer o comportamento do Kismet e a sua API, foi efetuado um estudo sobre as várias métricas, por forma a perceber quais seriam as mais adequadas para este processo.

A arquitetura do projeto foi desenvolvida no final de janeiro e, no mês seguinte, foi criada a base de dados, enquanto foi inicializado o software em Java (back-end) e Vue (front-end), processo de desenvolvimento que durou até maio. Por último, entre junho e julho, foram efetuados múltiplos testes sobre o software desenvolvido.

O presente relatório foi redigido entre novembro de 2021 e julho do ano seguinte.

Tabela 1 - Cronograma das tarefas macro deste projeto

Tarefas	2021				2022						
	9	10	11	12	1	2	3	4	5	6	7
Idealização do conceito											
Recolha de projetos na área e estudo de viabilidade											
Estudo do tráfego WiFi e os seus tipos de *frames*											
Testes das potencialidades do Kismet e a sua API											
Análises das métricas a serem utilizadas											
Desenvolvimento da arquitetura do projeto											
Criação da base de dados e campos necessários											
Desenvolvimento da aplicação em Java (*Back-end*)											
Desenvolvimento da aplicação em VUE *(Front-end*)											
Testes à aplicação utilizando vários tipos de ataques											
Elaboração do Relatório											

1.3.Estrutura do relatório

O presente documento está estruturado em sete capítulos, sendo o primeiro reservado à introdução ao mesmo.

O segundo capítulo é dedicado ao enquadramento do projeto, com foco no estado da Arte. Neste capítulo são estudados e comparados vários softwares que interagem com a segurança das redes WiFi.

No terceiro capítulo é apresentado um estudo sobre o tráfego WiFi e os seus vários *frames* de forma a retirar métricas que consigam relacionar a problemática das redes WiFi com o seu tipo de tráfego.

No quarto capítulo é apresentada a arquitetura da solução proposta para este projeto, descrevendo em detalhe os vários módulos lógicos que esta integra, bem como a sua base de dados.

O quinto capítulo é dedicado à implementação do projeto, realçando os vários módulos físicos do projeto, como estes interagem entre si e especificando toda a parte de alarmística.

No sexto capítulo são realizados vários testes onde são apresentadas as várias experiências realizadas (ataques à rede WiFi) bem como os resultados obtidos.

Por último, no capítulo sete, é apresentada uma conclusão sobre o trabalho desenvolvido e onde são também referidos alguns tópicos para trabalho futuro.

2. Estado da arte

Neste capítulo irão ser abordadas conceitos e problemáticas relacionadas com as redes WiFi, bem como alguns softwares que fazem a monotorização, avaliação e gestão das mesmas.

2.1. Funcionamento e dados das redes WiFi

Pode existir a perceção errada de que os dados WiFi são o mesmo que o tráfego WiFi, mas tratam-se de conceitos distintos. Os dados WiFi apenas dizem respeito aos dados gerados enquanto o dispositivo não está associado a uma rede WiFi e, apenas depois desse ponto, é gerado o tráfego WiFi. No entanto, esse tipo de dados (tráfego) não é relevante para gerar alarmísticas, pelo que não faz sentido estar a usar poder computacional para os guardar e analisar.

O processo de conceção de acesso um dispositivo a uma rede WiFi é definido em 3 estados distintos, sendo que os dois últimos podem ser englobados num estado apenas, denominado por acesso à rede.

- *Discovering* de redes WiFi
- Autenticação/Encriptação
- Associação/Desassociação

2.1.1 *Discovering* de redes WiFi

Quando um dispositivo se pretende ligar a uma rede WiFi, o primeiro passo é efetuar um *scanning* para conseguir encontrar uma rede compatível e só depois desta etapa é possível efetuar o processo seguinte, a associação e autenticação. Para efetuar o *scanning* são necessários vários parâmetros, tais como BSSID (Basic Service Set Identifier), lista de canais, *MinChannelTime* e *MaxChannelTime*, entre outros. Normalmente estes tipos de parâmetros são definidos pelos construtores das placas de rede, no entanto podem ser alterados posteriormente pelo utilizador dessa mesma placa.

Existem dois métodos distintos de efetuar o *scanning*, o passivo e o ativo. No caso das redes WiFi não visíveis, normalmente denominadas por *hidden*, o tipo de *scanning* de rede terá de ser passivo, pois um *scanning* ativo não funciona com este tipo de redes WiFi. Normalmente os dispositivos efetuam os dois tipos de *scanning* em simultâneo em

todos os canais onde são permitidas redes WiFi, tendo em atenção que países diferentes podem utilizar canais distintos para a distribuição das redes WiFi. Como tal, o *scanning* ativo é apenas efetuado nos canais permitidos pela regulamentação do país. Todos os outros canais são automaticamente excluídos.

- *Scanning* Passivo – neste tipo de *scanning*, os clientes/dispositivos estão à escuta de um *beacon frame* mas sempre alternando pelos vários canais pois não sabem em qual deles existem dispositivos a enviar esses frames (conforme Figura 1). Os *beacon frames* são utilizados pelos Access Point (AP) para se anunciarem a outros AP's ou dispositivos.

  Quando um AP pretende enviar um *beacon*, ele define um intervalo de tempo denominado de *Target Beacon Transmission Time* e, assim que esse período terminar, ele irá enviar outro *beacon*, e assim sucessivamente. No entanto, pode não ser possível enviar *beacons* caso a rede WiFi esteja ocupada, sendo possível avaliar a sua ocupação através do protocolo CSMA/CA. Quando um *beacon frame* está pronto para ser enviado, e se o canal estiver ocupado, o AP faz um compasso de espera gerando um valor aleatório para o tempo de espera e volta a confirmar se a rede naquele canal está ocupada quando esse tempo de espera tiver passado. Se não estiver ocupada, ele envia o *beacon*.

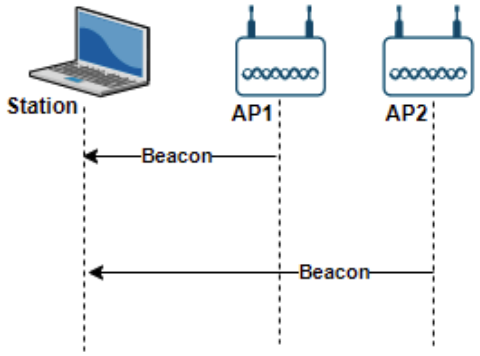


Figura 1 – Scanning Passivo [1]

- *Scanning* Ativo – neste método de *scanning*, o cliente/dispositivo inicia a procura enviando um *probe* [2] (frame de comunicação que contém os parâmetros de acesso a uma rede) para todos os canais disponíveis e espera por uma resposta. Se algum AP estiver nesse canal, irá responder ao pedido, ficando assim o cliente com a informação da sua existência (conforme Figura 2).

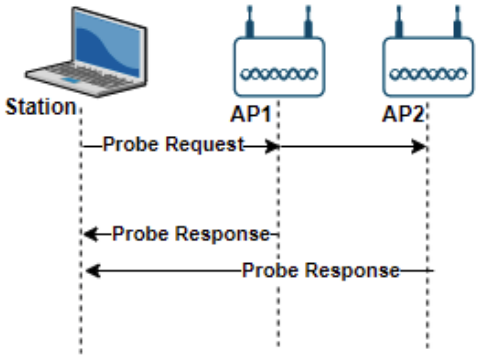


Figura 2 – Scanning Ativo [1]

O objetivo de uma *probe* é conseguir descobrir o AP e as suas redes associadas (SSID e/ou BSSID).

Sempre que uma *probe* é enviada, inicia automaticamente um tempo de contagem denominado por *probe timer countdown*. Este tempo é normalmente muito inferior ao tempo de intervalo entre o envio de *probes*, com valores na ordem dos 10 milissegundos. Assim que este *timer* terminar, o dispositivo que enviou a *probe* vai processar a resposta do mesmo e, caso não exista nenhuma resposta, segue para o próximo canal e repete o processo, enviando outra *probe*.

Mais informação detalhada sobre os vários tipos de *scanning* pode ser consultada no artigo [3].

2.1.2 Acesso a redes WiFi

O acesso às redes WiFi está subdivido em duas partes: autenticação e associação/desassociação.

- Autenticação

Assim que um dispositivo descobre a rede WiFi a que se pretende ligar através do *scanning* passivo ou ativo, é possível começar o processo de autenticação. As razões que levaram à criação deste processo de autenticação prendem-se com a validação do tipo de dispositivo, nomeadamente a sua compatibilidade com a rede WiFi, e o tipo de encriptação usada.

Este processo de autenticação inicia-se com o envio de um *frame* de autenticação que, quando é recebido pelo AP, responde com um *frame* de *acknowledgment* seguido de uma *autentication response* (conforme Figura 3). Este antigo processo de autenticação

também era usado para fazer a autenticação do protocolo WEP (Wired Equivalent Privacy), no entanto caiu em desuso pelo facto de ser obsoleto.

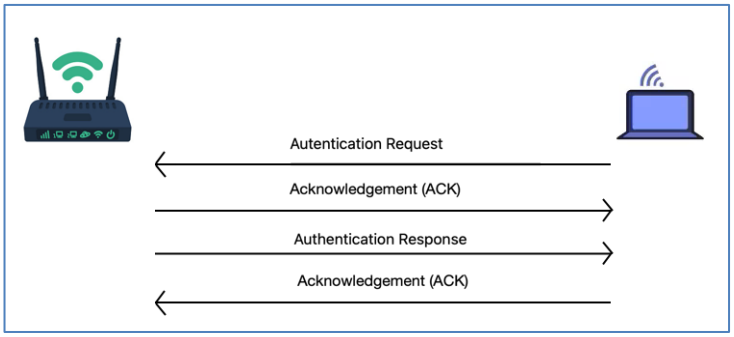


Figura 3 - Autenticação WiFi (artigo [4]) [5]

- Associação

Assim que o processo de autenticação for concluído, inicia o processo de associação que consiste num pedido de *association request* feito pelo dispositivo/cliente ao AP, que responde com uma *association response*. Desta forma, o dispositivo/cliente consegue juntar-se à rede e obter um *association* ID. No caso de existir autenticação do tipo WPA (WiFi Protected Access) / WPA2 (WiFi Protected Access 2) ou 802.1X, será necessário um passo adicional de autenticação para posteriormente possibilitar o envio de dados através da rede.

- Desassociação

Quando um dispositivo está associado a um AP, ambas as partes podem terminar essa associação enviando um *frame* de desassociação. O cliente/dispositivo pode enviar um *frame* de desassociação para deixar a rede ou para indicar que se vai ligar a outra rede.

Este processo não utiliza nenhum tipo de encriptação e apenas é necessário utilizar o *Mac Address* do dispositivo que se pretende desassociar. Um cliente desassociado continua autenticado e pronto para se associar novamente.

Mais detalhes sobre as ligações WiFi podem ser consultados no documento [3] .

2.2.Estados dos dispositivos nas redes WiFi

No processo de ligação de dispositivos à rede WiFi, os dispositivos têm 3 estados diferentes:

- Não autenticado nem associado;

Exemplo: Um dispositivo que está a passar por uma zona onde existe uma rede WiFi mas sem qualquer intenção de se ligar à mesma, no entanto, desde que o dispositivo esteja com o suporte WiFi ligado, é sempre efetuado o processo de *scan*.

- Autenticado mas não associado;

Exemplo: Um dispositivo que acabou o processo de autenticação mas ainda não finalizou o processo de associação.

- Autenticado e associado.

Exemplo: Um dispositivo totalmente ligado a um AP e pronto a transmitir dados.

2.3.Problemática das redes WiFi

Atualmente as redes WiFi são imprescindíveis em qualquer atividade económica e até mesmo a nível pessoal, no entanto elas acarretam enormes problemáticas já bastante conhecidas, das quais se salientam:

- **Deauthentication**

O processo de *deauthentication*[1] atua na camada 2 e é considerado um procedimento bastante simples. Utiliza a falta de proteção do protocolo 802.11 ao lidar com *frames* de forma não encriptada, nomeadamente o *frame* de *deauthentication*, e pode ser aproveitado para um ataque à rede WiFi. Esta falha de encriptação nos pacotes de gestão da ligação faz com que um dispositivo que se pretenda desassociar de uma rede WiFi apenas tenha de enviar o seu *Mac Address*, um dado facilmente obtido por alguém que faça *sniffing* na rede.

Quando um dispositivo se pretende desligar de uma rede WiFi, é enviado um *frame* de *deauthentication*, no entanto este tipo de *frame* não tem autenticação, não sendo assim possível verificar se quem o enviou é realmente quem diz ser, criando uma falha de segurança.

No momento em que o AP recebe este *frame*, ele responde com um pacote de *deauthentication acknowledging* e encerra a ligação.

[1] Mais informação sobre este tipo de ataque pode ser consultada no artigo [5]

Se um atacante repetir o processo de *deauthentication*, é possível criar um *denial-of-service* pois, mesmo que o dispositivo se tente ligar várias vezes, a ligação irá ser constantemente interrompida por pedidos de *deauthentication* (Figura 4).

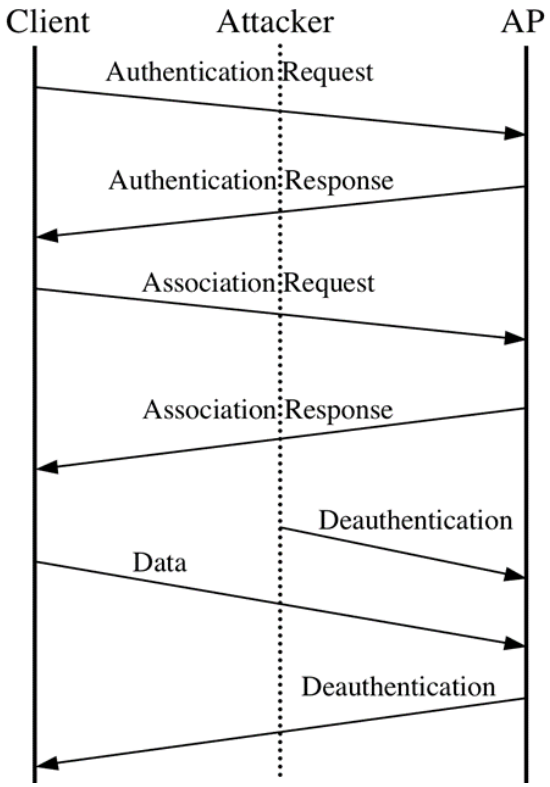


Figura 4 - Pedidos de Deauthentication [5]

Em 2009 tentaram-se resolver várias falhas conhecidas com o protocolo 802.11w, que tem a capacidade de encriptação de vários tipos de pacotes. Apesar deste protocolo conseguir proteger os equipamentos contra este tipo ataque (*deauthentication*), também permitiu que surgissem novas formas de *denial-of-service*. Outro dos problemas deste novo protocolo é que apenas está disponível para um número muito limitado de dispositivos de rede, o que faz com que muitos equipamentos ainda possam sofrer este tipo de ataques.

Existem várias CVE's (Common Vulnerabilities and Exposures) que associam alguns AP a ataques de *deauthentcation*, tais como CVE-2020-3206, CVE-2019-5062 ou CVE-2019-3944.

- **Número de tentativas de autenticação**

Um atacante pode tentar autenticar-se por *brute-force* ou até mesmo tentar debilitar o funcionamento de um router enviando milhares de pedidos de autenticação. Mesmo que

nunca consiga fazer a autenticação, a existência de todos estes pedidos que têm de ser processados pelo router pode eventualmente exceder o seu poder de processamento e afetar todo o restante tráfego da rede, fazendo assim um ataque de *denial-of-service*. A monitorização dos pedidos de autenticação, nomeadamente os pacotes descartados (*drop package*), deverá ser uma métrica a ter em conta. Para mais informação sobre ataques de *brute-force*, é possível consultar a referência [6].

Os *Rogue* AP (*Access Points*) são, na sua essência, *Access Points* não autorizados. Alguns *Access Points* públicos, por exemplo em hotéis, cafés ou aeroportos, não dispõem de nenhum sistema de autenticação, sendo que um atacante pode criar um AP fazendo-se passar por um AP legítimo, processo conhecido como *Rogue* AP.

Quando um utilizador se liga a este *Rogue* AP, é possível desenvolver um conjunto de ataques de rede com a finalidade de executar ataques de *phishing* às credenciais de acesso, alteração de tráfego ou impersonificação do mesmo, em muitos dos casos, mesmo que a página utilize encriptação SSL.

Outro problema relacionado com os *Rogue* AP está no facto dos dispositivos, dependendo da forma de configuração, se ligarem automaticamente ao AP com sinal mais forte, portanto existe a possibilidade de o utilizador pensar que está ligado a uma rede segura mas, no entanto, estar ligado a um *Rogue* AP (conforme Figura 5).

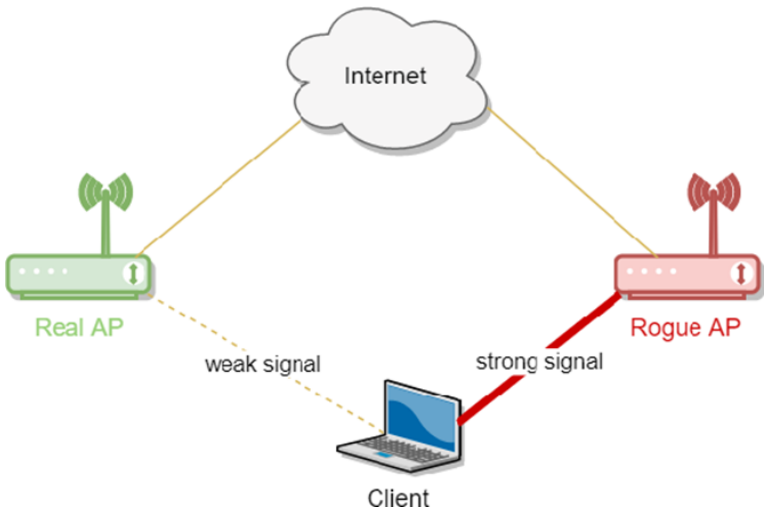


Figura 5 - Rogue AP [7]

Mais informação sobre este tipo de ataque pode ser encontrada no artigo [8].

- **Nova rede**

A facilidade na criação de um AP, e consequentemente de novas redes, pode despoletar um problema de segurança. Um atacante consegue facilmente criar um novo AP e alguns utilizadores menos atentos podem ligar-se a esta rede simplesmente por lapso ou até porque a rede não tem password. No entanto, ligar-se a um AP desconhecido pode trazer vários perigos, sendo um dos mais comuns o *Man-in-the-middle*[2], em que um atacante consegue intercetar a informação que é transmitida pelos dispositivos ligados, como é visível na Figura 6.

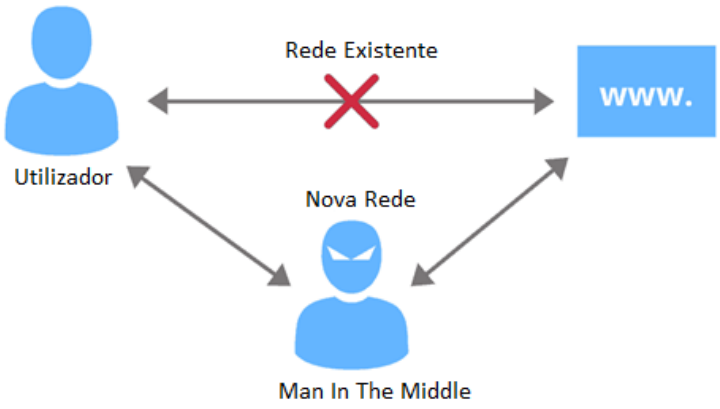


Figura 6 – Man-In-The-Middle [9]

2.3.1 Número de pacotes da rede

Todo o tráfego gerado numa rede é, na realidade, pacotes de informação, quer sejam de entrada ou de saída. No entanto, existe a possibilidade de alguém tentar obstruir a rede, fazendo pedidos repetidos e ocupando todo o buffer do router, levando-o a descartar alguns pacotes por falta de buffer. Um exemplo é o *syn flood*[3], em que um atacante efetua uma série de pedidos de *three-way handshake*[4] mas nunca finaliza essa ligação, fazendo com que o router não consiga aceitar mais ligações (conforme ilustrado na Figura 7).

[2] Man-in-the-middle é uma forma de ataque em que os dados trocados entre duas partes são de alguma forma intercetados, registados e, possivelmente, alterados pelo atacante sem que as vítimas se apercebam.
[3] Mais informação sobre este ataque pode ser consultada em [15]
[4] O Three-way Handshake é um processo usado nas redes TCP/IP para estabelecer a ligação cliente/servidor

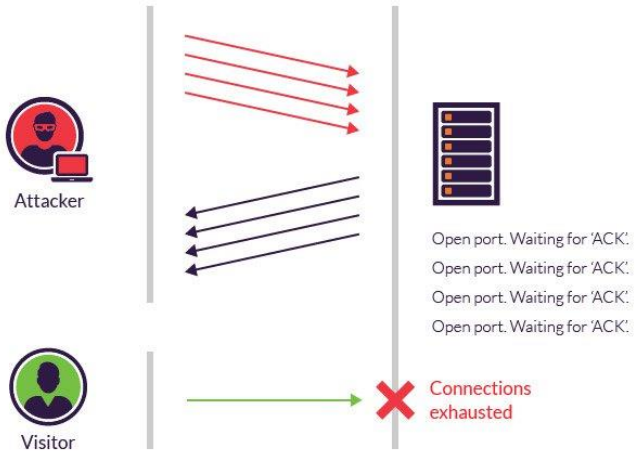


Figura 7 - Syn Flood ataque [7]

2.3.2 *Mac Address* desconhecidos

Algumas redes WiFi podem ser restringidas com uma filtragem por *Mac Address*, garantindo que apenas os dispositivos com os *Mac Address* conhecidos se liguem a essa rede.

Esta é uma das formas que os administradores de rede dispõem para controlar o acesso à rede, no entanto este cenário traduz-se num trabalho bastante significativo de registo e monitorização de endereços MAC (Mac Address) a serem acrescentados ou retirados das listas, por parte dos serviços de administração da rede.

Por outro lado, a facilidade com que é possível detetar endereços MAC's válidos e efetuar *arp spoofing* permite a um atacante interligar-se a uma rede, como se fosse um utilizador válido, o que torna um pouco inglório o trabalho dos serviços de administração e se traduz num mecanismo extremamente falível no que respeita à segurança de acesso à rede.

2.4.Ferramentas existentes

Atualmente existem alguns softwares capazes de fazer uma análise e monitorização de redes WiFi, no entanto a grande maioria é desenvolvida a pensar na disponibilidade da rede WiFi num espaço físico, de forma a garantir uma maior largura de banda aos utilizadores, e não de um ponto de vista de segurança da própria rede WiFi, descurando a análise e monitorização ao nível de segurança.

2.5.NetSpot

O “NetSpot”[5] é um dos exemplos mais populares no ramo de monitorização de redes WiFi. Trata-se de um software não gratuito que permite analisar redes WiFi, enumerar a quantidade de utilizadores ligados e disponibilizar métricas que permitam melhorar o estado dessa rede, tentando obter uma rede mais estável ou que tenha um maior alcance. É possível medir a intensidade da rede em vários pontos (conforme Figura 8) e posteriormente são sugeridas algumas melhorias, como a alteração de canais de rede WiFi, entre outras.

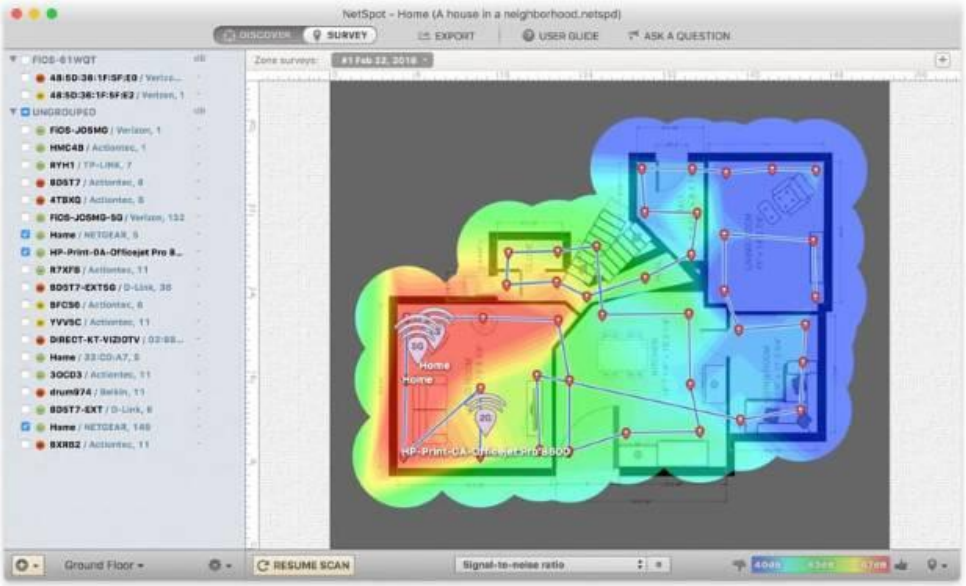


Figura 8 - Net [10]

2.5.1 Modo enumeração de redes WiFi do NetSpot

O “NetSpot” tem a capacidade de recolher várias métricas das redes WiFi e de as apresentar numa tabela interativa (Figura 9). A interface do “NetSpot”, juntamente com a tabela interativa, permitem resolver alguns problemas de rede e ainda melhorar a cobertura da mesma, graças à sua capacidade de monitorização e recolhas de métrica de análise do desempenho, do sinal, da interferência e do ruído das redes WiFi.

[5] Mais informação sobre o NetSpot pode ser encontrada em [10]

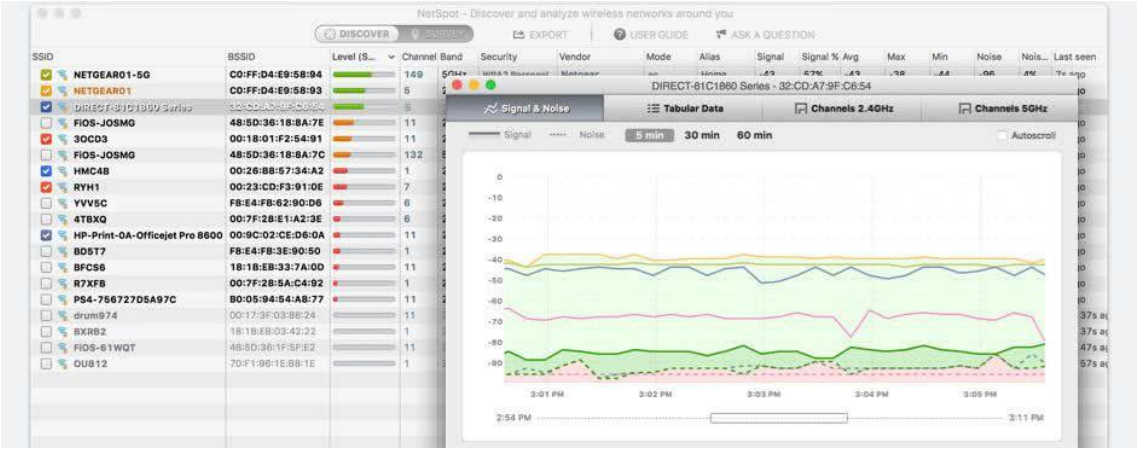


Figura 9 - Tabela interativa

2.5.2 Modo monitorização da rede WiFi

Com este modo, o "NetSpot" faz uma análise da rede WiFi de forma rápida, utilizando um sistema de mapas de calor interativo codificado por cores. Este mapa tem a capacidade de mostra várias redes e dispositivos de WiFi em tempo real no mapa (Figura 10).

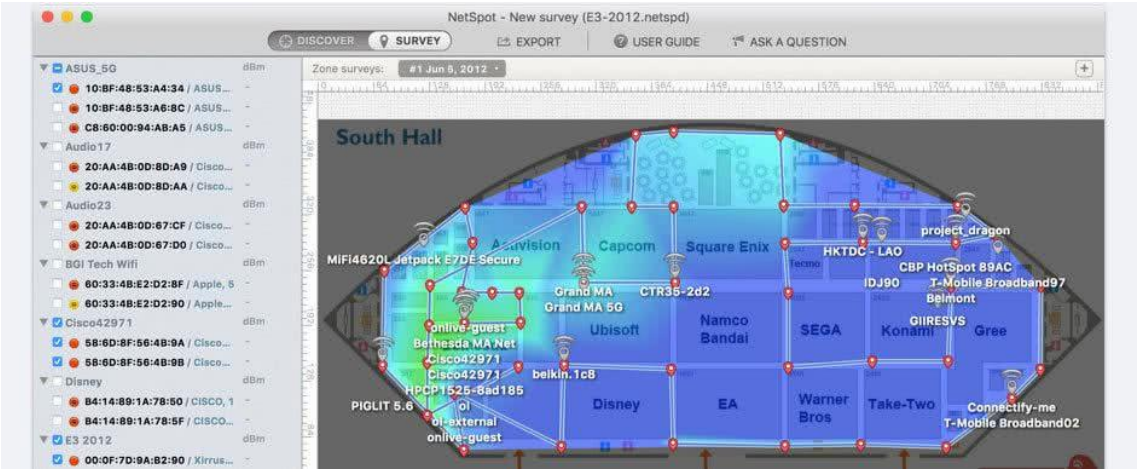


Figura 10 - NetSpot análise Wifi

2.5.3 Extração de dados

Esta aplicação permite fazer a extração de dados de diversas formas, nomeadamente:

- Exportar como PDF (Portable Document Format)
- Exportar como CSV (Comma-Separated Values)
- Gravação de mapas como PNG (Portable Network Graphics)
- Importar e exportar BSSID
- Partilha de dados entre vários projetos
- Entre outros.

Para mais informação sobre este software é possível consultar o website oficial do "NetSpot" em https://www.netspotapp.com/pt/.

2.6. AirCrack-NG

O software "Aircrack-ng"[6] é um conjunto de várias ferramentas *open-source* desenvolvidas para testar a segurança das redes WiFi, conseguindo assim obter várias capacidades:

- Monitorização de pacotes: capturando e exportando dados para ficheiros de texto;
- Ataque: *deauthentication*, criação de um rogue AP ou injeção de pacotes;
- Teste: possibilidade de testar placas de rede e respetivos drivers (captura e injeção);
- Craking, WEP e WPAS PKS (WPA1 e WPA2).

Conforme mencionado anteriormente, o Aircrack-ng está divido em várias ferramentas:

- "aircrack-ng": quebra chaves WEP e WPA (Por Força-bruta);
- "airdecap-ng": de-criptografa arquivos capturados com criptografia WEP ou WPA com a chave conhecida;
- "airmon-ng": coloca placas de rede em diferentes modos;
- "aireplay-ng": injeção de pacotes (Somente em Linux);
- "airodump-ng": coloca tráfego WiFi num arquivo .cap e mostra informação das redes;
- "airtun-ng": cria interfaces virtuais.

Apesar das várias ferramentas, o "Aircrack-ng" não disponibiliza nenhuma interface ou ferramenta que permita analisar os dados extraídos.

[6] Informação detalhada sobre este software pode ser consultada em [16]

2.7.Kismet

O "Kismet"[7] é um software *open-source* que tem como principal funcionalidade a análise de redes wireless e, entre elas, as redes WiFi. Com esta plataforma é possível:

- Detetar comunicações
- *Sniffing* de rede
- Analisar drivers
- Detetar intrusões de redes wireless.

2.7.1 Kismet Captura de dados

A captura de dados deste software é extremamente abrangente pois conta com um sistema de captura de dados remota, que permite que os dados das redes wireless sejam extraídos simultaneamente por vários dispositivos remotos e posteriormente enviados para uma unidade central por um canal seguro, utilizando o protocolo TCP ou *websockets*. Este software conta também com a sua própria base de dados para conseguir assim armazenar todos os dados recebidos. Imaginando uma instituição que tenha uma área física de 5000 metros quadrados, é possível ter vários dispositivos espalhados a analisar toda a rede WiFi em simultâneo e posteriormente enviar todos esses dados para o servidor principal.

2.7.2 Kismet API

O "Kismet" dispõe de uma API através de um *webserver* que está instalado no seu core. Com recurso a esta API, é possível integrar o "Kismet" com outros softwares, sendo possível extrair os dados com pedidos do tipo GET ou, para pedidos mais complexos, existe a possibilidade de efetuar os mesmos com o método POST, adicionando parâmetros no corpo do pedido. Esta API suporta vários formatos de saída, sendo o seu *default* JSON (JavaScript Object Notation). No entanto, é possível configurar outros formatos recorrendo a *run-time plug-ins*.

Todos os *endpois* têm um mecanismo de segurança associado onde requerem autenticação, à exceção de alguns *endpoints* de informação do estado do serviço (Kismet).

[7] Mais informação sobre este software pode ser consultada em [17]

Com esta segurança, é possível garantir que apenas os utilizadores com as respetivas credenciais têm acesso à informação.

2.7.3 Kismet Logs e Database

O "Kismet" tem um sistema de *logs* que possibilita guardar todos os seus registos, podendo escolher entre os seguintes formatos de ficheiros para os *logs*:

- "*.kismet*", o principal formato utilizado pelo Kismet, que combina todos os dados obtidos por este software, nomeadamente pacotes, dispositivos, records, alarmes, mensagens de sistema, localizações de GPS (Global Positioning System), entre outros. Este formato de log consiste em uma base de dados sqlite3 (Figura 11).

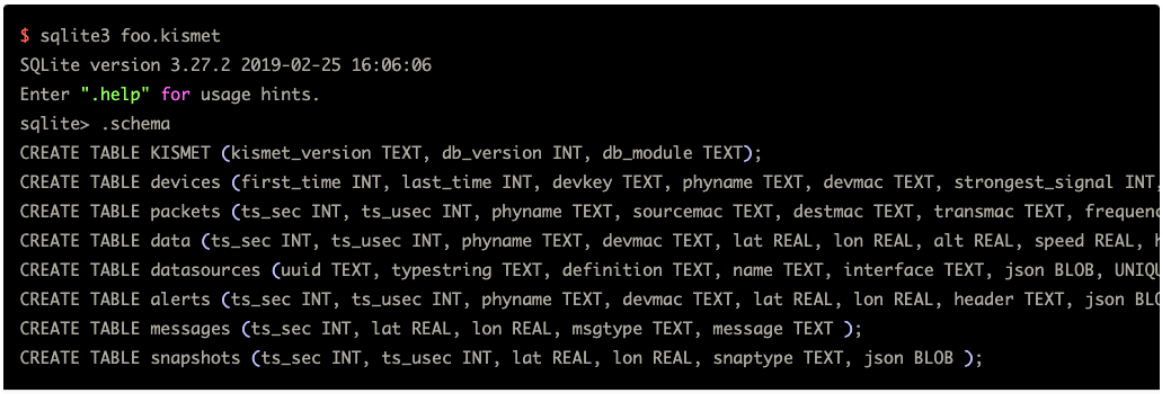


Figura 11 - Kismet Sqlite

Este tipo de log tem a particularidade de ser possível escolher a validade dos *logs*, escolhendo uma data máxima, o que faz com que o mesmo seja automaticamente excluído nessa data. O valor dessa validade é configurável (exemplo na Figura 12).

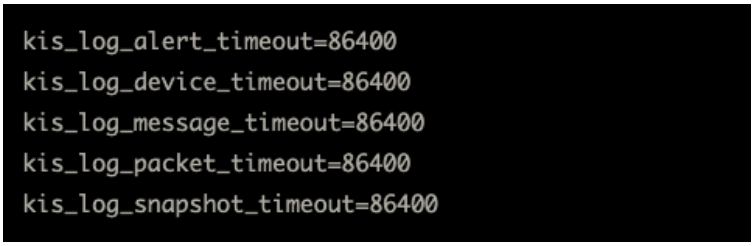


Figura 12 - Kismet Logs Timer

- "*.pcapppi*", formato antigo referente ao protocolo *pcap* que usa cabeçalhos PPI. Este protocolo tem o inconveniente de não permitir guardar todos os dados relativos aos pacotes recolhidos pelo Kismet.
- "*.pcapng*", o formato mais recente baseado no protocolo *pcap*, e que tem a particularidade de poder ser facilmente exportado e utilizado em softwares como

"Wireshark" e "TShark", apesar de nem todas as ferramentas do Kismet suportarem este novo protocolo. Com este formato o Kismet consegue ainda guarda dados de várias *sources* em simultâneo no mesmo ficheiro.

Apesar de existirem vários formatos diferentes, existe a possibilidade de usar mais do que um formato em simultâneo, criando assim vários formatos de *logs*.

2.7.4 Kismet Dashboard

O "Kismet" dispõe de um *dashboard* com uma interface baseada em web que permite consultar a informação recolhida, desde a informação mais simples até à mais complexa. É ainda possível adicionar vários *plug-ins* com uma UI em JavaScript, facilitando a personalização deste *dashboard*.

Para aceder ao *dashbaord* é necessário fazer o processo de autenticação com *username* e *password*. Posteriormente é possível observar os dados da aplicação (Figura 13), bem como dispositivos, mensagens, alarmes, entre outros.

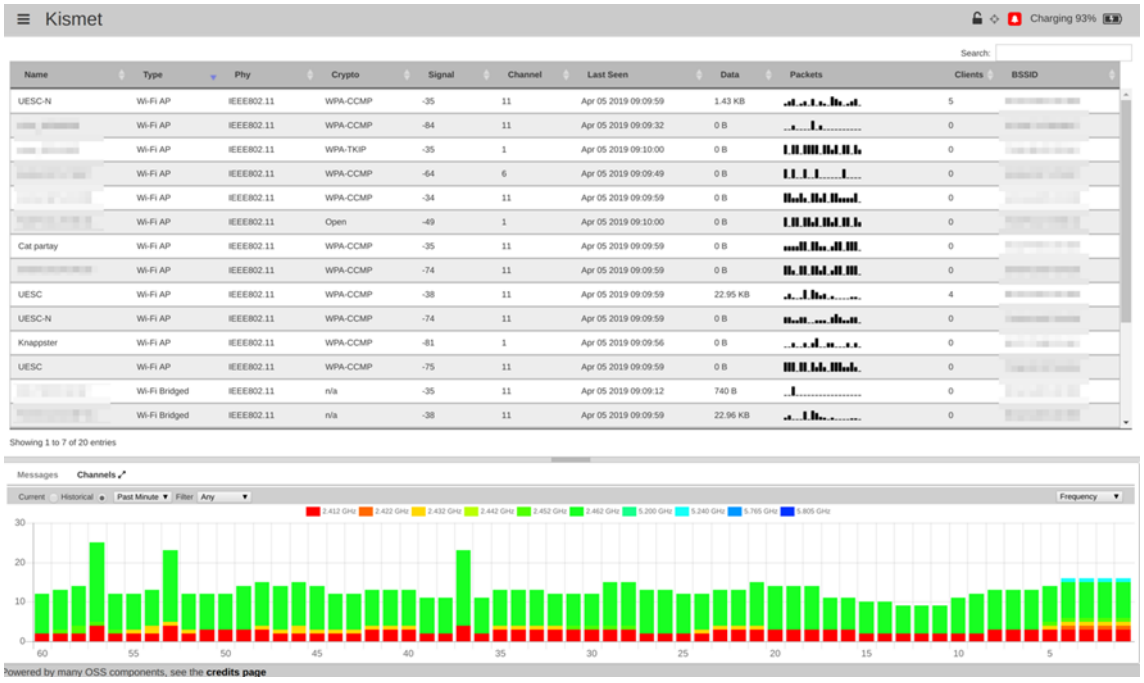


Figura 13 - Kismet Dashboard

A parte alarmística deste software pode ser um pouco limitativa no que diz respeito ao tipo de alarmes (Figura 14), sendo estes mais utilizados para reportar potenciais problemas com o software em si, e não com a parte de segurança da rede.

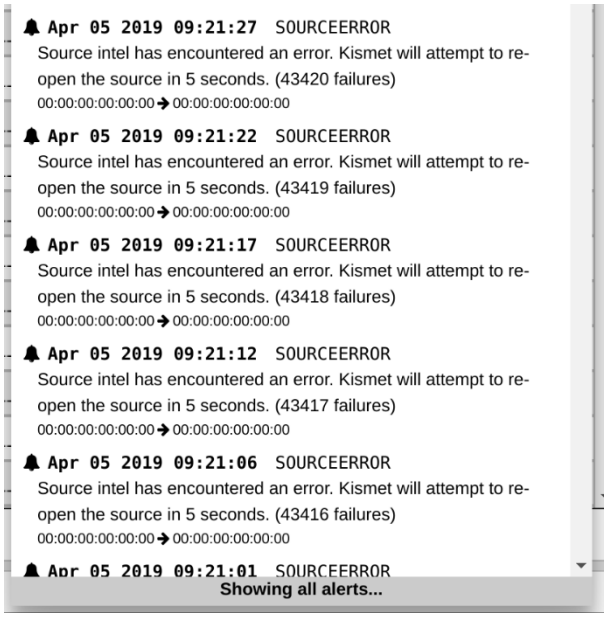


Figura 14 - Kismet Alarmes

O *dashbaord* conta também com uma parte destinada aos dispositivos encontrados e às redes que eles utilizam, mostrando os vários dados que captou (Figura 15).

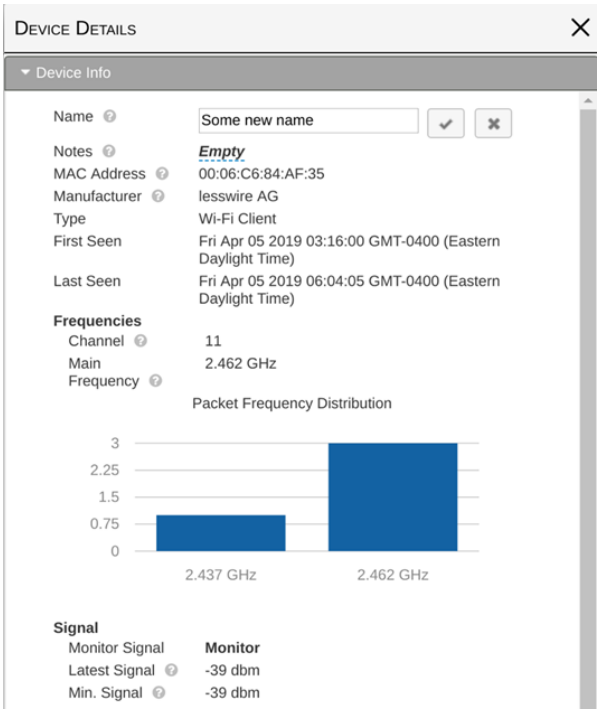


Figura 15 - Kismet Dados Wifi

Existe ainda a opção de visualizar os dados relativos às várias redes WiFi encontradas (Figura 16), podendo assim rapidamente perceber a quantidade de dados que têm como destino a rede selecionada.

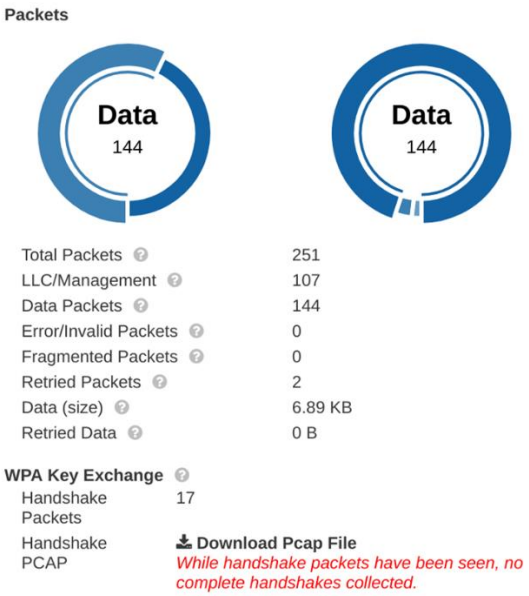


Figura 16 - Kismet Dados Dispositivo

Mais informação sobre este software pode ser encontrada no website oficial do Kismet, em https://www.kismetwireless.net/.

2.8. Comparativo de softwares

Considerando os vários softwares analisados anteriormente, é possível afirmar que o Kismet tem alguma vantagem relativamente aos seus concorrentes diretos, como o "aircrack-ng", devido ao seu sistema de captura de dados remota, e ainda ao seu sistema de base de dados, uma vez que o "aircrack-ng" apenas permite exportar os dados recolhidos para um ficheiro de texto, podendo ser um pouco limitativo.

No entanto, nenhum deste softwares executa em pleno as funções propostas neste projeto, porque apesar da forte componente de análise de rede WiFi, a parte de alarmística é bastante fraca ou praticamente inexistente, como no caso do "aircrack-ng". É possível observar esta comparação na tabela abaixo (Tabela 2).

Tabela 2 - Comparativo Softwares

Software	Mapeamento de Redes	Alarmística	Recolhe local de Pacotes WiFi	Recolha remota de Pacotes WiFi	API Integrada	DB Integrada
NetSpot	✓	✓	X	X	X	X
AirCrack	X	X	✓	X	X	X
Kismet	X	X	✓	✓	✓	✓

2.9. Síntese

Tal como mencionado no ponto anterior, não existe nenhuma aplicação que faça simultaneamente a recolha dos dados de redes WiFi e posteriormente a parte de alarmística. Surge então a necessidade de se criar uma solução para tentar colmatar as problemáticas já conhecidas das redes WiFi.

Tendo em conta a análise dos pontos anteriores, foi criada uma solução baseada no software *open-source* “Kismet”, tirando partido da possibilidade de fazer captura de dados WiFi de múltiplos pontos em simultâneo.

3. Estudo do tráfego WiFi

Foi efetuado o levantamento e estudo do tipo de tráfego que existe na rede WiFi a fim de avaliar os dados necessários para recolha, armazenamento e análise.

3.1. Funcionamento dos vários tipos de *management frames*

De entre os vários tipos de *frames* existentes, os do tipo *management frames*, presentes no tráfego WiFi, e relevantes para o presente projeto, são:

- *Beacon Frame*

  Os *beacon frames* são utilizados no *scanning* passivo e têm a configuração apresentada na Figura 17 - Beacon Frame.

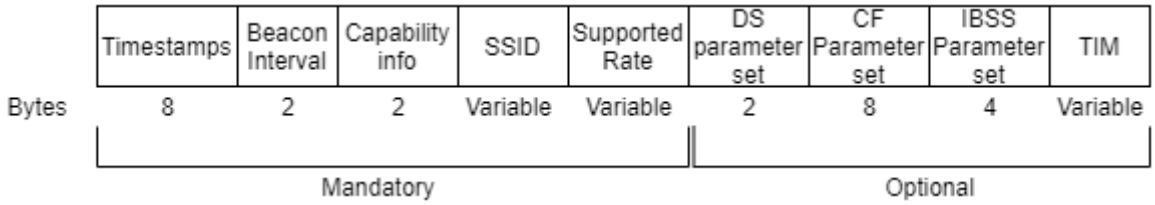


Figura 17 - Beacon Frame [11]

1. *Timestamp* utiliza 8 bytes e permite a sincronização entre vários dispositivos, representando em microssegundos o tempo em que o AP está ligado. Assim que este tempo chegar ao seu valor máximo (2^64 microsegundos), ele dá *reset* e volta ao valor 0. Este campo de *timestamp* é comum no *beacon frame* e no *probe request frame*.
2. *Beacon Interval* utiliza 2 bytes para representar o número de *time units* (TUs) entre o *Target Beacon Transmission Time* (TBTT), cujo valor por *default* é 100TU (102.4 milissegundos).
3. *Capability information*, com 2 bytes, contém informação sobre as capacidades do dispositivo/rede (Figura 18).

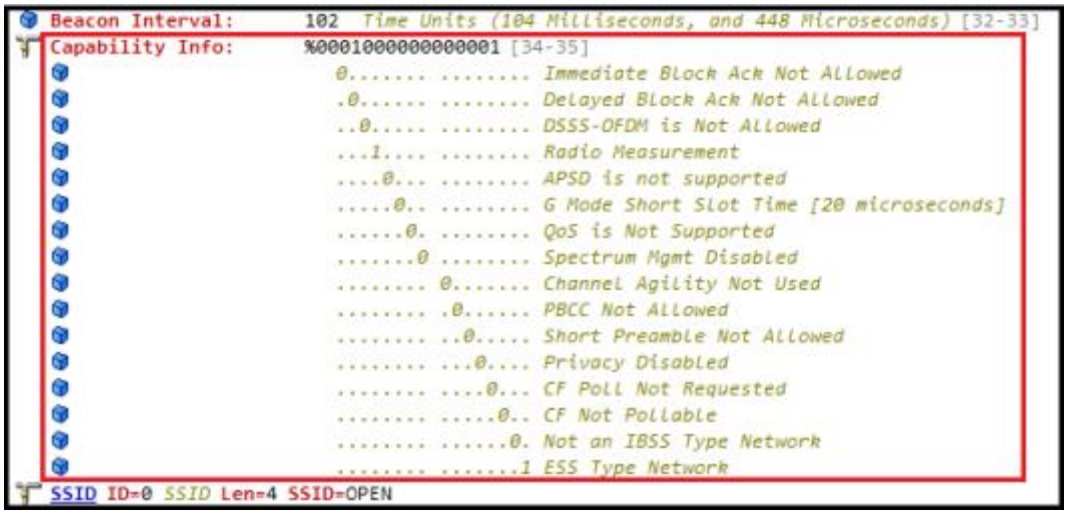


Figura 18 - Capability Information [2]

4. SSID, presente em todos os *beacon frames*, *probe request frames*, *probe* response *frames* e *association frame*, é um campo de valor variável e contém o *Service Set ID* da rede.
5. *Supported Rates*, contém informação das *data rates* suportadas pelo dispositivo.

- *Probe Request/Response Frame*

Na Figura 19 pode-se observar um *frame* de *request/responset*, sendo que a parte revelante é o *frame body*.

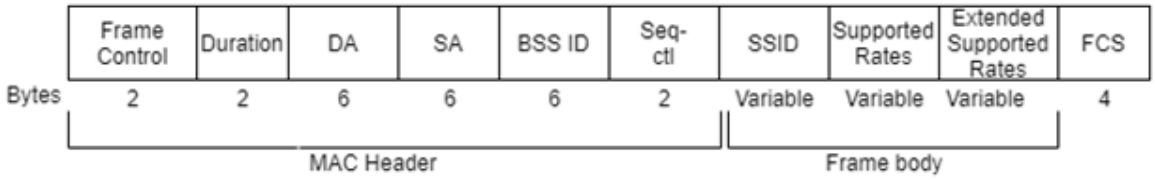


Figura 19 - Probe Request/Response Frame [11]

O dispositivo que envida o *probe request* pode especificar um SSID, processo conhecido como "*direct probe request*", no entanto este valor também pode ir a 0, sendo assim um *wildcard* SSID, também conhecido como "*null probe Request*".

Este *frame* também indica as frequências suportadas e, se o dispositivo suportar mais de 8 frequências, o campo *Extended supported rates* é obrigatório. Caso contrário, é um campo opcional.

- *Authentication frame*

  O formato do *frame* de autenticação é possível observar na Figura 20.

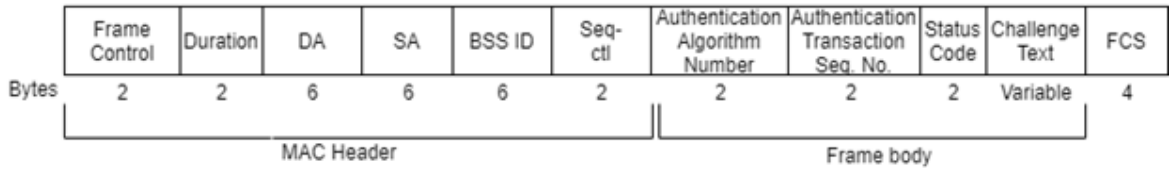


Figura 20 - Authentication Frame [11]

O corpo deste *frame* conta com quatro campos:

1. *Authentication algorithm number*, sendo 0 para sistemas abertos e 1 para *shared key* (chave partilhada).
2. *Authentication transaction Sequence number*, indica o estado de progresso.
3. *Status Code*, onde o 0 se refere a sucesso e 1 a falhas.
4. *Challenge Text*, utilizado para a autenticação por *Shared key*.

- *Association Request / Response Frame*

O primeiro *frame* enviado na fase de associação (Figura 21) é efetuado pelo dispositivo que se pretende ligar ao AP. Este *frame*, do tipo *unicast*, é apenas direcionado a um AP. Assim que ele recebe o *association request frame* verifica se existe compatibilidade com todos os parâmetros do protocolo 802.11 entre o dispositivo que fez o pedido e o próprio AP. Caso os parâmetros entre os dois sejam diferentes, ele analisa se essa diferença é bloqueante. Se for, o AP responde com o *status code* 1, indicando que rejeita a associação. Se não for bloqueante ou não existir qualquer diferença, então responde com o código 0, indicando sucesso, e envia os parâmetros do AP para o dispositivo que efetuou o pedido.

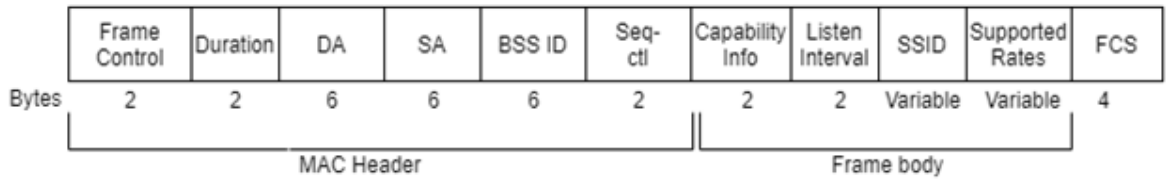


Figura 21 - Association Request/Reponse Frame [11]

- *Disassociation Frame* e *Deauthentication Frame*

O *frame* de *disassociation* e *deauthentication* tem um tamanho bastante reduzido e que utiliza apenas um campo, que indica o *Reason code* (Figura 22).

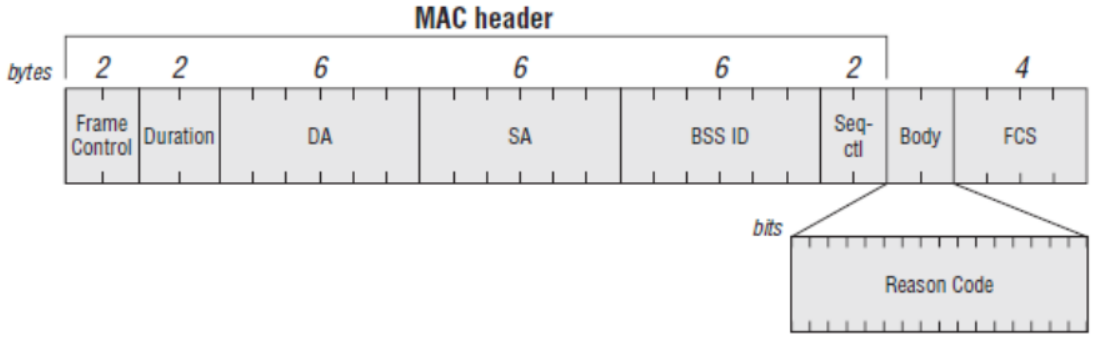


Figura 22 - Diassociaton e Deauthentication Frame [11]

Para obter mais informações sobre os vários tipos de *frames*, é possível consultar a referência [12] ou a RFC 5416 [13].

3.2. Captura de tráfego para vários ataques

Foi capturado e posteriormente analisado o tráfego gerado por vários tipos de ataques à rede WiFi e aos equipamentos da mesma, com o objetivo de analisar a necessidade e peso dos dados enviados durante as comunicações, com vista à deteção de atividades anómalas.

Os vários ataques foram efetuados com recurso a uma máquina virtual com o Sistema Operativo Kali20201 e vários softwares já construídos para o efeito.

Na parte de captura de dados, foi utilizada uma segunda máquina virtual com o mesmo sistema operativo, juntamente com os softwares “AirCrack-ng” e “Kismet”, e ainda uma placa de WiFi em modo monitor para conseguir capturar todos os dados.

Mais informações sobre os vários tipos de ataques podem ser consultadas em [14].

Foram executados os seguintes ataques:

3.2.1 Ataque de *Deauthenticate*

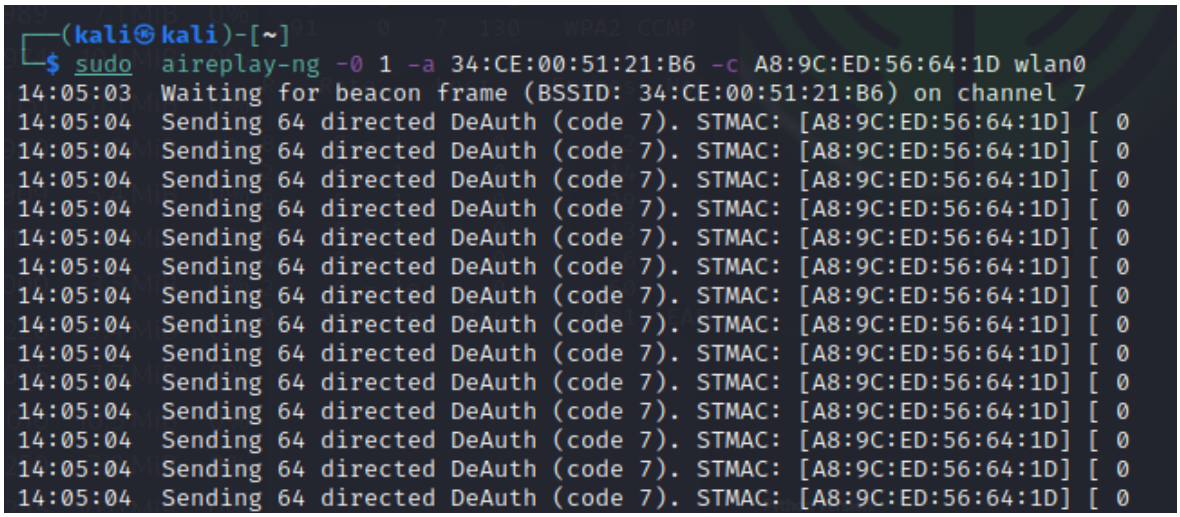
Foi executado um ataque de *deauthenticate* ao dispositivo com o endereço MAC A8:9C:ED:56:64:1D, que estava ligado ao *Access Point* com o MAC

34:CE:00:51:21:B6, utilizando a ferramenta *aireplay-ng*, através do comando apresentado na Tabela 3.

Tabela 3 - Comando Deauthenticate

```
sudo  aireplay-ng -0 0 -a 34:CE:00:51:21:B6 -c A8:9C:ED:56:64:1D wlan0
```

Assim que este comando foi executado, foram efetuados múltiplos pedidos de *deauthenticate* para o AP onde este dispositivo estava ligado, conforme resultado apresentado na Figura 23.



```
┌──(kali㉿kali)-[~]
└─$ sudo  aireplay-ng -0 1 -a 34:CE:00:51:21:B6 -c A8:9C:ED:56:64:1D wlan0
14:05:03  Waiting for beacon frame (BSSID: 34:CE:00:51:21:B6) on channel 7
14:05:04  Sending 64 directed DeAuth (code 7). STMAC: [A8:9C:ED:56:64:1D] [ 0
14:05:04  Sending 64 directed DeAuth (code 7). STMAC: [A8:9C:ED:56:64:1D] [ 0
14:05:04  Sending 64 directed DeAuth (code 7). STMAC: [A8:9C:ED:56:64:1D] [ 0
14:05:04  Sending 64 directed DeAuth (code 7). STMAC: [A8:9C:ED:56:64:1D] [ 0
14:05:04  Sending 64 directed DeAuth (code 7). STMAC: [A8:9C:ED:56:64:1D] [ 0
14:05:04  Sending 64 directed DeAuth (code 7). STMAC: [A8:9C:ED:56:64:1D] [ 0
14:05:04  Sending 64 directed DeAuth (code 7). STMAC: [A8:9C:ED:56:64:1D] [ 0
14:05:04  Sending 64 directed DeAuth (code 7). STMAC: [A8:9C:ED:56:64:1D] [ 0
14:05:04  Sending 64 directed DeAuth (code 7). STMAC: [A8:9C:ED:56:64:1D] [ 0
14:05:04  Sending 64 directed DeAuth (code 7). STMAC: [A8:9C:ED:56:64:1D] [ 0
14:05:04  Sending 64 directed DeAuth (code 7). STMAC: [A8:9C:ED:56:64:1D] [ 0
14:05:04  Sending 64 directed DeAuth (code 7). STMAC: [A8:9C:ED:56:64:1D] [ 0
14:05:04  Sending 64 directed DeAuth (code 7). STMAC: [A8:9C:ED:56:64:1D] [ 0
14:05:04  Sending 64 directed DeAuth (code 7). STMAC: [A8:9C:ED:56:64:1D] [ 0
```

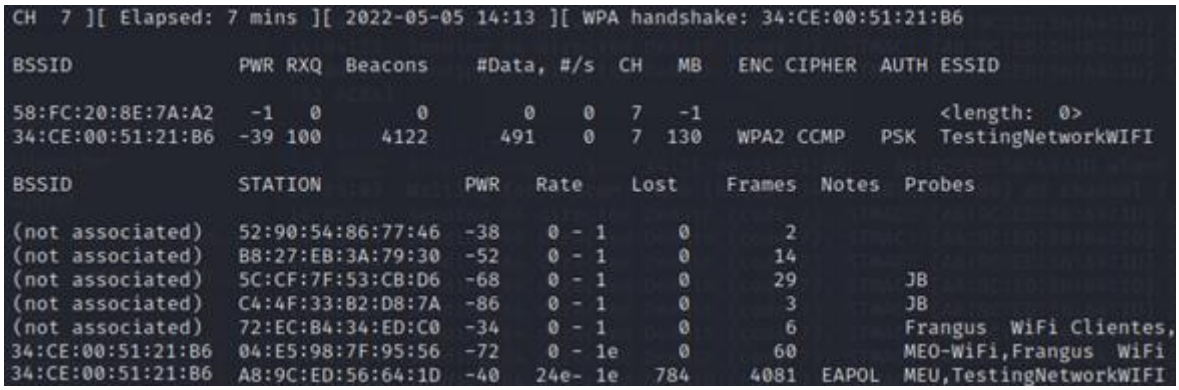
Figura 23- Aireplay-ng

Foi utilizado o “airdump-ng” com o comando da Tabela 4 para capturar todos os pacotes de rede, nomeadamente os pacotes de *deauthenticate*. Foi também ligado um segundo dispositivo ao AP para ser possível comparar o número de pacotes numa situação normal e num cenário de ataque.

Tabela 4 - Captura de pacotes

```
sudo airodump-ng wlan0 -c 7
```

Na Figura 24 é possível visualizar que existem dois dispositivos ligados ao mesmo AP (58:FC:20:8E:7A:A2), sendo que o último tem um elevado número de pacotes (*Lost e Frames*) quando comparando com o outro dispositivo, devido aos vários pedidos de *deauthenticate*.



```
CH  7 ][ Elapsed: 7 mins ][ 2022-05-05 14:13 ][ WPA handshake: 34:CE:00:51:21:B6

BSSID              PWR RXQ  Beacons    #Data, #/s  CH   MB   ENC CIPHER  AUTH ESSID

58:FC:20:8E:7A:A2   -1   0        0        0    0   7   -1                    <length:  0>
34:CE:00:51:21:B6  -39 100     4122      491    0   7  130   WPA2 CCMP   PSK  TestingNetworkWIFI

BSSID              STATION            PWR   Rate    Lost    Frames  Notes  Probes

(not associated)   52:90:54:86:77:46  -38    0 - 1      0         2
(not associated)   B8:27:EB:3A:79:30  -52    0 - 1      0        14
(not associated)   5C:CF:7F:53:CB:D6  -68    0 - 1      0        29         JB
(not associated)   C4:4F:33:B2:D8:7A  -86    0 - 1      0         3         JB
(not associated)   72:EC:B4:34:ED:C0  -34    0 - 1      0         6         Frangus  WiFi Clientes,
34:CE:00:51:21:B6  04:E5:98:7F:95:56  -72    0 - 1e     0        60         MEO-WiFi,Frangus  WiFi
34:CE:00:51:21:B6  A8:9C:ED:56:64:1D  -40   24e- 1e   784      4081  EAPOL  MEU,TestingNetworkWIFI
```

Figura 24- Airodump-ng

3.2.2 *Rogue* AP

O ataque de *rogue* AP foi efetuado com recurso ao software "create_ap"[8], no entanto este software não vem pré-instalado no S.O. Kali2021 e, como tal, foram executados os comandos da Tabela 5 para efetuar a sua instalação.

Tabela 5 - Instalar software RogueAP

sudo apt-get install haveged hostapd git util-linux procps iproute2 iw dnsmasq iptables bettercap
git clone https://github.com/oblique/create_ap
cd create_ap
sudo make install
cd .. && rm -rf create_ap

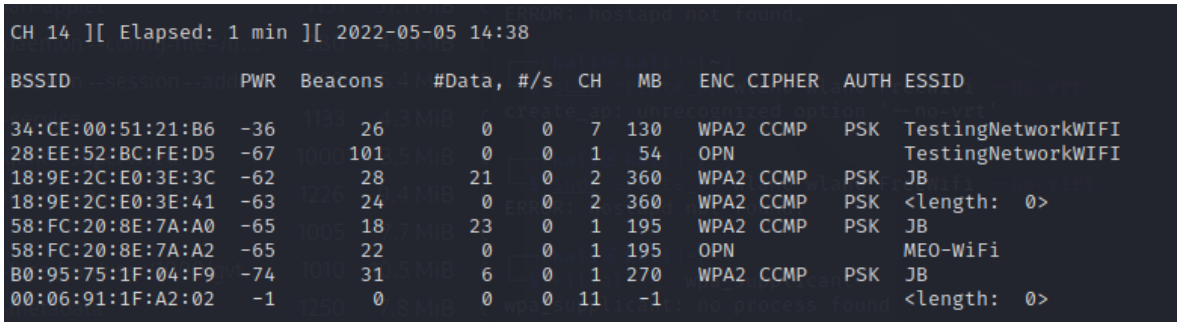
Com o software instalado, foi necessário executar um comando, presente na Tabela 6, para criar um *rogue* AP. Este, contém a placa de WiFi (wlx28ee52bcfed5) a segunda placa de WiFi ou rede onde o tráfego será redirecionado (ens33), juntamente com o nome da rede pretendido. Neste cenário será a TestingNetworkWIFI.

Tabela 6 - Criar RogueAP

sudo create_ap wlx28ee52bcfed5 ens33 TestingNetworkWIFI

[8] Mais informação sobre o software pode se encontrada em https://github.com/oblique/create_ap

Utilizando o software “airodump-ng” é possível detetar que existem duas redes com o mesmo SSID, tal como é visível na Figura 25, nas duas primeiras entradas da tabela.



```
CH 14 ][ Elapsed: 1 min ][ 2022-05-05 14:38

BSSID              PWR  Beacons    #Data, #/s  CH   MB   ENC CIPHER  AUTH ESSID

34:CE:00:51:21:B6  -36       26        0    0   7  130   WPA2 CCMP   PSK  TestingNetworkWIFI
28:EE:52:BC:FE:D5  -67      101        0    0   1   54   OPN              TestingNetworkWIFI
18:9E:2C:E0:3E:3C  -62       28       21    0   2  360   WPA2 CCMP   PSK  JB
18:9E:2C:E0:3E:41  -63       24        0    0   2  360   WPA2 CCMP   PSK  <length:  0>
58:FC:20:8E:7A:A0  -65       18       23    0   1  195   WPA2 CCMP   PSK  JB
58:FC:20:8E:7A:A2  -65       22        0    0   1  195   OPN              MEO-WiFi
B0:95:75:1F:04:F9  -74       31        6    0   1  270   WPA2 CCMP   PSK  JB
00:06:91:1F:A2:02   -1        0        0    0  11   -1                    <length:  0>
```

Figura 25 - Rogue AP Airodump-ng

3.2.3 *MacAddress SPOOF* com *Rogue* AP

Para realizar a duplicação de um *Mac Address* e assim realizar um *MAC Spoof*, foi alterado o *Mac Address* de uma placa WiFi utilizando o comando “macchanger”, Tabela 7. O novo *Mac Address* é de um AP já existente na rede. Foi também utlizado o software Kismet para capturar os dados da rede WiFi.

Tabela 7 - MacChanger

ifconfig wlan0 down
macchanger -m A8:9C:ED:56:64:1D wlan0
ifconfig wlan0 up

Com o dipositivo já com o *Mac Address* alterado, foi utilizado o comando do ponto anterior para criar um novo AP. Assim que o dispositivo com o MAC alterado começa a ser utilizado com AP, a aplicação Kismet identifica este tipo de comportamento, *MAC Spoof*, Figura 26.

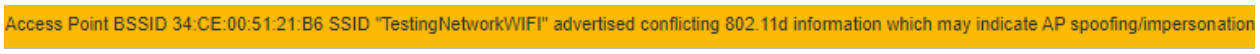


Figura 26 - Kismet *Mac Adress Spoof*

3.3. Métricas a ser analisadas

O Kismet já consegue identificar vários tipos de ataques, no entanto existem alguns que ainda não são possíveis de identificar utilizando apenas os alarmes do Kismet.

Os alarmes implementados pelo Kismet são:

- Ataque de *Deauthenticate*
- *Rogue* AP
- *Mac Address SPOOF* com *Rogue* AP

Existem ainda eventos que não estão contemplados pelo Kismet e que poderiam criar novos alarmes, nomeadamente:

- Novos dispositivos encontrados
- Novas redes WiFi
- Novo dispositivo cliente ligado em rede WiFi conhecida
- Número de pacotes de erro demasiado elevado
- AP com WPS (WiFi Protected Setup) Ativo

Para conseguir identificar os eventos mencionados é necessário analisar algumas métricas disponibilizadas pelo Kismet e posteriormente gerar alarmes dessas mesmas métricas.

No caso de novos dispositivos e também dispositivos em redes conhecidas, foram analisados os seguintes campos:

- *Mac Address*
- Fabricante
- Primeira vez que o dispositivo foi visto
- Última vez que o dispositivo foi visto
- Redes associadas

No que diz respeito às redes WiFi, existem vários campos relevantes, nomeadamente:

- Nome da rede (SSID)
- Frequência
- Força do sinal
- Canal de comunicação
- Total de pacotes
- Pacotes de erro
- Pacotes LLC (Link Layer Control) and Management

- Primeira vez que a rede foi vista
- Última vez que a rede foi vista

De salientar que os dados disponíveis (pelo Kismet) das redes WiFi já contêm o número de pacotes total e o número de pacotes de erro, que vão identificar problemas com a rede ou *device* específico.

Outra métrica importante é o estado do WPS nas várias redes, dado que é enviado individualmente pelo Kismet, específico a cada rede.

Apesar de atualmente não existir nenhum alarme que tenha por base os *beacons* transmitidos, existem alguns valores que no futuro podem ser utilizados para tal, nomeadamente:

- *Beacon* SSID
- Primeira vez que foi visto
- Última vez que foi visto

4. Arquitetura da solução proposta

Após efetuado o levantamento do estado da arte e recolha de inúmera informação sobre o tipo de tráfego relacionado com as redes WiFi, propõe-se uma solução que integre funcionalidades disponibilizadas por algumas das ferramentas avaliadas e que permita a recolha e monitorização de métricas de segurança em redes WiFi.

4.1. Modelo lógico

Foi desenhado e desenvolvido o modelo lógico, que é possível observar na Figura 27, que serviu como guia para a implementação física do projeto.

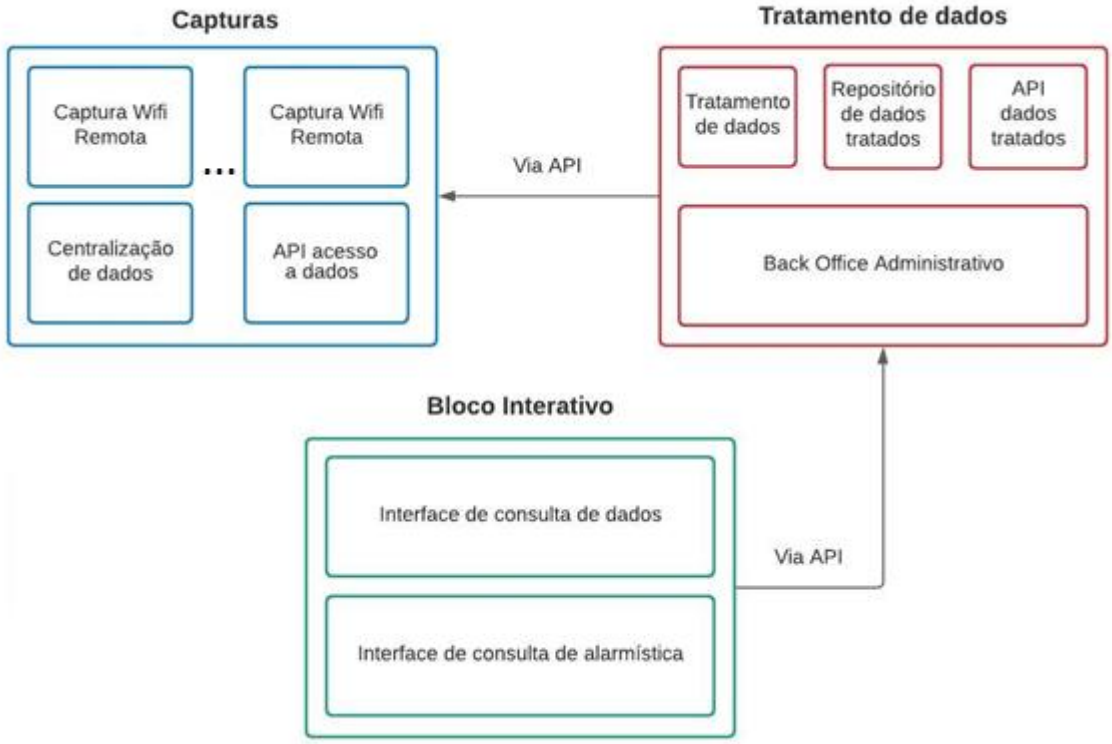


Figura 27 - Modelo lógico

4.2. Descrição modular

Cada um dos módulos apresentados tem uma função específica no modelo, sendo que todos são dependentes uns dos outros e, com tal, a aplicação só pode funcionar em pleno estando todos presentes.

4.2.1 Capturas

O modelo da Figura 27 serve especificamente para capturar dados WiFi, captura esta que pode ser efetuada localmente ou de maneira remota com recurso ao software "Kismet Remote Capture". Internamente está dividido em 3 partes: um bloco de captura de dados, outro bloco de centralização de dados que conta com uma base de dados interna do "Kismet" e, por último, uma API que permite que outras aplicações comuniquem e recolham os dados que se encontram na base de dados do "Kismet".

Cada um dos blocos pertencentes ao módulo de capturas é independente, sendo possível ser composto por múltiplos blocos de capturas remotas.

4.2.2 Tratamento de dados

No módulo de tratamento de dados existem quatro diferentes componentes, iniciando pela parte de tratamento dos dados em si, componente utilizado para tratar os dados recebidos pelas chamadas à API do "Kismet". Posteriormente, os dados tratados são armazenados na componente do repositório. Existe também uma API que permite consultar os dados aí guardados e, por último, o *back office* administrativo tem como propósito guardar e alterar as várias configurações da plataforma.

4.2.3 Bloco Interativo

No último bloco existem duas interfaces que têm contacto direto com o utilizador. Os dados chegam a esta interface devidamente tratados e prontos a serem mostrados ao utilizador final por meio de uma API disponibilizada pelo bloco de tratamento de dados. Estes dados estão divididos em 2 subcategorias, dados de rede WiFi (dispositivos clientes, AP e redes) e toda a componente de alarmística.

4.3. Testes preliminares na implementação do Kismet e utilização da API

Foi realizada a instalação do Kismet numa máquina virtual seguindo as instruções do site oficial: https://www.kismetwireless.net/docs/readme/quickstart/.

Para realização dos testes foram utlizadas várias API's disponibilizadas pelo "Kismet", conseguindo obter dados de dispositivos AP, dispositivos clientes, redes WiFi, *beacons* e alarmes gerados pelo "Kismet". Para informação detalhada sobre os *endpoints* é possível consultar https://www.kismetwireless.net/docs/devel_group.html.

4.3.1 *Endpoint* para obter dados de dispositivos e redes

Foi utilizado um *endpoint* (Tabela 8) para obter os dados de todo o tipo de dispositivos.

Tabela 8 - *Endpoint* obter todos os dispositivos

Tipo de pedido: Get Endereço: https:/IP/devices/views/all/devices.json

Com este pedido foi possível obter uma variedade de dados, tal como é visivel na Figura 28.


```
[
    {
        "kismet.device.base.first_time": 1639861878,
        "kismet.device.base.macaddr": "58:FC:20:8E:7A:A0",
        "kismet.device.base.crypt": "WPA2-PSK",
        "kismet.device.base.key": "4202770D00000000_A07A8E20FC58",
        "kismet.device.base.packets.error": 0,
        "kismet.device.base.packets.total": 3023,
        "kismet.device.base.manuf": "Altice Labs S.A.",
        "kismet.device.base.basic_type_set": 13,
        "dot11.device": {
            "dot11.device.wps_m3_count": 0,
            "dot11.device.client_disconnects": 0,
            "dot11.device.num_responded_ssids": 1,
            "dot11.device.probe_fingerprint": 0,
            "dot11.device.typeset": 259,
            "dot11.device.client_disconnects_last": 0,
            "dot11.device.last_sequence": 0,
            "dot11.device.response_fingerprint": 0,
            "dot11.device.client_map": {
                "18:9E:2C:E0:3E:40": {
                    "dot11.client.decrypted": 0,
```


Figura 28 - Reposta Kismet todos os dispositivos

Uma vez que é retornada demasiada informação, é possível fazer um pedido para um dispositivo específico, Tabela 9, usando um id gerado pelo Kismet, denominado de *kismet.device.base.key* (exemplo de um pedido no anexo A).

Tabela 9 - Endpoint Api Kismet um dispositivo apenas

Tipo de pedido: GET Endereço: https:/IP/devices/by-key/{{deviceID}}/device.json

Neste *endpoint* os dados extraídos são os mesmos que no *endpoint* anterior, mas apenas para um dispositivo e não para todos os existentes na rede. É possível então extrair os seguintes dados:

- Kismet ID do dispositivo
- Tipo de dispositivo
- *Mac Address*
- Quando foi visto pela primeira vez
- Quando foi visto pela última vez

É possível também obter dados relativos a redes neste pedido, nomeadamente:

- O nome da rede
- Frequência da rede
- Força do sinal WiFi
- Canal da rede WiFi
- Total de pacotes
- Pacotes com erro
- Pacotes de LLC (Management)
- Primeira vez que a rede foi vista
- Última vez que a rede foi vista

Caso seja um dispositivo cliente, é ainda possível observar o número de vezes que o dispositivo se tentou ligar à rede.

4.3.2 Obter dados de *Beacons*

Os dispositivos clientes transmitem *beacons* antes de se ligarem a uma rede WiFi. É possível então extrair essa informação do Kismet utilizando um *endpoint* onde apenas os dispositivos clientes estão disponíveis (Tabela 10).

Tabela 10 - Pedido Kismet todos os dispositivos clientes

devices/views/all/devices.json body:{“regex”: ["kismet.device.base.type", "Wi-Fi Device"]}

No corpo da mensagem dos pedidos (body) existe um campo onde é possível especificar o tipo de pedido que pretendemos. Neste cenário foi especificado o tipo de dispositivo que se pretende encontrar, *Wifi-Device*, dispositivos clientes. Com este pedido foi possível obter os seguintes dados:

- *Beacon SSID*
- Primeira vez que o *beacon* foi visto
- Última vez que o *beacon* foi visto
- Modelo do dispositivo (quando disponível)

4.3.3 Obter dados de alarmes

O Kismet dispõe de um sistema de alarmes que são categorizados por severidade, tal como é visível na Tabela 11:

Tabela 11- Severidade dos alarmes do Kismet

SEVERIDADE	DEFINIÇÃO	USO
0	INFO	Informação de alarmes e erros
5	LOW	Alarmes de risco reduzido
10	MEDIUM	Risco médios como tentativas de *Denial-of-service*
15	HIGH	Risco elevado, ataques severos de *Denial-Of-Service*, *deauthenticate*, entre outros.
20	CRITICAL	Erros críticos, como *exploits* a serem utilizados na rede.

Utilizando um *endpoint* específico para os alarmes, Tabela 12, foi possível obter várias informações relevantes.

Tabela 12 - Pedido Kismet todos os alarmes

Tipo de pedido: GET Endereço: http://ip/alerts/all_alerts.json

A resposta obtida contém a seguinte informação Figura 29.

```
[
    {
        "kismet.alert.hash": 2322863616,
        "kismet.alert.severity": 5,
        "kismet.alert.frequency": 0,
        "kismet.alert.dest_mac": "10:13:31:CC:DD:D9",
        "kismet.alert.text": "IEEE80211 network BSSID 10:13:31:CC:DD:D9 client 40:24:B
        "kismet.alert.class": "SPOOF",
        "kismet.alert.other_mac": "00:00:00:00:00:00",
        "kismet.alert.channel": "0",
        "kismet.alert.location": {
            "kismet.common.location.geopoint": [
                0,
                0
            ]
        },
        "kismet.alert.phy_id": 0,
        "kismet.alert.device_key": "00_0",
        "kismet.alert.transmitter_mac": "10:13:31:CC:DD:D9",
        "kismet.alert.timestamp": 1643834862.129082,
        "kismet.alert.header": "NOCLIENTMFP",
        "kismet.alert.source_mac": "40:24:B2:B0:3D:A4"
    },
```


Figura 29 - Respostas API Kismet Alarmes

Para os testes em questão foram extraídas as seguintes informações dos alarmes do Kismet:

- Descrição
- Tipo
- Severidade
- Transmissor
- *Mac Address* que originou o alarme
- *Mac Address* destino, se existir
- Cabeçalho do alarme
- Data em que o alarme foi gerado

4.4.Base de dados

Com vista a armazenar a informação tratada obtida das chamadas a API do kismet, foi criada uma base de dados relacional com recurso ao *MySQL* onde é possível guardar todas as informações relevantes para o projeto, de forma a poder consultá-las sempre que for necessário. É possível observar os esquemas desta base de dados na Figura 30.

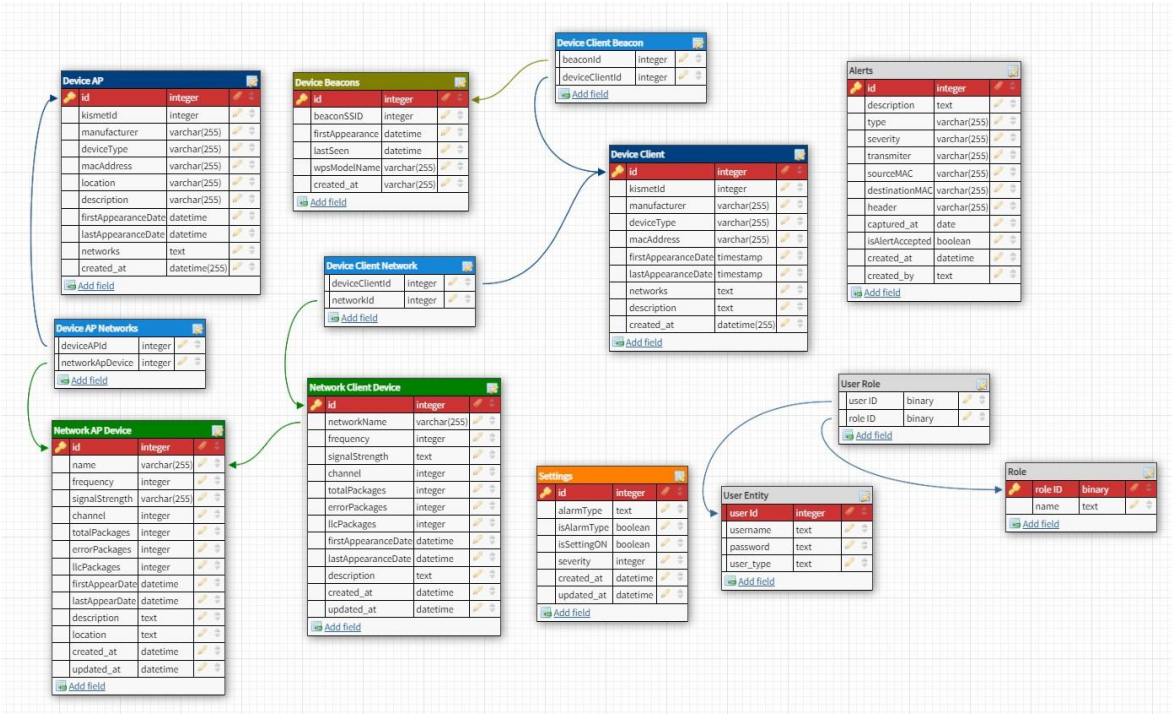


Figura 30 - Esquema da Base de dados

4.4.1 Tabelas da base de dados

A base de dados está dividida entre várias tabelas ligadas entre si, nomeadamente:

- *Device* AP

  Nesta tabela constam todos os dados relevantes para a identificação dos dispositivos *AccessPoint*/Router, tal como *Mac Address* do *device* ou *Kismet ID*, bem como as datas que permitem verificar quando o dispositivo foi visto pela primeira e última vez.

- *Network* AP *Device*

  O mapeamento das redes WiFi que são disponibilizadas pelos *Access Points* são guardados nesta tabela, sendo que existe uma tabela intermédia "Dispositivo AP *Networks*" que faz a ligação entre os dispositivos *AP* e as suas redes. São

guardados vários dados relativos a cada rede, tal como nome, canal, frequência, o número de pacotes que passam pela rede, entre outros.
Todos estes dados são revelantes para realizar uma análise detalhada à rede.

- *Device Client*
  Esta tabela guarda os dados dos dispositivos clientes encontrados na rede. Assim que um novo dispositivo seja encontrado, ele é adicionado à tabela, mesmo que este não se conecte a uma rede Wifi. São guardados dados que identifiquem o dispositivo tal como *Mac Address* ou Kismet ID.

- *Device Beacons*
  Os *beacons* são informações valiosas dos dispositivos clientes e, como tal, são guardados na base de dados, tendo uma tabela intermédia "*Device Client Beacon*" para fazer a ligação entre os *beacons* e o dispositivo cliente que os emitiu. É guardada informação sobre o SSID do *beacon*, quando foi visto pela primeira e última vez, entre outras informações relevantes.

- *Network Client Device*
  Nesta tabela são guardadas informações inerentes às redes em que os dispositivos clientes se ligaram. Contem informações relevantes da rede, tal como o nome da rede, a frequência, a força do sinal, o número de vezes que o dispositivo se tentou ligar, o número total de pacotes e o número de pacotes com erro, entre outras informações necessárias para uma análise a esta ligação.
  Se um dispositivo cliente se ligar a várias redes, é possível que tenha várias entradas na tabela *network* para o mesmo dispositivo. Para fazer este mapeamento existe a tabela intermédia "*Device Client Network*" que relaciona *Device Client* e *Network Client Device*.

- *Alerts*
  Os alarmes são parte fundamental desta aplicação e, como tal, foi criada uma tabela para guardar este tipo de eventos, sejam eles despoletados pelo "Kismet" ou pelo software que faz a análise dos dados recebidos pelo "Kismet". Os dados

guardados permitem identificar quem gerou aquele alarme, a origem do problema bem como a sua severidade.

- *Settings*
  Existem vários tipos de *settings* que podem ser configuradas pelos administradores, nomeadamente se pretendem vários tipos de alarmes e também as suas severidades. É nesta tabela que toda essa informação é guardada.

- *User Entity* e *Role*
  Estas duas tabelas guardam informação sobre o utilizador para ser utilizada na autenticação. A informação sobre a password está encriptada através da utilização do *bcrypt*. O relacionamento entre estas duas tabelas é feito com recurso à tabela "User Role".

Existem ainda alguns campos, como o *created_at*, que indicam a data de criação dos dados na tabela, que são transversais a todas as tabelas, com exceção das tabelas intermédias. Nas tabelas de redes e dispositivos existem ainda dois campos que indicam quando aquela rede/dispositivo foi vista pela primeira e pela última vez, conseguindo assim um controlo sobre quando aquele evento aconteceu ou se ainda está a decorrer.

Para garantir que os dados se encontram atualizados na base de dados, foram efetuados pedidos recorrentes à API do "Kismet", atualizando a informação de forma constante. A periocidade dos pedidos poderá ser definida no painel de administração da solução.

4.5.Popular Base de dados

A base de dados é preenchida com os dados que foram capturados pelo "Kismet", alarmes e *settings*. Para obter os dados oriundos do "Kismet" irão ser utilizadas várias chamadas à sua API. Todos os pedidos são efetuados utilizando um *header* com um *token* de autenticação, para garantir que apenas utilizadores autenticados conseguem efetuar pedidos. Relativamente aos alarmes, eles podem ser criados de duas maneiras: com os dados recebidos do "Kismet" ou por funções da solução a desenvolver. Por último, as

settings serão geridas pelo utilizador superAdmin com chamadas à API da solução proposta.

São utilizados, para acesso aos dados oriundos do "Kismet", vários *endpoints* dependendo da informação que é pretendida, nomeadamente:

- Obter todos os dados dos dispositivos AP e as suas *networks*, Tabela 13:

Tabela 13 - Pedido API dados de dispositivos AP

Tipo de pedido: GET Endereço: http://{{host}}/devices/views/phydot11_accesspoints/devices.json

- Obter todos os dados de dispositivos cliente, as suas *Networks* e *Beacons* associados, Tabela 14:

Tabela 14 - Pedido API dados de dispositivos *c*liente

Tipo de pedido: Post Endereço: http://{{host}}/devices/views/all/devices.json Body: {"regex": ["kismet.device.base.type", "Wi-Fi Client"]}

- Obter os *alerts* gerados pelo Kismet, Tabela 15:

Tabela 15 - Pedido API dados Alerts

Tipo de pedido: Get Endereço: http://{{host}}/alerts/last-time/1643833340/alerts.json

Após o pedido à API, as respostas vêm formatadas em *JSON* (Figura 31), sendo necessário utilizar um software para as converter em objetos e, só depois disso, as guardar na base de dados.

```
[
    {
        "kismet.device.base.first_time": 1639861878,
        "kismet.device.base.macaddr": "58:FC:20:8E:7A:A0",
        "kismet.device.base.crypt": "WPA2-PSK",
        "kismet.device.base.key": "4202770D00000000_A07A8E20FC58",
        "kismet.device.base.packets.error": 0,
        "kismet.device.base.packets.total": 3023,
        "kismet.device.base.manuf": "Altice Labs S.A.",
        "kismet.device.base.basic_type_set": 13,
        "dot11.device": {
            "dot11.device.wps_m3_count": 0,
            "dot11.device.client_disconnects": 0,
            "dot11.device.num_responded_ssids": 1,
            "dot11.device.probe_fingerprint": 0,
            "dot11.device.typeset": 259,
            "dot11.device.client_disconnects_last": 0,
            "dot11.device.last_sequence": 0,
            "dot11.device.response_fingerprint": 0,
            "dot11.device.client_map": {
                "18:9E:2C:E0:3E:40": {
                    "dot11.client.decrypted": 0,
```


Figura 31 - Resposta da API JSON

Foi desenvolvida uma aplicação em Java com o objetivo de fazer os pedidos à API do "Kismet" e criado um método, com o auxílio da biblioteca Jackson[9], para efetuar a conversão do formato JSON (reposta do Kismet) para um objeto em Java (ou mais que um, caso seja necessário) e guardar os valores desse objeto na base de dados.

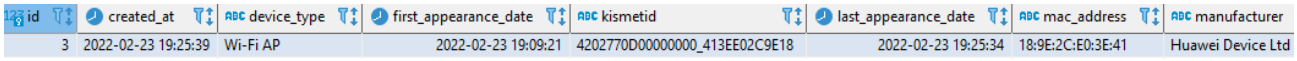
Na Tabela 16 é apresentada a função utilizada para a conversão de JSON para um objeto de Java, nomeadamente um dispositivo AP e as suas redes, que também são objetos Java. O código interno da função foi retirado por motivos da sua extensão, no entanto é possível consultar todo o código da função, que está disponível no anexo C.

Tabela 16 - Conversão JSON para Objeto Java

```
public DeviceAP JSONToDeviceAP(String response) {
....
    return deviceAP;
}
```

[9] Mais informação sobre esta biblioteca pode ser encontrada em [18]

Assim que a conversão para o objeto Java é finalizada, ele guarda os valores na base de dados. Neste exemplo, tratando-se de um dispositivo AP, seria adicionada uma linha na tabela dos dispositivos AP, tal como exemplifica a Figura 32.



id	created_at	device_type	first_appearance_date	kismetid	last_appearance_date	mac_address	manufacturer
3	2022-02-23 19:25:39	Wi-Fi AP	2022-02-23 19:09:21	4202770D00000000_413EE02C9E18	2022-02-23 19:25:34	18:9E:2C:E0:3E:41	Huawei Device Ltd

Figura 32 - Nova entrada base de dados *Device* AP

Outro exemplo são os *alerts* (Tabela 17), onde é convertido o JSON da resposta em *alert* e guardado na DB (Database). O código da função pode ser consultado no anexo D.

Tabela 17 - Conversão JSON para Objeto Java

```
public static Alert toKismetAlert(String response) {
   ....
}
```

Estes pedidos são realizados recorrentemente, dependendo do valor configurado pelo administrador. Sempre que o Java trata a resposta da API do "Kismet", confirma se aqueles dados já se encontrem na base de dados. Caso isto seja verdade, ele descarta os dados do pedido, mas, caso sejam dados novos ou atualizados, eles serão introduzidos na base de dados.

A comunicação entre a aplicação Java com Spring Boot[10] e a base de dados em *MySQL* é efetuada utilizando o *object-realtion mapping*(ORM) JPA (Java Persistence API) juntamente com *Hibernate*[11].

Para além da conversão dos *endpoints* mencionada acima, foram ainda utilizados vários *endpoints* para obter diversas informações, descritas no ponto 4.3.

A aplicação desenvolvida em Java que efetua a comunicação com o *Kismet* encontra-se referenciada neste capítulo de arquitetura proposta uma vez que foi necessária para testar as várias *API's* disponibilizadas pelo *Kismet*.

[10] Mais informação sobre estas tecnologias pode ser encontrada em [19]
[11] Mais informações sobre *Hibernate* pode ser encontrada em [20]

5. Implementação

Foi implementado um sistema utilizando o Kismet com base nos testes realizados no ponto 4.3, que consiste num servidor "Kismet" e dois dispositivos de captura remota, juntamente com um servidor de Java (*Tomcat*), uma base de dados *MySQL* e o servidor Web *Apache*.

5.1. Implementação física

A implementação física foi desenvolvida com recurso à virtualização, recorrendo ao suporte *VMware Workstaion Player*[12], conseguindo assim, com apenas uma máquina física, virtualizar os vários servidores necessários para a implementação deste projeto.

No anexo E encontra-se um guia de instalação com todos os passos necessários para replicar este cenário.

5.1.1 Esquema físico

Foi criado o esquema físico apresentado na Figura 33 para o desenvolvimento deste projeto.

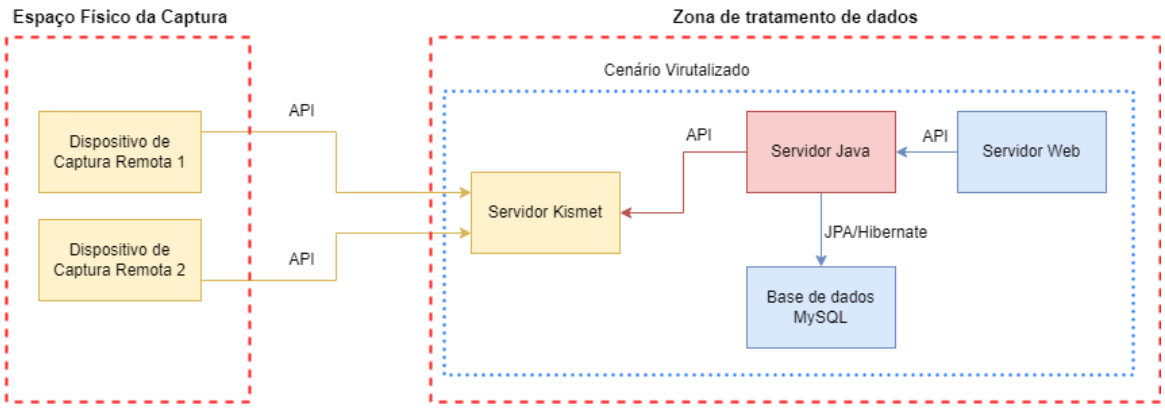


Figura 33 - Esquema Físico

O esquema da Figura 33 está dividido em duas partes distintas: o espaço físico de captura e a zona de tratamento de dados. Como a comunicação entre ambas as partes é efetuada por uma API, os dois componentes podem estar separados geograficamente e utilizar a web para se comunicar, garantindo assim capturas completamente remotas. Para

[12] Informação sobre VMware pode ser encontra em [21]

ter uma garantia de segurança na comunicação poderia existir uma VPN (*Virutal Private Network*) para interligar ambas as partes.

O espaço de captura consiste em duas máquinas com o único propósito de captura de dados WiFi, não necessitando de muito poder computacional e permitindo a utilização de minicomputadores ou até de um Raspberry pi, desde que a placa de rede WiFi tenha a possibilidade de ser colocada em modo monitor. Neste cenário, foi testado um sistema misto entre virtualização e *bare-metal*, utilizando um dos servidores de captura com uma máquina virtual e outro com um servidor físico. Ambas os servidores utilizaram Ubuntu 20.04 e o software do Kismet Capture. Assim que a ligação ao servidor é efetuada com sucesso, é exibida uma mensagem de capture (Figura 34).

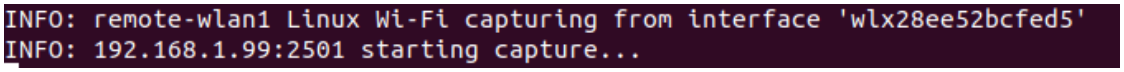


Figura 34 - Captura Remota

A zona de tratamento de dados está dividida em 2 subcomponentes: um servidor de "Kismet", utilizado para receber os dados dos dispositivos de captura remota, e um servidor em Java que manipula e gere os dados do "Kismet", utilizando uma base de dados local para o seu armazenamento.

O servidor de "Kismet", por sua vez, está alojado numa máquina virtual com SO (Operating System) Ubuntu 20.04, sendo o ponto onde os vários dispositivos de captura remota se vão ligar para enviar informação sobre dos dados WiFi que capturaram, tal como é visível na Figura 35.

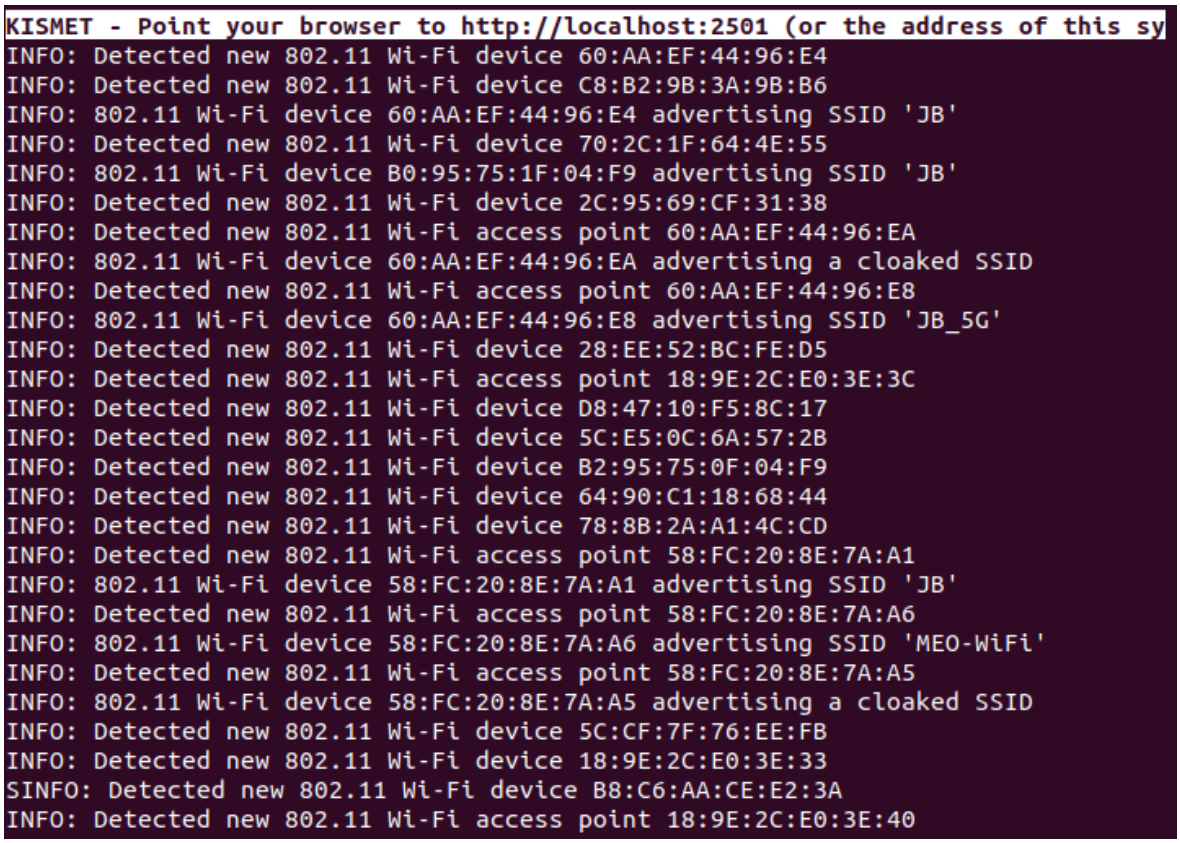


Figura 35 - Kismet Servidor

Neste cenário existem apenas dois dispositivos de captura remota. No entanto, podem ser adicionamos mais dispositivos utilizando apenas um servidor central. Por defeito, o ponto de comunicação entre os dispositivos remotos e o servidor é o 2501, podendo este ser alterado.

A aplicação em Java utiliza uma máquina virtual com o SO Ubuntu 20.04 e o *Tomcat* como servidor. Esta máquina realiza os pedidos ao servidor do Kismet com recurso a uma API disponibilizada pelo mesmo, transformando os dados recebidos em formato *JSON* para objetos em java e, posteriormente, guardando-os numa base de dados *MySQL*. A comunicação com a base de dados é efetuada com recurso ao JPA/*Hibernate* utilizado pelo Java.

A base de dados utiliza a mesma máquina virtual que o servidor de Java por razões de segurança e performance, mas existe a possibilidade da base de dados se encontrar remota.

O servidor Web trata toda a parte de GUI (Graphical User Interface) da web necessário para os utilizadores conseguirem utilizar a aplicação, tanto *front-office* como *back-office*. O Software de *front-office* e *back-office* foi desenvolvido com recurso à

framework VUE.js. A comunicação entre o *font-end* (VUE) e o *back-end* (Java) é feita utilizando uma API criada para tal.

5.2. Comunicação entre Java e Kismet

A comunicação entre a aplicação Java e o "Kismet Server" é efetuada através de endpoints do Kismet já anteriormente mencionados. O método *getDevicesAndAlerts*, que se encontra na Tabela 18, faz essa orquestração de pedidos, chamando no seu interior 3 outros métodos:

- "getAllApDevicesResponse()", devolve todos os *Access Points* reconhecidos pelo Kismet desde o último pedido efetuado.
- "getAllClientDevicesResponse()", devolve todos os dispositivos clientes reconhecidos pelo Kismet desde o último pedido efetuado.
- "getAllAlertsResponse()", devolve todos os alarmes gerados desde o último pedido.

Todos estes métodos são chamados a cada X segundos, sendo X uma *setting* definida pelo utilizador, identificada na Tabela 18 como "*timeToDoRequestsInMilliseconds"*.

Tabela 18 - Código comuinicação Java - Kismet

```
@PostConstruct
   public void getDevicesAndAlerts() throws JsonProcessingException,
UnsupportedEncodingException {
      int timeToDoRequestsInMilliseconds = 0;
      Setting settingTimeBetweenRequest = settingService.getSettingByType("REQUESTS
TIME IN SECONDS");
      if (settingTimeBetweenRequest == null ||
!settingTimeBetweenRequest.getIsSettingOn()) {
         timeToDoRequestsInMilliseconds = kismetTimeBetweenRequests * 10000;
      } else {
         timeToDoRequestsInMilliseconds = settingTimeBetweenRequest.getSeverity() * 1000;
      }
      try {
```

```
        Timer timer = new Timer();
        timer.schedule(new TimerTask() {
            @SneakyThrows
            @Override
            public void run() {
                System.out.println("#Inicio request");
                String apResponseBody = GetAllApDevicesResponse();
                if (StringUtils.isNotEmpty(apResponseBody)) {
                    jsonResponseToDevicesAP.JSONToDevicesAP(apResponseBody);
                }
                String clientsResponseBody = GetAllClientDevicesResponse();
                if (StringUtils.isNotEmpty(clientsResponseBody)) {
                    jsonResponseToDevicesClient.JSONToDevicesClient(clientsResponseBody);
                }
                String AlertsResponseBody = GetAllAlertsResponse();
                if (StringUtils.isNotEmpty(AlertsResponseBody)) {
                    jsonResponseToKismetAlert.JSONToAlerts(AlertsResponseBody);
                }
                System.out.println("#Fim request  " + new Date());
            }
        }, 0, timeToDoRequestsInMilliseconds);
    } catch (Exception e) {
        e.printStackTrace();
        LOGGER.warning("Exception on Starter " + Instant.now());
        getDevicesAndAlerts();
    }
}
```

Caso exista alguma exceção, como um *timeout* se o servidor não responder atempadamente ou uma falha nas credenciais de acesso ao "Kismet", elas são devidamente tratadas para garantir que a aplicação não para de funcionar de maneira inesperada. Todo este processamento é efetuado numa *thread* específica, separada da *thread* principal do software, para garantir paralelismo na aplicação, podendo esta estar a

responder a pedidos do *Front-end* e ao mesmo tempo efetuar e processar pedidos ao servidor do Kismet.

5.3. Interação com o utilizador

Foi desenvolvida uma Graphical User Interface (GUI) através de uma aplicação baseada em web para proporcionar a interação dos utilizadores com a solução desenvolvida.

Esta abordagem permite ter uma GUI transversal a vários tipos de plataformas/Sistemas Operativos. Para a sua criação foi utilizada a *framework* VUE.js que, por sua vez, consome dados de uma API desenvolvida em Java.

No anexo B existe um guia de funcionalidades da aplicação onde podem ser consultadas em pormenor todas as funcionalidades da aplicação.

5.3.1 Tipo de Utilizadores

Na plataforma existem 3 tipos de utilizadores:

- Não autenticados - estes utilizadores apenas podem aceder à página de log in, Figura 36. Tendo em conta o objetivo e sensibilidade da informação obtida, o acesso à mesma é restrita a pessoal devidamente autenticado;
- Admin - depois de efetuar o log in estes utilizadores têm acesso a todos os dados do *font-end*;
- SuperAdmin - utilizador com maior nível de acesso uma vez que, depois de efetuado o seu log in, pode aceder tanto ao *front-end* como ao *back-end*, podendo assim mudar algumas funcionalidades da plataforma no menu de *settings*.

Figura 36 - Página Log in

5.3.2 Front-end

Tal como mencionado anteriormente, foi utilizada a *framework* VUE.js para desenvolver a plataforma, que está dividida em várias secções. Todas as secções do *front-end* estão disponíveis para todos os utilizadores autenticados, não fazendo distinção entre *admin* e *SuperAdmin*.

5.3.3 Dashboard

O *dashboard*, Figura 37, é o ecrã para onde o utilizador é redirecionado quando efetua o log in com sucesso e onde é possível visualizar os seguintes dados:

- Número total de dispositivos;
- Número de redes encontradas;
- Número total de dispositivos clientes;
- Quantidade de *beacons* de todos os dispositivos Clientes;
- Número total de dispositivos AP;
- Número total de alarmes da plataforma.

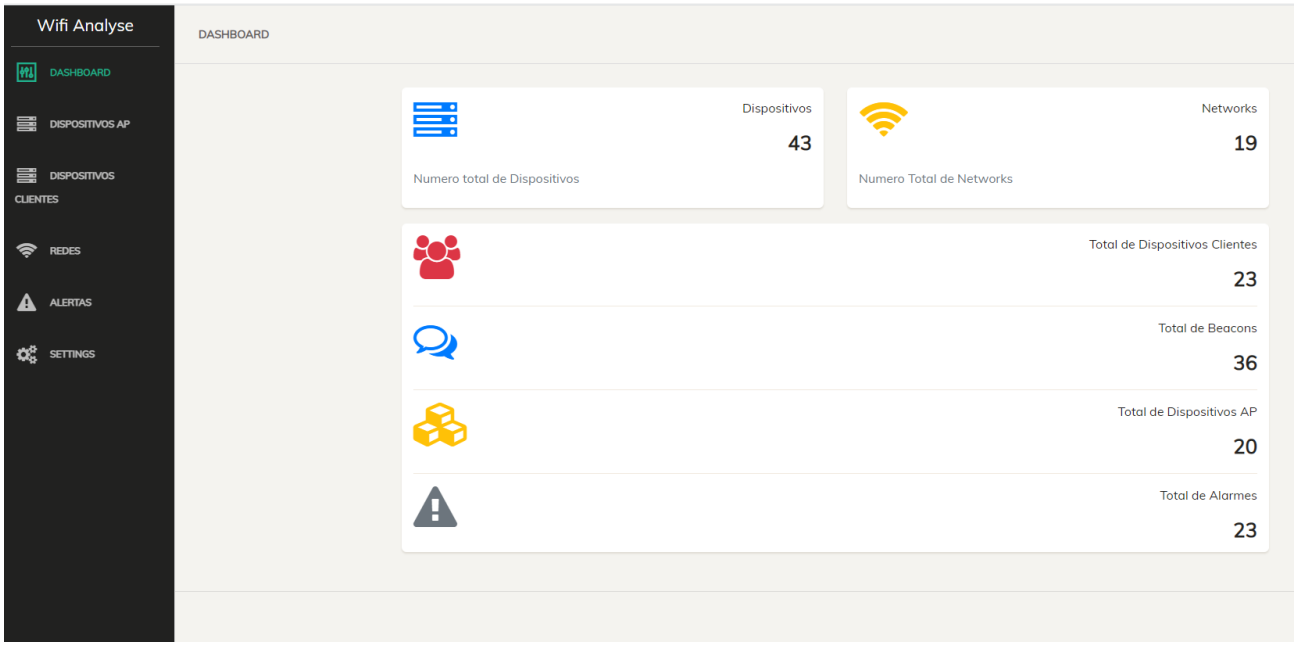


Figura 37 - Front-end Dashboard

Existe também um menu de navegação lateral onde é possível percorrer pelas várias funcionalidades da plataforma, sendo que a opção de "*Settings*" apenas está disponível se o utilizador for do tipo *SuperAdmin*.

5.3.4 Alertas

A alarmística é uma componente fundamental neste projeto, pois é assim que o utilizador da aplicação percebe o que está a acontecer na sua rede. Existe um componente denominado por Alarmes que mostra todos os alarmes da plataforma em forma de tabela (Figura 38), onde é possível verificar se um determinado alerta já foi validado pelo utilizador.

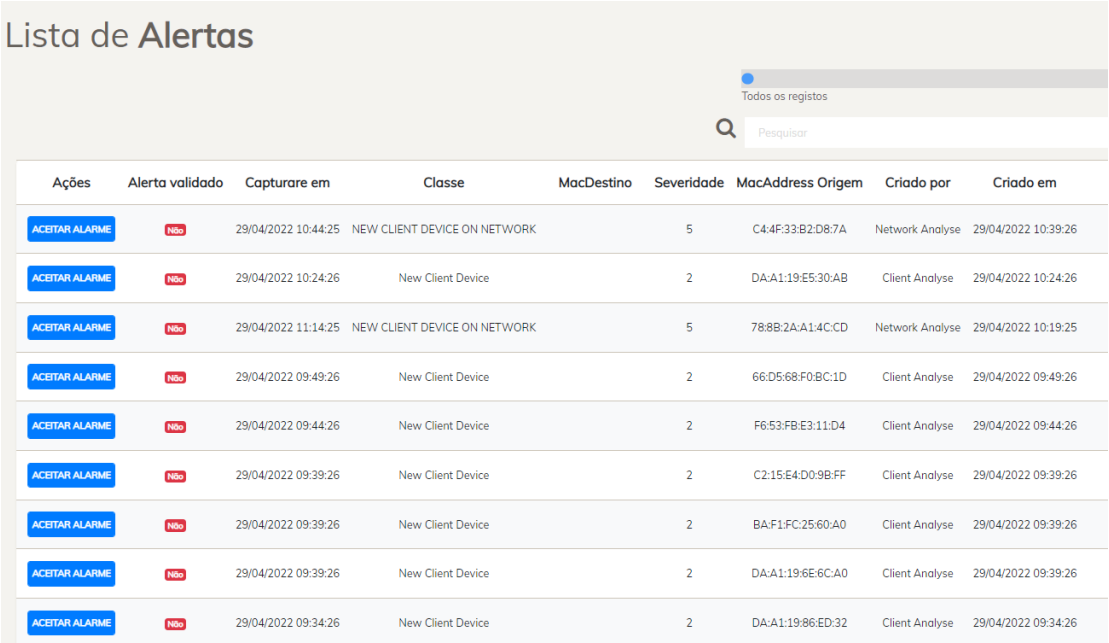


Figura 38 - Front-end listar alarmes

5.3.5 Back-office

O *back-office* destina-se exclusivamente a utilizadores do tipo *SuperAdmin*. Apenas estes utilizadores conseguem ver o menu de *settings* que se encontra no barra lateral, Figura 39, que permite o acesso ao mesmo.

Acedendo a este menu de *back-office*, é possível alterar configurações de sistema, nomeadamente o tempo entre pedidos ao servidor do "Kismet", ou se um determinado evento é alarme, por exemplo, quando um novo dispositivo cliente é encontrado na rede WiFi.

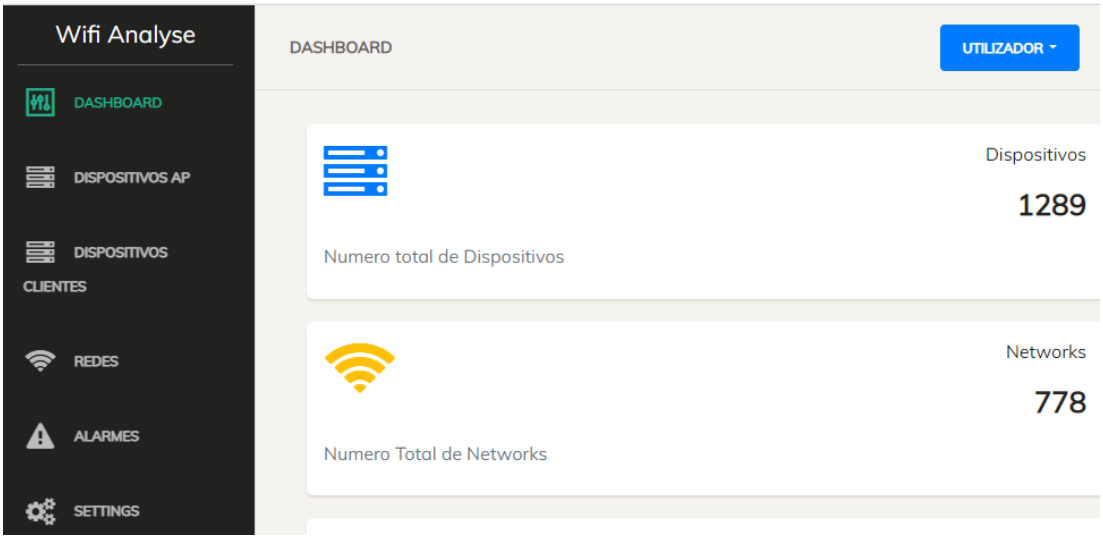


Figura 39 - Barra Lateral com opção settings

5.3.6 Settings

O componente das *settings* guarda informação relevante a toda a aplicação. Na Figura 40 é possível visualizar algumas das *settings* mais importantes, tal como o tempo entre os pedidos ao servidor do "Kismet".

Lista de *settings* disponíveis:

- *DENIAL*
- *SPOOF*
- *NEW NETWORK*
- *NEW CLIENT DEVICE*
- *NEW AP DEVICE*
- *NEW CLIENT DEVICE ON NETWORK*
- *ERROR PACKAGES TOO HIGH*
- *WPS ACTIVE*
- *REQUESTS TIME IN SECONDS*
- *NUMBER ERROR PACKAGES ALARM*
- *SEND EMAILS WITH ALERTS BIGGER THEN*
- *LIST OF EMAILS*
- *SEND SMS WITH ALERTS BIGGER THEN*
- *LIST OF SMS NUMBERS*

DESATIVAR	Sim	Sim	DENIAL	10
DESATIVAR	Sim	Sim	SPOOF	5
DESATIVAR	Sim	Sim	NEW NETWORK	2
DESATIVAR	Sim	Sim	NEW CLIENT DEVICE	2
DESATIVAR	Sim	Sim	NEW AP DEVICE	2
DESATIVAR	Sim	Sim	NEW CLIENT DEVICE ON NETWORK	5
DESATIVAR	Sim	Sim	ERROR PACKAGES TOO HIGH	6
DESATIVAR	Sim	Sim	WPS ACTIVE	1
DESATIVAR	Sim	Não	NUMBER ERROR PACKAGES ALARM	50
ATIVAR	Não	Não	SEND EMAILS WITH ALERTS BIGGER THEN	6
DESATIVAR	Sim	Não	LIST OF EMAILS	
ATIVAR	Não	Não	SEND SMS WITH ALERTS BIGGER THEN	9
DESATIVAR	Sim	Não	LIST OF SMS NUMBERS	
DESATIVAR	Sim	Não	REQUESTS TIME IN SECONDS	60

Figura 40 – *Back-office* lista de *settings*

Todos os alarmes podem ser manipulados, indicando se se pretende ou não que determinado evento crie um novo alarme ou a severidade do mesmo. A severidade de um evento poderá ser mais ou menos relevante, dependendo dos parâmetros de segurança de cada organização.

5.4. Implementação de alarmísticas

Foram implementados vários alarmes de modo que o utilizador tenha rapidamente acesso a informação relevante sobre a sua rede.

5.4.1 Alarmística através de e-mail

Existe a possibilidade de se receberem alarmes por e-mail se essa *setting* estiver ativa. Neste projeto foi utilizado o servidor de mail mailtrap.io[13] para que seja possível enviar e consultar os e-mails de maneira simulada (Figura 41).

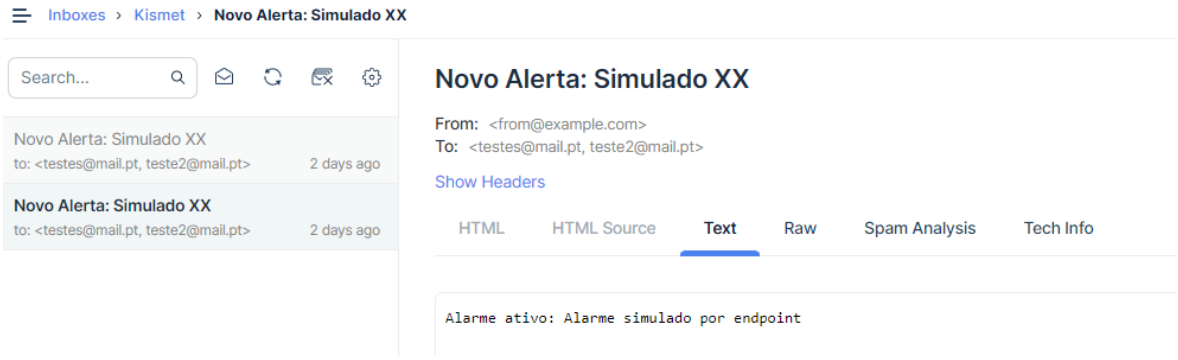


Figura 41 - Email Mailtrap

A comunicação com este Provider de e-mail é feita através de uma API disponibilizada pelo mailtrap.io para esse efeito. O código que efetua essa chamada encontra-se na tabela Tabela 19:

Tabela 19 – Código para envio de email

```
    public void sendEmailFromAlarm(String sendTo, String alertType, String messageBody) {
        messageBody = "Alarme ativo: " + messageBody;
        String subject = "Novo Alerta: " + alertType;
        sendEmail(sendTo, subject, messageBody);

    }

public class EmailSender {
…
            Message message = new MimeMessage(session);
            message.setFrom(new InternetAddress(from));
            message.setRecipients(Message.RecipientType.TO,
                    InternetAddress.parse(to));
            message.setSubject(subject);
            message.setText(messageBody);
```

[13] Para mais informações pode consultar [20]

```
        // Send Email
        Transport.send(message);
…
}
```

5.4.2 Alarmística através de SMS (Short Message Service)

O envio de SMS é uma das funcionalidades implementadas que permite ao utilizador receber uma mensagem quando existir um alarme que tenha uma severidade superior a um valor pré-definido nas *settings*.

Neste projeto só é possível enviar SMS para números validados no software Twilio[14] (e apenas para um) por se tratar de uma opção gratuita (Figura 42).

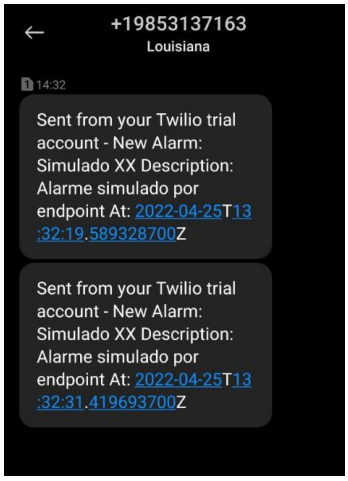


Figura 42 - Alarme SMS

A comunicação com o Twilio é efetuada através de uma API e o código desenvolvido para essa comunicação pode ser consultado na Tabela 20.

Tabela 20 - Código para envio de SMS

```
public class SMSSender {


   @Value("${sms.twilio.account_sid}")
   private String ACCOUNT_SID;
```

[14] Mais informações podem ser encontradas em [23]

```
    @Value("${sms.twilio.auth_token}")
    private String AUTH_TOKEN;

    public void sendSMS(String sendTo, String messageBody) {
        Twilio.init(ACCOUNT_SID, AUTH_TOKEN);
    }

    public void sendSMSForAlarm(String sendTo, String subject, String messageBody) {
        StringBuilder message = new StringBuilder().append("New Alarm: ");
        message.append(subject);
        message.append(" Description: ").append(messageBody);
        message.append(" At: ").append(Instant.now());
        sendSMS(sendTo, message.toString());
    }
}
```

5.4.3 Alarmes nativos do Kismet

O Kismet já tem no seu serviço alguns alarmes relevantes que foram utilizados pela aplicação em Java. De 10 em 10 minutos (ou outro tempo configurável), quando o servidor Java faz um pedido ao servidor do "Kismet", são devolvidos os vários alarmes gerados pelo "Kismet". Assim que o pedido é recebido, é confirmado que esse alarme foi gerado nos últimos minutos para não processar os dados se estes forem antigos. É sempre verificado o valor da *setting* desse alarme específico e, caso esteja ativo, o mesmo é guardado na base de dados. Caso contrário, o alarme é descartado. Parte deste desenvolvimento pode ser visto na Tabela 21.

Tabela 21 - Novo Alarme Kismet

```
if (alert.getCaptured_at().getEpochSecond() + kismetTimeBetweenRequests + 120 >
Instant.now().getEpochSecond()) {
        alert.setIsAlertAccepted(false);
        if (settingService.isSettingAlarmEnabled(alert.getClassType())) {
            Optional<Alert> isNewAlert = this.isNewAlertOrUpdate(alert);
            if (isNewAlert.isEmpty()) {
```

```
            return this.saveAlert(alert);
        } else {
            alert.setId(isNewAlert.get().getId());
            this.saveAlert(alert);
        }
    }
}
```

Os alarmes mais relevantes que são gerados pelo servidor do "Kismet" são:

- *SPOOF* e *Rogue* AP

O "Kismet" consegue perceber quando um dispositivo tenta fazer *SPOOF* ao *Mac Address* de um *Access Point* (Figura 43). Existe uma configuração específica no "Kismet", denominada por "CHANCHANGE*"*, que gera alarmes para *SPOOF*. É possível ver mais sobre este tipo de comportamento interno do Kismet na documentação oficial, em https://www.kismetwireless.net/docs/readme/alerts_and_wids/.

Figura 43 - Kismet SPOOF

- Ataque *Deauthenticate*

Quando existe um ataque de *deauthenticate*, o "Kismet" gera automaticamente um alarme de *DEATHFLOOD* onde é visível qual o dispositivo que está a ser atacado (Figura 44). Em alguns cenários, este ataque emitiu também um alarme do tipo *Denial* devido ao "Kismet" o considerar também do tipo *denial-of-service*.

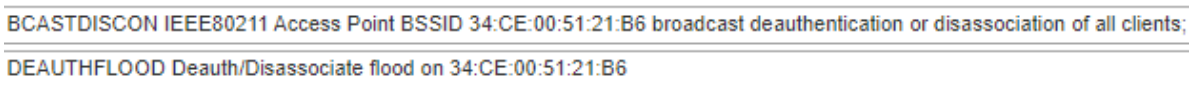


Figura 44 - Kismet Deauthenticate Alert

5.4.4 Outros alarmes gerados na análise dos dados WiFi

Existem também outros alarmes que não são gerados pelo Kismet e, como tal, a aplicação Java executa várias validações aos dados recebidos e gera os alarmes necessários consoante os valores das *settings* da aplicação.

Testes criados pela aplicação Java:

- Novos dispositivos

Quando é encontrado na rede WiFi um novo dispositivo, independentemente do seu tipo, dispositivo cliente ou AP, este gera um alarme. Existem, no entanto, alarmes distintos para dispositivos cliente e dispositivos AP, podendo o *SuperAdmin* desligar algum ou ambos, caso assim o deseje. No código abaixo (Tabela 22), é visível o alarme de um novo dispositivo cliente juntamente com a chamada da função *notificationAlerts.SendNotifications(alert)*, que verifica se é necessário enviar alguma notificação adicional, tal como SMS ou e-mail.

Tabela 22 - Alarme novo dispositivo cliente

```
    if (settingService.isSettingAlarmEnabled("NEW CLIENT DEVICE")) {
        isNewClientAlarm(deviceClient);
    }
 private void isNewClientAlarm(DeviceClient deviceClient) {
    if (deviceClientRepository.findAllByKismetID(deviceClient.getKismetID()).size() == 0) {
        Alert alert = new Alert();
        Optional<Alert> alertIsPresent = isNewAlertOrUpdate("New Client Device: " +
deviceClient.getMacAddress());
        if (alertIsPresent.isPresent()) {
            alert = alertIsPresent.get();
            alert.setCaptured_at(Instant.now());
            alertRepository.save(alert);
        } else {
            alert.setCaptured_at(Instant.now());
            alert.setCreated_at(Instant.now());
            alert.setClassType("New Client Device");
            alert.setSourceMAC(deviceClient.getMacAddress());
            alert.setCreated_by("Client Analyse");
```

```
            alert.setSeverity(settingService.getSeverityByAlarmType(alert.getClassType()));
            alert.setIsAlertAccepted(false);
            alert.setDescription("New Client Device: " + deviceClient.getMacAddress());
            notificationAlerts.SendNotifications(alert);
            alertRepository.save(alert);
        }
    }
}
```

- Novas redes

Assim que uma nova rede WiFi for encontrada, ela gera automaticamente um novo alerta indicando que existe uma nova rede juntamente com alguns dos seus dados.

Foi desenvolvido o código da Tabela 23 para gerar o novo alarme.

Tabela 23 - Alarme nova rede WiFi

```
if (settingService.isSettingAlarmEnabled("NEW NETWORK")) {
            isNewNetworkAlarm(apNetwork, macAddress);
        }


private void isNewNetworkAlarm(APNetwork apNetwork, String MacAddress) {
    if
(apNetworkRepository.findAllByDeviceKismetId(apNetwork.getDeviceKismetId()).size() == 0)
{
        Alert alert = new Alert();
        alert.setCaptured_at(Instant.now());
        alert.setCreated_at(Instant.now());
        alert.setClassType("New Network");
        alert.setSourceMAC(MacAddress);
        alert.setCreated_by("Network Analyse");
        alert.setSeverity(settingService.getSeverityByAlarmType(alert.getClassType()));
        alert.setIsAlertAccepted(false);
        alert.setDescription("Nova Network: " + apNetwork.getName());
        alertService.saveAlert(alert);
```

```
    }
```

- Novo *device* cliente em rede conhecida

Garantir que apenas os clientes com permissões se conseguem ligar aos AP é fulcral para a segurança da rede. Para garantir esta segurança, caso exista um novo dispositivo ligado a uma rede conhecida, será gerado um alerta. Um dispositivo é considerado desconhecido se não tiver nenhuma descrição sobre ele. Se este dado não for relevante, o *SuperAdmin* pode desativar este alarme. Foi desenvolvido o código apresentado na Tabela 24 para criar este alarme.

Tabela 24 - Alarme novo cliente em rede conhecida

```
        if (settingService.isSettingAlarmEnabled("NEW CLIENT DEVICE ON NETWORK")) {
            isNewNetworkClientAlarm(clientNetwork, isDeviceKnown, macAddress);
        }
   private void isNewNetworkClientAlarm(ClientNetwork clientNetwork, Boolean
isDeviceKnown, String macAddress) {
      if (!isDeviceKnown) {
         boolean isNetworkKnown =
apNetworkService.findAllApNetworksByKismetId(clientNetwork.getDeviceClientKismetId())
               .stream().anyMatch(apNetwork -> (apNetwork.getLocation() != null ||
apNetwork.getDescription() != null));
         if (!isNetworkKnown) {
            Alert alert = new Alert();
            Optional<Alert> alertIsPresent = isNewAlertOrUpdate("Novo Client em rede
conhecida: " + clientNetwork.getNetworkName() + " Client:" + macAddress);
            if (alertIsPresent.isPresent()) {
               alert = alertIsPresent.get();
               alert.setCaptured_at(Instant.now());
            } else {
               alert.setCaptured_at(Instant.now());
               alert.setCreated_at(Instant.now());
               alert.setClassType("NEW CLIENT DEVICE ON NETWORK");
               alert.setSourceMAC(macAddress);
```

```
                alert.setCreated_by("Network Analyse");
                alert.setSeverity(settingService.getSeverityByAlarmType(alert.getClassType()));
                alert.setIsAlertAccepted(false);
                alert.setDescription("Novo Client em rede conhecida: " +
clientNetwork.getNetworkName() + " Client:" + macAddress);
            }
            alertService.saveAlert(alert);
        }
    }
  }
```

- Número de pacotes de erro demasiado elevado

Sempre que o servidor Java faz um pedido ao "Kismet", ele compara o número de pacotes de erros dos dispositivos clientes com o que tem na sua base de dados. Caso essa diferença seja demasiado elevada (valor configurado por *setting*), é acionado um alarme, cujo código que o gera se encontra na Tabela 25.

O número de pacotes de erro tem de ser ajustado tendo em consideração o tempo entre pedidos. Quanto maior o tempo entre pedidos do Java para o Kismet, maior poderá ser o número de pacotes com erro até ser gerado um alarme, para garantir que não existem falsos positivos.

Tabela 25 - Alarme pacotes de erro elevado

```
          if (apNetwork.getErrorPackages() - apNetworkOnDB.get().getErrorPackages() >
numberPackagesToTriggerError
                  && settingService.isSettingAlarmEnabled("ERROR PACKAGES TOO HIGH")) {
              errorPackagesAlarm(apNetwork, macAddress,
apNetworkOnDB.get().getErrorPackages());
          }


   private void errorPackagesAlarm(APNetwork apNetwork, String macAddress, Long
lastTimeErrorPackage) {
      Alert alert = new Alert();
      alert.setCaptured_at(Instant.now());
```

```
        alert.setCreated_at(Instant.now());
        alert.setClassType("ERROR PACKAGES TOO HIGH");
        alert.setSourceMAC(macAddress);
        alert.setCreated_by("Client Analyse");
        alert.setSeverity(settingService.getSeverityByAlarmType(alert.getClassType()));
        alert.setIsAlertAccepted(false);
        alert.setDescription("Device AP Error packages too high: " + macAddress + " Error
Packages:" + apNetwork.getErrorPackages() + " Package from last time: " +
lastTimeErrorPackage);
        alertService.saveAlert(alert);
    }
```

- WPS Ativo

Ter o WPS ativo pode ser um potencial problema, podendo um atacante tentar um ataque do tipo *reaver*[15]. Assim, quando é detetada uma rede com essa configuração, é acionado um alarme (Tabela 26). Caso já exista um alarme desse tipo que não está aceite, não vai ser gerado um segundo alarme, apenas a data do evento vai ser atualizada para a última ocorrência.

Tabela 26 - Novo Alarme WPS Ativo

```
    if (apNetwork.getIsWPSActive() && settingService.isSettingAlarmEnabled("WPS ACTIVE"))
{
            isWPSActiveNetworkAlarm(apNetwork, macAddress);
        }
    private void isWPSActiveNetworkAlarm(APNetwork apNetwork, String macAddress) {
        List<Alert> alerts = alertService.findAllAlertsByDescription("WPS Active: " +
apNetwork.getName() + " MAC: " + macAddress);
        if (alerts.size() == 0 || alerts.stream()
            .filter(alert -> !alert.getIsAlertAccepted())
            .count() == 0) {
            Alert alert = new Alert();
```

[15] Mais informação sobre este tipo de ataque pode encontrado em https://outpost24.com/blog/wps-cracking-with-reaver

```
            alert.setCaptured_at(Instant.now());
            alert.setCreated_at(Instant.now());
            alert.setClassType("WPS ACTIVE");
            alert.setSourceMAC(macAddress);
            alert.setCreated_by("Network Analyse");
            alert.setSeverity(settingService.getSeverityByAlarmType(alert.getClassType()));
            alert.setIsAlertAccepted(false);
            alert.setDescription("WPS Active: " + apNetwork.getName() + " MAC: " +
macAddress);
            alertService.saveAlert(alert);
        } else {
            Alert alertOnDB = alerts.stream().filter(alert ->
!alert.getIsAlertAccepted()).findFirst().get();
            alertOnDB.setUpdated_at(Instant.now());
            alertService.saveAlert(alertOnDB);

        }
    }
```

6. Testes

No decorrer do projeto foram criados cenários de testes com o intuito de simular vários eventos onde fosse possível testar todas as funcionalidades desenvolvidas para os vários componentes do projeto. Foi montado o cenário da Figura 45, composto pelos seguintes componentes:

- Router WiFi 2.4Ghz com SSID TestingNetwokWiFi e *Mac Address* 34:CE:00:51:21:B6
- Smartphone (dispositivo cliente) com *Mac Address* A8:9C:ED:56:64:1D
- Tablet (dispositivo cliente) com Mac Address 98:3B:16:3B:4A:EE
- Máquina de "Kismet Remote Capture" para redes WiFi
- Servidor "Kismet" para receber as capturas remotas
- Servidor Java (Tomcat) e servidor de *Front-end* (Apache)

As especificações técnicas dos servidores podem ser consultadas no anexo F.

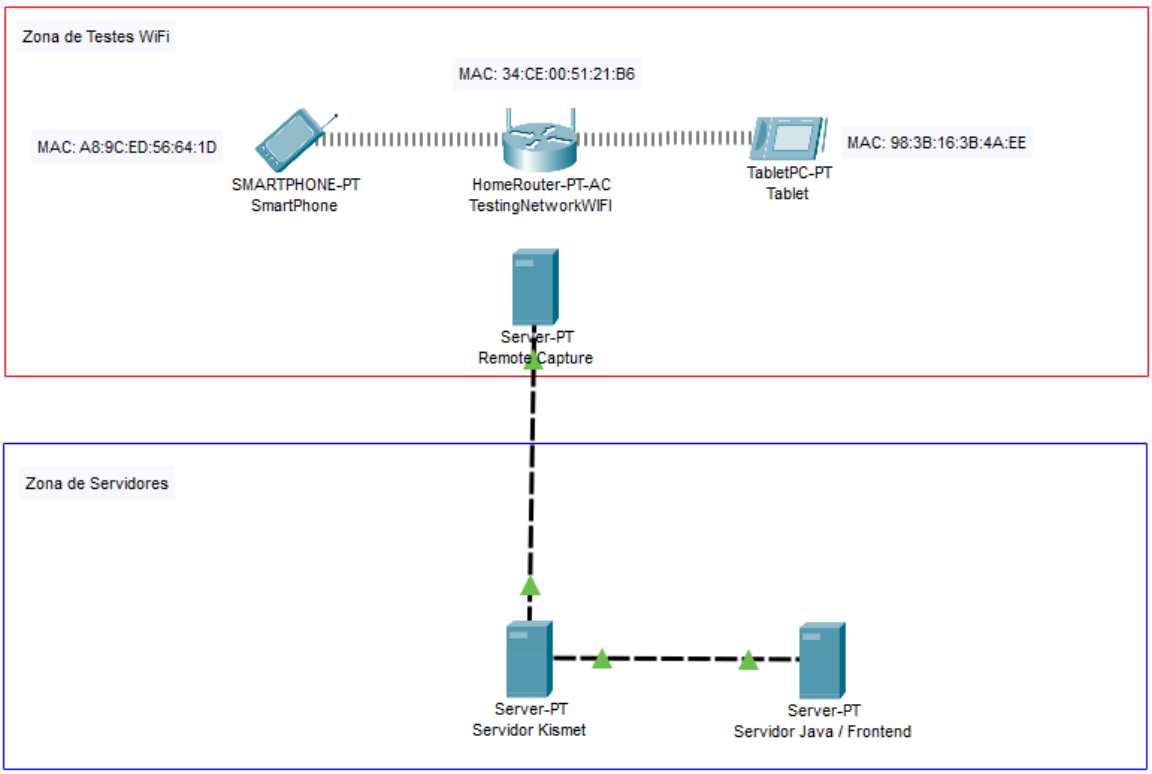


Figura 45 - Esquema de testes WiFi

Foi utilizada uma ligação de rede por cabo entre a máquina de captura remota, o servidor do "Kismet" e o servidor Java/*Front-end*. Todos os outros componentes comunicam através de rede WiFi 2.4Ghz.

Utilizou-se também uma máquina virtual, denominada por atacante, com o sistema operativo Kali 2020.3 para efetuar alguns ataques. Este equipamento, que apenas é utilizado em alguns dos testes efetuados, não se encontra no esquema acima pois não faz parte da rede. Antes de se iniciarem os testes foram configuradas algumas *settings* de forma a tirar o máximo proveito da aplicação, que se especificam seguidamente.

A *setting* que indica o tempo entre pedidos do servidor Java à aplicação do "Kismet", *REQUESTS TIME IN SECONDS*, foi configurada para 300 segundos, garantindo assim que se obtenham novos dados a cada 5 minutos. Todas as *settings* de alarmes estão ativas para garantir que todos os alarmes estão disponíveis para os testes (Figura 46).

Por último, as *settings* de notificações de e-mail e SMS também foram ativadas para ser possível testar estas funcionalidades.

Sim	DENIAL	10
Sim	SPOOF	5
Sim	NEW NETWORK	2
Sim	NEW CLIENT DEVICE	2
Sim	NEW AP DEVICE	2
Sim	NEW CLIENT DEVICE ON NETWORK	5
Sim	ERROR PACKAGES TOO HIGH	6
Sim	WPS ACTIVE	1
Não	NUMBER ERROR PACKAGES ALARM	50
Não	SEND EMAILS WITH ALERTS BIGGER THEN	6
Não	LIST OF EMAILS	
Não	SEND SMS WITH ALERTS BIGGER THEN	9
Não	LIST OF SMS NUMBERS	
Não	REQUESTS TIME IN SECONDS	60

Figura 46 - Ativar todas as settings de alarme

Entre cada teste, todos os dados da base de dados são limpos para garantir que o resultado de um teste não afeta o resultado de outros.

6.1. Teste de Rogue Access Point

Foram desenvolvidos dois testes com softwares distintos para testar um ataque de *Rogue Access Point*, e em ambos foram simulados *Access Points* com a máquina atacante. O primeiro teste decorreu utilizando o software *create_ap*[16], no entanto ele não vem

[16] Mais informação sobre o software pode se encontrada em [24]

instalado no Kali 2020.3, pelo que foram utilizados os comandos da Tabela 27 para proceder à sua instalação.

Tabela 27 - Comandos instalação create_ap

sudo apt-get install haveged hostapd git util-linux procps iproute2 iw dnsmasq
iptables bettercap
git clone https://github.com/oblique/create_ap
cd create_ap
sudo make install
cd .. && rm -rf create_ap

Assim que finalizou a instalação do *create_ap* foi executado o comando constante na Figura 47 para criar um *Rogue Access Point*. Neste caso, o SSID do *Access Point* é "TestingNetworkWIFI", garantindo assim que tem o mesmo SSID que o da rede WiFi já existente.

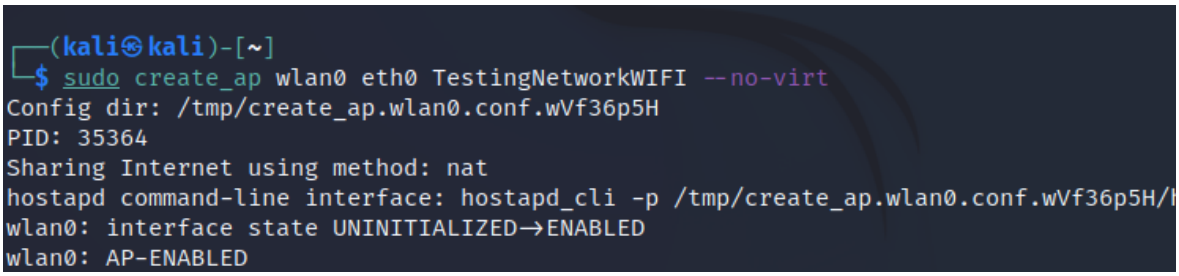


Figura 47 - Arranque do Rogue AP com software create_ap

Para garantir que o "Kismet" reconhece o MAC do SSID original como sendo fidedigno, é necessário alterar uma configuração onde são definidos os MAC's de cada SSID. Esta configuração encontra-se no ficheiro *kismet.conf* e é necessário especificar o MAC que corresponde ao SSID. Neste cenário foi adicionado 34:CE:00:51:21:B6 ao SSID TestingNetworkWIFI (Tabela 28).

Tabela 28 - Configuração Kismet MACs permitidos

apspoof=Foo3:ssid="TestingNetworkWIFI",validmacs=34:CE:00:51:21:B6

Assim que o *Rogue* AP é reconhecido pelo "Kismet" e posteriormente enviado para o software Java, é gerado um novo alarme com esta ocorrência, Figura 48 - Novo Alarme Rogue AP.

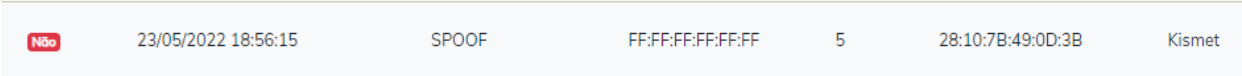


Figura 48 - Novo Alarme Rogue AP

O alarme contém uma descrição com dados relevantes do acontecimento. Neste caso, indica qual o MAC, o SSID e a razão pela qual este evento pode ser um ataque de *spoofing*, Tabela 29.

Tabela 29 – Descrição do alarme Rogue AP com create_ap

IEEE80211 Access Point BSSID 28:10:7B:49:0D:3B SSID TestingNetworkWIFI changed advertised encryption from Open to WPA2 WPA2-PSK TKIP AES-CCMP which may indicate AP spoofing/impersonation

O segundo teste de *Rogue Access Point* foi efetuado utilizando o software "Pumpkin3"[17], que não vem instalado no Kali e, como tal, foi necessário executar os comandos da Tabela 30 para o instalar e posteriormente executar esse software.

Tabela 30 - Comandos instalação Pumpkin3

sudo apt install libssl-dev libffi-dev build-essential
git clone https://github.com/P0cL4bs/wifipumpkin3.git
cd wifipumpkin3
sudo apt install python3-pyqt5
sudo python3 setup.py install
sudo wifipumpkin3

Foi configurado o "Pumpkin3" para ter o mesmo SSID que o router original, "TestingNetworkWifi", conforme representa a Figura 49.

[17] Mais informação sobre o software pode se encontrada em [25]

Figura 49 - Arranque do Pumpkin3

Assim que for executado o comando *Start* dentro do "Pumpink3", o *Rogue Access Point* começa a funcionar (Figura 50).

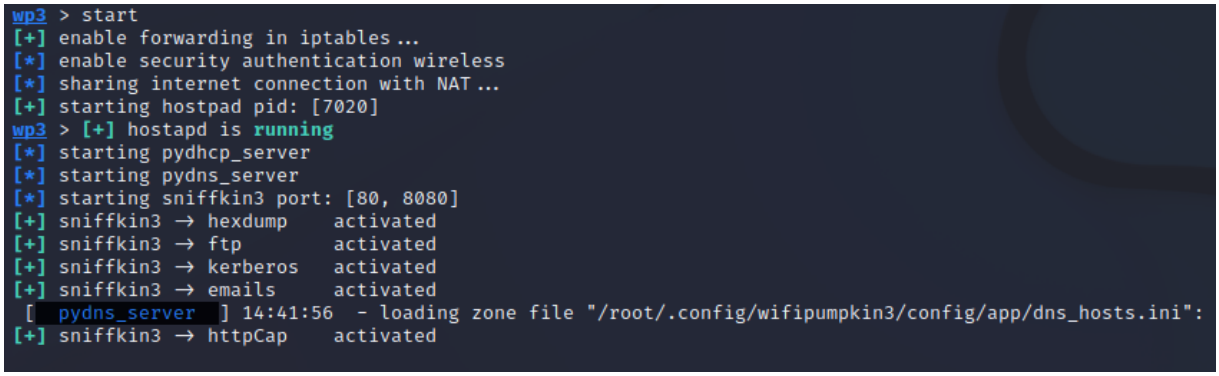


Figura 50 - Arranque do Rogue AP com "Pumpkin3"

Quando o novo AP é detetado pelo "Kismet", é gerado um novo alerta que, posteriormente, vai ser adiciono ao Java e ao *Front-end* (Figura 51).

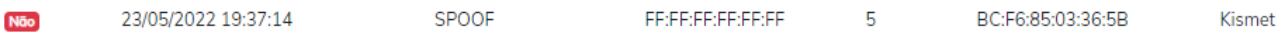


Figura 51 - Alarme Kismet Rogue AP com Pumpkin3

A descrição do alarme (Tabela 31) é similar à descrição do alarme de *Rogue* AP criado pelo alarme do primeiro teste de *Rogue* AP.

Tabela 31 - Descrição do alarme *Rogue* AP com Pumpkin3

IEEE80211 Access Point BSSID BC:F6:85:03:36:5B SSID "TestingNetworkWIFI" changed advertised encryption from Open to WPA2 WPA2-PSK TKIP which may indicate AP spoofing/impersonation

Em ambos os testes, o software responde de forma positiva mostrando ao utilizador um alarme do evento ocorrido.

6.2. Teste de *SPOOF* com *Rogue* AP

Foi efetuado um ataque de *SPOOF* com um *Rogue* AP através da alteração do MAC da placa WiFi do AP com o comando *macchanger*[18], para o MAC do *Access Point* que já estava em funcionamento na rede 34:CE:00:51:21:B6. Depois do novo MAC estar configurado, é criado o *Access Point* com o comando *create_ap*, tal como no ponto anterior. Esta sequência de comandos é visível na Tabela 32.

Tabela 32 - Comandos para alterar MAC com software macchanger

```
ifconfig wlan0 down
macchanger -m 34:CE:00:51:21:B6 wlan0
ifconfig wlan0 up
sudo create_ap wlx28ee52bcfed5 ens33 TestingNetworkWIFI –no-virt
```

Assim que este dispositivo é detetado na rede WiFi é criado um novo alarme de SPOOF (Figura 52).

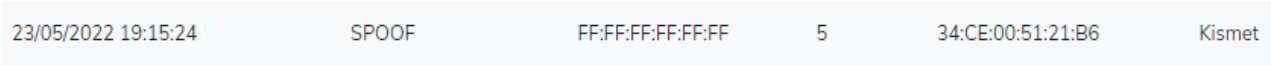


Figura 52 - Alarme de SPOOF com Rogue AP

6.3. Ataque *Deauthenticate*

Foram criados testes com dois softwares distintos para ter garantias que este tipo de ataque é devidamente tratado pela aplicação.

O primeiro teste foi efetuado utilizando o software "aircrack-ng", com o comando "aireplay-ng" (Tabela 33). O intuito deste ataque é enviar repetidamente pacotes de *deauthenticate* impedindo o dispositivo com o MAC A8:9C:ED:56:64:1D de estar ligado ao *Access Point* com o MAC 34:CE:00:51:21:B6.

Tabela 33 - Comando de ataque *Deauthenticate* com "aireplay-ng"

```
sudo  aireplay-ng -0 0 -a 34:CE:00:51:21:B6 -c A8:9C:ED:56:64:1D wlan0
```

[18] Mais informação sobre o software pode se encontrada em [26]

Assim que foi iniciado o ataque de *deauthenticate*, o Kismet detetou-o e foi criado um alarme de *Deauth/denail*, Figura 53 - Alarme de Deauthenticate com aireplay-ng.

Alerta validado	Capturare em	Classe	MacDestino	Severidade	MacAddress Origem	Criado por	Criado em
Não	21/05/2022 22:20:23	DENIAL	A8:9C:ED:56:64:1D	10	34:CE:00:51:21:B6	Kismet	21/05/2022 22:20:34

Figura 53 - Alarme de Deauthenticate com aireplay-ng

O segundo ataque de *deauthenticate* foi efetuado com o software MDK4[19], que não vem instalado no Kali, pelo que foi executado o comando da Tabela 34 para fazer a sua instalação.

Tabela 34 - Instalação software mdk4

sudo apt install mdk4

Assim que foi concluída a instalação do "MDK4", foi possível começar o ataque especificando o *Access Point* com o MAC 34:CE:00:51:21:B6 e o dispositivo cliente com o MAC A8:9C:ED:56:64:1D. Na Figura 54 é apresentado um ataque com o "MDK4", enviando repetidamente pacotes de *deauthenticate*.

```
┌──(kali㉿kali)-[~]
└─$ sudo mdk4 wlan0 d -c 6 -B 34:CE:00:51:21:B6 -S A8:9C:ED:56:64:1D
Disconnecting A8:9C:ED:56:64:1D from 34:CE:00:51:21:B6 on channel 6
Packets sent:      1 - Speed:    1 packets/sec
Disconnecting 34:CE:00:51:21:B6 from 34:CE:00:51:21:B6 on channel 6
Packets sent:   1008 - Speed: 1007 packets/sec
Disconnecting 34:CE:00:51:21:B6 from 34:CE:00:51:21:B6 on channel 6
Packets sent:   1827 - Speed:  819 packets/sec
Disconnecting A8:9C:ED:56:64:1D from 34:CE:00:51:21:B6 on channel 6
Packets sent:   2951 - Speed: 1124 packets/sec
Disconnecting A8:9C:ED:56:64:1D from 34:CE:00:51:21:B6 on channel 6
Packets sent:   4090 - Speed: 1139 packets/sec
Disconnecting 34:CE:00:51:21:B6 from 34:CE:00:51:21:B6 on channel 6
Packets sent:   4939 - Speed:  849 packets/sec
```

Figura 54 - Ataque *Deauthenticate* com "mdk4"

O ataque ativou um alarme (Figura 55) que indica um ataque de *deauthenticate* a um dispositivo.

[19] Para mais informações sobre MDK4 consultar [27]

Alerta validado	Capturare em	Classe	MacDestino	Severidade	MacAddress Origem	Criado por	Criado em
Não	23/05/2022 18:15:43	DENIAL	34:CE:00:51:21:B6	10	34:CE:00:51:21:B6	Kismet	23/05/2022 18:16:13

Figura 55 - Alarme de *Deauthenticate* com "mdk4"

Ambos os ataques de *deauthenticate* são identificados com sucesso pelo software, gerando novos alarmes.

6.4.Novos dispositivos encontrados

Para ter um controlo total da rede WiFi, é necessário ter uma visão global sobre todos os dispositivos WiFi, mesmo que estes não estejam conectados a uma rede.

Foi ligado o WiFi do dispositivo com o MAC A8:9C:ED:56:64:1D mas, no entanto, este dispositivo não se ligou a nenhuma rede e apenas ficou a procurar redes WiFi (*scanning*). Assim que o "Kismet" detetou um novo dispositivo, foi criado um novo alarme (Figura 56), pois este dispositivo cliente ainda não constava na base de dados.

28/05/2022 13:52:38	New Client Device	2	A8:9C:ED:56:64:1D	Client Analyse

Figura 56 - Alarme de novo dispositivo smartphone

A descrição deste alarme indica o MAC do novo cliente, Tabela 35.

Tabela 35 - Descrição do alarme de novo dispositivo

New Client Device: A8:9C:ED:56:64:1D

Foi efetuado um segundo teste, ligando o WiFi de outro dispositivo com o MAC 98:3B:16:3B:4A:EE e, tal como no teste anterior, não foi ligado a nenhuma rede WiFi. Este dispositivo também desencadeou um novo alarme, Figura 57.

Não	28/05/2022 13:42:46	New Client Device	2	98:3B:16:3B:4A:EE	Client Analyse

Figura 57 - Alarme de novo dispositivo tablet

Estes novos dispositivos também podem ser consultados na listagem com todos os dispositivos clientes (Figura 58) ou AP que o "Kismet" já reconheceu, juntamente com os detalhes de cada dispositivo.

Ações	MacAdress	Construtor	Visivel a partir de:	Visto pela última vez:	Descrição	Tipo de dispositivo
EDITAR REDES	B8:C6:AA:CE:E2:3A	Eardа Technologies Ltd	25/04/2022 22:02:32	25/04/2022 22:02:50		Wi-Fi Device
EDITAR REDES	52:B2:80:F8:C4:D9	Unknown	25/04/2022 22:02:32	25/04/2022 22:03:40		Wi-Fi Device
EDITAR REDES	F4:D4:88:8A:B7:2E	Apple	25/04/2022 22:02:37	25/04/2022 22:03:45		Wi-Fi Ad-Hoc
EDITAR REDES	78:8B:2A:A1:4C:CD	Zhen Shi Information Technology (Shanghai) Ltd	25/04/2022 22:02:37	25/04/2022 22:02:37		Wi-Fi Device
EDITAR REDES	70:2C:1F:64:4E:55	Wisol	25/04/2022 22:03:16	25/04/2022 22:03:17		Wi-Fi Device
EDITAR REDES	00:25:00:FF:94:73	Apple	25/04/2022 22:02:32	25/04/2022 22:03:40		Wi-Fi Ad-Hoc
EDITAR REDES	F4:D4:88:8A:B7:2E	Apple	26/04/2022 09:19:44	26/04/2022 22:12:56		Wi-Fi Ad-Hoc
EDITAR REDES	70:2C:1F:64:4E:55	Wisol	26/04/2022 09:22:11	26/04/2022 22:13:14		Wi-Fi Device

Figura 58 - Lista de dispositivos Clientes

6.5.Novas redes WiFi

Para executar este teste foi criada uma nova rede com o software *create_ap* e o SSID "AP_Testing" (Figura 59).

```
└─$ sudo create_ap wlan0 eth0 AP_Testing --no-virt
Config dir: /tmp/create_ap.wlan0.conf.iYQhjdTk
PID: 12879
Network Manager found, set wlan0 as unmanaged device ... DONE
Sharing Internet using method: nat
hostapd command-line interface: hostapd_cli -p /tmp/create_ap.wlan0.conf.iYQhjdTk/hostapd_ctrl
wlan0: interface state UNINITIALIZED→ENABLED
wlan0: AP-ENABLED
```

Figura 59 - Criar nova rede com software create_ap

Foi gerado um novo alarme (Figura 60) indicando que foi encontrada uma nova rede WiFi pelo "Kismet Remote Capture".

Não	28/05/2022 14:44:10	New Network	2	28:10:7B:49:0D:3B	Network Analyse	28/05/2022 14:44:10

Figura 60 - Alarme de nova rede com *create_ap*

A descrição do alarme (Tabela 36) indica o SSID da nova rede encontrada, "AP_Testing".

Tabela 36 Descrição do alarme de nova rede

Nova Network: AP_Testing

Para não utilizar apenas um router virtual, foi efetuado um segundo teste com um router físico e o resultado foi similar ao exemplo anterior, Figura 61.

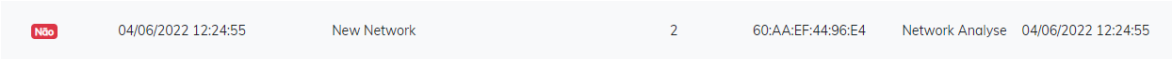


Figura 61 - Alarme de nova rede com router real

Existe ainda uma listagem com todos os *Access Points* (Figura 62), que pode ser consultada para obter mais dados sobre os mesmos.

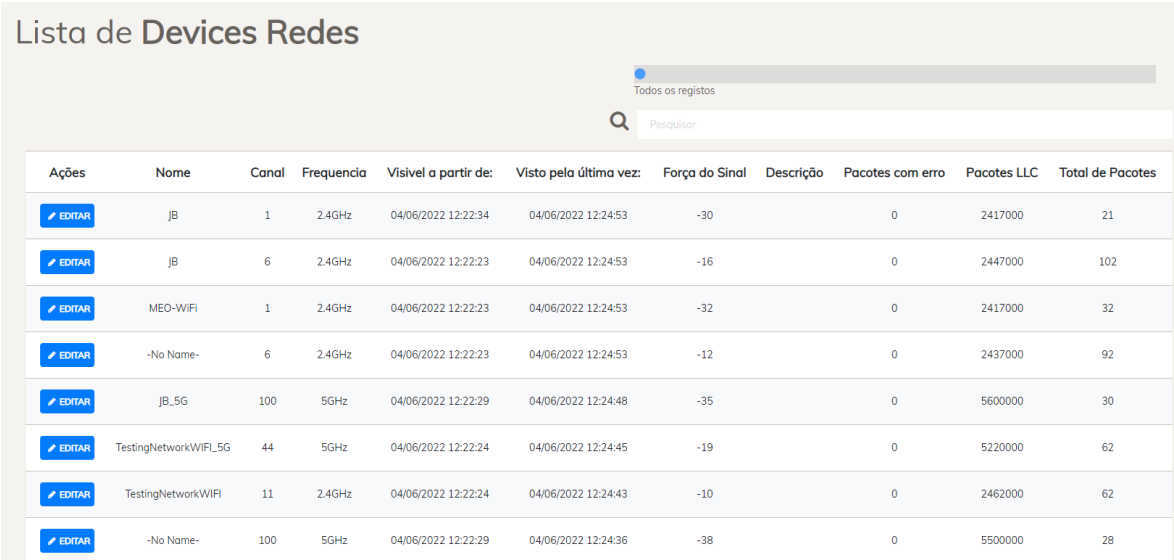


Figura 62 - Lista de *Access Points*

O software reconhece novas redes WiFi, tanto para redes geradas por virtualização como redes de routers físicos, alertando o utilizador para as mesmas.

6.6.Novo dispositivo cliente em rede WiFi conhecida

Para garantir que apenas os dispositivos autorizados se podem ligar às redes WiFi das organizações é necessário fazer um *scanning* constante de todos os dispositivos ligados. Neste cenário, uma rede é considerada conhecida (ou de uma organização) quando existe uma descrição da mesma na base de dados.

Foi ligado um novo dispositivo cliente com o MAC A8:9C:ED:56:64:1D à rede conhecida com SSID “TestingNetworkWIFI”. Esta ligação fez despoletar um novo alarme, indicando que a rede tinha um novo cliente, Figura 63.

Figura 63 - Alarme de novo dispositivo em rede conhecida

A descrição do alarme, Tabela 37, indica o SSID da rede e o MAC do dispositivo cliente que se ligou à mesma.

Tabela 37 - Descrição do alerta de novo dispositivo em rede conhecida

Novo Client em rede conhecida: TestingNetworkWIFI Client:A8:9C:ED:56:64:1D

Com este tipo de alarme é possível observar todos dos dispositivos ligados a cada rede e saber quando algum dispositivo novo se liga.

6.7. Número de pacotes de erro demasiado elevado

O número de pacotes de erro é uma métrica que pode indicar algum problema ou ataque na rede. Para efetuar este ataque, foram efetuados os comandos da Tabela 38 com o intuito de gerar pacotes de erro.

Tabela 38 - Criação de pacotes de erro com "aireplay-ng"

sudo aireplay-ng -3 -b 34:CE:00:51:21:B6 -h A8:9C:ED:56:64:1D wlan0 sudo aireplay-ng -1 0 -a 34:CE:00:51:21:B6 -h A8:9C:ED:56:64:1D wlan0

No entanto, depois de efetuar este ataque, o "Kismet Remote Capture" não reconheceu nenhum pacote de erro. Foi então efetuada uma segunda tentativa de gerar pacotes de erro com o software *besside-ng*, utilizando o comando da Tabela 39, mas sem sucesso.

Tabela 39 - Criação de pacotes de erro com besside-ng

besside-ng -c 1 -b 34:CE:00:51:21:B6 wlan0

O "Kismet Remote Capture" não conseguiu em nenhum dos casos capturar pacotes de erro, apenas reconhece o número total de pacotes, Figura 64.

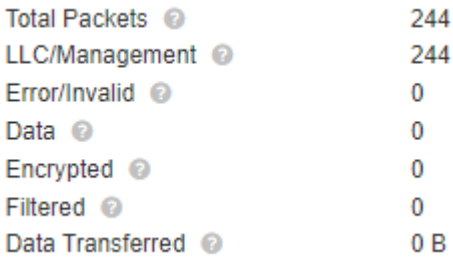


Figura 64 - Kismet Pacotes

Esta limitação do "Kismet" faz com que não seja possível utilizar pacotes de erro como métrica para gerar alarmes. Caso fosse possível o "Kismet" reconhecer estes pacotes, existe uma *setting* (Figura 65) que determina o número de pacotes necessários para gerar um alerta, número que é utilizado entre pedidos ao servidor "Kismet". Imaginando que são efetuados pedidos a cada 5 minutos, ele verifica se existem mais de 50 pacotes de erro naquele tempo.

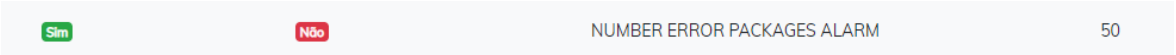


Figura 65 - *Setting* de pacotes de erro

6.8. WPS Ativo

O *Access Point* com MAC 34:CE:00:51:21:B6 encontra-se com o WPS ativo e é reconhecido pelo "Kismet" e, consequentemente, gera um alarme (Figura 66).

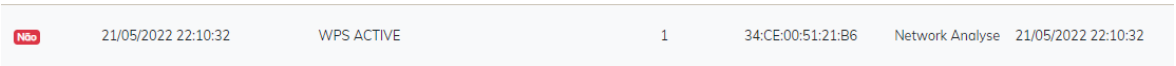


Figura 66 - Alarme de WPS ativo

A descrição do alarme (Tabela 40) indica o SSID e MAC do dispositivo que tem o MAC ativo.

Tabela 40 - Descrição do Alarme de WPS ativo

WPS Active: TestingNetworkWIFI MAC: 34:CE:00:51:21:B6

6.9. Testar os vários tipos de alarmísticas

Atualmente existem três tipos de alarmísticas distintas que dependem da severidade do alarme:

- Alarme no Website

- Envio de Email
- Envio de SMS

A parte lógica de alarmística foi implementada no Java utilizando o *design pattern Chain of Responsibility* [20], o que possibilita que, no futuro, possam ser adicionados outros tipos de alarmísticas de forma bastante simples.

Os alarmes do Website são a forma mais simples de alarmística (Figura 67), podendo o utilizador consultar todos os alarmes gerados e aceitar os mesmos, dando assim a indicação que está ciente daquele evento.

Ações	Alerta validado	Capturare em	Classe	MacDestino	Severidade	MacAddress Origem	Criado por	Criado em
ACEITAR ALARME	Não	04/06/2022 12:24:59	New Client Device		2	D6:22:A4:27:E9:B2	Client Analyse	04/06/2022 12:24:59
ACEITAR ALARME	Não	04/06/2022 12:24:59	New Client Device		2	6A:65:49:55:3B:4A	Client Analyse	04/06/2022 12:24:59
ACEITAR ALARME	Não	04/06/2022 12:24:59	NEW CLIENT DEVICE ON NETWORK		5	A8:9C:ED:56:64:1D	Network Analyse	04/06/2022 12:24:59
ACEITAR ALARME	Não	04/06/2022 12:24:59	NEW CLIENT DEVICE ON NETWORK		5	5C:CF:7F:76:EE:FB	Network Analyse	04/06/2022 12:24:59
ACEITAR ALARME	Não	04/06/2022 12:24:58	NEW CLIENT DEVICE ON NETWORK		5	58:FC:20:8E:7A:9F	Network Analyse	04/06/2022 12:24:58
ACEITAR ALARME	Não	04/06/2022 12:24:58	NEW CLIENT DEVICE ON NETWORK		5	58:FC:20:8E:7A:9F	Network Analyse	04/06/2022 12:24:58
ACEITAR ALARME	Não	04/06/2022 12:24:58	NEW CLIENT DEVICE ON NETWORK		5	F4:D4:88:8A:B7:2E	Network Analyse	04/06/2022 12:24:58

Figura 67 - Alarmística Website

Para o envio de e-mails existem duas *settings* que têm de se encontrar obrigatoriamente ativas:

- SEND EMAILS WITH ALERTS BIGGER THEN
- LIST OF EMAILS

A primeira *setting* identifica qual o grau mínimo de severidade dos alarmes a que serão enviados emails, neste teste foi utlizado o valor de severidade 6.

A segunda *setting* contém uma lista de endereços de e-mails, podendo enviá-los para mais do que um endereço ao mesmo tempo.

[20] Mais informação sobre este *design pattern* pode se encontrada em [28]

Foi ajustada a severidade do alarme de *Denail-Of-Service* para o valor de severidade 10 (Figura 68), para garantir que era enviado um e-mail com este alarme.

Figura 68 - Setting Alarme Denial-of-service

Foi executado um ataque de *Denail-Of-Service* e, assim que foi gerado um novo alarme para este evento, foi também enviado um e-mail para o "mailtrap" (Figura 69).

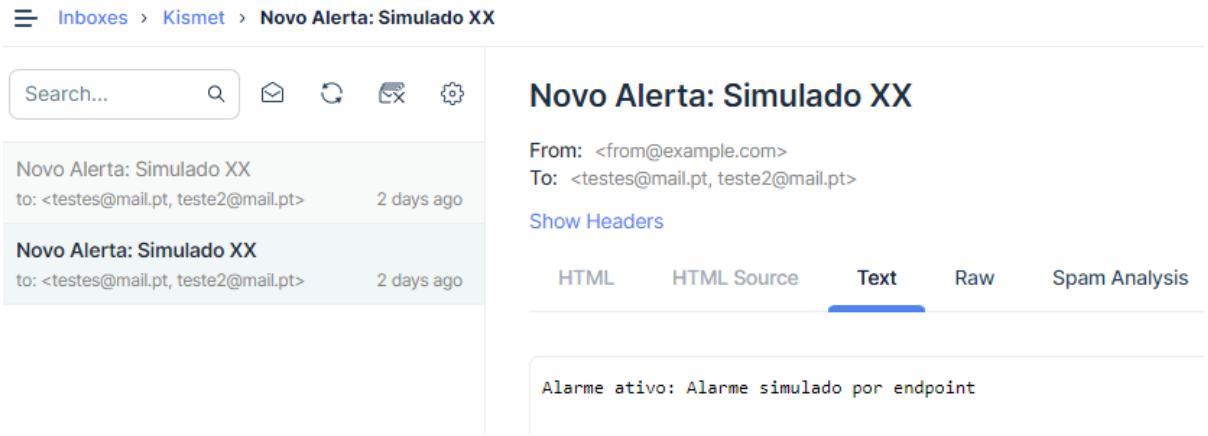


Figura 69 - MailTrap mail de alerta

A alarmística de SMS, tal como o e-mail, também tem *settings* que necessitam estar ativas para ser possível o seu envio, nomeadamente:

- *SEND SMS WITH ALERTS BIGGER THEN*
- *LIST OF SMS NUMBERS*

O valor de severidade para receber alarmes por SMS foi ajustado para 9 e foi adicionado um número válido (91XXXXXXX) na lista de números para envio de SMS. Tal como no exemplo anterior, foi utilizado um ataque de *deauthenticate* que gera um alarme de severidade 10.

Assim que este ataque foi identificado pelo servidor Java, foi feito um pedido à API da plataforma “Twillio” que por sua vez envia uma SMS (Figura 70) para os números especificados na *setting* de números para SMS.

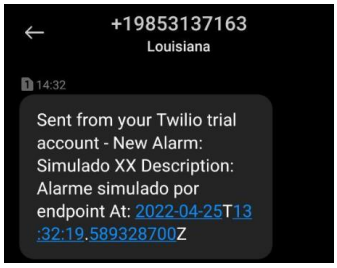


Figura 70 - SMS alerta de Alarme

7. Conclusão

As redes WiFi são um recurso essencial para as organizações, sendo que cada vez mais é necessário garantir o acesso às mesmas a dispositivos móveis como smarthpones, tablets ou computadores portáteis, em todos os lugares possíveis, de uma forma rápida e preferencialmente sem interrupções.

Apesar do WiFi ser muito vantajoso para os utilizadores, por não ter a necessidade de um cabo de *ethernet* ao seu redor e de ter facilmente uma extensão de cobertura sem interrupção do serviço, ele acarreta várias problemáticas, nomeadamente a dificuldade em delimitá-la ou até alguns *frames* de comunicação que não são encriptados, tal como o *frame* de *deauthenticate*.

Os equipamentos de rede, tal como os routers mais recentes, já têm alguns protocolos de rede focados na segurança informática, no entanto a grande maioria dos equipamentos ainda não dispõem desse tipo de segurança, nomeadamente na encriptação de pacotes de *management*.

Muitas destas problemáticas já estão identificadas pelo mercado, no entanto não existem soluções completas com uma forte alarmística que as consigam colmatar. Foi nesse sentido que foi desenvolvido este projeto, com sucesso, uma vez que demonstra uma solução exequível de ser utilizada com vista à monitorização da segurança em redes WiFi.

Com a elaboração deste projeto deu-se um contributo fundamental na área de monitorização das redes WiFi, que pode ser comprovado pelos testes elaborados no decorrer do mesmo. Foram ainda identificadas algumas necessidades que serão referenciadas em trabalhos futuros, com vista a uma futura operacionalização final da solução.

7.1. Trabalhos futuros

No que se refere ao desenvolvimento futuro, existem algumas melhorias que poderiam ser feitas para conseguir colmatar alguns problemas do Kismet na recolha de dados, nomeadamente a recolha de pacotes de erro ou a distinção mais precisa de *Rogue* AP.

Seria também interessante utilizar os *beacons* dos dispositivos clientes para conseguir criar um mapa das redes a que estes dispositivos já se ligaram anteriormente.

A adição de funcionalidades de análise com base em Inteligência Artificial e de *Machine Learning*, que consiga analisar todos estes dados de forma que possam gerar alarmes com o mínimo de erro possível, seria também relevante.

Por fim, com os dados atualmente armazenados, seria pertinente adicionar a funcionalidade de um mapeamento em tempo real do dispositivo, com base na sua localização, sabendo exatamente onde se encontra cada dispositivo e podendo ainda detetar quando ele se move e em que momento isso acontece.

8. Bibliografia ou Referências Bibliográficas

[1] NXP, "[802.11] Wi-Fi Connection/Disconnection proces," NXP, 06 10 2020. [Online]. Available: https://community.nxp.com/t5/Wireless-Connectivity-Knowledge/802-11-Wi-Fi-Connection-Disconnection-process/ta-p/1121148. [Acedido em 02 10 2021].

[2] Y. Li, J. Barthelemy, S. Sun, P. Perez e B. Moran, "A Case Study of WiFi Sniffing Performance Evaluation," *IEEE Access,* pp. 129224 - 129235, 10 July 2020.

[3] Tektronix, "Wi-Fi: Overview of the 802.11 Physical Layer and Transmitter Measurements," [Online]. Available: https://www.cnrood.com/en/media/solutions/Wi-Fi_Overview_of_the_802.11_Physical_Layer.pdf. [Acedido em 14 02 2022].

[4] C. J. Liu Chinese Academy of Sciences, X. Ye, J. Zhang e J. Li, "Security Verification of 802.11i 4-Way Handshake Protocol," em *IEEE International Conference on Communications,* Beijing, China, 2008.

[5] H. Noman, M. N. Shahidan e H. I. Mohammed, "An Automated Approach to Detect Deauthentication and Disassociation Dos Attacks on Wireless 802.11 Networks," *JCSI International Journal of Computer Science Issues, , ,* vol. Volume 12, nº Issue 4, 2015.

[6] D. Bongard, "Offline bruteforce attack on WiFi Protected Setup," @reversity, [Online]. Available: https://dl.aircrack-ng.org/wiki-files/doc/others/Hacklu2014_offline_bruteforce_attack_on_wps.pdf. [Acedido em 2022 03 01].

[7] Imperva, "TCP SYN Flood," [Online]. Available: https://www.imperva.com/learn/ddos/syn-flood/. [Acedido em 15 03 2022].

[8] X. G. K. D. Wenjia Wu, “A novel received signal strength–based approach for practical rogue access point detection,” *International Journal of Distributed Sensor Networks*, 28 08 2018.

[9] mlythics, “What is a man-in-the-middle attack?,” mlythics, [Online]. Available: https://learning.mlytics.com/cyber-attacks/what-is-a-man-in-the-middle-attack/.

[10] NetSpot , “NetSpot,” NetSpot, [Online]. Available: https://www.netspotapp.com/. [Acedido em 16 03 2022].

[11] R. Nayanajith, “802.11 Mgmt : Deauth & Disassociation Frames,” 11 10 2016. [Online]. Available: https://mrncciew.com/2014/10/11/802-11-mgmt-deauth-disassociation-frames/. [Acedido em 30 09 2021].

[12] M. S. Gast, 802.11 Wireless Networks: The Definitive Guide, O'Reilly Media, Inc., April 2005.

[13] Network Working Group, “RFC 5416,” [Online]. Available: https://datatracker.ietf.org/doc/html/rfc5416. [Acedido em 29 03 2022].

[14] W. L. M. M. Arthur Salmon, Applied Network Security: Proven tactics to detect and defend against all kinds of network attack, Packt, 2018.

[15] Ionos, “SYN flood attack: variants and countermeasures,” 21 09 20. [Online]. Available: https://www.ionos.com/digitalguide/server/security/syn-flood/. [Acedido em 16 03 2022].

[16] Aircrack-ng, “Aircrack-ng,” [Online]. Available: https://www.aircrack-ng.org/. [Acedido em 11 03 2022].

[17] Kismet, “Kismet,” Kismet, [Online]. Available: https://www.kismetwireless.net/. [Acedido em 18 03 2022].

[18] L. FasterXML, “Jackson,” [Online]. Available: https://github.com/FasterXML/jackson. [Acedido em 02 04 2022].

[19] Hibernate, [Online]. Available: https://hibernate.org/. [Acedido em 14 04 2022].

[20] S. Boot. [Online]. Available: https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#reference. [Acedido em 14 04 2022].

[21] VMware, “VMware Workstation,” [Online]. Available: https://www.vmware.com/products/workstation-player/workstation-player-evaluation.html. [Acedido em 14 04 2022].

[22] https://mailtrap.io/, “Mail Trap,” [Online]. Available: https://mailtrap.io/. [Acedido em 30 03 2022].

[23] Twilio, “Twilio,” [Online]. Available: https://www.twilio.com/sms. [Acedido em 20 04 2022].

[24] Oblique, “Create AP,” [Online]. Available: https://github.com/oblique/create_ap. [Acedido em 25 04 2022].

[25] P. Team, “Pumpkin3,” [Online]. Available: https://github.com/P0cL4bs/wifipumpkin3. [Acedido em 18 04 2022].

[26] Kali, “Macchanger,” [Online]. Available: https://www.kali.org/tools/macchanger/. [Acedido em 29 04 2022].

[27] Aircrack-ng, “MDK4,” [Online]. Available: https://github.com/aircrack-ng/mdk4. [Acedido em 01 05 2022].

[28] Refactoring Guru, “Chain of Responsibility in Java,” [Online]. Available: https://refactoring.guru/design-patterns/chain-of-responsibility/java/example. [Acedido em 29 05 2022].

9. Anexos

Anexo A - Exemplo de pedido de dados para um dispositivo.


```
    "kismet.device.base.first_time": 1640690153,
    "kismet.device.base.macaddr": "A6:D1:C0:C3:7A:2E",
    "kismet.device.base.crypt": "WPA2-PSK",
    "kismet.device.base.key": "4202770D00000000_2E7AC3C0D1A6",
    "kismet.device.base.packets.error": 0,
    "kismet.device.base.packets.total": 571,
    "kismet.device.base.manuf": "Apple",
    "kismet.device.base.basic_type_set": 1,
    "dot11.device": {
        "dot11.device.wps_m3_count": 0,
        "dot11.device.client_disconnects": 0,
        "dot11.device.num_responded_ssids": 0,
        "dot11.device.probe_fingerprint": 0,
        "dot11.device.typeset": 1,
        "dot11.device.client_disconnects_last": 0,
        "dot11.device.last_sequence": 0,
        "dot11.device.response_fingerprint": 0,
        "dot11.device.beacon_fingerprint": 1509448566,
        "dot11.device.last_beaconed_ssid_record": {
            "dot11.advertisedssid.ht_center_2": 0,
            "dot11.advertisedssid.ssid": "test2",
            "dot11.advertisedssid.ssidlen": 5,
            "dot11.advertisedssid.maxrate": 72.200000,
            "dot11.advertisedssid.ietag_checksum": 1582106754,
            "dot11.advertisedssid.crypt_set": 268436162,
            "dot11.advertisedssid.wps_state": 0,
            "dot11.advertisedssid.beacon": 1,
            "dot11.advertisedssid.wpa_mfp_required": 0,
            "dot11.advertisedssid.ccx_txpower": 0,
            "dot11.advertisedssid.first_time": 1640690164,
            "dot11.advertisedssid.beacons_sec": 230,
            "dot11.advertisedssid.dot11e_qbss_stations": 0,
            "dot11.advertisedssid.wpa_mfp_supported": 0,
            "dot11.advertisedssid.channel": "11",
            "dot11.advertisedssid.probe_response": 0,
            "dot11.advertisedssid.ssid_hash": 1693566721,
            "dot11.advertisedssid.dot11d_country": "",
            "dot11.advertisedssid.cloaked": 0,
            "dot11.advertisedssid.dot11r_mobility": 0,
            "dot11.advertisedssid.ht_center_1": 0,
            "dot11.advertisedssid.dot11e_qbss": 0,
            "dot11.advertisedssid.wps_version": 0,
            "dot11.advertisedssid.dot11r_mobility_domain_id": 0,
            "dot11.advertisedssid.wps_config_methods": 0,
            "dot11.advertisedssid.beaconrate": 10,
            "dot11.advertisedssid.last_time": 1640690623,
```

```
            "dot11.advertisedssid.dot11e_channel_utilization_perc": 0,
            "dot11.advertisedssid.ht_mode": "HT20",
            "dot11.advertisedssid.cisco_client_mfp": 0
        },
        "dot11.device.datasize": 0,
        "dot11.device.num_probed_ssids": 0,
        "dot11.device.bss_timestamp": 7018291200,
        "dot11.device.num_client_aps": 0,
        "dot11.device.neighbor_report_capable": 0,
        "dot11.device.advertised_ssid_map": [
            {
                "dot11.advertisedssid.ht_center_2": 0,
                "dot11.advertisedssid.ssid": "test2",
                "dot11.advertisedssid.ssidlen": 5,
                "dot11.advertisedssid.maxrate": 72.200000,
                "dot11.advertisedssid.ietag_checksum": 1582106754,
                "dot11.advertisedssid.crypt_set": 268436162,
                "dot11.advertisedssid.wps_state": 0,
                "dot11.advertisedssid.beacon": 1,
                "dot11.advertisedssid.wpa_mfp_required": 0,
                "dot11.advertisedssid.ccx_txpower": 0,
                "dot11.advertisedssid.first_time": 1640690164,
                "dot11.advertisedssid.beacons_sec": 230,
                "dot11.advertisedssid.dot11e_qbss_stations": 0,
                "dot11.advertisedssid.wpa_mfp_supported": 0,
                "dot11.advertisedssid.channel": "11",
                "dot11.advertisedssid.probe_response": 0,
                "dot11.advertisedssid.ssid_hash": 1693566721,
                "dot11.advertisedssid.dot11d_country": "",
                "dot11.advertisedssid.cloaked": 0,
                "dot11.advertisedssid.dot11r_mobility": 0,
                "dot11.advertisedssid.ht_center_1": 0,
                "dot11.advertisedssid.dot11e_qbss": 0,
                "dot11.advertisedssid.wps_version": 0,
                "dot11.advertisedssid.dot11r_mobility_domain_id": 0,
                "dot11.advertisedssid.wps_config_methods": 0,
                "dot11.advertisedssid.beaconrate": 10,
                "dot11.advertisedssid.last_time": 1640690623,
                "dot11.advertisedssid.dot11e_channel_utilization_perc": 0,
                "dot11.advertisedssid.ht_mode": "HT20",
                "dot11.advertisedssid.cisco_client_mfp": 0
            }
        ],
        "dot11.device.link_measurement_capable": 0,
        "dot11.device.num_advertised_ssids": 1,
        "dot11.device.wps_m3_last": 0,
        "dot11.device.num_associated_clients": 0,
        "dot11.device.associated_client_map": {},
        "dot11.device.max_tx_power": 0,
        "dot11.device.last_bssid": "A6:D1:C0:C3:7A:2E",
        "dot11.device.num_fragments": 0,
```

```
        "dot11.device.datasize_retry": 0,
        "dot11.device.last_beacon_timestamp": 1640690623,
        "dot11.device.min_tx_power": 0,
        "dot11.device.num_retries": 0,
        "dot11.device.wpa_present_handshake": 0
    },
    "kismet.device.base.packets.rrd": {
        "kismet.common.rrd.day_vec": [
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0
        ],
        "kismet.common.rrd.blank_val": 0,
        "kismet.common.rrd.last_time": 1640691370,
        "kismet.common.rrd.serial_time": 1640691370,
        "kismet.common.rrd.minute_vec": [
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
```

```
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0
        ],
        "kismet.common.rrd.hour_vec": [
            0,
            0,
            0,
```

```
        0,
        0,
        0,
        0,
        0,
        0,
        0,
        0,
        0,
        0,
        0,
        0,
        0,
        2,
        2,
        2,
        0,
        0,
        0,
        0,
        0,
        0,
        0,
        0,
        0,
        0,
        0,
        0,
        0,
        0,
        0,
        0,
        0,
        0,
        0,
        0,
        0,
        0,
        0,
        0,
        0,
        0,
        0,
        0,
        0,
        0,
        0,
        0,
        0,
        0,
        0,
```

```
            0,
            0,
            0,
            0,
            0,
            0
        ]
    },
    "kismet.server.uuid": "3596C672-440F-11EC-B526-4B49534D4554",
    "kismet.device.base.packets.rx": 0,
    "kismet.device.base.type": "Wi-Fi AP",
    "kismet.device.base.basic_crypt_set": 2,
    "kismet.device.base.freq_khz_map": {
        "2457000": 56,
        "2452000": 48,
        "2437000": 4,
        "5085000": 39,
        "5080000": 38,
        "2472000": 116,
        "5070000": 27,
        "2462000": 115,
        "2447000": 1,
        "5075000": 36,
        "5090000": 3,
        "2467000": 87,
        "2432000": 1
    },
    "kismet.device.base.frequency": 5085000,
    "kismet.device.base.packets.tx": 0,
    "kismet.device.base.phyname": "IEEE802.11",
    "kismet.device.base.signal": {
        "kismet.common.signal.last_noise": 0,
        "kismet.common.signal.min_signal": -84,
        "kismet.common.signal.signal_rrd": {
            "kismet.common.rrd.blank_val": 0,
            "kismet.common.rrd.last_time": 1640691370,
            "kismet.common.rrd.serial_time": 1640691370,
            "kismet.common.rrd.minute_vec": [
                0,
                0,
                0,
                0,
                0,
                0,
                0,
                0,
                0,
                0,
                0,
                0,
                0,
                0,
                0,
```

```
                0,
                0,
                0,
                0,
                0,
                0,
                0,
                0,
                0,
                0,
                0,
                0,
                0,
                0,
                0,
                0,
                0,
                0,
                0,
                0,
                0,
                0,
                0,
                0,
                0,
                0,
                0,
                0,
                0,
                0,
                0,
                0,
                0,
                0,
                0,
                0,
                0,
                0,
                0,
                0,
                0,
                0,
                0,
                0,
                0,
                0,
                0
            ]
        },
        "kismet.common.signal.last_signal": -36,
        "kismet.common.signal.type": "dbm",
```

```
        "kismet.common.signal.encodingset": 1,
        "kismet.common.signal.carrierset": 1,
        "kismet.common.signal.maxseenrate": 10,
        "kismet.common.signal.min_noise": 0,
        "kismet.common.signal.max_noise": 0,
        "kismet.common.signal.max_signal": -16
    },
    "kismet.device.base.num_alerts": 0,
    "kismet.device.base.mod_time": 1640690623,
    "kismet.device.base.packets.data": 0,
    "kismet.device.base.datasize": 0,
    "kismet.device.base.last_time": 1640690623,
    "kismet.device.base.commonname": "test2",
    "kismet.device.base.channel": "11",
    "kismet.device.base.related_devices": {},
    "kismet.device.base.name": "test2",
    "kismet.device.base.packets.crypt": 0,
    "kismet.device.base.packets.filtered": 0,
    "kismet.device.base.seenby": [
        {
            "kismet.common.seenby.num_packets": 571,
            "kismet.common.seenby.uuid": "5FE308BD-0000-0000-0000-28EE52BCFED5",
            "kismet.common.seenby.first_time": 1640690153,
            "kismet.common.seenby.last_time": 1640690623
        }
    ],
    "kismet.device.base.packets.llc": 571
```


Anexo B - Guida de funcionalidades da aplicação

Neste guia estão explicadas cada uma das funcionalidades da plataforma.

- **Login**

A página inicial da plataforma é a página de Login (Figura 71), sendo que, sem efetuar o login, não é possível ter acesso aos dados da plataforma.

A password do utilizador está guardada na base de dados *Mysql* e encriptada pela função de *hashing bcrypt*.

Figura 71 - Front-end Login

- **Dashboard**

O dashboard (Figura 72) é o ecrã para onde o utilizador é redirecionado quando efetua o log in com sucesso. Aqui é possível visualizar estatísticas dos seguintes dados:

- Número total de dispositivos;
- Número de redes encontradas;
- Número total de dispositivos clientes;
- Quantidade de *beacons* de todos os dispositivos clientes;
- Número total de dispositivos AP;
- Número total de alarmes da plataforma.

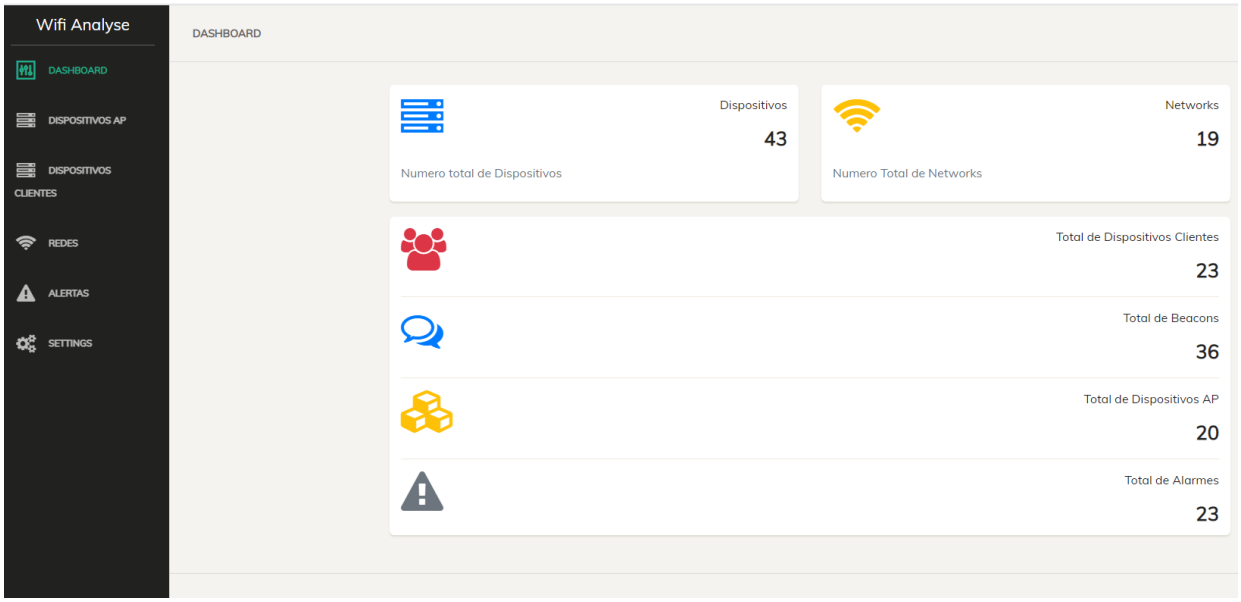


Figura 72 - Front-end Dashboard

Existe também um menu de navegação lateral onde é possível percorrer as várias funcionalidades da plataforma, onde a opção de *Settings* apenas está disponível se o utilizador for do tipo *SuperAdmin*.

- **Dispositivos AP**

Na página dos dispositivos de *Acess Point* são listados todos os dispositivos desse tipo juntamente com alguns dados mais relevantes, como o *Mac Address* ou a data em que foi visto pela última vez (Figura 73).

É possível filtrar por data pesquisando, por exemplo, apenas os AP visto na última hora ou no último dia.

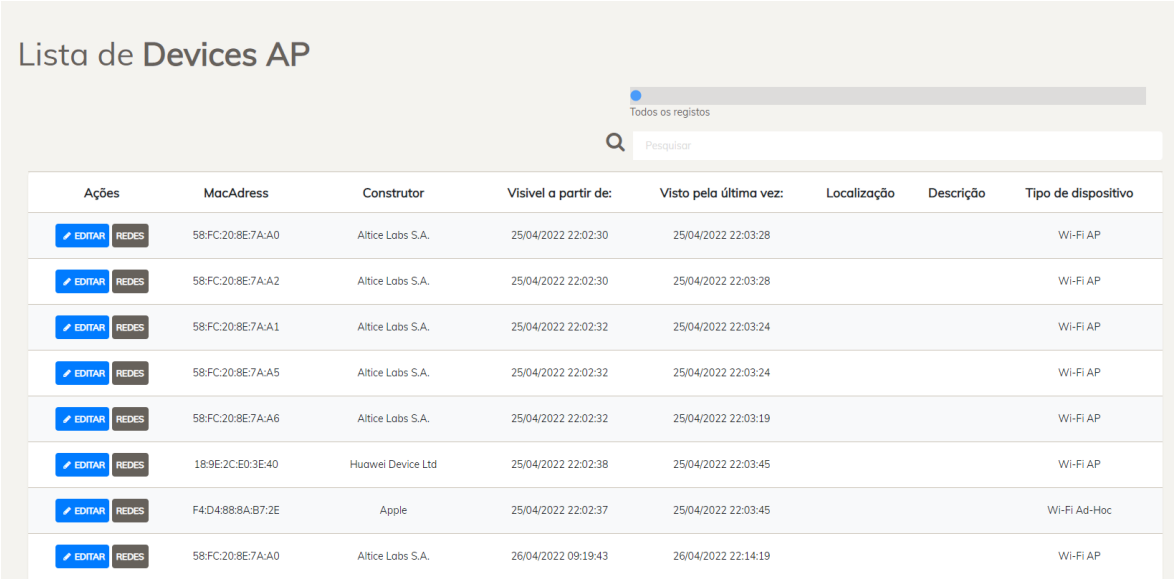


Figura 73 - Front-end Lista de devices AP

Se se optar por editar/visualizar os dados de um AP específico, ser-se-á direcionado para a página de dados do AP (Figura 74) onde são visíveis os dados do dispositivo. Nesta página é possível editar os dados no AP nomeadamente a sua localização e descrição.

Figura 74 - Front-end mostrar device AP

As redes dos dispositivos AP também podem ser consultadas ao clicar no botão “redes” na lista de dispositivos AP. Neste ecrã é possível ver vários dados de cada rede nomeadamente a frequência e a força do sinal, entre outros.

- **Dispositivos Cliente**

A informação sobre os dados clientes é uma parte bastante importante do presente projeto e, como tal, existe uma listagem de dispositivos clientes onde é possível observar uma listagem dos mesmos. Caso o dispositivo cliente não se tenha ligado a nenhuma rede, ele ainda vai aparecer nesta lista de dispositivos. É possível filtrar a tabela da Figura 75 por data ou por qualquer elemento da tabela, nomeadamente por *Mac Address*.

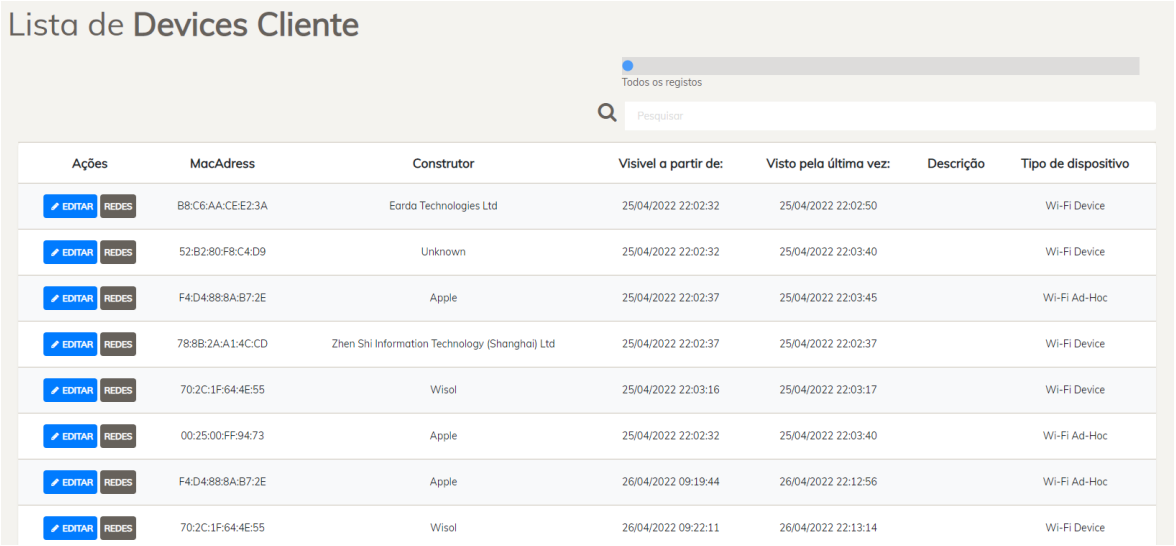


Figura 75 - Dispositivos Clientes

Existe ainda a possibilidade de ver as informações de apenas um cliente (Figura 76), clicando em "editar" na tabela de clientes. Aqui é possível escrever uma descrição para o dispositivo selecionado, por exemplo, a quem o mesmo pertence.

Um dos dados importantes destes dispositivos são os *beacons* que enviam. Com esta informação é possível perceber que aquele dispositivo esteve ligado numa rede específica. No ecrã de informações do cliente existe uma lista de *beacons* que este dispositivo emitiu.

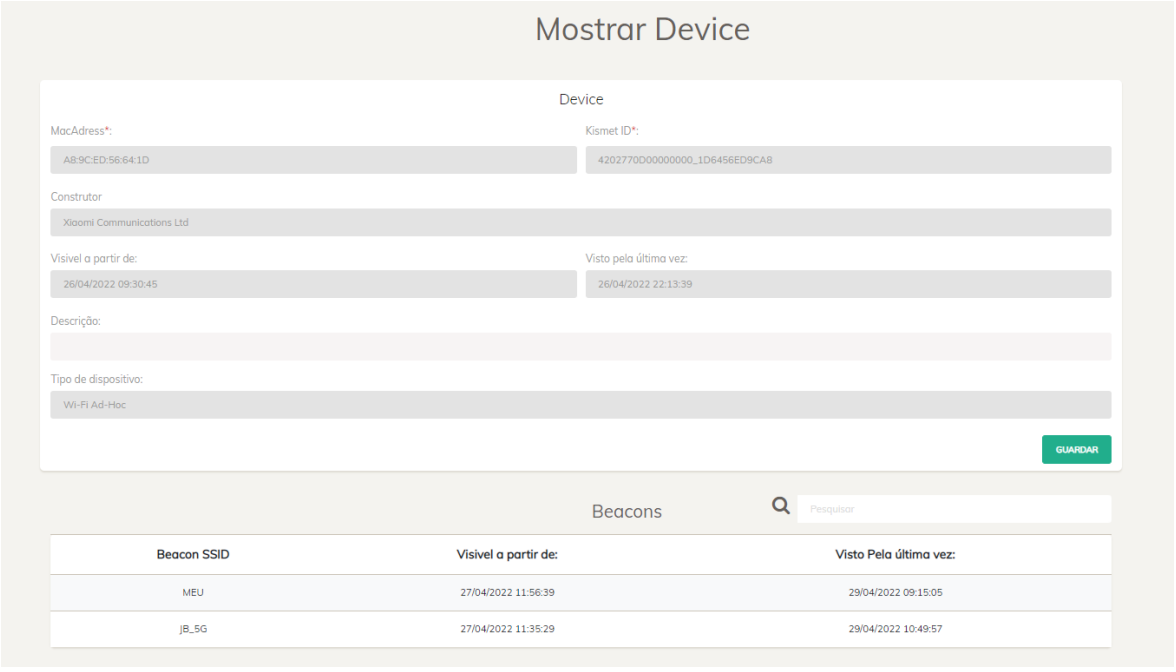


Figura 76 - Front-end mostrar device cliente

É possível ainda visualizar as redes com que cada dispositivo cliente se ligou. Um dispositivo pode ter-se ligado a várias redes, então existe uma tabela para agregar esta informação (Figura 77).

Figura 77 - Front-end mostrar redes cliente

- **Redes dos APs**

No menu de navegação existe uma opção denominada por redes, que permite mostrar todas as redes transmitidas pelos *Access Points* em forma de tabela (Figura 78). Tal como nas tabelas anteriores, existe a possibilidade de pesquisar por qualquer elemento e ainda a possibilidade de filtrar por data.

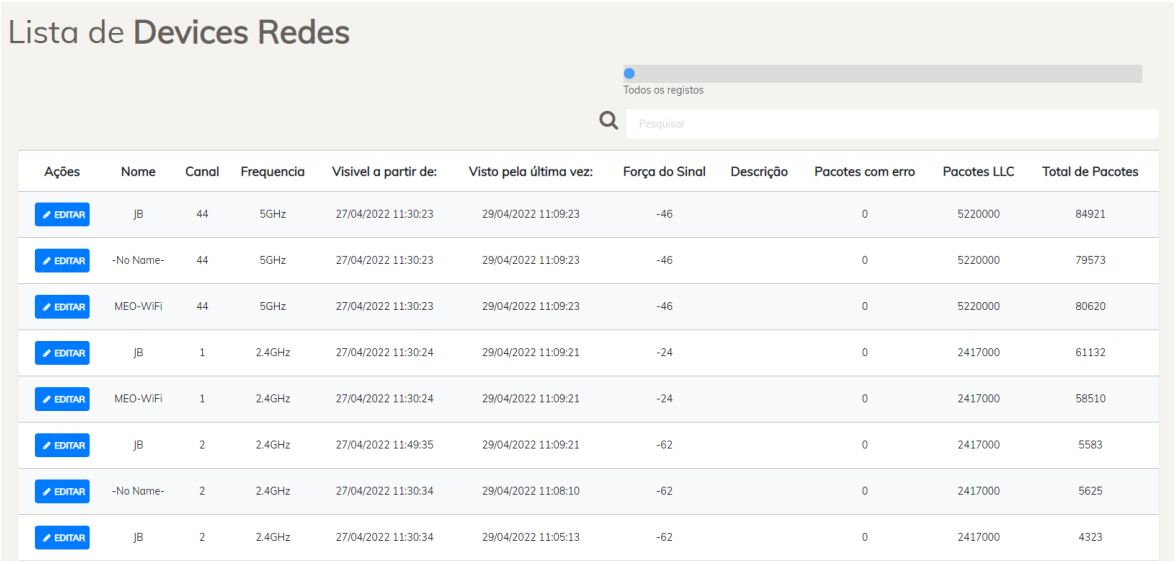


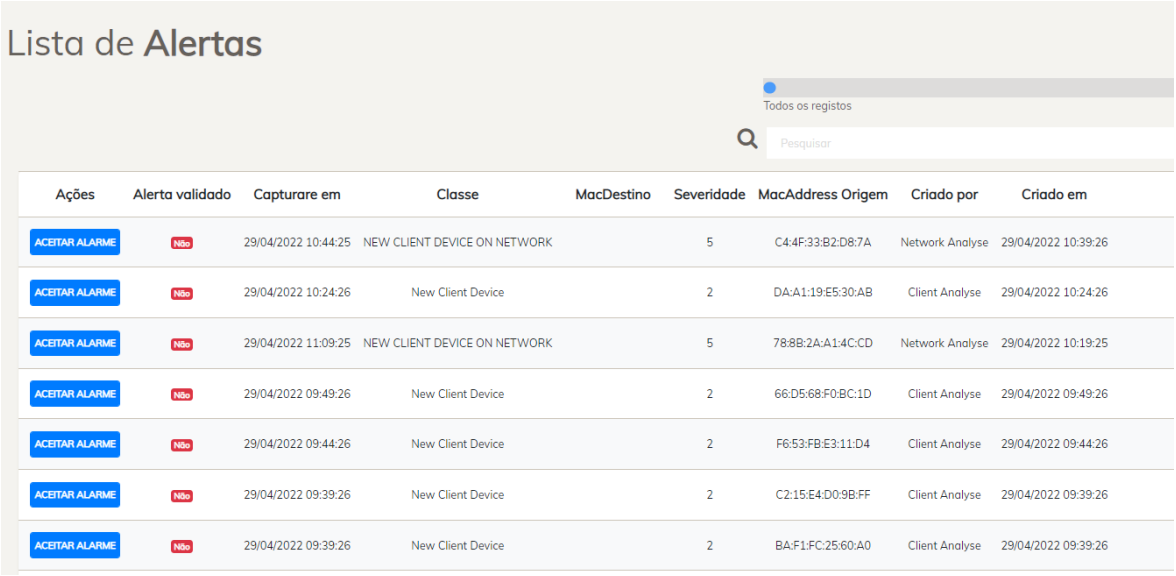
Figura 78 - Front-end listar redes AP

Se o utilizador clicar no botão de editar vai ser redirecionado para a página de edição de rede (Figura 79) que pode ser especialmente importante para alterar informação sobre aquela rede, nomeadamente a sua descrição ou a localização.

Figura 79 - Front-end editar redes AP

- **Alarmes**

A alarmística é um componente fundamental neste projeto, pois é assim que o utilizador da aplicação percebe o que está a acontecer na sua rede. Existe um componente denominado por Alarmes que mostra todos os alarmes da plataforma em forma de tabela (Figura 80). É ainda possível verificar se um determinado alerta já foi validado pelo utilizador.



Ações	Alerta validado	Capturare em	Classe	MacDestino	Severidade	MacAddress Origem	Criado por	Criado em
ACEITAR ALARME	Não	29/04/2022 10:44:25	NEW CLIENT DEVICE ON NETWORK		5	C4:4F:33:B2:D8:7A	Network Analyse	29/04/2022 10:39:26
ACEITAR ALARME	Não	29/04/2022 10:24:26	New Client Device		2	DA:A1:19:E5:30:AB	Client Analyse	29/04/2022 10:24:26
ACEITAR ALARME	Não	29/04/2022 11:09:25	NEW CLIENT DEVICE ON NETWORK		5	78:8B:2A:A1:4C:CD	Network Analyse	29/04/2022 10:19:25
ACEITAR ALARME	Não	29/04/2022 09:49:26	New Client Device		2	66:D5:68:F0:BC:1D	Client Analyse	29/04/2022 09:49:26
ACEITAR ALARME	Não	29/04/2022 09:44:26	New Client Device		2	F6:53:FB:E3:11:D4	Client Analyse	29/04/2022 09:44:26
ACEITAR ALARME	Não	29/04/2022 09:39:26	New Client Device		2	C2:15:E4:D0:9B:FF	Client Analyse	29/04/2022 09:39:26
ACEITAR ALARME	Não	29/04/2022 09:39:26	New Client Device		2	BA:F1:FC:25:60:A0	Client Analyse	29/04/2022 09:39:26

Figura 80 - Front-end listar alarmes

Existe também a possibilidade de o utilizador receber um email ou uma SMS dependendo da severidade do alarme. Estes 2 tipos de alarmística têm de ser configurados pela página de settings.

- **Back-office**

O *back-office* destina-se exclusivamente a utilizadores do tipo *SuperAdmin*. Caso o utilizado seja do tipo menciona, é possível aceder ao ecrã de settings. Neste componente é possível alterar configurações de sistema.

- **Settings**

O componente das *settings* guarda informação relevante e transversal a toda a aplicação. Na Figura 81 é possível ver algumas das settings mais importantes, tal como o tempo entre os pedidos ao servidor do Kismet para obter novas informações, ou se o utilizador deve enviar um email quando existem alarmes com a severidade maior que um certo valor.

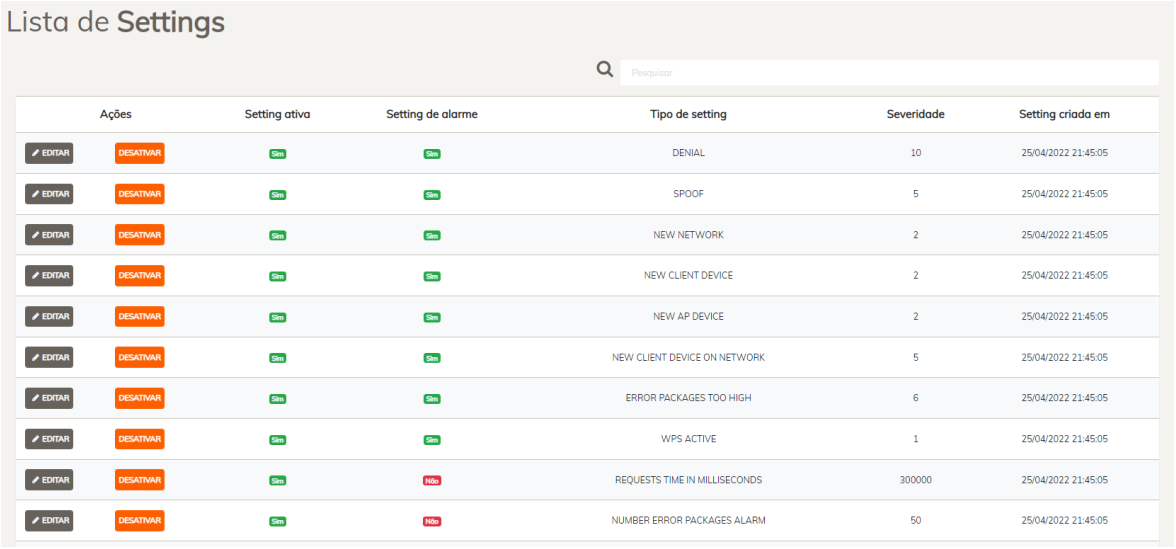


Figura 81 - Back-end listar settings

Todos os alarmes podem ser editados, como mostra a Figura 82, especificando se é pretendido ou não que aquele determinado evento crie um novo alarme ou a severidade do mesmo. Para uma determinada organização, a severidade de um evento poderá ser mais relevante do que para outras.

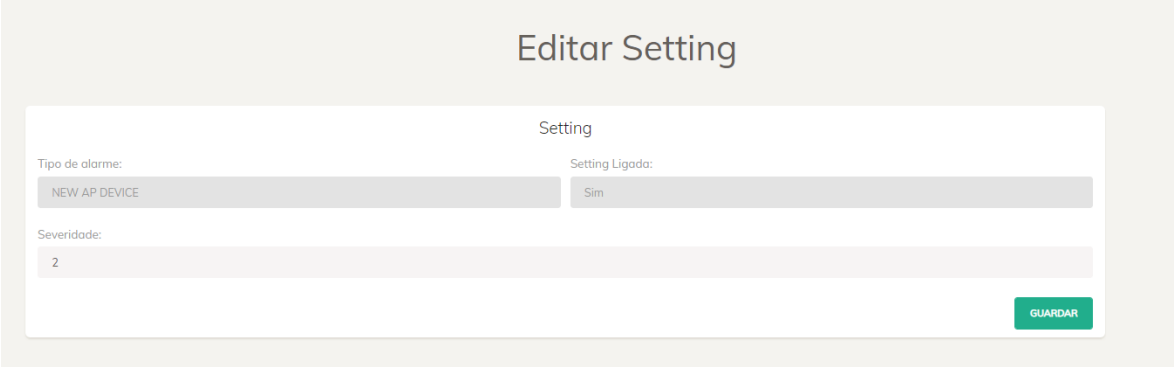


Figura 82 - Back-end editar setting

Caso estejamos a editar uma *setting* que contenha uma lista de endereços (Emails ou SMS) é possível ver/editar os destinatários dos mesmos.

Anexo C - Conversão de Json para objeto Java de um Device AP

```
public DeviceAP jsonToDeviceAP(String response) {
    ObjectMapper mapper = new ObjectMapper();
    JsonNode j = null;
    try {
       j = mapper.readTree(response);
    } catch (JsonProcessingException e) {
       e.printStackTrace();
    }

    DeviceAP deviceAP = new DeviceAP();

    deviceAP.setKismetID(j.get("kismet.device.base.key").textValue());

    deviceAP.setMacAddress(j.get("kismet.device.base.macaddr").textValue());

    if (j.get("kismet.device.base.manuf") != null) {
       deviceAP.setManufacturer(j.get("kismet.device.base.manuf").textValue());
    }
    if (j.get("kismet.device.base.type") != null) {
       deviceAP.setDeviceType(j.get("kismet.device.base.type").textValue());
    }
    if (j.get("kismet.device.base.seenby") != null) {
       Iterator<JsonNode> iterator = j.get("kismet.device.base.seenby").iterator();
       while (iterator.hasNext()) {
          JsonNode nodeIterator = iterator.next();
          if (nodeIterator.get("kismet.common.seenby.first_time") != null) {
             long epochfrist =
nodeIterator.get("kismet.common.seenby.first_time").longValue();
             deviceAP.setFirstAppearanceDate(new Date(epochfrist *
1000).toInstant());
          }
          if (nodeIterator.get("kismet.common.seenby.last_time") != null) {
```

```
                long epochlast =
nodeIterator.get("kismet.common.seenby.last_time").longValue();
                deviceAP.setLastAppearanceDate(new Date(epochlast *
1000).toInstant());
            }
          }
      }
      ///ADD NETWORKS
      if (j.get("dot11.device") != null &&
j.get("dot11.device").get("dot11.device.advertised_ssid_map") != null) {

deviceAP.getApNetworks().addAll(apNetworkService.jsonToNetwork(response,
deviceAP.getMacAddress()));
      }
      if (settingService.isSettingAlarmEnabled("NEW AP DEVICE")) {
         isNewAPAlarm(deviceAP);
      }
      deviceAP.setCreated_at(Instant.now());
      Optional<DeviceAP> getLastDevice =

this.findFirstByKismetIDAndLastAppearanceDate(j.get("kismet.device.base.key").te
xtValue(),
                  deviceAP.getLastAppearanceDate());
      if (getLastDevice.isEmpty()) {

         return this.saveDeviceAP(deviceAP);
      }

      deviceAP.setId(getLastDevice.get().getId());
      deviceAP.setApNetworks(getLastDevice.get().getApNetworks());
      return this.saveDeviceAP(deviceAP);
   }
```

Anexo D - Conversão de Json para objeto Java de alerta

```
public Alert toKismetAlert(String response) {
    ObjectMapper mapper = new ObjectMapper();
    JsonNode j = null;
    try {
       j = mapper.readTree(response);
    } catch (JsonProcessingException e) {
       e.printStackTrace();
    }
    Alert alert = new Alert();

    alert.setDescription((j.get("kismet.alert.text").textValue()));
    String classType = j.get("kismet.alert.class").toString();
    String classTypeWithoutQuotes = classType.replace("\"", "");
    alert.setClassType(classTypeWithoutQuotes);
    alert.setSeverity(settingService.getSeverityByAlarmType(alert.getClassType()));
    alert.setTransmitter(j.get("kismet.alert.transmitter_mac").textValue());
    alert.setSourceMAC(j.get("kismet.alert.source_mac").textValue());
    alert.setDestinationMAC(j.get("kismet.alert.dest_mac").textValue());
    alert.setHeader(j.get("kismet.alert.header").textValue());
    alert.setCreated_at(Instant.now());
    alert.setCreated_by("Kismet");
    if (j.get("kismet.alert.timestamp") != null) {
       long epoch = j.get("kismet.alert.timestamp").longValue();
       alert.setCaptured_at(new Date(epoch * 1000).toInstant());
    }
    if (alert.getCaptured_at().getEpochSecond() + (kismetTimeBetweenRequests /
10) + 120 >
          Instant.now().getEpochSecond()) {
       alert.setIsAlertAccepted(false);
       if (settingService.isSettingAlarmEnabled(alert.getClassType())) {
          Optional<Alert> isNewAlert = this.isNewAlertOrUpdate(alert);
          if (isNewAlert.isEmpty()) {
```

```
                return this.saveAlert(alert);
            } else {
                alert.setId(isNewAlert.get().getId());
                this.saveAlert(alert);
            }
        }
    }
    return alert;
}
```

Anexo E - Guia de Instalação do cenário de desenvolvimento

O cenário de desenvolvimento apresenta alguma complexidade na sua instalação devido aos vários módulos que utiliza. Este guia explica passo a passo o procedimento para o instalar/configurar, por forma a ser possível replicar o cenário utilizado. Podem ser utilizadas várias máquinas ou simplesmente colocar todos os serviços a funcionar no mesmo computador, dependendo do propósito da instalação. A ordem em que os componentes são instalados é importante e não deve ser ignorada. Caso não se pretenda seguir essa ordem, algumas máquinas terão de ser reiniciadas para garantir que o sistema comunica entre si. Cada sistema tem pelo menos 1GB de RAM, sendo que, para um cenário real, dependendo do número de pedidos/clientes, poderá não ser suficiente.

- **Servidor Kismet**

Sistema Operativo: Ubuntu 20.04

1. Instalar dependências necessárias para a instalação do Kismet com o comando da Tabela 41:

Tabela 41 - Instalar Dependncias Kismet

```
sudo apt install build-essential git libwebsockets-dev pkg-config zlib1g-dev libnl-3-dev libnl-genl-3-dev libcap-dev libpcap-dev libnm-dev libdw-dev libsqlite3-dev libprotobuf-dev libprotobuf-c-dev protobuf-compiler protobuf-c-compiler libsensors4-dev libusb-1.0-0-dev python3 python3-setuptools
```

```
python3-protobuf python3-requests python3-numpy python3-serial python3-usb python3-dev python3-websockets librtlsdr0 libubertooth-dev libbtbb-dev
```

2. Adicionar o repositório do Kismet e instalar mesmo utilizando o comando da Tabela 42:

Tabela 42 - Adicionar Repositório e Instalar Kismet

```
$ wget -O - https://www.kismetwireless.net/repos/kismet-release.gpg.key | sudo apt-key add -
$ echo 'deb https://www.kismetwireless.net/repos/apt/git/focal focal main' | sudo tee /etc/apt/sources.list.d/kismet.list
$ sudo apt update
$ sudo apt install kismet
```

3. Para inicializar o Kismet basta utilizar o seguinte comando na bash, Tabela 43:

Tabela 43 - Inicializar Kismet

```
$kismet
```

Quando o Kismet for inicializado pela primeira vez através do endereço http://IP:2501 vai ser pedido um *Username* e *Password*, credenciais que vão ser utilizadas mais tarde para enviar dados pelo Kismet Capture e também pela API do Kismet.

É aconselhável a criação de um mecanismo que dê *start* ao Kismet assim que esta máquina arrancar/reiniciar, por exemplo um script em *bash* a ser chamado pelo *crontab*.

- **Kismet Capture**

Sistema Operativo: Ubuntu 20.04

Esta máquina tem de ter obrigatoriamente uma placa WiFi que seja possível de colocar em modo monitor e outra placa que permita a ligação à rede. (outra placa WiFi ou uma placa de rede por cabo)

Existem duas possibilidades para a instalação do Kismet Capture:

1. É possível instalar o Kismet na sua totalidade, tal como foi a instalação do Kismet Servidor no ponto anterior e posteriormente utilizar apenas a parte do Kismet Capture. Esta é a opção mais segura porque são instaladas todas as dependências.
2. É também possível instalar apenas o Kismet Capture, sendo que este não foi o método utilizado no cenário de testes.

Assim que o Kismet estiver instalado, é necessário começar a capturar e enviar os dados Wifi e, para tal, é utilizado o comando da Tabela 44:

Tabela 44 - Wifi Remote Captura

kismet_cap_linux_wifi --connect 164.90.195.33:2501 (https://164.90.195.33:2501/) --user joel --password figueiredo --source=wlx28ee52bcfed5:name=remote-Twlan1

Neste comando é necessário identificar o endereço IP do servidor de Kismet, o seu *username* e *password*, juntamente com a placa WiFi que vai fazer a captura. Neste caso o nome da placa WiFi seria wlx28ee52bcfed5.

Assim que for executado o comando, se ambas as máquinas tiverem comunicação, Kismet e Kismet Capture, o servidor começa a receber dados enviados pelo Kismet Capture.

- **Servidor de base de dados MySQL**

Sistema Operativo: Ubuntu 20.04

Foi utilizado o seguinte guia para instalação da base de dados mysql:

https://www.digitalocean.com/community/tutorials/how-to-install-mysql-on-ubuntu-20-04-pt

Assim que o MySQL estiver instalado é necessário criar uma nova base de dados onde a aplicação em Java irá guardar os seus dados.

O nome da base de dados tem de ser o mesmo que irá estar no ficheiro de configurações do Java, neste caso kismetFilteredDB.

- **Servidor de Java Tomcat**

Sistema Operativo: Ubuntu 20.04

Foi utilizado o guia para instalar o servidor de Java (versão 11) Tomcat:

https://www.digitalocean.com/community/tutorials/how-to-install-apache-tomcat-10-on-ubuntu-20-04

No entanto, existem duas pequenas alterações ao guia mencionado acima. No ficheiro opt/tomcat/webapps/manager/META-INF/context.xml, em vez de ser comentado o código como era mencionado no guia, vai ser alterado para o conteúdo da Tabela 45:

Tabela 45 - Alteração ficheiros Tomcat

```
<Valve className="org.apache.catalina.valves.RemoteAddrValve"
        allow="^.*$" />
```

A segunda alteração é no ficheiro /opt/tomcat/conf/tomcat-users.xml, e deve ser adicionado o seguinte, Tabela 46:

Tabela 46 - Alteração ficheiros Tomcat Users

```
<role rolename="admin-gui"/>
<role rolename="admin-script"/>
<role rolename="manager-status"/>
<role rolename="manager-script"/>
<role rolename="manager-gui"/>

<user username="user" password="pass" roles="admin-script,admin-gui,manager-script,manager-gui,manager-status"/>
```

Assim que o Tomcat estiver a funcionar é necessário fazer o upload do ficheiro WAR que contém todos os dados do nosso software de Java. No entanto, antes de fazer upload deste ficheiro, é necessário ajustar alguns parâmetros para garantir que é possível comunicar com o servidor de Kismet e a base de dados correta.

No projeto Java, no ficheiro resources/application.properties, que é visível na Tabela 47, estão todos os dados necessários para efetuar conceções externas a outras aplicações, nomeadamente, IPs, portos, usernames e passwords.

Tabela 47 - Ficheiro Propriedades do Java

```
server.port=8080
# MYSQL Config
spring.jpa.database-platform=org.hibernate.dialect.MySQL5InnoDBDialect
spring.jpa.hibernate.ddl-auto=update
spring.datasource.url=jdbc:mysql://jfigueiredo.pro:3306/kismetFilteredDB
spring.datasource.username=joel
spring.datasource.password=figueiredo
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
kismet.endpoint.address=http://joel:figueiredo@192.168.1.113:2501/
kismet.username=joel
kismet.password=figueiredo
kismet.cookie=9FE5BD034BAB150196EDDA3A3DA60B06
kismet.time.between.requests=600
spring.jpa.properties.hibernate.enable_lazy_load_no_trans=true
mailtrap.username=ca3d707a0446ff
mailtrap.password=0c770d8b04523e
sms.twilio.account_sid=AC94fcb4402916fb9029c02e99784e6a0b
sms.twilio.auth_token=bc6818c34e6eaa9a299acc9c5a2b8fb9
```

Dados que têm de ser alterados:

1. Comunicação com a base de dados:

   Têm de ser alterado o endereço e o nome da base de dados bem como o seu *username* e *password* (Tabela 48).

Tabela 48 - Propriedades Base de dados

```
spring.datasource.url=jdbc:mysql://192.168.1.113:3306/kismetFilteredDB
spring.datasource.username=joel
spring.datasource.password=figueiredo
```

2. Ligação ao Kismet Server

Na ligação do Kismet é necessário alterar o IP, porto, *username* e *password* da aplicação (Tabela 49).

Tabela 49 - Propriedades Kismet Server

kismet.endpoint.address=http://username:password@164.90.195.34:2501/ kismet.username= username kismet.password= password

3. Ligação com o servidor de email:

Os emails são simulados utilizando o serviço mailtrap. Para garantir que existe comunicação com esta aplicação, é necessário alterar o *username* e *password* (Tabela 50).

Tabela 50 - Propriedades Mailtrap

mailtrap.username=ca3d707a0446ff mailtrap.password=0c770d8b04523e

4. Ligação com o servidor de SMSs:

O serviço de SMS tem de ser configurado com um account_sid e um auth_token, como é visível na Tabela 51.

Tabela 51 - Propriedades Twilio

sms.twilio.account_sid=AC94fcb4402916fb9029c02e99784e6a0b sms.twilio.auth_token=bc6818c34e6eaa9a299acc9c5a2b8fb9

Assim que todas estas configurações estiverem efetuadas, é necessário executar o comando maven que nos vai gerar o ficheiro WAR (Tabela 52):

Tabela 52 - Maven Gerar WAR

mvn package

Este comando vai apagar o ficheiro anterior e criar um novo ficheiro do tipo WAR com todo o conteúdo Java no seu interior.

Quando este processo estiver efetuado, é necessário fazer upload do ficheiro WAR para o servidor de Tomcat, instruções que estão integradas no guia de instalação do tomcat.

- **Servidor de Web Apache**

Sistema Operativo: Ubuntu 20.04

Foi utilizado o seguinte guia para instalar o servidor apache:

https://www.digitalocean.com/community/tutorials/how-to-install-the-apache-web-server-on-ubuntu-20-04-pt

Quando o servidor estiver instalado é necessário copiar os ficheiros web para a pasta /var/www/domainName/

Anexo F - Especificações dos Servidores

Especificações técnicas dos servidores e máquinas utilizadas durante a fase de testes:

- Máquina de Remote Capture para redes Wifi

Maquina Virutal: VMware VMware® Workstation 16 Pro 16.0.0 build-16894299

CPU: 2 Cores

RAM: 4GB

Rede: VM Network Card (bridge mode)

WiFi: TP-Link Archer T2U Dual Band Wireless AC600

Disco: 30GB

SO: Ubuntu 20.04

- Servidor Kismet para receber as capturas remotas e Servidor Java (Tomcat) e servidor de Front-end (Apache)

Marca: LENOVO

Versão: ThinkCentre M72e

CPU: Intel(R) Core(TM) i3-2100 CPU @ 3.10GHz

RAM: 8GB SODIMM DDR3 Synchronous 1333 MHz

Rede: RTL8111/8168/8411 PCI Express Gigabit Ethernet Controller

Disco: KINGSTON SV300S3 120GB

SO: Ubuntu 20.04