Facultade de Informática

UNIVERSIDADE DA CORUÑA

TRABALLO FIN DE GRAO
GRAO EN ENXEÑERÍA INFORMÁTICA
MENCIÓN EN COMPUTACIÓN

WIFI-HASHING
Recopilación automatizada de hashes y análisis de patrones de contraseñas en redes Wi-Fi

Estudiante: José Pablo Galego Carro
Directores: Adrián Carballal Mato
Carlos Fernández Lozano

A Coruña, 3 de septiembre de 2019.

A quienes intentan que el conocimiento y la educación estén por encima del dinero

Agradecimientos

A mis tutores, Adrián y Carlos, por la idea del trabajo y por la libertad para ejecutarla.
A todos los buenos profesores/as que me han dado clase estos cuatro años.
A mis compañeros y compañeras, por hacer la vida en la Facultad más llevadera.
A mi familia, amigos y amigas, por aguantarme.

Resumen

En este trabajo se busca ofrecer una instantánea de la seguridad de los puntos de acceso Wi-Fi del Área Metropolitana de A Coruña. Primero se discuten las opciones para la obtención de una herramienta que permita la recogida y almacenamiento de información auditable de redes Wi-Fi, desde localización a intensidad de la señal, protocolo de seguridad o la lista de clientes conectados. Posteriormente se efectúa un análisis que procura la identificación de patrones de contraseñas en redes Wi-Fi con protocolos de seguridad WEP, WPA y WPA2. Para ello se ha utilizado una herramienta de recuperación de contraseñas llamada Hashcat que permite ejecutar ataques de diccionario o de fuerza bruta, entre otros, con diversas colecciones de palabras. Al final, la cobertura de los puntos de acceso de los que se ha descifrado la contraseña se visualiza en un mapa de calor que representa varios niveles de calidad de la señal según el alcance.

Abstract

This paper seeks to provide a snapshot of the security of Wi-Fi access points in the metropolitan area of A Coruña. First, we discuss the options for obtaining a tool that allows the collection and storage of auditable information from Wi-Fi networks, from location to the signal strength, security protocol or the list of connected clients. Subsequently, an analysis is carried out that seeks to identify password patterns in Wi-Fi networks with WEP, WPA and WPA2 security protocols. For this purpose, a password recovery tool called Hashcat has been used to execute dictionary or brute force attacks, among others, with various word collections. In the end, the coverage of the access points from which passwords have been decrypted is displayed on a heat map that represents various levels of signal quality depending on the signal strength.

Palabras clave:

- WEP
- WPA/WPA2
- Seguridad Wi-Fi
- *Cracking* de contraseñas
- Hashcat

Keywords:

- WEP
- WPA/WPA2
- Wi-Fi security
- Password cracking
- Hashcat

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

1.1 Motivación

Decir que los estudios sobre la seguridad de los sistemas siempre son necesarios es una obviedad que por sí sola ya justificaría este trabajo, sin embargo, cuando lo que se prueba es la seguridad de un sistema que casi literalmente ”está en todas partes” entonces se hace más que imperativo. Se explica en esta sección por qué el Wi-Fi es una tecnología omnipresente en la actualidad y cómo las distintas iteraciones en los protocolos de seguridad de las contraseñas han hecho que, de facto, el eslabón más débil de la cadena sean las y los usuarios.

1.1.1 Nos rodean los Wi-Fis

Según el INE [1], a 2018, el 86,1% de los hogares españoles (un poco más de 14 millones) tiene acceso a Internet de banda ancha a través de las distintas tecnologías disponibles: fibra óptica o ADSL, Internet móvil (3G, 4G o 5G), satélite, etc. Con la excepción del Internet móvil, las demás tecnologías se ocupan de llevar la conexión hasta casas y organizaciones pero no proporcionan por sí mismas el acceso a Internet para los dispositivos, es necesario añadir un punto de acceso que los comunique. A día de hoy el dispositivo por excelencia para este cometido es lo que se conoce por router, que combina las funcionalidades de módem y de router propiamente dicho para crear una red privada a la que se conectan los distintos dispositivos, bien a través de cables Ethernet o bien de Wi-Fi.

Desde el año 2000, momento de la aparición de los primeros routers que incorporaban la tecnología Wi-Fi, ésta se ha convertido indiscutiblemente en la forma más popular de conectarse a la Red en el entorno doméstico y empresarial, habiendo dos hitos que representan perfectamente esta popularidad: en 2005 se añade el término Wi-Fi al diccionario Merriam-Webster y en 2012 ya se había implantado en el 25% de los hogares de todo el mundo [2].

El Wi-Fi facilitó la conexión a Internet de dispositivos muy diversos sin la necesidad de

estar físicamente enlazado por cable al router. Las ventajas son evidentes: abaratamiento y facilidad de implantación de redes LAN, proliferación de aplicaciones y dispositivos móviles, posibilidad de crear espacios con conectividad de manera inmediata, movilidad de usuarios, etc. [3]. Sin embargo, como contrapartida, la conexión se volvió potencialmente más peligrosa porque quien quisiera atacar la red ya no necesitaba acceso físico al cable sino que los datos se transmitían por un medio público, el "aire" (o, técnicamente, el espectro radioeléctrico). Cualquiera en el rango de una red inalámbrica podría intentar acceder a ella.

El primer protocolo de seguridad que pretendió paliar esta vulnerabilidad de las redes Wi-Fi fue el WEP (*Wired Equivalent Privacy*), lanzado a la vez que los primeros productos con la certificación de la Wi-Fi Alliance. Sin embargo, a finales de 2001 el protocolo ya se consideraba inseguro [4] y se desarrolló WPA con la intención de que todo dispositivo que soportase cifrado WEP pudiera soportar el nuevo protocolo con una simple actualización de firmware, una solución de compromiso mientras se desarrollaba el estándar WPA2. El protocolo WPA mantuvo algunos de los mismos problemas y vulnerabilidades que el WEP (ataque "chop chop" [5], inyección de paquetes [6], etc.) pero para lo que nos ocupa en este trabajo, es decir, la seguridad de las contraseñas de puntos de acceso, sí supuso un considerable avance con la introducción del *handshake* de cuatro vías.

En 2004 comienzan a certificarse los primeros dispositivos compatibles con WPA2 y a partir de 2006 todos los nuevos dispositivos que deseasen obtener la certificación Wi-Fi debían implementarlo. A 2019, WPA2 es el protocolo más extendido pero ya no el más reciente ni seguro. Durante estos años se han ido descubriendo vulnerabilidades (descifrado de las claves de grupo [7] o el famoso KRAK [8] que, como con WPA, si bien no comprometen la seguridad de la clave, sí ponen en peligro el tráfico entre cliente y punto de acceso.

Para paliar los problemas descubiertos a WPA2, la versión 3 fue anunciada en enero de 2018 y se encuentra en la fase de implementación opcional por los fabricantes. En esta nueva versión el *handshake* de cuatro vías ha sido sustituido por un protocolo que se ha llamado "autenticación simultánea entre iguales" que promete ser resistente a los ataques de diccionario *offline*[1]. Con todo, ya se ha encontrado una vulnerabilidad de este protocolo en el modo de compatibilidad con WPA2 [9] que seguiría permitiendo los ataques *offline* que se ejecutan en este trabajo.

1.1.2 Sobre el *handshake* de cuatro vías o por qué tu vecino duerme un poco más tranquilo

Mientras cada año se descubrían nuevas vulnerabilidades de los protocolos WPA/WPA2, ninguna de ellas comprometía la confidencialidad de la clave de autenticación. El *handshake* de cuatro vías ha probado ser eficaz para preservar la seguridad de esa parte concreta del protocolo. Se entrará a explicar con algo más de detalle el mecanismo más adelante. Por ahora

[1] En los ataques *offline*, una vez obtenido el archivo que contiene el *hash* que se quiere descifrar no hay ninguna restricción respecto al número de posibilidades que se pueden probar hasta conseguirlo, al contrario de lo que ocurre por lo general en los servicios en Internet.

llega comprender que cada vez que un cliente quiere conectarse a un punto de acceso éstos se intercambian cuatro mensajes para confirmar que cada uno conoce la clave de autenticación. Esta clave está "hasheada", es decir, se le ha aplicado una función que transforma esa clave de tamaño arbitrario (en WPA2 de 8 a 63 caracteres) en otro conjunto de caracteres de tamaño fijo, de ahí que también se las denomine "función resumen". Un ejemplo de función *hash* se puede ver a continuación:

Entrada	**MD5(Entrada)**
Hola	f688ae26e9cfa3ba6235477831d5122e
En un lugar de La Mancha	a87fa8d67bf1067a8216e7b4a6536b23

Cuadro 1.1: Ejemplo de función *hash*

En criptografía, una cualidad deseable de una función *hash* es que no sea invertible en la práctica, es decir, que computacionalmente no sea factible calcular la función inversa y recuperar la entrada a la función *hash*. Esto es lo que ha permitido que la obtención de contraseñas desde que se ha extendido el protocolo WPA/WPA2 se base exclusivamente en el tiempo y la capacidad de cómputo de la que se disponga. El proceso para "dehashear" una contraseña frecuentemente implica tres pasos:

1. Cargar una gran cantidad de términos, combinaciones de ellos, o combinación de términos y símbolos para construir una palabra.
2. Aplicar el algoritmo de "hasheado" sobre la palabra.
3. Comparar la salida del algoritmo anterior con el *hash* que se busca descifrar.

Por tanto, sobre el papel, la única amenaza a la confidencialidad de la clave WPA/WPA2 es el avance tecnológico. Cuanto más pueden computar los dispositivos y más rápida se vuelve la entrada/salida, más claves se pueden probar en un tiempo razonable.

1.1.3 Me prometiste que diez caracteres me llegarían

"Sobre el papel, la única amenaza a la confidencialidad de la clave WPA/WPA2 es el avance tecnológico". Sobre el papel, porque con cada nueva filtración de datos acceso (como las de LinkedIn, la "Collection #1", Verifications IO, etc.), se aprende un poco más sobre los hábitos a la hora de elegir una contraseña[2]. Se tiende a usar contraseñas cortas y obvias, con combinaciones de números, letras o palabras sencillas de recordar. Cuando las páginas web (u otros sistemas) obligan a incluir un número o carácter especial se añade al final de la contraseña [12] [13] y como ya se cree haber creado una contraseña segura, en parte por desconocer las de los demás, se reutiliza en otros lugares [14] aumentando la probabilidad de que aparezca en alguna filtración.

[2]Las implicaciones éticas de utilizar este tipo de filtraciones para investigación son objeto de discusión [10] [11].

En el mejor de los casos, cuando se producen estas filtraciones las contraseñas publicadas están ”hasheadas” como en el ejemplo que se mostró en el apartado anterior. Que una contraseña ”hasheada” es más segura que una contraseña en texto plano es evidente, también lo es que hay funciones *hash* mejores que otras y que, una vez un *hash* se ha hecho público en Internet cualquiera puede intentar descifrarlo. Gracias a que la computación se distribuye entre muchos nodos, el tiempo para descifrarlo se reduce considerablemente y pronto esas colecciones son texto plano en gran parte o completamente, además de públicas. En el peor de los casos, las compañías guardaban las contraseñas directamente en texto plano y nada de esto es necesario. La información que proporcionan estas filtraciones es especialmente valiosa porque se trata de contraseñas reales con las que hacer análisis estadísticos y recopilar enormes diccionarios. En definitiva, la “vulnerabilidad” de las contraseñas WPA/WPA2 es el ser humano que la escoge.

1.2 Objetivos

La finalidad de este proyecto es, por un lado, obtener una herramienta que permita la recogida masiva y almacenamiento de información auditable de redes Wi-Fi. Principalmente para conseguirlo será necesario *automatizar* lo máximo posible las tareas de detección de los puntos de acceso que pueden ser atacados y después *distribuir* esos ataques en el máximo de adaptadores inalámbricos disponibles de forma que en ambientes de mucha concentración de puntos de accesos la herramienta esté preparada para *escalar*. Se habrá de buscar una herramienta que cumpla estos entre otros requisitos o implementar una propia.

Por otro, una fase de análisis que incluya la identificación de patrones de contraseñas en redes Wi-Fi con protocolos de seguridad WEP, WPA y WPA2. La fase de análisis debe producir un diccionario mínimo generado a partir de los patrones detectados. Los resultados de la recolección y descifrado se visualizarán a través de un mapa de calor que representará el alcance de red de cada punto de acceso cuya contraseña haya sido descifrada. De esta forma demostrar la hipótesis de que la baja calidad de las contraseñas de la zona permite ir “saltando” de punto a punto de un extremo a otro de la zona elegida para el análisis estando conectado a Internet en todo momento.

Capítulo 2

Cuestiones legales

La recopilación de *handshakes* que se ha realizado en este trabajo y el descifrado posterior no puede considerarse una actividad con relevancia penal. El artículo 197 ter del Código Penal relativo al descubrimiento y revelación de secretos dispone:

Será castigado con una pena de prisión de seis meses a dos años o multa de tres a dieciocho meses el que, sin estar debidamente autorizado, produzca, adquiera para su uso, importe o, de cualquier modo, facilite a terceros, con la intención de facilitar la comisión de alguno de los delitos a que se refieren los apartados 1 y 2 del artículo 197 o el artículo 197 bis:

a) un programa informático, concebido o adaptado principalmente para cometer dichos delitos; o

b) una contraseña de ordenador, un código de acceso o datos similares que permitan acceder a la totalidad o a una parte de un sistema de información

Esta actividad no cumple con el requisito de tipicidad penal, es decir, el hecho cometido no se ajusta a la descripción que se hace en la ley. Si bien se está "produciendo sin estar debidamente autorizado un código de acceso que permite acceder a la totalidad o a una parte de un sistema de información" (la red inalámbrica privada que crea el router Wi-Fi), esta producción no es con el "objetivo de cometer o facilitar la comisión por terceros de los delitos a que se refieren los apartados 1 y 2 del artículo 197 o el artículo 197 bis" ya que todo este proceso se realiza en un entorno separado con fines académicos y sin intención de probar físicamente en esas redes que el valor descifrado es correcto.

Capítulo 3

Planificación y costes

El propósito de la planificación de los proyectos es definir el alcance del proyecto, estimar el tiempo que habrá de dedicarse y crear un calendario durante el cual se han de realizar las tareas descritas en el plan.

	Nombre	Duracion	Inicio	Terminado	Predecesores	Nombres del Recurso
1	**Documentación**	24 days	**28/01/19 8:00**	**28/02/19 17:00**		
2	Herramientas disponibles	8 days	28/01/19 8:00	6/02/19 17:00		Analista programador[50%]
3	Python y librerías	16 days	7/02/19 8:00	28/02/19 17:00		Analista programador[50%]
4	**Script**	58 days	**1/03/19 8:00**	**21/05/19 17:00**		
5	**Primera iteración**	27 days	**1/03/19 8:00**	**8/04/19 17:00**		
6	Análisis y diseño	5 days	1/03/19 8:00	7/03/19 17:00	2;3	Analista programador[50%]
7	Codificación y pruebas	22 days	8/03/19 8:00	8/04/19 17:00	6	Analista programador[50%]
8	**Segunda iteración**	8 days	**9/04/19 8:00**	**18/04/19 17:00**		
9	Análisis y diseño	3 days	9/04/19 8:00	11/04/19 17:00	7	Analista programador[50%]
10	Codificación y pruebas	5 days	12/04/19 8:00	18/04/19 17:00	9	Analista programador[50%]
11	**Tercera iteración**	12 days	**19/04/19 8:00**	**6/05/19 17:00**		
12	Análisis y diseño	3 days	19/04/19 8:00	23/04/19 17:00	10	Analista programador[50%]
13	Codificación y pruebas	9 days	24/04/19 8:00	6/05/19 17:00	12	Analista programador[50%]
14	**Cuarta iteración**	11 days	**7/05/19 8:00**	**21/05/19 17:00**		
15	Análisis y diseño	3 days	7/05/19 8:00	9/05/19 17:00	13	Analista programador[25%]
16	Codificación y pruebas	8 days	10/05/19 8:00	21/05/19 17:00	15	Analista programador[25%]
17	Pruebas de campo y debugging	11 days	7/05/19 8:00	21/05/19 17:00	13	Analista programador[25%]
18	Auditorías	16 days	22/05/19 8:00	12/06/19 17:00	16;17	Analista programador[50%]
19	Crackeo de contraseñas	32 days	6/06/19 8:00	19/07/19 17:00	18SS	Clúster
20	Memoria del proyecto	55 days	13/06/19 8:00	28/08/19 17:00	18	Analista programador[50%]

Figura 3.1: Lista de tareas del proyecto

La figura muestra la lista de tareas y subtareas del proyecto junto con la fecha de inicio, de fin, su duración, las relaciones de precedencia y a quién se le asignó cada una.

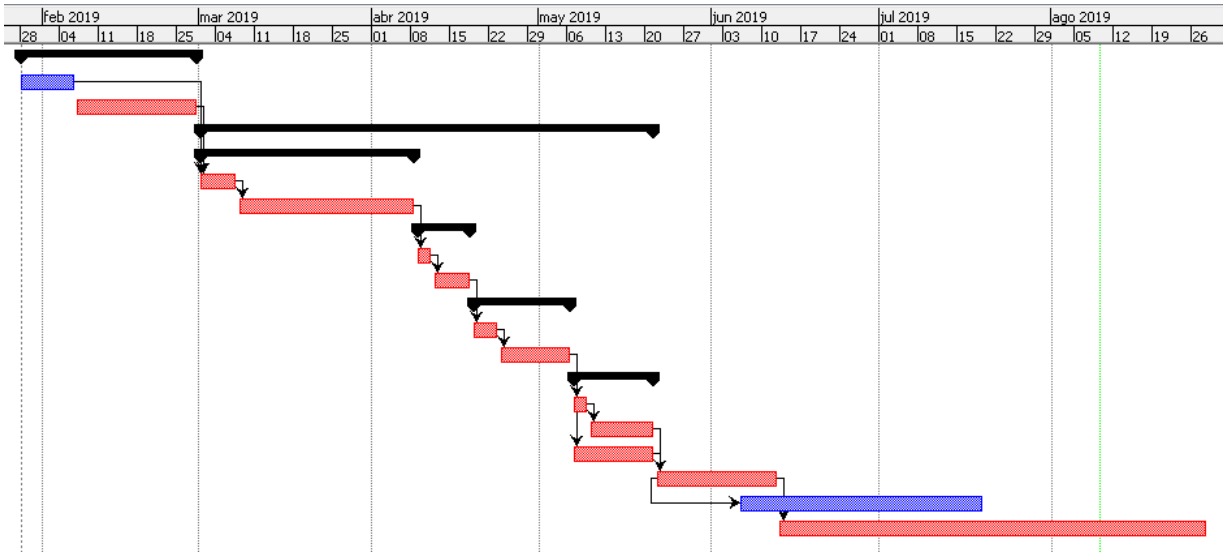

Figura 3.2: Diagrama de Gantt del proyecto

En el diagrama de Gantt se representa en rojo las tareas que representan el camino crítico, es decir, la ruta de trabajo que marca la duración del proyecto.

Los dos recursos disponibles son un/a analista programador/a a media jornada y un clúster de altas prestaciones. Las horas totales de trabajo fueron 612, a un coste de 20€ la hora el proyecto tuvo un coste de personal de 12.240€. El acceso al clúster BIOCAI se proporcionó por el grupo RNASA gratuitamente. Una estimación de lo que en el mercado puede valer una capacidad de computación ligeramente superior se hace más adelante en este documento. Se ha considerado que 2.000€ al mes es un coste razonable, por dos meses el coste material total es 4.000€. No se han tenido en cuenta los costes de ordenador, Raspberry Pi y antenas porque ya se disponía de todo ello antes de empezar el proyecto, pero sería de alrededor de 600€. Tampoco los costes fijos de electricidad e Internet. El coste final del proyecto se estima en 16.240€.

Capítulo 4

Teoría previa

Para entender mejor los conceptos que se utilizan en este trabajo conviene dedicar unas páginas a explicar qué es el *handshake* de cuatro vías, por qué se diseñó de esa forma y cómo herramientas como Hashcat ejecutan el descifrado. También se tratará qué podemos entender por *seguridad de una contraseña* y por qué debido al comportamiento de los usuarios y usuarias casi siempre se puede tener confianza en que un porcentaje no desdeñable de contraseñas serán descifradas.

4.1 *Four way handshake*

Se ha dividido la explicación del *handshake* de cuatro vías en lo que informalmente podría llamarse ”cifrado” y ”descifrado”. Dicho de otro modo, primero todo el proceso implicado en el cálculo de la PTK a partir de la PSK, paso a paso y explicando por qué las opciones más sencillas no son óptimas desde el punto de vista de la seguridad de la contraseña, para así conseguir una imagen nítida de cómo un punto de acceso (router) y un cliente confirman que ambos conocen la clave de autenticación de manera privada y segura. Después, en el segundo apartado, se explica qué se calcula en el proceso contrario de descifrado, ya que como se comprobará no se trata de ir deshaciendo uno a uno los pasos del cifrado.

4.1.1 Proceso

El proceso de autenticación a través de *handshake* de cuatro vías se produce de la siguiente manera [15]:

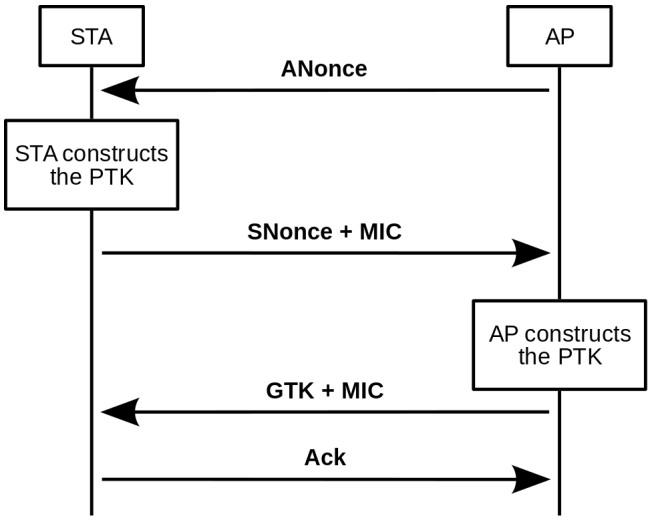

Figura 4.1: Esquema del *handshake* de cuatro vías (Wikipedia)

El principio básico detrás del *handshake* de cuatro vías es que la contraseña del punto de acceso (PSK) no puede transmitirse al medio público bajo ningún concepto. Por ello, lo primero que hace el cliente antes de iniciar el intercambio de mensajes es construir la PMK:

$$PMK = PBKDF2(HMAC\text{–}SHA1, PSK, SSID, 4096, 256)$$

Esta es la versión que usan los protocolos WPA y WPA2. Para entenderla y comprender las razones de su elección es interesante desgranarla e ir construyéndola paso a paso. Se podría empezar por una versión muy simple de esta misma función.

$$PMK = HMAC\text{–}SHA1(PSK)$$

HMAC–SHA1 es una función de *hashing* y la PSK se usa como entrada. Este proceso devuelve una cadena de caracteres de tamaño finito, el *resumen*, incoherente a ojos de un humano. Se podría usar esta cadena como PMK, sin embargo, esta aproximación tiene una amenaza muy grave: las tablas *hash*. El resultado de introducir dos palabras iguales en la función HMAC-SHA1 es igual, lo que es lo mismo que decir que dos contraseñas iguales generan el mismo *hash*. El vector de ataque es claro, generar gran cantidad de ellos con distintas contraseñas, crear una tabla *hash*. Estas tablas se distribuyen por Internet y la única operación para descifrar un *hash* es encontrarlo en esa colección.

Por tanto se hace imprescindible utilizar una función pseudoaleatoria que impida que dos contraseñas iguales generen el mismo *hash*. El resultado de aplicar una función pseudoaleatoria no es completamente aleatorio, como se intuye por el nombre, ya que está determinado por un conjunto de valores iniciales. El conjunto de valores iniciales en el protocolo WPA/WPA2 es el SSID, es decir, el nombre del punto de acceso. A este proceso se le conoce como *salt* o "saltear".

$$PMK = FUNCION\_PSEUDOALEATORIA(HMAC\text{–}SHA1, PSK, SSID)$$

Aunque esta versión es mejor que la inicial, aún se le puede añadir un punto más de complejidad. Hay que diseñar una función que repita este proceso un número determinado de veces para generar una clave de un número dado de bits. Esta función es PBKDF2 y cuando repite 4096 veces la aplicación de la función pseudoaleatoria genera una clave de 256 bits, la PMK.

$$PMK = PBKDF2(HMAC\text{–}SHA1, PSK, SSID, 4096, 256)$$

Ahora ya se puede recibir el primer mensaje del *handshake* proveniente del punto de acceso, el ANonce. Con el ANonce el cliente dispone de toda la información necesaria para construir la PTK, sólo se han de concatenar PMK, ANonce, SNonce, AMA, SMA. El resultado de la concatenación se pasa como entrada a otra función pseudoaleatoria que nos devolverá una cadena de 64 bytes (512 bits), la PTK.

$$PTK = PRF512(CONCAT(PMK, ANonce, SNonce, AMA, SMA))$$

La PTK se va a dividir en cinco partes de la siguiente forma:

PTK				
16 bytes	16 bytes	16 bytes	8 bytes	8 bytes
KCK	KEK	TK	MIC-tx	MIC-rx

Cuadro 4.1: Descomposición de la clave PTK

En este momento el cliente envía el segundo mensaje del *handshake*, compuesto por su SNonce y un MIC. Como el cliente ya dispone de la PTK, además de enviar el SNonce, aprovecha para protegerlo con un MIC. El MIC lo ha calculado a partir del SNonce y el KCK. WPA y WPA2 utilizan distintas funciones *hash* para este paso:

$$WPAMIC = HMAC(KCK, SNonce, MD5SHA)$$
$$WPA2MIC = HMAC(KCK, SNonce, SHA1)$$

Así, cuando el punto de acceso recibe SNonce y MIC puede calcular también la PTK (hasta ahora le faltaba el SNonce) y teniendo SNonce y PTK, proceder a validar el MIC (esto es, calcularlo y comparar si es igual que el que ha recibido). Si la validación del MIC es positiva, el punto de acceso confirma que él y el cliente están usando la misma PTK, por tanto la misma PMK y por tanto, la misma PSK, que es de lo que se trataba. En este punto del *handshake* quien esté interceptando los mensajes tendría la información suficiente para intentar descifrar la contraseña, es decir, los mensajes 3 y 4 del proceso no son de relevancia para lo que nos ocupa y en consecuencia se obviarán en la explicación.

4.1.2 Descifrado

Interceptado el segundo mensaje del *handshake* sólo falta la PSK para poder reconstruir la PTK. Para computarla debe presumirse una PSK. Es aquí donde finalmente se utiliza un diccionario, ataque de fuerza bruta o cualquier otro método de generación de palabras. Para cada palabra se generará la PTK correspondiente, de ésta se extraerá el KCK y se calculará el MIC. Si el MIC coincide con el MIC genuino, entonces se ha encontrado la contraseña.

Como es evidente, cada prueba con cada palabra es totalmente independiente de las demás y no es necesaria comunicación entre nodos, lo que permite la paralelización perfecta del proceso de descifrado [16]. Se trata claramente de un caso donde es preferible tener gran cantidad de procesadores de poca potencia sobre varios de mucha potencia. Las tarjetas gráficas encajan perfectamente en esta descripción.

4.2 La seguridad de las contraseñas

Existen muchos estudios sobre la idoneidad de las contraseñas como forma de autenticación en los sistemas, no sólo desde la parte técnica informática[1] sino también desde la parte de la psicología de los usuarios y usuarias.

Las contraseñas son un concepto fácil de reflejar en el diseño de sistemas, con una implementación habitualmente sencilla y fáciles de entender para quien use el sistema. Como se ha dicho, los usuarios y usuarias suelen ser el eslabón más débil de la cadena y comprometen la seguridad de los sistemas al olvidar las contraseñas, escribirlas, compartirlas con otras personas y escogerlas fáciles de adivinar. Sigue siendo una cuestión abierta al estudio en qué medida las contraseñas son débiles por falta de motivación o por limitaciones cognitivas de los seres humanos. Algunos estudios concluyen que no hay todavía evidencia convincente de que personas conscientes sobre políticas que mejoran la seguridad de las contraseñas elijan contraseñas que resistan mejor al descifrado por un/a atacante con conocimientos, ya que suelen considerarlas engorrosas y poco prácticas [18] [19]. Los intentos de educar al "usuario medio" sobre la creación de contraseñas más seguras a través del asesoramiento y la apli-

[1] Ya en 1979 la seguridad de las contraseñas en los sistemas UNIX les causaba preocupación a Robert Morris y a Ken Thompson [17].

cación obligatoria de políticas de contraseñas suelen tener poco éxito. Los requisitos de las contraseñas se cumplen mínimamente [2] o se ignoran o malinterpretan los consejos que se proporcionan para su creación [21].

Otros trabajos intuyen que la poca diferencia entre usuarios/as normales y ”conscientes” se debe a que la mayoría tiende a variar la complejidad de las contraseñas conforme varía la importancia de las cuentas que protegen. La mayoría de las personas que participa en los estudios tiene una sensación similar del compromiso al que hay que llegar entre seguridad y comodidad a la hora de elegir una contraseña y entienden que reutilizar contraseñas o usar contraseñas débiles en sus cuentas importantes podría poner en riesgo sus datos [19].

Mientras que el enfoque técnico informático sí que es aplicable al caso concreto de las contraseñas Wi-Fi, esto es, *qué hace segura una contraseña*, los resultados del enfoque piscológico en general no parecen extrapolables. Primero, porque en la mayoría de trabajos revisados [13] [19] [21] [22] [23] [24] [25] los procesos psicológicos de elección de una contraseña (tanto crearla como decidir reusar otra) implican que *el sistema* pide una contraseña. Esto es radicalmente distinto a los routers Wi-Fi donde, por defecto, la contraseña viene fijada y es *el usuario/a* el que voluntariamente la cambia. Y segundo, porque en general la contraseña Wi-Fi sólo es necesaria introducirla una vez por dispositivo, siendo responsabilidad del sistema operativo el guardarla de forma segura para cada vez que se realice una nueva autenticación.

Las estrategias más populares para reducir las debilidades inherentes al uso de contraseñas incluyen las siguientes [22]:

1. Longitudes de contraseña de al menos 8 caracteres: las contraseñas más largas aumentan el tiempo que tardan los programas de ”crackeo” en descrifrarlas.

2. Combinar aleatoriamente mayúsculas y minúsculas con símbolos especiales: la inclusión de mayúsculas, minúsculas y símbolos (!£$%, etc.) en las contraseñas requiere usar métodos de fuerza bruta y aumenta el número de permutaciones de caracteres que deben probarse. Una forma propuesta para aproximarse es crear frases nemotécnicas: ”Quiero generar una contraseña segura contra el Software de cracking” produce ”QgucsceSdc” cogiendo la primera letra.

3. Sin palabras del diccionario: para minimizar los ataques de diccionario.

4. Cambiar la contraseña periódicamente: en caso de que un intruso obtenga una contraseña válida, la mayoría de los sistemas le permitirán continuar accediendo hasta que se detecte la intrusión. Si los usuarios cambian periódicamente sus contraseñas, el intruso se verá forzado a identificar la nueva contraseña.

Respecto a 2, los humanos rara vez generan contraseñas aleatorias (un conjunto de contraseñas perfectamente al azar produciría una distribución uniforme de caracteres). El software

[2] Por ejemplo, una política que exige incluir al menos tres dígitos en una contraseña a menudo dará como resultado que el usuario/a simplemente agregue ”123” al final de una contraseña insegura. Las herramientas de *cracking* incluyen amplios conjuntos de reglas como esas [20].

de ”crackeo” optimiza los ataques de fuerza bruta utilizando tablas de frecuencia de caracteres aprovechando una propiedad inherente al lenguaje: ciertos caracteres aparecen con más frecuencia que otros. Según [25], cuando a los usuarios y usuarias se les indica que creen contraseñas basadas en frases nemotécnicas, la mayoría seleccionan frases de letras de canciones, películas, literatura o programas de televisión. El texto de estas fuentes a menudo está disponible en Internet. Esto abre la posibilidad de que se pueda construir un diccionario efectivo también para estas contraseñas.

Respecto a 4, desde hace varios años se considera que no es una recomendación correcta en general [26] [27]. Las contraseñas deben cambiarse cuando se tenga la intuición o el conocimiento de que han sido comprometidas o se hayan vuelto potencialmente inseguras debido al avance tecnológico.

Estas estrategias hacen que componer contraseñas y memorizarlas sea especialmente complicado, más teniendo en cuenta que la proliferación de servicios en Internet obligaría a manejar un buen conjunto de contraseñas si no se quiere reutilizar ninguna. Sin embargo, este es un problema que, como se ha dicho, no afecta a las contraseñas Wi-Fi ya que *sólo es necesario usarlas una vez*. Salvada esta dificultad no hay ninguna excusa para no utilizar contraseñas largas y complejas seleccionadas totalmente al azar. Sabiendo que estos requisitos son contrarios a las propiedades de la memoria humana [23], varios expertos en seguridad han argumentado que escribir la contraseña y guardarla en un lugar seguro podría ser una buena opción para el usuario medio [28]. No es de extrañar que en estudios se haya encontrado relación entre escribir las contraseñas y que éstas sean más seguras [24].

Una contraseña larga y compleja es imperativa en el caso de la PSK del protocolo WPA/WPA2, ya que el ”crackeo” de los *handshakes* se realiza *offline*. La cuestión clave es responder a la pregunta sobre *cuánto de compleja* y *cuánto de larga debe ser*, para lo que no hay una respuesta única y segura. Si partimos de la base de que buscamos generar una contraseña sólo descifrable a través de ataques de fuerza bruta, evitando palabras de diccionario, podemos pensar la siguiente aproximación práctica.

El espacio de búsqueda de la contraseña será la variación con repetición de x elementos de orden y

$$VR_{x,y} = x^y$$

donde x es la cardinalidad del conjunto de caracteres que se van a usar e y la longitud de la palabra. La cardinalidad está limitada por los caracteres permitidos por el protocolo WPA/WPA2, los caracteres ASCII imprimibles:

	Dígitos	**Letras Mayúsculas**	**Letras Minúsculas**	**Símbolos**
Cardinalidad	10	26	26	33

Puede decirse que la forma más sencilla y efectiva de aumentar la seguridad de las contraseñas para un usuario medio es aumentar la longitud. Aumentar la longitud produce un crecimiento exponencial del espacio de búsqueda, mientras que aumentar la complejidad aumentando la cardinalidad del conjunto de caracteres sólo produce un crecimiento polinomial[3]. Esto es algo que se ha comprobado empíricamente; las siguientes dos políticas de composición de contraseñas deberían dar como resultado contraseñas con la misma entropía: una que sólo requiere que las contraseñas tengan al menos 16 caracteres de longitud y otra que requiere al menos ocho caracteres pero también una letra mayúscula, un número, un símbolo y una comprobación de diccionario, de acuerdo con las mejores pautas disponibles [29]. Sin embargo, la política de los 16 caracteres produce contraseñas significativamente menos predecibles y, según varias métricas, es menos onerosa para los usuarios [24].

Otra de las variables a tener en cuenta es la amenaza, identificar *quién puede intentar descifrarla*. Esta variable puede dividirse en capacidad de computación y tiempo. La capacidad de computación c es el número de $hashes/tiempo$ que el hardware supuesto puede probar y el tiempo t es el periodo que se puede estar computando. Un ejemplo de evaluación de amenaza para un usuario medio puede ser "la hija del vecino sabe de ordenadores". Podemos suponer un hardware de consumo y dos semanas de tiempo. En Internet se suelen encontrar *benchmarks* que proveen esta proporción simplemente haciendo una búsqueda como "<Programa de cracking> <Hardware>".

Por tanto, una contraseña se puede considerar segura para el/la usuario/a en concreto

$$\text{ContaseñaSegura}(x, y, c, t) = \begin{cases} \text{Sí} & \text{Si } \frac{x^y}{c} \geq t \\ \text{No} & \text{Si } \frac{x^y}{c} < t \end{cases}$$

Esta es una estimación sencilla y práctica que tiene en cuenta el estado actual del hardware y software de *cracking* y no la seguridad intrínseca de la contraseña[4]. Además toma siempre el peor escenario para quien está atacando (la contraseña correcta es la última opción que prueba), mientras que en otros trabajos se han presentado métricas más avanzadas basadas en la distribución estadística de las contraseñas y que consideran el mejor escenario [18].

En definitiva, los dos vectores de ataque que se explotarán en este trabajo para descifrar las contraseñas Wi-Fi interceptadas serán, primero confiar en la existencia de contraseñas en un espacio de búsqueda abarcable por fuerza bruta con la infraestructura y el tiempo disponibles, segundo confiar en que el descuido de la gente haya llevado a utilizar contraseñas típicas que fueron objeto de alguna filtración. El éxito medio de estas aproximaciones suele situarse en torno al 30% en proyectos de descifrado masivo,[5] [6] mientras que los números publicados de diversos estudios sobre la eficacia de descifrado de contraseñas varían sustancialmente. Según [18]:

[3] Crecer en longitud permite además no usar caracteres especiales, que pueden producir problemas con malas implementaciones del protocolo WPA/WPA2.

[4] Para cuya medición se han propuesto diversas formas [18] [29] [30]

[5] gpuhash.me (28,70%)

[6] wpa-sec (31,93%)

- La mayoría de los estudios han descifrado el 20–50% de las contraseñas con tamaños de diccionario en el rango de las 2^{20}–2^{30} palabras.
- Todos los estudios muestran rendimientos decrecientes para diccionarios más grandes.
- Hay pocos datos sobre la eficiencia de los diccionarios pequeños, ya que la mayoría de los estudios emplean el diccionario más grande que pueden procesar.

4.3 El arte de ”crackear” contraseñas

Se van a adelantar a continuación con unos ejemplos simples algunos de los conceptos que se utilizarán a lo largo del trabajo. Tomamos contraseñas de 10 caracteres y usamos como referencia una tarjeta gráfica de consumo actual, la NVIDIA GTX1080, capaz de calcular alrededor de 400.000 *hashes* al segundo [31]. El espacio de búsqueda se corresponde con el número de posibilidades para cada combinación de caracteres ASCII: sólo numérico, alfanumérico sólo con letras minúsculas, alfanumérico combinando mayúsculas y minúsculas (mixto) y alfanumérico mixto con símbolos especiales. El tiempo para recorrer el espacio es la división entre el espacio de búsqueda y la potencia de la tarjeta gráfica, o lo que es lo mismo, el tiempo que tardaría en descifrar la contraseña si ésta fuera la última que probase de todo el espacio de búsqueda, el peor escenario.

Contraseña	Espacio de búsqueda	Tiempo para recorrer el espacio
1111111111	10.000.000.000	Alrededor de 7 horas
3548346841		
holahola12	2.758.547.353.515.625	Alrededor de 220 años
83dk12w7p1		
HolaHola12	604.661.760.000.000.000	Alrededor de 47.934 años
83Dk12w7P1		
HolaHola1#	38.941.611.811.810.745.401	Alrededor de 3.087.076 años
8#Dk1^w7P1		

Cuadro 4.2: Cálculo de contraseñas por fuerza bruta

Estos tiempos se corresponden con lo que se llama ataques de “fuerza bruta”. Las únicas variables que requiere un ataque de fuerza bruta para ejecutarse es qué caracteres son aceptables y cuál es la longitud de lo que se intenta descifrar. Por eso para este ataque no hay distinción entre “holahola12” y “83dk12w7p1” cuando cualquiera, a simple vista, tiene la intuición de que una *debe ser* más fácil de descifrar que la otra.

Esa intuición es la que condensan las colecciones de palabras o diccionarios. Se sabe que la contraseña “holahola12” es más probable estadísticamente que “83dk12w7p1”, no sólo porque “hola” es una palabra del castellano, sino porque se ha encontrado en el mundo real, alguien ya la ha usado[7]. Un ataque de fuerza bruta que conoce las dos variables mencionadas en el

[7] Según Pwned Passwords ”holahola12” se ha encontrado en 721 filtraciones.

párrafo anterior, con el tiempo necesario, siempre va a descifrar la contraseña. El problema es que muchas veces no se sabe qué caracteres se han usado, qué longitud tiene la palabra, ni se dispone del tiempo necesario, y por eso se busca un compromiso: se reduce el espacio de búsqueda a un subconjunto a cambio de reducir las posibilidades de éxito descifrando la clave.

La opción más sencilla es utilizar alguno de los diccionarios más populares que circulan por Internet, cuya extensión varía de los cientos de miles de palabras a los miles de millones. Muchos de ellos tienden hacia una mayoría de términos anglosajones pero para esta estrategia, en la que no se toma en cuenta ninguna información del objetivo, sino que se basa en la pura probabilidad, no es una desventaja. La otra opción es desarrollar un diccionario propio especializado, cuando se conocen algunos datos del objetivo como país, nombre del propietario del punto de acceso, política de longitud de contraseñas de la empresa, etc. y partir de ahí proponer unos patrones que generen las palabras del diccionario.

Por ejemplo, se va a ejecutar una auditoría en un punto de España y el patrón escogido es “dos palabras del castellano más uno o dos dígitos o símbolos especiales”. Se empezaría escogiendo las palabras: la RAE publica las más usadas del castellano [32], de ellas se cogen las 10.000 primeras (“hola” está en el número 7373) y se descartan las que tienen menos de 4 y más de 8 caracteres, lo que deja 5935, y cada una se concatena con las otras 5934 y consigo misma. Esto genera 35.224.225 palabras base. Añadirle un dígito o caracter especial genera 1.549.865.900, a lo que se suma que añadirle dos dígitos o caracteres especiales genera 68.194.099.600. Finalmente un total de 69.743.965.500 palabras, que la tarjeta gráfica de referencia puede probar en dos días en el peor de los casos.

Las dos estrategias pueden ser complementarias y, de hecho, en este trabajo lo son. Un primer barrido con diccionarios genéricos debe descifrar una cantidad suficiente de contraseñas como para poder inferir una serie de patrones con los que generar un diccionario mínimo específico para el Área Metropolitana de A Coruña. Así por una parte se dispone de nuevas palabras candidatas más especializadas y por otra no sería necesario utilizar hardware de altas prestaciones para descifrar las contraseñas de la zona.

Capítulo 5

Recolección de *handshakes* de puntos de acceso

En este capítulo se tratarán por un lado las diferentes opciones de software disponibles para interceptar *handshakes* y recopilar información de los puntos de acceso y sus dispositivos conectados junto con las restricciones de hardware de la auditoría y la selección final de aparatos para ejecutarla. Por otro, el análisis de los resultados de esta auditoría.

5.1 Desarrollo de una herramienta para adquirir información auditable de redes Wi-Fi

A continuación se enuncian los requisitos que debería cumplir la utilidad que se vaya a usar para las auditorías. Con la intención de "no reinventar la rueda" se revisan todas las posibles soluciones ya implementadas. Finalmente se exponen los pasos seguidos para crear una herramienta propia.

5.1.1 Requisitos

La primera parte de este apartado se dirigió a definir los requisitos funcionales y no funcionales que debía tener la herramienta software. Un requisito funcional es aquel que describe una *función* que debe realizar un sistema o componente, entendiendo como función el comportamiento que hay entre una entrada y una salida. Los requisitos no funcionales, por su parte, no se refieren a funciones concretas del sistema sino que juzgan cómo opera en general (si es eficiente, robusto, escalable, etc).

Comenzando por los requerimientos funcionales, el sistema debía:

- Escoger los puntos de acceso objetivo de forma automática. Un punto de acceso objetivo reúne los siguientes requisitos:
  - No es una red abierta.
  - Hay clientes conectados, ya que en caso contrario no es posible conseguir un *handshake* de cuatro vías.
  - Tiene una potencia de señal suficiente (determinada por el/la usuario/a), para evitar perder tiempo en auditorías con baja probabilidad de éxito.
- Sobre cada auditoría, guardar:
  - Momento de la realización.
  - Localización GPS (longitud y latitud).
- Para cada punto de acceso, recolectar los siguientes datos:
  - BSSID (Dirección MAC).
  - Canal.
  - Potencia de la señal.
  - Privacidad.
    - WEP.
    - WPA.
    - WPA/WPA2. Correspondiente a routers que implementan WPA2 con WPA como *fallback method*.
    - WPA2.
  - Cifrado [Sólo para WPA y WPA2]
    - TKIP.
    - CCMP/TKIP. Correspondiente a los routers que implementan WPA2 con WPA como *fallback method* cuando el dispositivo que se quiere conectar no soporta cifrado AES.
    - CCMP.
  - Autenticación:
    - PSK.
    - MGT. Correspondiente a WPA/WPA2-Enterprise. Utiliza un servidor RADIUS para la autenticación, descifrarlos queda fuera del alcance de este trabajo.
  - ESSID.
  - Tarjeta de red en uso durante la auditoría (opcional).
- Para cada cliente conectado, guardar los siguientes datos:
  - Station MAC (Dirección MAC del cliente).

  - BSSID.
- Recolección automática de un *handshake* entre clientes y punto de acceso:
  - Versión pasiva, sin deautenticar clientes.
  - Versión activa, deautenticando clientes.
- Guardar toda esta información en una base de datos.
- Ser operable a través de línea de comandos.

Los requisitos no funcionales de esta herramienta serían los siguientes. El sistema debería ser:

- Fácil de usar.
- Completamente automático o requerir mínima intervención humana.
- Razonablemente robusto.
- Escalable.
- Configurable.
- Tener pocas dependencias.

5.1.2 Herramientas ya desarrolladas

Antes de desarrollar un software nuevo es buena idea comprobar si existe ya una herramienta que cubra las necesidades del proyecto para evitar ”reinventar la rueda”. Es cierto que siempre habrá necesidades específicas de cada proyecto y es posible que no se haya desarrollado todavía ningún software que se ajuste completamente a ellas, pero no por ello deja de ser conveniente ejecutar esta búsqueda mientras sea posible encontrar herramientas que satisfagan parcialmente los requisitos y permitan una integración o extensión poco costosa. Se ha utilizado el servicio de alojamiento GitHub para la búsqueda de proyectos que se pudieran ajustar a la lista de requisitos funcionales y no funcionales arriba mencionada.

Airscript-ng[1] *A python script to simplify the process of auditing wireless networks*
Airscript-ng es una suite de auditoría *wireless* muy completa pero poco adecuada para las tareas de este proyecto. Requiere descargar muchas dependencias (si bien lo hace automáticamente si se le indica) y necesita intervención humana continua. La base de código es suficientemente grande como para que no compense modificarlo y adaptarlo para este proyecto.

[1] https://github.com/Sh3llcod3/Airscript-ng

KeyGrab[2] *Grabs and saves handshakes from WPA/WPA2 encrypted networks*
KeyGrab es un *script* escrito en C que se ocupa de lanzar secuencialmente "ifconfig" para cambiar la tarjeta de red a modo monitor, "macchanger" para aleatorizar la MAC y los programas de la suite Aircrack-ng. Al igual que Airscript-ng, también necesita intervención continua del usuario/a para seleccionar los routers objetivo. La última versión disponible no es funcional, envía continuamente paquetes de deautenticación impidiendo que los dispositivos se conecten de nuevo al punto de acceso. Por esto y por la elección de lenguaje se ha descartado.

WiFiBroot[3] *A WiFi Pentest Cracking tool for WPA/WPA2*
WiFiBroot requiere la instalación de las librerías de Python "scapy" y "pbkf2" y cambiar la tarjeta de red a modo monitor previamente al lanzamiento del programa (parte que se podría automatizar fácilmente, eso sí) pero no es funcional. Permite deautenticar a los dispositivos conectados a un punto de acceso pero no detecta que se hayan conectado de nuevo. No compensa "debuggear" si además hay que ampliar la funcionalidad.

Wificracker[4] *Small python script to help you crack nearby wireless networks in no time*
Wificracker es una versión funcional en Python de lo que intenta KeyGrab. Guía al usuario por las distintas herramientas necesarias para conseguir un *handshake*. Tiene el mismo problema que las anteriores, necesita intervención continua, pero es interesante como inspiración para el desarrollo de una posible primera iteración de un software propio.

WiFi-Rifle[5] *Creating a wireless rifle de-authentication gun*
WiFi-Rifle sólo implementa a bajo nivel la deautenticación en masa de clientes, no es relevante para este proyecto.

Wifi Spy[6] *Sniff Wifi traffic, log device addresses*
No se ha conseguido ejecutar Wifi Spy correctamente.

Comprobado que ningún software cumplía todos los requisitos, se consideró para los más prometedores (los que más requisitos cumplían) la opción de extender su funcionalidad. Ampliar las funciones de un software implica hacer un juicio sobre su *extensibilidad*, o esfuerzo necesario para implementar la extensión, y requiere entender la estructura interna y el flujo de datos. Esta opción, una vez comparada en coste con la de implementar un programa *ad-hoc* que cubriese la necesidad concreta, fue descartada.

5.1.3 Desarrollo de una herramienta propia

Hay que tener en cuenta que el proceso manual para conseguir un *handshake* tan sólo implica la ejecución de determinados comandos por terminal en un determinado orden, como demuestra Wificracker. Automatizar el proceso con un lenguaje de *scripting* y añadir las

[2] https://github.com/ryansisco/KeyGrab
[3] https://github.com/hash3liZer/WiFiBroot
[4] https://github.com/yelhamer/wificracker
[5] https://github.com/sensepost/WiFi-Rifle
[6] https://github.com/Geovation/wifispy

funcionalidades que se requieren se consideró menos costoso que estudiar el código fuente de estas herramientas para ampliarlas con las funciones de las que carecían.

Metodología

Se utilizó la metodología iterativa e incremental para desarrollar esta herramienta. Consiste en aplicar de forma reiterada varias secuencias del modelo en cascada (análisis, diseño, codificación y pruebas), donde cada resultado de un ciclo en cascada constituye un incremento del software. Con esta metodología se construye una implementación parcial pero operativa del sistema y posteriormente se va aumentando su funcionalidad o refinando un incremento anterior.

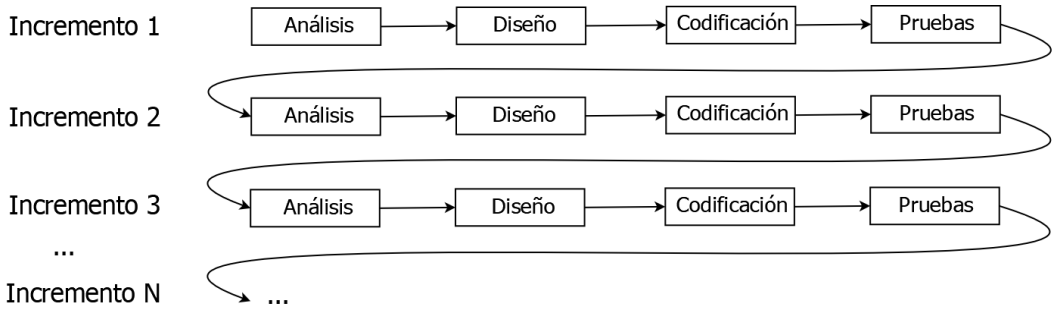

Figura 5.1: Esquema de la metodología incremental

Diseño e implementación del programa

El lenguaje escogido fue Python. Python es un lenguaje de alto nivel diseñado con énfasis en la legibilidad que permite escribir *scripts* de forma muy sencilla gracias a su librería estándar. La práctica totalidad de distribuciones Linux traen Pyhton preinstalado, de ahí que esté prácticamente asegurado que una máquina Linux pueda ejecutar este *script*. Esto va en consonancia con el requisito no funcional de limitar al máximo las dependencias del software. Se pretende que una máquina Linux ”normal”, es decir, con cualquier distribución que no sea especializada en seguridad, pueda ejecutar el programa.

Siguiendo con este principio, la suite elegida para ejecutar las tareas de auditoría fue Aircrack-ng. Aircarck-ng es, sin lugar a dudas, la colección de herramientas de auditoría Wi-Fi más popular del mundo y aunque funciona principalmente en Linux, también lo hace en Windows, macOS, FreeBSD, OpenBSD, NetBSD o Solaris. En consecuencia, la mayoría de distribuciones Linux provee en sus repositorios de alguna versión de la suite que, aunque no suele ser la más actual, es probablemente suficiente, ya que el *script* sólo utiliza funcionalidades que han estado presentes desde hace 10 años[7]. Finalmente, para manejar los CSV que

[7] Changelog de Aircrack-ng

genera Aircrack-ng se ha utilizado la librería pandas. La herramienta, en su versión final, se encuentra alojada en GitHub.

En la primera iteración del software se buscaba adquirir familiaridad con la forma en que Python ejecuta comandos de terminal y con los objetos que representan sus salidas. Posteriormente proporcionar la siguiente funcionalidad:

1. Cambiar las tarjetas de red a modo monitor.
2. Buscar puntos de acceso y seleccionar sólo los que tuvieran clientes conectados.
3. Para cada punto de acceso, monitorear si se producía un *handshake* después de forzar una deautenticación de cliente.

En la segunda iteración se buscaba que el *script* recopilase la información especificada en los requisitos para cada clase de objetos, auditoría, routers, *handshakes* y clientes:

1. Implementar las clases que representan dicha información (Audit, Router).
2. Diseñar la mejor forma de concurrencia para que el *script* escale según aumenta el número de tajetas de red disponibles. Cada auditoría es independiente de las demás, no es necesario sincronizar información entre hilos de procesamiento salvo la lista de puntos de acceso que quedan por auditar. Para ello simplemente la única instancia clase ”Audit” (patrón *singleton*) provee un método ”get” que devuelve el siguiente router a auditar. El singleton ”Audit” almacena una lista protegida por un semáforo para evitar *race conditions* que provoquen múltiples auditorías a la vez sobre un mismo punto de acceso.
3. Diseñar el esquema de base de datos para guardar toda la información que exigen los requisitos.
4. Implementar la base de datos en sqlite3.

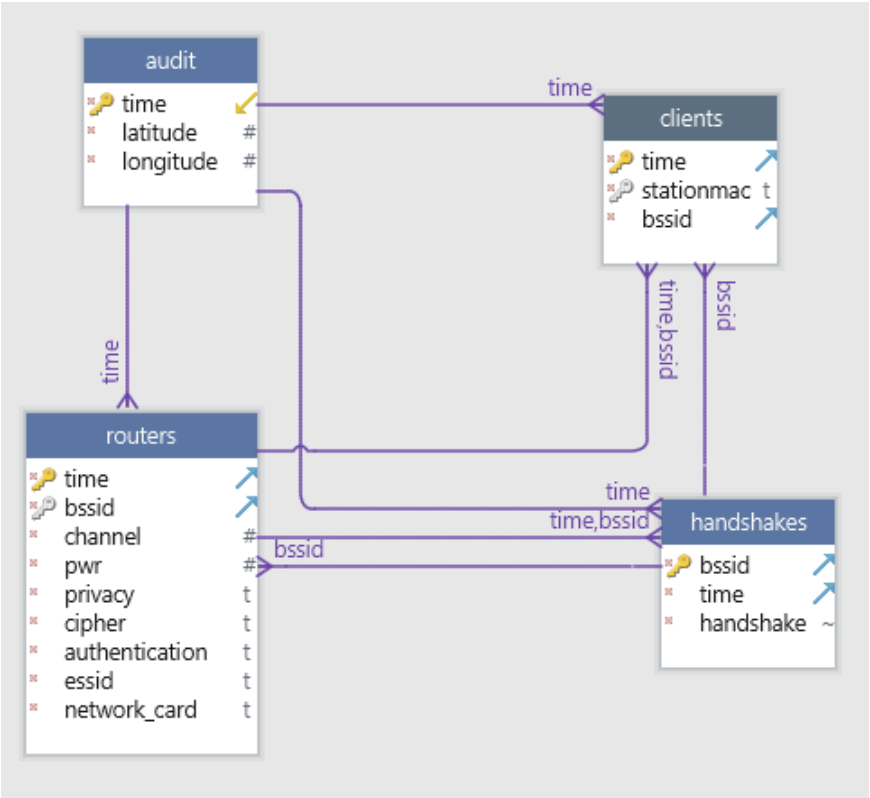

Figura 5.2: Esquema de la base de datos

En la tercera iteración se buscó que mostrase información relevante durante el proceso. Los mensajes debían ser informativos y autocontenidos, ya que con procesos concurrentes no se podía garantizar que aparecieran por pantalla en el orden que les correspondía. A partir de este momento la herramienta ya se consideró lo suficientemente madura como para hacer pruebas en un entorno real.

En la cuarta iteración se buscaba añadir determinadas funcionalidades extra que mejorasen el tiempo de las auditorías o el desempeño de la herramienta:

1. Desplazar a un fichero de configuración todos aquellos valores "hardcodeados" desde los que se controlan las funcionalidades del *script*.
2. Máximo de clientes a deautenticar por punto de aceso.
3. Deautenticación educada de los clientes (sólo deautentica al siguiente si no ha conseguido *handshake* con el anterior) o bruta (todos los clientes de la red). La versión bruta permite reducir el tiempo de intercepción de un *handshake* cuando no importa ser detectables, ya que se está dejando sin conexión a todos los clientes.
4. Aleatorizar la MAC de las tarjetas de red. No sólo por privacidad, también para evitar bloqueos de sistemas de detección.

5. Mínimo de potencia de señal del punto de acceso para intentar captura de *handshake*. Se perdía mucho tiempo intentando interceptar *handshakes* de redes con pocas posibilidades de éxito. Empíricamente se comprobó que todos los *handshakes* se habían obtenido en las pruebas de campo eran de redes con más de -30dB.

A lo largo de estas iteraciones se fueron solucionando errores y carencias tanto del propio *script* como de las herramientas que usa:

- Airodump-ng no produce archivos CSV acordes a ningún estándar. No sólo genera dos tablas en el mismo archivo, sino que introduce varios valores en algunas columnas. Cuando esto último ocurre, la decisión que se ha tomado es quedarse con el primero de ellos.
- La única forma en que Airodump-ng informa de que ha capturado un *handshake* es mostrándolo por pantalla. En cada ejecución hay un archivo de texto temporal donde se redirige la salida por pantalla y se busca si aparecen las palabras "WPA HANDSHAKE".
- Airmon-ng no sigue una política de nombrado de interfaces consistente cuando cambia las tarjetas de red a modo monitor. Por ejemplo, una interfaz de nombre "wlan0" en modo *managed* podía cambiar a "mon0" en modo monitor, sin embargo cuando los nombres de las interfaces son más largos[8] los nombres se volvían completamente arbitrarios. Se utilizó el comando "ip" en su lugar porque permite dejar el nombre tal y como está.
- En ocasiones las redes no proporcionan toda la información que se pretende recabar por lo que quedan algunos atributos en NULL. Como ya se había empezado a usar el script cuando apareció este problema se añadieron valores por defecto para no cambiar la restricción de no nulos en la base de datos. Estos valores por defecto se escogieron cuidadosamente para que no afectaran a los análisis que pudieran hacerse posteriormente con esos datos.
- Lista blanca de direcciones MAC a la que no atacar. Al utilizar la Raspberry Pi en versión *headless*, es decir, controlada a través de la red sin monitor, teclado o ratón, es necesario que haya una zona Wi-Fi para conectarse con el smartphone. Sin una lista blanca el script no discriminaba esta red y podía dejar sin conexión entre smartphone y Raspberry.

Flujo del programa

Antes de ejecutar el *script* es necesario revisar el archivo de configuración config.py y añadir los nombres de las tarjetas de red que se usarán para la auditoría, todas las demás opciones pueden mantenerse por defecto. En el futuro podría automatizarse este proceso también, es

[8] Por ejemplo, desde la política adoptada por Debian 9 y derivados de añadir la dirección MAC al nombre de la interfaz.

decir, que el *script* obtenga por sí mismo las interfaces disponibles y que las presente para que se escojan.

```
# Network card interfaces from "$ ifconfig" or "$ ip link show"
NETWORK_CARDS = ["wlan0", "wlan1"]
```

Hecho esto, el *script* se lanza con permisos de superusuario con los siguientes comandos:

```
cd /directorio/del/proyecto/Wifi-Hashing/src
sudo python3 wifihashing.py
```

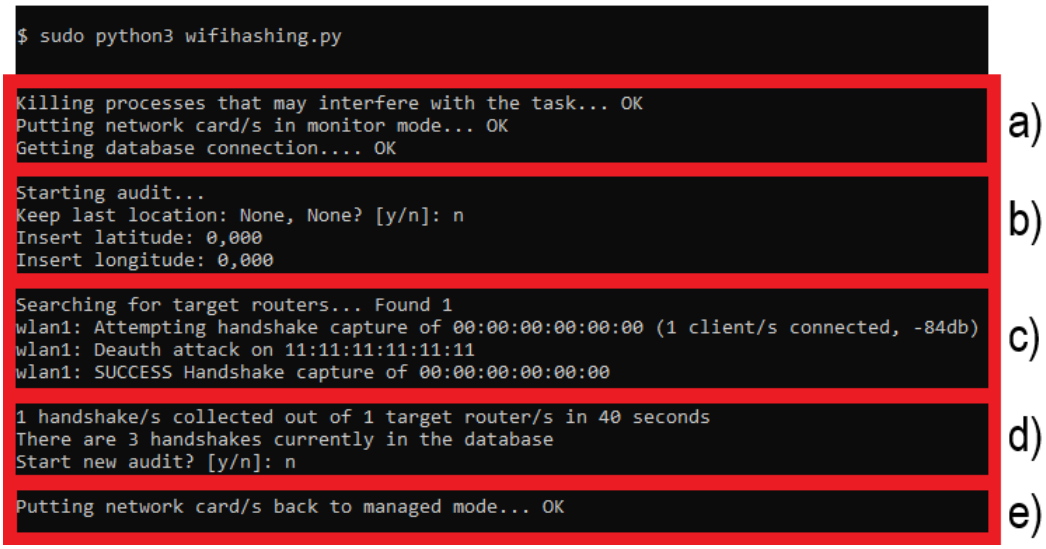

Figura 5.3: Ejemplo de salida por pantalla de wifihashing.py, dividido por secciones

Es necesario lanzar el script con permisos de superusuario porque el comando "ip" solicita esos privilegios si se quiere cambiar las tarjetas de red a modo monitor. En el futuro se podría intentar que sólo esa parte del *script* se ejecute con permisos de superusuario, siguiendo el principio del menor privilegio posible.

La interfaz que muestra el script por pantalla se ha dividido en secciones para facilitar la explicación de qué está reportando. La sección a) se corresponde con la inicialización del script, las secciones b) a d) son el bucle principal y la sección e) la finalización del script. Más en detalle:

a) Inicialización del script. Implica parar los procesos que pueden interferir en la auditoría (en particular NetworkManager y wpa_supplicant), cambiar las antenas a modo monitor y obtener la conexión con la base de datos.

b) Al inicio de la auditoría se exige insertar la localización en latitud y longitud. Esta es una de las partes del *script* que no se ha automatizado. Sería posible automatizarla en el

futuro si se dispone de un receptor GPS compatible conectado al dispositivo realizando la auditoría pero añadiría una nueva dependencia.

c) Núcleo de la auditoría. Esta es una actividad que se realiza concurrentemente si hay varias tarjetas de red inalámbricas, cada antena disponible estará auditando un punto de acceso hasta que se agoten. Se reporta qué tarjeta está auditando determinado punto de acceso y si se ha conseguido interceptar un *handshake*.

d) Se informa de cuántos *handshakes* se han conseguido de los routers objetivo en un determinado tiempo. También informa de cuántos hay en ese momento en la base de datos. A la vez que esto se muestra por pantalla, por detrás el *script* está insertando toda la información en la base de datos. Aquí se vuelve a pedir intervención del usuario o usuaria para indicar si se quiere realizar otra auditoría o terminar la ejecución del *script*. Si se indica que sí, el programa vuelve al punto b).

e) Finalmente, se termina la auditoría cambiando las antenas de nuevo a modo *managed* y cerrando la conexión con la base de datos.

La salida del script son datos sobre la auditoría, los puntos de acceso, clientes y *handshakes* contenidos en la base de datos Sqlite3. Los ficheros originales generados por las herramientas de la suite Aircrack-ng, de los que se extrae toda la información (routers y clientes en formato CSV y *handshakes* en formato cap), se guardan también en sus respectivos directorios. Así se dispone de las fuentes de datos originales de las que recuperar toda la información en caso de que la base de datos se corrompa.

Visto a un nivel más alto, el flujo del programa sigue el siguiente esquema:

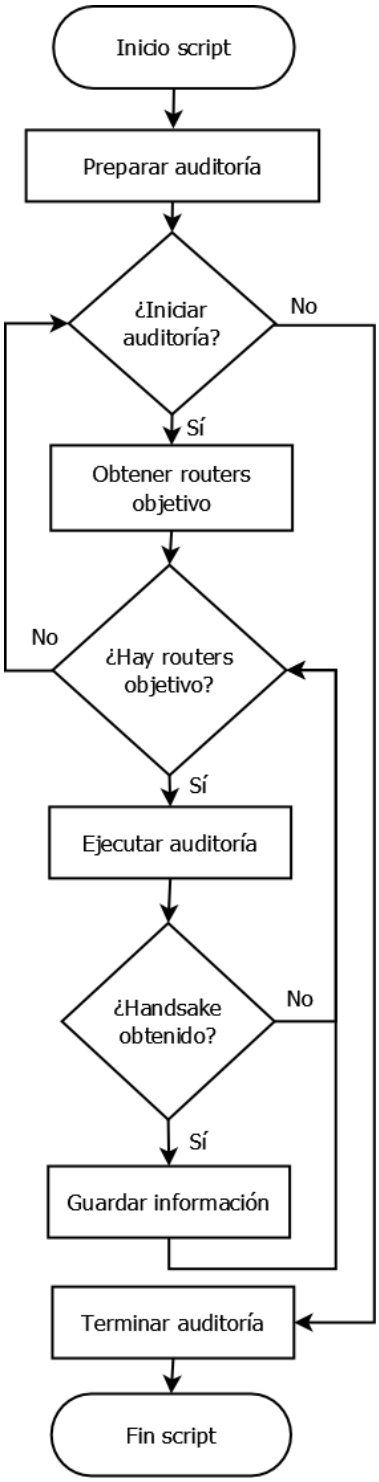

Figura 5.4: Diagrama de flujo del script

5.2 Recolección de la información

Con la herramienta disponible el siguiente paso en el estudio fue comenzar a interceptar los *handshakes*. A continuación se explica por qué se eligió la zona paralela a los Jardines de Méndez Núñez, por qué se ejecutaron las auditorias con un ordenador de placa simple y finalmente se hace un comentario de los resultados.

5.2.1 Zona

La zona de los alrededores de los Jardines de Méndez Núñez fue la elegida para la recolección de la información. Abarca un área de unos $105.785\,m^2$ y un perímetro de unos $1.518\,m$ y es una zona interesante por la presencia de viviendas y negocios de todo tipo, desde bancos a pequeñas tiendas, pasando por farmacias, restaurantes y bares, etc. Se ha considerado que es suficientemente representativa del Área Metropolitana de A Coruña.

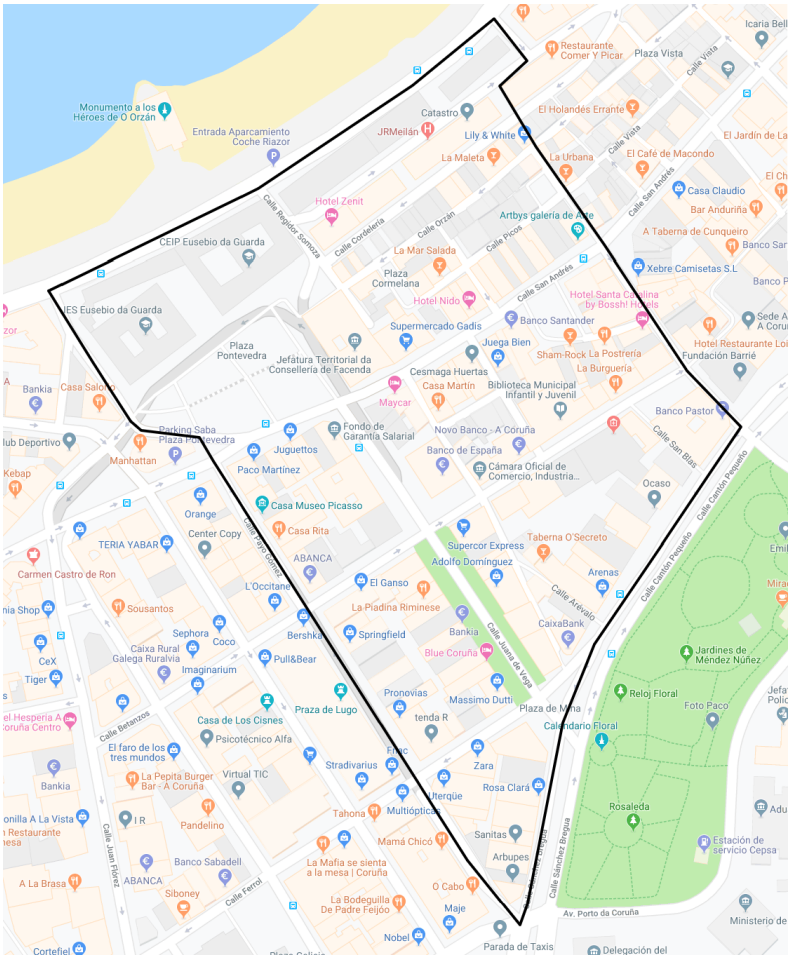

Figura 5.5: Mapa de la zona de la auditoría (Google Maps)

También es interesante conocer la concentración de viviendas y residentes (en viviendas principales) por cuestiones técnicas: para capturar un *handshake* se necesita que haya dispositivos conectados al punto de acceso, lo que en la mayoría de casos se traduce en que "haya gente" en las viviendas usando dispositivos con Wi-Fi. Según los datos censales del INE de 2011[9], la zona abarca en gran parte o completamente las siguientes secciones censales:

Sección censal	**Residentes en viviendas principales**	**Viviendas**
15-030-01-004	1.590	775
15-030-01-012	975	475
15-030-03-001	780	310
Total	3.345	1560

Como se puede observar, la concentración de población no es particularmente alta, 2,14 personas por vivienda, hecho que *a priori* dificultará la obtención de *handshakes*.

5.2.2 Hardware

A pesar de que la zona no es territorialmente extensa, recorrerla durante horas transportando un portátil a la espalda no era factible. Teniendo en cuenta que la ejecución del script requiere mínima potencia computacional y que los ordenadores de placa simple (*single-board computers*) ya se han utilizado en muchos proyectos de seguridad se optó por esta tecnología, en particular una Raspberry P1 3 A+, suplementada con dos antenas Wi-Fi Alfa AWUS036NHA y una Xiaomi Mi Power Bank 2S como fuente de alimentación.

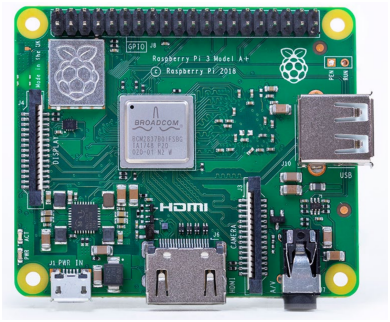

Figura 5.6: Raspberry Pi 3 A+

El modelo 3 A+ de la Raspberry Pi[10] es la revisión final de la serie 3. Equipa los siguientes componentes: un SoC (*System on a chip*) Broadcom BCM2837B0 Cortex-A53 (ARMv8) de

[9] http://www.ine.es/censos2011/tablas/Wizard.do?WIZARD=5&reqCode=paso4
[10] Raspberry Pi 3 A+ product brief

64 bits a 1.4GHz, 512MB LPDDR2 SDRAM, adaptador inalámbrico de 2.4GHz y 5GHz IEEE 802.11.b/g/n/ac, 1 puerto USB 2.0, una entrada Micro SD para la carga del sistema operativo y almacenamiento y una entrada de alimentación de 5V/2.5A. Son especificaciones humildes pero que permiten ejecutar perfectamente distribuciones Linux sin escritorio gráfico y para este proyecto son más que suficientes.

Figura 5.7: Antena Alfa AWUS036NHA

La antena Alfa AWUS036NHA[11] tiene las siguientes características: Chipset Atheros AR9271, cumple los estándares IEEE 802.11/b/g/n, soporta la banda de frecuencia inalámbrica de los 2.4GHz, posee una antena dipolo de 2.4GHz 5dBi, soporta los protocolos WEP, WPA, WPA2, WPA Mixed y WPS y la entrada es de tipo miniUSB. Es una antena relativamente antigua pero con la ventaja de incorporar un chipset muy conocido y compatible. No es un problema que sólo soporte la frecuencia de los 2.4GHz ya que los routers nuevos que soportan los 5GHz también crean una red en los 2.4GHz.

Figura 5.8: Xiaomi Mi Power Bank 2S

[11] Alfa AWUS036NHA product detail

La Xiaomi Mi Power Bank 2S[12] es una batería externa de célula de polímero de litio con una capacidad de 10.000 mAh. Los parámetros de salida de puerto único son 5.1V/2.4A - 9V/1.6A MAX - 12V/1.2A MAX. Proporciona suficiente voltaje para alimentar la Raspberry Pi con las dos antenas Alfa AWUS036NHA conectadas por USB gracias a un *hub* que multiplica la entrada a 4 puertos.

El sistema operativo que se instaló en la Raspberry fue Raspbian Stretch Lite (basado en Debian 9) para asegurar la mayor compatibilidad con el hardware. La versión Lite es la adecuada para operar la Raspberry *headless*. La red la proporcionaba un smartphone a través de una zona Wi-Fi. Con el servicio "openssh-server" corriendo en la Raspberry es posible conectarse a través de un cliente "ssh", en este caso desde el propio smartphone:

```
ssh pi@ip.del.host
```

5.2.3 Sobre la auditoría

La auditoría se extendió 19 horas distribuidas en 11 días, entre el 21 de mayo y el 11 de junio, durante los cuales se recorrieron alrededor de 15 kilómetros. El objetivo de *handshakes* a recolectar se marcó en 100-200 para asegurar un número razonable de contraseñas descifradas. Inicialmente no se siguió ningún método a la hora de recorrer la zona. Una vez se consiguió sobrepasar los 100 *handshakes* las localizaciones dejaron de ser aleatorias para intentar "rellenar huecos" de aquellos lugares en los que todavía no se había conseguido interceptar ninguno. Los números que representan los resultados de esta auditoría masiva son los siguientes: Se realizaron 333 auditorías individuales en las que se obtuvieron 239 *handshakes*, 142 de los cuales son únicos, por tanto un 42,64% de probabilidad de intercepción en cada auditoría. De estos 142, 133 se corresponden con autenticación PSK y 9 con autenticación a través de servidor RADIUS. En consecuencia, la cifra final de *handshakes* a descifrar es 133, lo que equivale a uno por cada $795\,m^2$.

[12] Especificaciones Xiaomi Mi Power Bank 2S

Capítulo 6

Descifrado y análisis de patrones de contraseñas

Este capítulo tratará sobre Hashcat (el programa de descifrado de *hashes*), la infraestructura sobre la que se ejecutó todo este proceso y la estrategia detrás de las decisiones de utilizar los distintos tipos de ataques. Después, con las contraseñas ya descifradas, se comentará qué patrones se hallaron y cómo se elaboró un diccionario mínimo para el Área Metropolitana de A Coruña. Finalmente, el último apartado está dedicado a la visualización y las implicaciones de los resultados.

6.1 Descifrado

Este es el apartado dedicado al descifrado de las contraseñas y se centra en las tres cuestiones esenciales del proceso: el software con el que descifrar, el hardware sobre el que correrá y la estrategia a seguir, dado que tiempo y recursos computacionales son limitados.

6.1.1 Hashcat

Se ha escogido Hashcat para descifrar los *handshakes*. Hashcat es una herramienta de recuperación de contraseñas de software libre que soporta cinco modos de ataque para alrededor de 200 algoritmos de *hashing* tanto por CPU, GPU como por otros aceleradores hardware en Linux, Windows y macOS. En este proyecto se ha utilizado la versión 4.2.1.

La última versión de Hashcat puede compilarse e instalarse en Linux y macOS de la siguiente forma[1]:

[1] Documentación para compilar Hashcat

```
git clone https://github.com/hashcat/hashcat.git
make
make install
```

También se pueden descargar los binarios desde la página principal del proyecto en https://hashcat.net/hashcat/. En la descarga se proporcionan ejecutables tanto para Linux y macOS como para Windows además de unos *scripts* que ejecutan descifrados de ejemplo con unos pequeños diccionarios incluidos.

Uso[2]:

```
hashcat [options]... [hash|hashfile|hccapxfile] [dictionary|mask|directory]...
```

Se han usado las siguientes opciones:

Opción corta, larga	Parámetro	Significado
-m, –hash-type	2500	Correspondiente con WPA-EAPOL-PBKDF2 de la categoría “Protocolos de red”
-a, –attack-mode	0 (Por defecto)	Diccionario
	3	Fuerza bruta
-w, –workload-profile	2 (Por defecto)	Rendimiento medio, puede que tenga impacto sobre el rendimiento del escritorio
	3	Alto rendimiento, el escritorio deja de responder

Cuadro 6.1: Opciones de Hashcat

Ejemplo de ataque de diccionario con el perfil de carga de trabajo por defecto:

```
hashcat −m 2500 example.hccapx example.dic
```

Ejemplo de ataque diccionario más reglas con carga de trabajo en alto rendimiento:

```
hashcat −w 3 −m 2500 example.hccapx example.dic −r rules/best64.rule
```

Ejemplo de ataque de fuerza bruta para números de 8 dígitos a través de una máscara:

```
hashcat −a 3 −m 2500 example.hccapx ?d?d?d?d?d?d?d?d
```

[2] Documentación de Hashcat

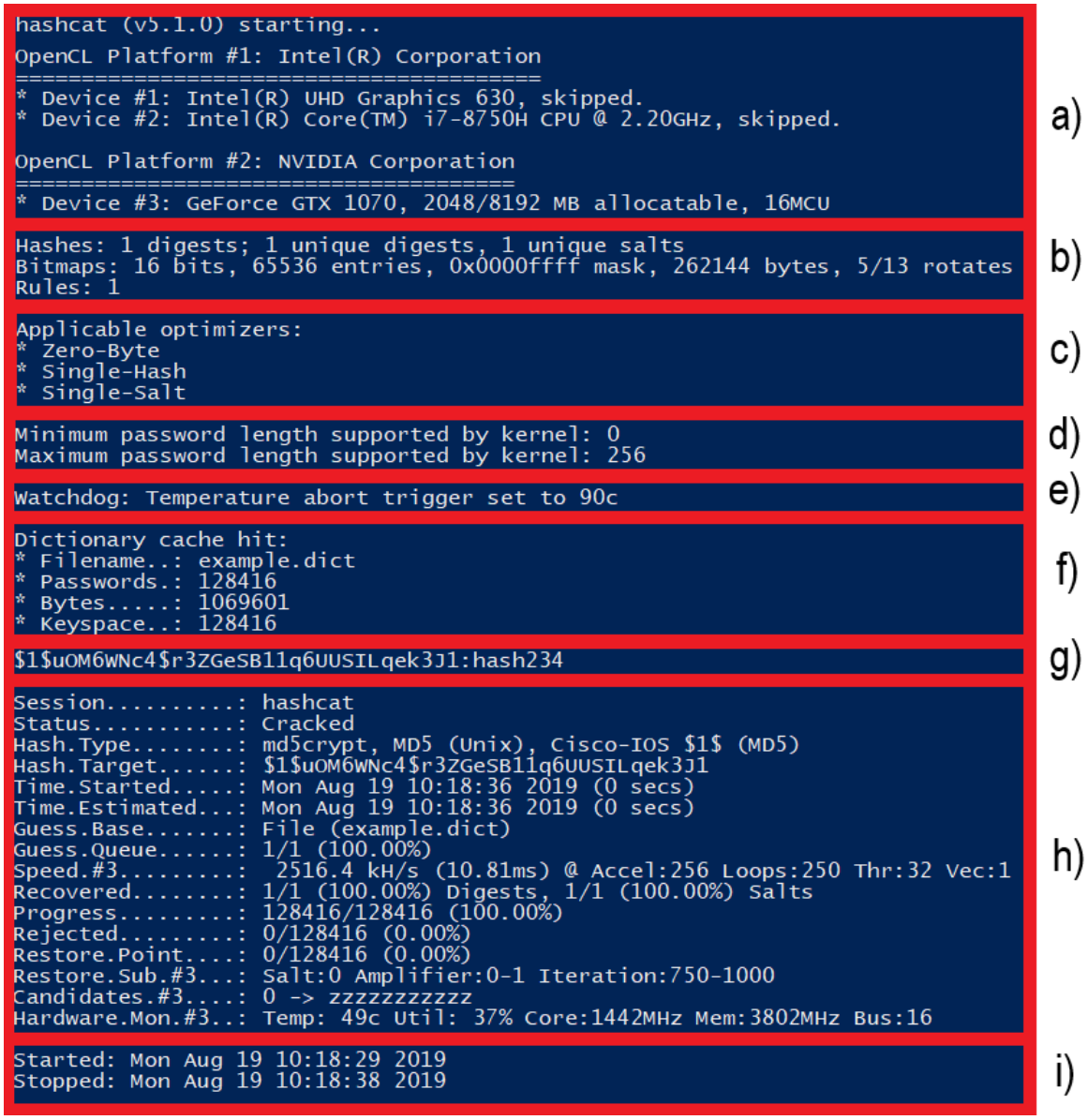

Figura 6.1: Ejemplo de salida por pantalla de Hashcat, dividido por secciones

La interfaz que muestra Hashcat por pantalla es muy espartana pero proporciona una gran cantidad de información. Se ha dividido en secciones para facilitar la explicación de qué está reportando. Podemos agrupar las secciones de la a) a la f), ambas inclusive, en ”información inicial”. Estas secciones son las que Hashcat muestra siempre al inicio de una sesión de descifrado. La sección h) se corresponde con la ”información en tiempo real”, en ella se da una instantánea del progreso de la sesión. La sección g) es un ejemplo de cómo muestra Hashcat una contraseña descifrada. Y finalmente la sección i) es la ”información de finalización” de la sesión, muestra cuándo empezó y cuándo terminó la sesión. Entrando un poco más en detalle

en cada una:

a) Muestra los dispositivos reconocidos por Hashcat agrupados por fabricante. Admite CPUs Intel y GPUs Intel, NVIDIA y AMD.

b) Aquí se contabilizan los *hashes* a descifrar, cuántos de ellos son únicos y cuántos *salts* hay. Esta información es interesante cuando se intenta descifrar los *handshakes* WPA/WPA2 ya que en cada fichero cap puede haber varios handshakes de un mismo punto de acceso. También contabiliza cuántas reglas se van a aplicar sobre la colección de palabras, siempre será una o más porque no hacer ninguna transformación sobre las palabras es una regla en sí.

c) Lista los algoritmos que pueden optimizar el cálculo de los *hashes*. Aunque dice *aplicable optimizers* no significa que el mensaje sea informativo y hayan de aplicarse sino que ya se han aplicado.

d) Qué longitud pueden tener las palabras que se prueben. En WPA/WPA2 de 8 a 63.

e) Monitor de temperatura. Por defecto abortará la sesión cuando el dispositivo alcance los 90 grados centígrados.

f) Muestra información sobre la colección de palabras que se va a probar: nombre, número de palabras, tamaño en bytes y el espacio de claves. En la imagen del ejemplo, número de palabras y espacio de claves es el mismo, ya que la única regla aplicada es ”no modificar la palabra”. Sin embargo, de haber aplicado alguno de los conjuntos de reglas que provee Hashcat por defecto, el espacio de claves sería mucho mayor que el número de contraseñas.

g) Ejemplo de cómo se muestra que ha descifrado una contraseña.

h) Características de la sesión en curso

- ”Session”. Por defecto ”hashcat”. Nombrar las distintas sesiones permite guardarlas en distintos archivos y restaurarlas más fácilmente.
- ”Status”. Hay tres estados: *running*, *exhausted*, *cracked*, para cuando el programa aún se está ejecutando, para cuando ha agotado la colección de palabras sin éxito y cuando ha descifrado la contraseña, respectivamente.
- ”Hash type” y ”Hash target”. El tipo de *hash* y el propio *hash* objetivo.
- ”Time started” y ”Time estimated”. El momento en el que ha empezado a descifrar contraseñas y cuánto tiempo estima que tardará en terminar el trabajo.
- ”Guess Base” y ”Guess Queue”. El tipo de colección de palabras (diccionario con o sin reglas, máscara o híbrido) y la cola (pueden ponerse varios diccionarios en cola, por ejemplo).
- ”Speed”. La velocidad a la que está probando palabras en $kilohashes/segundo$.

- ”Recovered”, ”Progress” y ”Rejected”. El número de contraseñas descifradas, el progreso probando palabras y el número de palabras que se han rechazado. Las palabras se rechazan por dos razones: el algoritmo tiene alguna limitación de longitud de palabra o hashcat ha añadido sus propias limitaciones en función del modo de ataque.
- ”Restore point”. El punto hasta el que se ha guardado la sesión. Si la sesión terminase de forma abrupta se retomaría desde ahí. Si la sesión se termina de forma ordenada se espera a que el programa llegue al siguiente punto de restauración y termina.
- ”Candidates”. Ejemplos de las palabras que se están probando.
- ”Hardware monitor”. Proporciona datos del hardware en tiempo real. Los más importantes son la temperatura y el porcentaje de utilización.

i) Muestra el momento en que ha empezado y terminado la sesión. Se distingue del ”Time started” del punto anterior en que empieza a contar desde el momento en que arranca el programa y por tanto tiene en cuenta también el tiempo de inicialización antes de empezar a descifrar.

6.1.2 Infraestructura

Para la fase de descifrado de las contraseñas se ha facilitado el acceso a dos tarjetas gráficas NVIDIA Tesla K80[3] y a otras dos NVIDIA Tesla K20m[4] del grupo RNASA. La Tesla K80 dispone de 4992 núcleos CUDA, 24 GB de memoria GDDR5 y 480 GB/s de ancho de banda de memoria agregado. Por su parte, la Tesla K20m dispone de 2496 núcleos CUDA, 5 GB de memoria GDDR5 y 206 GB/s de ancho de banda de memoria agregado. Los *handshakes* en formato ”hccapx” se repartieron en cuatro ficheros y cada uno de ellos se asignó a una de las tarjetas gráficas.

Figura 6.2: Tesla K80

Figura 6.3: Tesla K20m

[3] Especificaciones de la NVIDIA Tesla K80
[4] Especificaciones de la NVIDIA Tesla K20m

Para dar muestra del rendimiento de cada una se ha ejecutado un benchmark de Hashcat con el comando:

```
hashcat --benchmark
```

Los resultados de la Tesla K80 fueron:

```
Hashmode: 2500 - WPA-EAPOL-PBKDF2 (Iterations: 4096)
Speed.Dev .1.....:     81116 H/s (81.18 ms) @ Accel:64 Loops:32 Thr:1024 Vec:1
Speed.Dev .2.....:     77568 H/s (84.88 ms) @ Accel:64 Loops:32 Thr:1024 Vec:1
Speed.Dev .3.....:     81400 H/s (80.89 ms) @ Accel:64 Loops:32 Thr:1024 Vec:1
Speed.Dev .4.....:     78254 H/s (84.15 ms) @ Accel:64 Loops:32 Thr:1024 Vec:1
Speed.Dev .*.....:     318.3 kH/s
```

Los resultados de la Tesla K20m fueron:

```
Hashmode: 2500 - WPA-EAPOL-PBKDF2 (Iterations: 4096)
Speed.Dev .1.....:     75182 H/s (87.29 ms) @ Accel:64 Loops:32 Thr:1024 Vec:1
Speed.Dev .2.....:     75176 H/s (87.28 ms) @ Accel:64 Loops:32 Thr:1024 Vec:1
Speed.Dev .*.....:     150.4 kH/s
```

La capacidad de cómputo total es

$$2 \times 318,3\,kH/s + 2 \times 150,4\,kH/s = 937,4\,kH/s$$

Es decir, 937.400 *hashes* cada segundo.

Una infraestructura ligeramente superior en el servicio EC2 de Amazon, con 4 NVIDIA Tesla K80, 100 GB de almacenamiento y pagando *On demand* ya que sólo se usaría durante dos meses (para acceder a los descuentos hay que comprar un año de servicio como mínimo) tendría los siguientes costes por mes[5]:

Servicio	**Tecnología**	**Precio**
Computación	AWS EC2	2.628 USD
Almacenamiento	AWS EBS	40 USD
Total		2.668 USD

Por tanto, dos meses de computación con un potencia ligeramente superior a la proporcionada por el clúster BIOCAI del grupo RNASA tendría un coste de 5.336 USD, unos 4.763 euros a 06/08/2019.

[5] Calculadora de costes AWS

6.1.3 Estrategia

Cuanto mayores son el tiempo y recursos disponibles para ”crackear” una contraseña, menor es la utilidad de una estrategia. Simplemente se alimenta a la herramienta de elección con los distintos ataques que se pretendan realizar y se espera. Sin embargo, en un entorno donde tiempo o recursos, o ambos, son limitados es lógico dirigir los primeros esfuerzos hacia las opciones más obvias para intentar conseguir resultados en un período corto. La estrategia puede dividirse en tres fases: suposiciones obvias, pruebas en masa y variaciones comunes de palabras base.

Es oportuno hacer dos comentarios acerca de las estimaciones que se verán a continuación. La primera, para que las estimaciones sean comparables entre los distintos ataques se ha considerado siempre el conjunto al completo de *handshakes*. En la práctica, conforme se iban descifrando no se volvían a probar. La segunda, que los ataques de diccionario con reglas son más rápidos que los ataques de diccionario[6] debido al límite de ancho de banda PCI-e y a la latencia en la transferencia de información desde el host. Es decir, si se tiene un diccionario de palabras base y un conjunto de reglas, es mucho más eficiente en tiempo enviar sólo las palabras base a GPU y que se modifiquen en la propia GPU que generar toda la colección de modificaciones en CPU y enviarlas después a la GPU.

La primera fase se compuso de los siguientes ataques:

Los primeros intentos fueron dos ataques de diccionario con PasswordsPro y RockYou, dos de las colecciones de palabras más conocidas y utilizadas. PasswordsPro es una recopilación de palabras realizada por el equipo de InsidePro mientras que RockYou es el resultado de una filtración de datos de acceso de la empresa RockYou en 2009, una compañía que desarrollaba widgets para MySpace y aplicaciones para otras redes sociales como Facebook. RockYou utilizaba una base de datos no cifrada para guardar información personal y contraseñas y fue objetivo de un ataque de inyección SQL. Alrededor de 32 millones de cuentas de usuario y contraseñas fueron recuperadas en texto plano. La siguiente es una estimación de tiempo para los 133 objetivos a 937.400 H/s y sus respectivos resultados:

Diccionario	Palabras	Tiempo consumido	Descifrados
PasswordsPro	2.937.125	6m	0
RockYou	14.344.391	33m	13

Los siguientes dos fueron ataques de fuerza bruta con números de ocho y nueve dígitos. El espacio de búsqueda de los ocho y nueve dígitos son 10^8 y 10^9 de posibilidades respectivamente.

[6] Cómo proporcionar más trabajo a Hashcat para conseguir la máxima velocidad

Máscara	Palabras	Tiempo consumido	Descifrados
8 dígitos	100.000.000	3h 56m	11
9 dígitos	1.000.000.000	1d 15h 24m	2

A continuación se probó con los nombres de los propios puntos de acceso junto con un conjunto de reglas que proporciona Hashcat llamado "best64.rule". Algunos ejemplos de las modificaciones que realiza sobre las palabras base son: dar la vuelta, ponerla en mayúsculas, añadir un número al final o al inicio...

Diccionario	Reglas	Palabras	Tiempo consumido	Descifrados
ESSIDs	best64.rule	1.432.354	3m	2

El INE publica los 100 nombres más comunes de hombres y mujeres en España[7]. Sobre esta lista se aplicó también el set de reglas anterior.

Diccionario	Reglas	Palabras	Tiempo consumido	Descifrados
Nombres	best64.rule	2.017.400	4m	1

En estos dos casos los efectos aceleradores de las reglas no son de relevancia dada la mínima cantidad de ESSID (131) y de nombres (200).

Finalmente se probaron números de DNI. Los números de DNI están compuestos por 8 cifras y una letra. He comprobado en mi círculo cercano que la idea general es que la letra es aleatoria. Con esto entendí por qué algunos de ellos y ellas pensaban que el DNI podía ser una buena contraseña, a pesar de la intuición de que no debe estar bien usarlo. Como se sabe, la letra no es aleatoria, sino que se calcula a partir de los dígitos. Por tanto lo que podría ser un espacio de búsqueda de 36^9 términos pasa a ser efectivamente de un orden de magnitud menor (10^8).

Diccionario	Palabras	Tiempo consumido	Descifrados
DNI	100.000.000	3h 56m	0

A continuación la segunda fase fue ir probando diccionarios sin ningún criterio más que hacerlo en orden creciente de tamaño:

[7] Nombres más frecuentes en España según el INE

Diccionario	Palabras	Tiempo consumido	Descifrados
hk_hlm_founds	38.647.798	1h 31m	3
CrackStation	63.941.069	2h 31m	0
Top306M ProbableWordlists	306.429.377	12h 4m	4
HashesOrg	446.426.204	17h 35m	0
Andronicus	626.198.124	1d 40m	0
breachcompilation	1.012.024.699	1d 15h 55m	0
Rocktastic	1.133.849.621	1d 20h 41m	0
weakpass_2_wifi	2.347.498.477	3d 20h 31m	5
weakpass_2	2.649.982.129	4d 8h 26m	0

La tercera fase fue probar algunos de los diccionarios más pequeños con el conjunto de reglas "best64.rule". En estos casos es donde se nota, por un lado, la aceleración de utilizar reglas y por otro, el trabajo que hace hashcat rechazando determinadas palabras según las políticas mencionadas en la sección 6.1.1. Siguiendo estas políticas pueden llegar a rechazarse hasta un 90% de todas las palabras candidatas, por eso aunque "best64.rule" produce 10.087 variaciones de cada una ello no se traduce en tiempos de "crackeado" 10.087 veces superiores.

Diccionario	Reglas	Palabras	Tiempo consumido por una Tesla K20m	Descifrados
PasswordsPro	best64.rule	29.626.779.875	1d 22h	0
RockYou	best64.rule	144.691.801.408	10d 1h	1

Las últimas posibilidades que se probaron fueron contraseñas de diez dígitos a través de una máscara:

Máscara	Palabras	Tiempo consumido	Descifrados
10 dígitos	10.000.000.000	16d 10h 7m	2

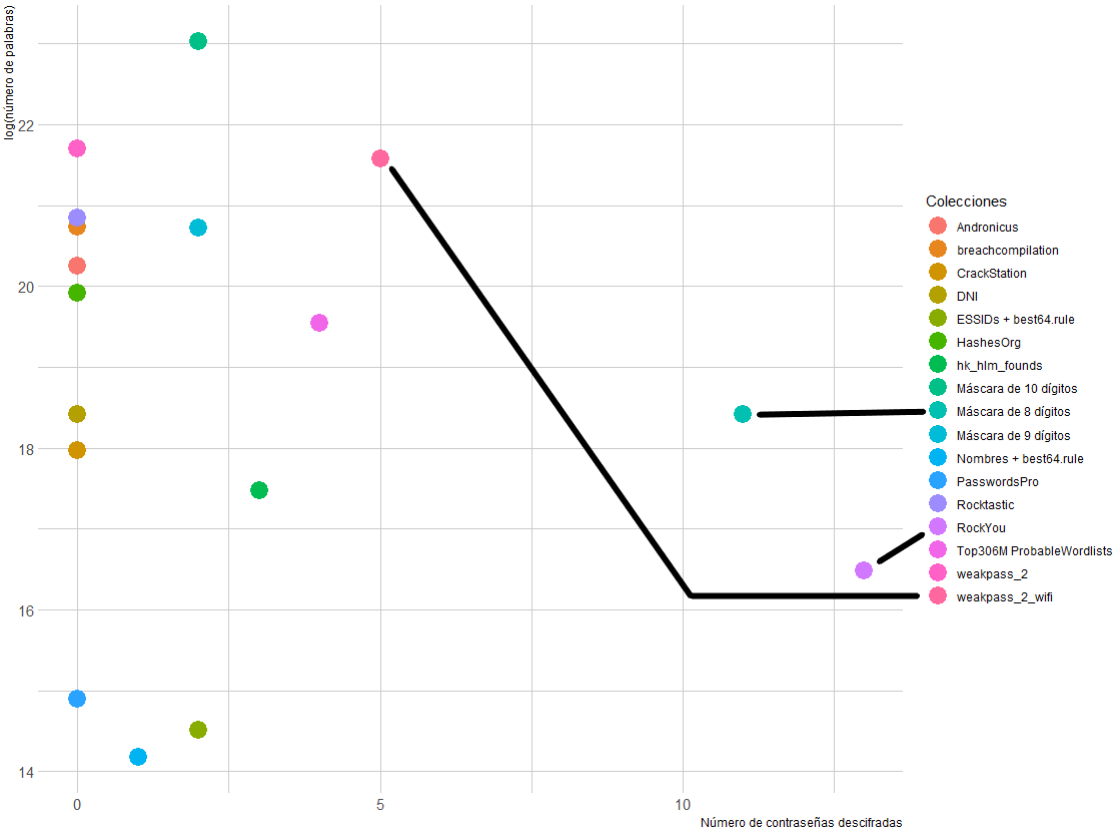

Figura 6.4: Relación entre el tamaño de las colecciones de palabras y contraseñas descifradas

En la figura 6.4 el tamaño de las colecciones se representa en escala logarítmica contra el ratio de éxito de cada una. Se han resaltado las tres colecciones que mejores resultados cosecharon. Es necesario utilizar el logaritmo natural dada la gran variación de tamaño entre colecciones, de la más pequeña a la más grande hay cuatro órdenes de magnitud.

El número final de contraseñas descifradas ascendió a 44, un 33,08% de éxito que encaja con otras pruebas empíricas ya mencionadas. Quizás el dato más preocupante sea que, siguiendo esta estrategia, en menos de dos días de computación ya se había recuperado el 22.13% de las contraseñas, aproximadamente dos tercios del total que se habrían de acabar descifrando. Como se puede comprobar visualmente en las figuras 6.5 y 6.6, en longitud dominan las contraseñas de 10 caracteres o menos y en tipos las numéricas y alfabéticas. No es de extrañar, ya que como se explicó en la sección 4.2 cuanto más aumenta la longitud o complejidad de las contraseñas, más difícil es descifrarlas.

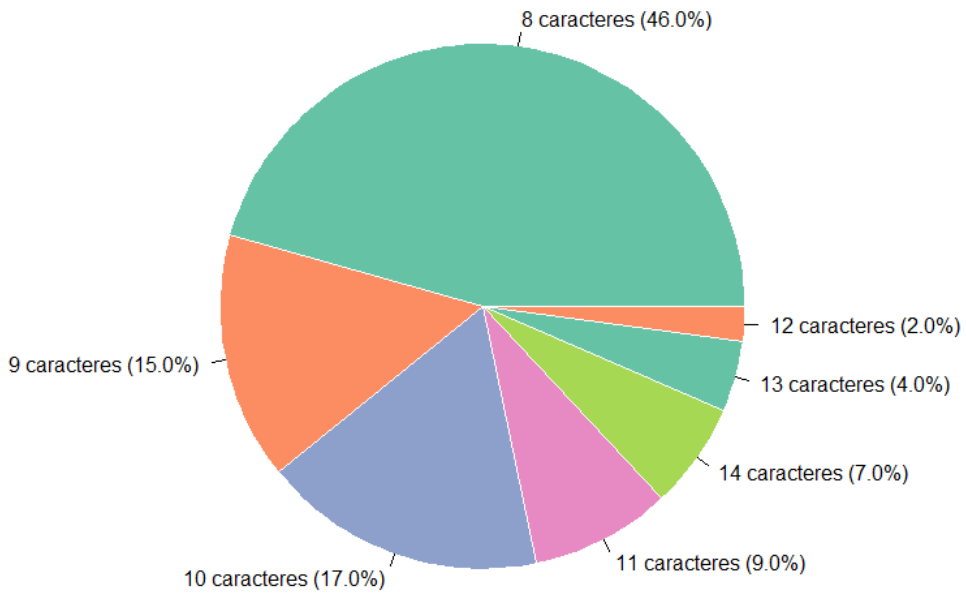

Figura 6.5: Distribución de longitudes de las contraseñas descifradas

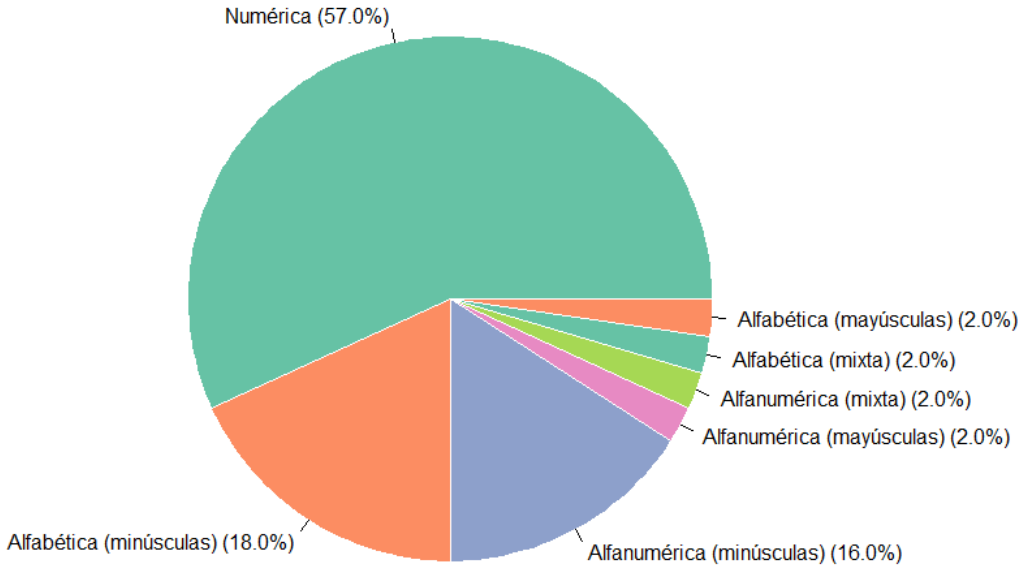

Figura 6.6: Distribución de tipos de las contraseñas descifradas

6.2 Análisis de patrones y elaboración de un diccionario mínimo

Un examen visual rápido de las contraseñas descifradas descubre un patrón subyacente a la gran mayoría. Para describirlo definimos dos conjuntos de palabras: las palabras base y las terminaciones. Para los conjuntos de palabras S_1 de palabras base y S_2 de terminaciones, una palabra podrá pertenecer al diccionario si se encuentra en el siguiente conjunto:

$$D = \{\, v_1 \dots v_n\, t \quad / \quad v \in S_1,\ t \in S_2,\ v_1 \neq \lambda,\ 8 \leq |v_1 \dots v_n\, t| \leq 63\}$$

Es decir, el conjunto de todas aquellas concatenaciones de palabras pertenecientes a los conjuntos S_1 y S_2 en las cuales la única palabra que no puede ser la tira nula (λ) es la primera (v_1) y siempre que la concatenación resultante tenga una longitud de entre 8 y 63 caracteres.

La unión de los siguientes conjuntos forma el conjunto S_1 de palabras base:

$B_{lambda} = \{\lambda\}$
$B_{nat} = \mathbb{N}$
$B_{nombres} =$ Conjunto de los 100 nombres más frecuentes en España según el INE
$B_{negocios} =$ Subconjunto aleatorio de los nombres de negocios de A Coruña
$B_{palabras} =$ Subconjunto de los sustantivos más usados según datos de la RAE

$$S_1 = B_{lambda} \cup B_{nat} \cup B_{nombres} \cup B_{negocios} \cup B_{palabras}$$

Los nombres de personas se obtuvieron de la lista de los 100 nombres de hombre y mujer más comunes según el INE. Sobre esos 200 nombres se aplicaron tres variaciones: poner todas las letras en mayúsculas, todas en minúsculas y sólo la primera en mayúscula. Por su parte los nombres de negocios se obtuvieron haciendo *scraping* de una conocida plataforma de mapas *online* y para ser aptos necesitaron una limpieza manual (eliminar espacios, caracteres no válidos, etc.).

Por su parte, el conjunto S_2 de terminaciones es la unión de los siguientes subconjuntos de naturales:

$T_{num} = \{u \quad / \quad u \in \mathbb{N},\ 0 \leq u \leq 9\}$
$T_{dia} = \{u \quad / \quad u \in \mathbb{N},\ 1 \leq u \leq 31\}$
$T_{mes} = \{u \quad / \quad u \in \mathbb{N},\ 1 \leq u \leq 12\}$
$T_{ano1} = \{u \quad / \quad u \in \mathbb{N},\ 0 \leq u \leq 20,\ 78 \leq u \leq 99\}$
$T_{ano2} = \{u \quad / \quad u \in \mathbb{N},\ 1978 \leq u \leq 1999,\ 2000 \leq u \leq 2020\}$

$$S_2 = T_{num} \cup T_{dia} \cup T_{mes} \cup T_{ano1} \cup T_{ano2}$$

Para el diccionario mínimo hace falta acotar más los patrones de las palabras. El diccionario mínimo, por su parte, se compone de la unión de los siguientes conjuntos de palabras:

$B_1 = \{w \quad / \quad w \in \mathbb{N},\ 10.000.000 \leq w \leq 19.999.999\}$
$B_2 = \{w \quad / \quad w \in \mathbb{N},\ 20.000.000 \leq w \leq 29.999.999\}$
$B_3 = \{w \quad / \quad w \in \mathbb{N},\ 269.000.000 \leq w \leq 270.999.999\}$
$B_4 = \{w^2 \quad / \quad w \in \mathbb{N},\ 0 \leq w \leq 9.999\}$
$B_5 = \{w^2 \quad / \quad w \in \mathbb{N},\ 0 \leq w \leq 99.999\}$
$B_6 = \{w^2 \quad / \quad w \in \mathbb{N},\ 0 \leq w \leq 999.999\}$
$B_7 = \{uvw \quad / \quad 1 \leq u \leq 31,\ 1 \leq v \leq 12,\ 1978 \leq w \leq 2020\}$
$B_8 = \{0123456789, 12345678, 123456789, 1234567890, 1223334444, 1122334455,$
$135792468, 1357924680, 246813579, 2468013579, 12345678910, 123456789123456789\}$
$B_9 = \{\ uv \quad / \quad u \in B_9,\ v \in$ Conjunto de letras mayúsculas y músculas ASCII$\}$
$B_{10} = B_{negocios}$
$B_{11} = B_{negocios} \times T_{ano2}$
$B_{12} = B_{negocios} \times T_{mes} \times T_{ano1}$
$B_{13} = B_{negocios} \times T_{num}$
$B_{14} = B_{13} \times T_{num}$
$B_{15} = B_{nombre}$
$B_{16} = B_{nombre} \times T_{ano2}$
$B_{17} = B_{nombre} \times T_{mes} \times T_{ano1}$
$B_{18} = B_{nombre} \times T_{num}$
$B_{19} = B_{18} \times T_{num}$
$B_{20} = \{vw \quad / \quad v \in B_{palabras},\ w \in T_{ano2},\ 4 \leq |v| \leq 7\}$
$B_{21} = \{vw \quad / \quad v \in B_{palabras},\ w \in B_8,\ 4 \leq |v| \leq 7\}$
$B_{22} = B_{palabras} \times B_{palabras}$
$B_{23} = B_{22} \times T_{num}$
$B_{24} = \{w \quad / \quad w \in B_{palabras},\ |w| \geq 8\}$
$B_{25} = B_{24} \times T_{num}\}$
$B_{26} = B_{25} \times T_{num}\}$

$$D_{min} = \{u \quad / \quad u \in \bigcup_{i=1}^{26} B_i,\ 8 \leq |u| \leq 63\}$$

Cada patrón de la lista generó un diccionario de palabras cuya longitud debía estar entre los 8 y 63 (ambos inclusive) caracteres ASCII para cumplir con los requisitos el protocolo WPA/WPA2. Tras unir todos los conjuntos quedó un único archivo de 288 megabytes. Sobre este archivo aún se ejecutó un último filtro para eliminar las palabras de más de 15 caracteres porque son estadísticamente insignificantes[8]. El archivo final es de 252 megabytes. Para poner en contexto el tamaño se puede ver en la figura 6.7 cómo se compara con los usados en este trabajo.

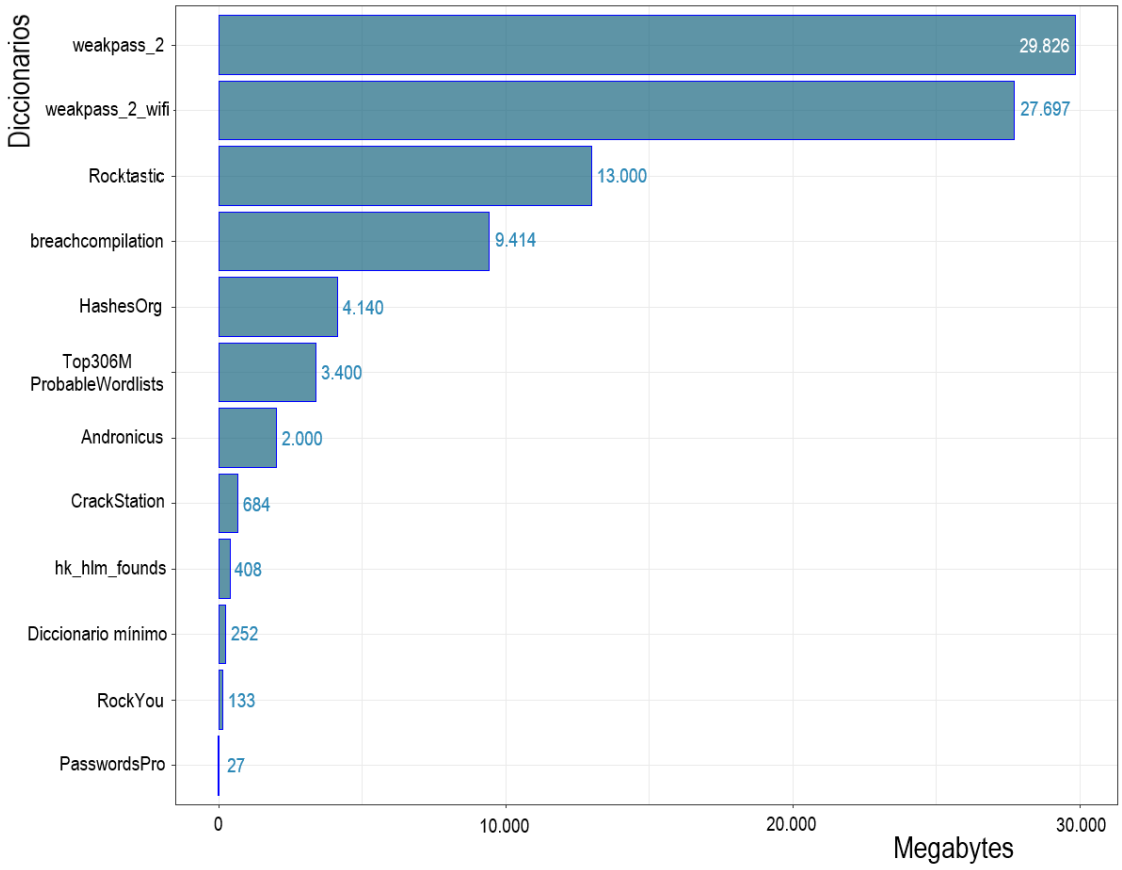

Figura 6.7: Comparativa de los tamaños de los diccionarios

[8] Smarter password cracking with PACK

Los cuadros 6.2 y 6.3 dan una perspectiva detallada de cómo se reparten en importancia los distintos subdiccionarios, así como los tipos de palabras que los componen.

Conjunto	Palabras
B_1	10.000.000
B_2	10.000.000
B_3	2.000.000
B_4	10.000
B_5	100.000
B_6	1.000.000
B_7	15.340
B_8	12
B_9	676
B_{10}	375
B_{11}	57.456
B_{12}	635.040
B_{13}	5.970
B_{14}	61.424
B_{15}	300
B_{16}	45.600
B_{17}	360.000
B_{18}	4.320
B_{19}	55.494
B_{20}	5.092
B_{21}	1.712.989
B_{22}	4.488
B_{23}	453.389
B_{24}	54
B_{25}	540
B_{26}	4.914

Cuadro 6.2: Reparto de cada conjunto en el diccionario mínimo

Tipo	Palabras
Numérico	23.125.351
Alfanumérico (minúsculas)	2.750.265
Alfanumérico (mayúsculas)	156.273
Alfanumérico (min/may)	504.097
Otro	5.809

Cuadro 6.3: Tipos de palabras en el diccionario mínimo

Esta misma información representada de una forma más visual se recoge a continuación en las figuras 6.8 y 6.9.

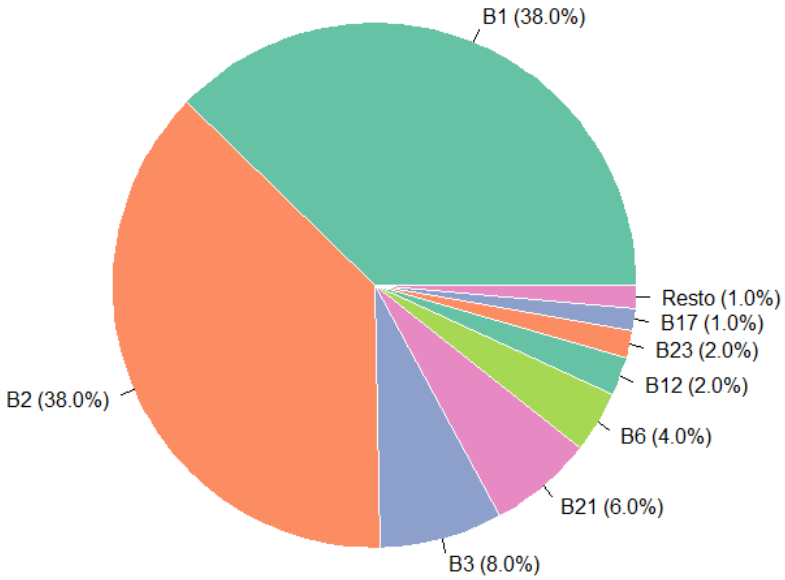

Figura 6.8: Proporción de cada patrón en el diccionario

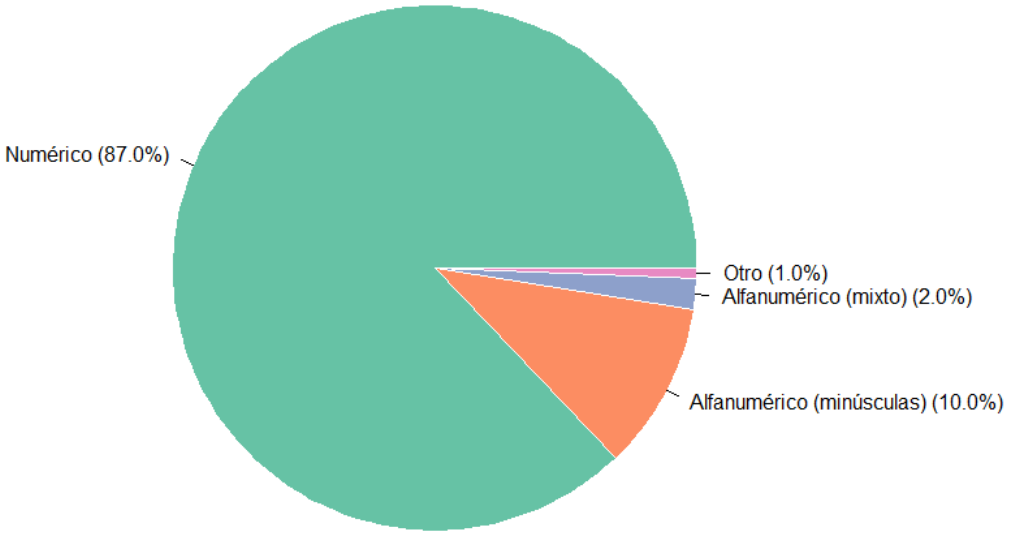

Figura 6.9: Proporción de cada tipo de palabra en el diccionario

6.3 Visualización e implicaciones de los resultados

En este último apartado antes de las conclusiones finales se mostrará de una forma visual y simple cómo se reparten por la zona auditada los puntos de acceso cuya contraseña ha sido comprometida. Después se hará un breve repaso a las implicaciones de romper esta primera barrera de seguridad.

6.3.1 Visualización

Gracias a que la herramienta desarrollada permitía guardar la información de localización de cada auditoría puede elaborarse una representación visual sobre mapa donde se muestra el alcance supuesto de cada punto de acceso del que se ha descifrado la contraseña. Aquí hay que hacer la precisión de que la posición donde se ha ubicado cada punto de acceso es realmente en donde se ha ejecutado *la auditoría*. Se da por buena porque en cada auditoría se esperó a que el GPS fijase posición con un margen de $7m$ o menos de error y porque sólo se fijaban objetivos con una buena calidad de señal (para aumentar las posibilidades de interceptar un *handshake* como se comentó en la sección 5.1.3).

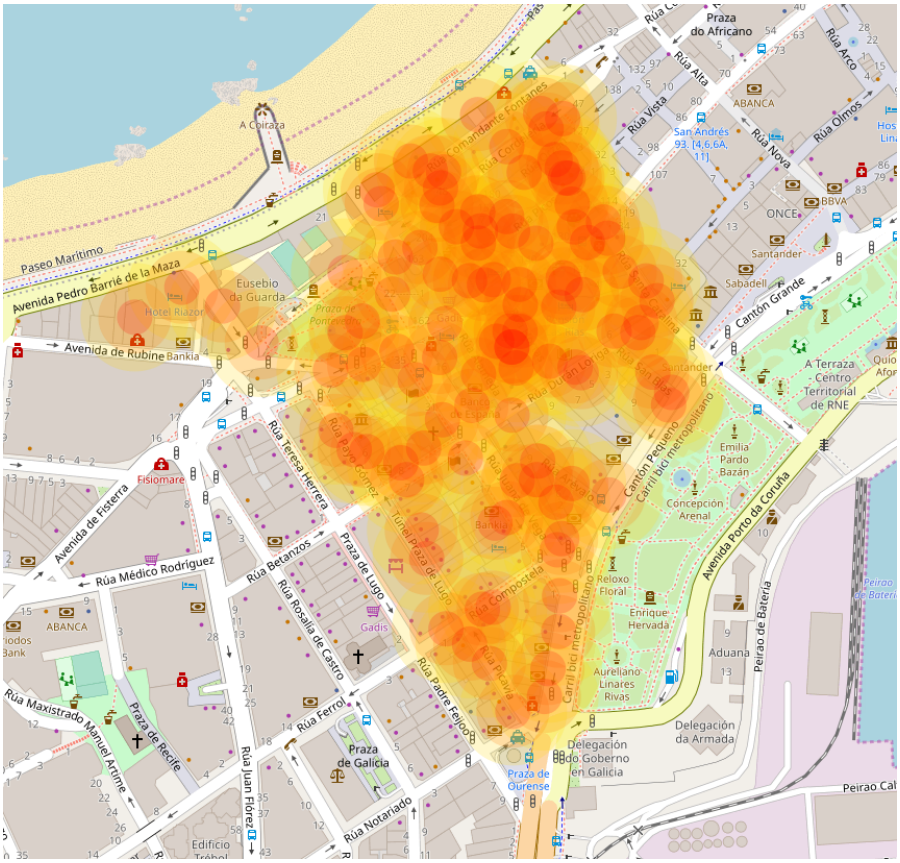

Figura 6.10: Mapa de calor de los routers de los que se ha descifrado la contraseña (Leaflet)

En las redes Wi-Fi el radio de alcance depende en gran medida del medio en el que se despliegue el router. La estimación que se ha hecho es muy básica ya que no tiene en cuenta elementos clave como obstáculos a la señal, altura (el punto de acceso puede estar en un piso alto de un edificio), modelo de router, etc. Puede llegar a variar de $15m$ en una casa a $100m$ al aire libre. Por simplicidad se han escogido los radios de 15, 30 y 45 metros para representar la calidad de la señal. Se puede comprobar que, disponiendo de un *handshake* por cada $795\,m^2$, y con un éxito medio en el descifrado de las contraseñas, de alrededor del 30%, se puede abarcar completamente la zona geográfica auditada. Sería posible recorrer toda la zona conectándose de router en router sin perder nunca la conexión a Internet.

6.3.2 Implicaciones

Las implicaciones en seguridad que supone descifrar las contraseñas de los routers varían en gravedad. Sin ánimo de exhaustividad las consecuencias de accesos no autorizados se podrían clasificar de menor a mayor importancia como en [3]:

- Consumo de ancho de banda: El ancho de banda de las redes Wi-Fi puede verse claramente mermado o incluso puede que las redes se vuelvan completamente inutilizables, provocando lo que se conoce como denegación de servicio (DoS o *denial of service*).
- Acceso no autorizado a comunicaciones y equipos: En general, las protecciones frente a equipos externos a la red local suelen ser más fuertes que aquellas que se aplican frente a equipos que pertenecen a la misma red local. Las comunicaciones pueden verse comprometidas a través de ataques *man-in-the-middle* (MITM), *hijacking* de DNS, redirecciones a páginas maliciosas, etc [33].
- Convertirse en el atacante: Un equipo comprometido puede servir como equipo atacante de sistemas remotos[9], tanto de ataques dirigidos como masivos si se integra en una *botnet*. En zonas urbanas densamente pobladas, los Wi-Fi forman una red de proximidad estrechamente interconectada que puede explotarse como sustrato para la propagación de malware capaz de lanzar ataques fraudulentos masivos [33] [34]. La seguridad de los routers Wi-Fi tiene implicaciones más allá del propio dispositivo.

[9] Con las posibles repercusiones legales que ello puede conllevar.

Capítulo 7

Conclusiones

En este trabajo se ha realizado un estudio de los patrones de contraseñas de redes Wi-Fi dentro del área metropolitana de A Coruña, escogiendo como representante la zona del Centro a la altura de los Jardines de Méndez Núñez. El muestreo se realizó de forma mayoritariamente aleatoria con una herramienta desarrollada específicamente para la tarea. Utilizando hardware y software de altas prestaciones se encontraron patrones subyacentes a las contraseñas que permitieron la generación de un diccionario mínimo específico para el Área Metropolitana de A Coruña.

La herramienta software que se implementó demostró ser suficientemente robusta y automática como para conseguir realizar una auditoría cada tres minutos e interceptar *handshakes* en un 40% de las ellas, además de permitir la mejora de estos números conforme aumentan las tarjetas de red disponibles, haciéndola también una opción escalable. Como trabajo futuro podría plantearse un mayor grado de automatización haciendo que el *script* por sí solo consiga las interfaces de las tarjetas de red y las coordenadas GPS si hay algún dispositivo conectado que las pueda obtener.

A su vez, el diccionario elaborado a raíz de los análisis de patrones permite descifrar más del triple de contraseñas (40) que el diccionario que mejores resultados cosechó (RockYou, 13) con menos del doble del tamaño, 256 megabytes. A pesar de esto hay que mencionar las limitaciones de generar un diccionario de patrones con una muestra tan pequeña. Una línea de trabajo futuro puede ser comprobar empíricamente la efectividad de este diccionario contra los otros que se han usado durante este trabajo en un área de más tamaño.

Por otra parte, si bien es cierto que la ideoneidad de las contraseñas como método de autenticación de usuarios está en disputa, parece que para la autenticación en redes Wi-Fi es óptima. Dejando la responsbilidad de recordar y almacenar de forma segura al sistema operativo del dispositivo, el usuario puede permitirse elegir una contraseña larga y compleja ante la comodidad de sólo introducirla una vez por dispositivo. Es por esto que, conociendo las posibilidades actuales en computación a nivel de consumo y el software de "crackeo" disponible se pueden hacer las siguientes recomendaciones:

- Las contraseñas numéricas de menos de 12 caracteres son inseguras.
- Las contraseñas por defecto de routers antiguos son inseguras, mejor cambiarlas siguiendo las recomendaciones que se mencionan a continuación.
- Si se cambia la contraseña de un router,
  - no escatimar en longitud y complejidad de la contraseña, el límite son 63 caracteres y la seguridad que se gana compensa en gran medida la incomodidad de introducirla una vez en cada dispositivo.
  - comprobar que la contraseña no ha aparecido en ninguna filtración con servicios como Pwned Passwords de ';–have i been pwned?.
- A 2019, para un usuario medio con un nivel medio de amenaza, cualquier combinación que genere un espacio de búsqueda superior a 100 billones de posibilidades ($VR_{x,y} > 10^{14}$) y que no utilice palabras de diccionario puede considerarse segura.
- Si se siguen estas recomendaciones y se crea una contraseña que es imposible de recordar, para un usuario medio es buena idea escribirla en un papel y guardarla en un lugar seguro.

Finalmente, con los resultados de este trabajo se puede concluir que la seguridad de las contraseñas del Área Metropolitana de A Coruña se sitúa en la media. El 33,08% de contraseñas descifradas entra en el rango de la mayoría de los estudios (20–50%) [18] y encaja con los números que manejan las herramientas online de descifrado masivo como gpuhash.me y wpa-sec. Con todo, aproximadamente dos tercios de este 33,08% son contraseñas absolutamente inseguras que deberían de cambiarse de inmediato.

Apéndice A

Instalación y configuración de la herrramienta

La última versión del *script* puede descargarse para Linux con el siguiente comando:

```
git clone https://github.com/PabloGalegoC/Wifi-Hashing
```

La jerarquía de carpetas es la siguiente:

```
Wifi-Hashing
├── data
│   ├── all
│   ├── clients
│   ├── handshakes
│   │   └── hashcat
│   ├── routers
│   └── audit_database.sqlite
├── src
│   ├── config.py
│   ├── fromblobtohashcat.py
│   ├── cap2hccapx.bin
│   └── wifihashing.py
└── setup.yml
```

Antes de ejecutar wifihashing.py es necesario cumplir todas las dependencias:

- Instalar el paquete ”pandas”.
- Instalar la suite aircrack-ng.

Para la Raspberry Pi 3 A+ es posible usar el archivo setup.yml que se ocupa de instalar todas las dependencias y prepararla para la ejecución del *script* a través de "ssh". El archivo de *setup* está escrito en YAML y se trata de un *playbook* de Ansible muy sencillo. Para ejecutarlo con el siguiente comando es necesario que "ansible" esté instalado.

```
ansible−playbook setup.yml −K
```

Las instalaciones a través de "pip" de paquetes "grandes" en la Raspberry Pi pueden demorarse demasiado porque han de ser compilados. Una de las ventajas de provisionar la Raspberry con este *playbook* es que se ocupa de descargar e instalar una versión precompilada para la arquitectura ARM del paquete "pandas" proporcionada por piwheels. La diferencia entre instalar la versión normal contra la precompilada es de ocho horas contra menos de cinco minutos.

Después es necesario comprobar que todo en el archivo de configuración config.py se ajusta a las necesidades de la auditoría. Como mínimo ha de modificarse el valor de NETWORK_CARDS con los nombres de las antenas que se vayan a usar.

```
NETWORK_CARDS = ["wlan0"]
```

Modificar las demás opciones de configuración es opcional:

- *Excluded MAC adresses*, la lista de direcciónes MAC a las que no se atacará. Por defecto vacía.
- *Search time*, tiempo en segundos buscando routers objetivo. Por defecto 20.
- *Max attempts*, por defecto cada auditoría acabará desconectando a todos los clientes, esta variable define cuántas veces ocurrirá. Por defecto 1.
- *Stealth deauth*, es un valor booleano que controla el comportamiento del *script* a la hora de deautenticar clientes del punto de acceso. Si está en "True", la deautenticación será "educada": se deautentica un cliente y se espera a conseguir un handshake, sólo en el caso de que no se consiga se deautentica al siguiente. Si está en "False" la deautenticación será "bruta": todos los clientes serán desconectados del punto de acceso a la vez. Por defecto "False".
- *Deauth wait*, el tiempo en segundos que esperará a que el cliente o clientes (dependiendo del valor de *stealth deauth*) se reconecten al punto de acceso para interceptar el *handshake*. Por defecto 30.

  Si *Stealth deauth* se ha marcado como "True":

  - *Max client deauth*, el problema de ir uno a uno desconectando clientes es que en redes con muchos clientes el *script* podría tardar demasiado en ejecutar una auditoría. Esta variable limita el comportamiento por defecto del *script* de deautenticar por defecto a todos los clientes. Por defecto 15.

  - *Deauth wait reduction*, también si desconectamos clientes uno a uno, cada vez que otro más se deautentica las posibilidades de interceptar un *handshake* aumentan, por tanto es interesante también que a cada nuevo cliente desconectado el tiempo que se espera a interceptar un *handshake se reduzca*. Por defecto 0,7. Esto significa que si por la reconexión del primer cliente se ha esperado el valor por defecto (30 segundos), por el siguiente se esperará $30 \times 0.7 = 21$ segundos.
    Con estos valores cada auditoría duraría alrededor de 100 segundos en el peor de los casos.

- *Min pwr*, el mínimo de señal (dB) para intentar interceptar un *handshake*. En aircrack-ng se representa desde -1 (mínima potencia de señal) a -100 (máxima). Por defecto -20.

- *Db relative path*, la ruta relativa respecto a la carpeta "src" en donde se guardará el archivo de la base de datos. Por defecto en la raíz de la carpeta "data".

Ahora ya puede ejecutarse "wifihashing.py" con el comportamiento deseado. Como se indica en la sección 5.1.3, hay que usar el siguiente comando:

```
sudo python3 wifihashing.py
```

Los *handshakes* capturados se almacenan en la base de datos como *blobs*, para extraerlos y transformarlos al formato que acepta Hashcat se puede usar la pequeña utilidad "fromblob-tohashcat.py". Los handshakes en formato "hccapx" se guardan en Wifi-Hasing/data/handshakes/hashcat.

Apéndice B

Glosario de acrónimos

AP *Access point* o punto de acceso, popularmente el router.

STA *Station*, también se le suele llamar *supplicant* o cliente, por ejemplo smartphones, portátiles, televisiones, etc.

PSK *Pre-Shared Key* o clave compartida previamente, la contraseña del punto de acceso. "Clave compartida previamente" entendida como "clave que ambos conocen antes de comenzar el proceso de *handshake*".

PMK *Pairwise Master Key* o contraseña maestra de la pareja, donde la "pareja" son punto de acceso y cliente.

SSID *Service Set IDentifier* o identificador del conjunto de servicios, se corresponde con el nombre del punto de acceso.

ANonce *Authenticator Number used once* o número del autenticador usado una vez, el autenticador es el punto de acceso.

SNonce *Supplicant Number used once* o número del suplicante usado una vez, el suplicante es el cliente.

AMA *Authenticator Mac Address* o dirección MAC del autenticador. La dirección MAC identifica de forma única a una tarjeta o dispositivo de red.

SMA *Supplicant Mac Address* o dirección MAC del suplicante.

PTK *Pairwise Transient Key* o clave transitoria de la pareja.

PRF *Pseudo Random Function* o función pseudoaleatoria, genera un número pseudoaleatorio.

MIC *Message Integrity Code* o código de integridad del mensaje.

GTK *Group Temporal Key* o Clave temporal de grupo.

ACK *Acknowledgement* o acuse de recibo.

PBKDF2 *Password-Based Key Derivation Function 2* o función de derivación de clave basada en contraseña versión 2.

HMAC–SHA1 función de *hashing* SHA1 usando HMAC como código de autenticación de mensajes.

Bibliografía

[1] INE, "Encuesta sobre equipamiento y uso de tecnologías de información y comunicación en los hogares," https://www.ine.es/jaxi/Datos.htm?path=/t25/p450/base_2011/a2018/l0/&file=01001.px, Nov 2018, accedido a 26-07-2019.

[2] W.-F. Alliance, "Infographic 20 years of wi-fi," https://www.wi-fi.org/sites/default/files/public/Infographic_20_years_of_Wi-Fi.pdf, 2019, accedido a 26-07-2019.

[3] R. Castro, "Avanzando en la seguridad de las redes wifi," vol. 73, pp. 23–33, 2005.

[4] S. Fluhrer, I. Mantin, and A. Shamir, "Weaknesses in the key scheduling algorithm of rc4," in *Selected Areas in Cryptography*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 1–24.

[5] M. Beck and E. Tews, "Practical attacks against wep and wpa," *IACR Cryptology ePrint Archive*, vol. 2008, p. 472, Jan 2008.

[6] J. Huang, J. Seberry, W. Susilo, and M. Bunder, "Security analysis of michael: The ieee 802.11i message integrity code," in *Embedded and Ubiquitous Computing - EUC 2005 Workshops*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 423–432.

[7] M. Vanhoef and Frank Piessens, "Predicting, decrypting, and abusing wpa2/802.11 group keys," in *25th USENIX Security Symposium (USENIX Security 16)*. Austin, TX: USENIX Association, Ago 2016, pp. 673–688. [Online]. Available: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/vanhoef

[8] M. Vanhoef and F. Piessens, "Key reinstallation attacks: Forcing nonce reuse in wpa2," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17. New York, NY, USA: ACM, 2017, pp. 1313–1328. [Online]. Available: http://doi.acm.org/10.1145/3133956.3134027

[9] M. Vanhoef and E. Ronen, "Dragonblood: A security analysis of wpa3's sae handshake." *IACR Cryptology ePrint Archive*, vol. 2019, p. 383, 2019.

[10] S. Egelman, J. Bonneau, S. Chiasson, D. Dittrich, and S. Schechter, "It's not stealing if you need it: A panel on the ethics of performing research using public data of illicit

origin," in *Financial Cryptography and Data Security*, J. Blyth, S. Dietrich, and L. J. Camp, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 124–132.

[11] D. R. Thomas, S. Pastrana, A. Hutchings, R. Clayton, and A. R. Beresford, "Ethical issues in research using datasets of illicit origin," in *Proceedings of the 2017 Internet Measurement Conference*, ser. IMC '17. New York, NY, USA: ACM, 2017, pp. 445–462. [Online]. Available: http://doi.acm.org/10.1145/3131365.3131389

[12] WPEngine, "Unmasked: What 10 million passwords reveal about the people who choose them," https://wpengine.com/unmasked/, accedido a 26-07-2019.

[13] S. Riley, "Password security: What users know and what they actually do," *Usability News*, vol. 8, no. 1, pp. 2833–2836, 2006.

[14] Google, "Online security survey," https://services.google.com/fh/files/blogs/google_security_infographic.pdf, Feb 2019, accedido a 26-07-2019.

[15] Wikipedia, "Ieee 802.11i-2004," http://en.wikipedia.org/w/index.php?title=IEEE%20802.11i-2004&oldid=902359769, 2019, accedido a 26-07-2019.

[16] ——, "Embarrassingly parallel," http://en.wikipedia.org/w/index.php?title=Embarrassingly%20parallel&oldid=905597318, 2019, accedido a 29-07-2019.

[17] R. Morris and K. Thompson, "Password security: A case history," *Communications of the ACM*, vol. 22, 05 2002.

[18] J. Bonneau, "The science of guessing: Analyzing an anonymized corpus of 70 million passwords," in *2012 IEEE Symposium on Security and Privacy*, May 2012, pp. 538–552.

[19] G. Notoatmodjo and C. Thomborson, "Passwords and perceptions," *Conferences in Research and Practice in Information Technology Series*, vol. 98, pp. 71–78, 01 2009.

[20] M. Weir, S. Aggarwal, M. Collins, and H. Stern, "Testing metrics for password creation policies by attacking large sets of revealed passwords," 01 2010, pp. 162–175.

[21] A. Forget, S. Chiasson, P. C. van Oorschot, and R. Biddle, "Improving text passwords through persuasion," 01 2008, pp. 1–12.

[22] K. Bryant and J. Campbell, "User behaviours associated with password security and management," *Australasian Journal of Information Systems*, vol. 14, 11 2006.

[23] J. Yan, A. Blackwell, R. Anderson, and A. Grant, "Password memorability and security: empirical results," *IEEE Security Privacy*, vol. 2, no. 5, pp. 25–31, Sep. 2004.

[24] S. Komanduri, R. Shay, P. G. Kelley, M. L. Mazurek, L. Bauer, N. Christin, L. F. Cranor, and S. Egelman, "Of passwords and people: Measuring the effect of password-composition policies," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '11. New York, NY, USA: ACM, 2011, pp. 2595–2604. [Online]. Available: http://doi.acm.org/10.1145/1978942.1979321

[25] A. Narayanan and V. Shmatikov, "Fast dictionary attacks on passwords using time-space tradeoff," 01 2005, pp. 364–372.

[26] A. Adams and A. Sasse, "Users are not the enemy," *Commun. ACM*, vol. 42, pp. 40–46, 12 1999.

[27] M. Mazurek, S. Komanduri, T. Vidas, L. Bauer, N. Christin, L. Cranor, P. Kelley, R. Shay, and B. Ur, "Measuring password guessability for an entire university," 11 2013.

[28] B. Schneier, "Write down your password," http://www.schneier.com/blog/archives/2005/06/write_down_your.html, 06 2005, accedido a 12-08-2019.

[29] W. E. Burr, D. F. Dodson, and W. Polk, "Sp 800-63-1. electronic authentication guideline," 2006.

[30] C. Castelluccia, M. Dürmuth, and D. Perito, "Adaptive password-strength meters from markov models," 02 2012.

[31] J. Gosney, "8x nvidia gtx 1080 hashcat benchmarks," https://gist.github.com/epixoip/a83d38f412b4737e99bbef804a270c40, 2016, accedido a 26-07-2019.

[32] RAE, "Corpus de referencia del español actual," http://corpus.rae.es/lfrecuencias.html, accedido a 26-07-2019.

[33] H. Kavak, J. J Padilla, D. Vernon-Bido, S. Diallo, and R. Gore, "The spread of wi-fi router malware revisited," 05 2017.

[34] H. Hu, S. Myers, V. Colizza, and A. Vespignani, "Wifi networks and malware epidemiology," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 106, pp. 1318–23, 02 2009.